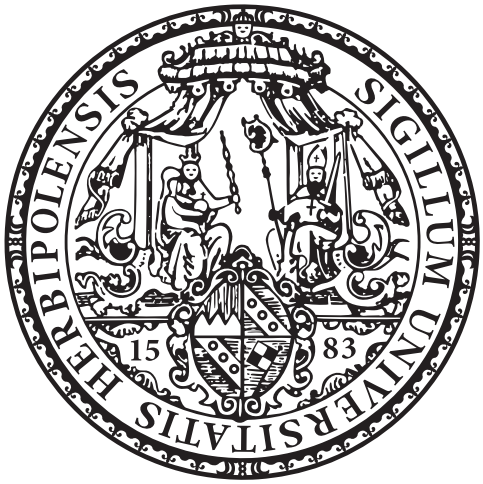


Detecting Anomalies in Transaction Data

vorgelegt von

Daniel Schlör

Dissertation zur Erlangung des naturwissenschaftlichen
Doktorgrades der Julius-Maximilians-Universität Würzburg

Würzburg, 2022

Abstract

Detecting anomalies in transaction data is an important task with a high potential to avoid financial loss due to irregularities deliberately or inadvertently carried out, such as credit card fraud, occupational fraud in companies or ordering and accounting errors. With ongoing digitization of our world, data-driven approaches, including machine learning, can draw benefit from data with less manual effort and feature engineering. A large variety of machine learning-based anomaly detection methods approach this by learning a precise model of normality from which anomalies can be distinguished. Modeling normality in transactional data, however, requires to capture distributions and dependencies within the data precisely with special attention to numerical dependencies such as quantities, prices or amounts.

To implicitly model numerical dependencies, Neural Arithmetic Logic Units have been proposed as neural architecture. In practice, however, these have stability and precision issues. Therefore, we first develop an improved neural network architecture, iNALU, which is designed to better model numerical dependencies as found in transaction data. We compare this architecture to the previous approach and show in several experiments of varying complexity that our novel architecture provides better precision and stability. We integrate this architecture into two generative neural network models adapted for transaction data and investigate how well normal behavior is modeled. We show that both architectures can successfully model normal transaction data, with our neural architecture improving generative performance for one model.

Since categorical and numerical variables are common in transaction data, but many machine learning methods only process numerical representations, we explore different representation learning techniques to transform categorical transaction data into dense numerical vectors. We extend this approach by proposing an outlier-aware discretization, thus incorporating numerical attributes into the computation of categorical embeddings, and investigate latent spaces, as well as quantitative performance for anomaly detection.

Next, we evaluate different scenarios for anomaly detection on transaction data. We extend our iNALU architecture to a neural layer that can model both numerical and non-numerical dependencies and evaluate it in a supervised and one-class setting. We investigate the stability and generalizability of our approach and show that it outperforms a variety of models in the balanced supervised setting and performs comparably in the one-class setting. Finally, we evaluate three approaches to using a generative model as an anomaly detector and compare the anomaly detection performance.

Zusammenfassung

Die Erkennung von Anomalien in Transaktionsdaten ist eine wichtige Zielsetzung mit hohem Potenzial finanzielle Verluste zu vermeiden, die auf absichtlich oder versehentlich begangenen Unregelmäßigkeiten wie beispielsweise Kreditkartenbetrug oder Bestell- und Abrechnungsfehler gründen. Mit der fortschreitenden Digitalisierung können datengetriebene Ansätze einschließlich maschinellen Lernens mit immer weniger manuellem Aufwand Nutzen aus den Daten ziehen. Viele Methoden zur Erkennung von Anomalien, die auf maschinellem Lernen basieren, verfolgen diesen Ansatz, indem sie ein präzises Modell der normalen Daten erlernen, mit dem sich dann Anomalien davon unterscheiden lassen. Die Modellierung von normalen Transaktionsdaten erfordert jedoch eine genaue Erfassung von Verteilungen und Abhängigkeiten innerhalb der Daten mit besonderem Augenmerk auf numerischen Abhängigkeiten von beispielsweise Mengen oder Geldbeträgen.

Zur impliziten Modellierung numerischer Abhängigkeiten wurden Neural Arithmetic Logic Units als neuronale Architektur vorgeschlagen. In der Praxis haben diese jedoch Stabilitäts- und Präzisionsprobleme. Daher entwickeln wir zunächst eine verbesserte neuronale Netzwerkarchitektur, iNALU, die darauf ausgelegt ist, numerische Abhängigkeiten, wie sie in Transaktionsdaten vorkommen, besser zu modellieren. Wir vergleichen diese Architektur mit ihrer Vorläuferarchitektur und zeigen in mehreren Experimenten, dass unsere Architektur höhere Präzision und Stabilität bietet. Wir integrieren unsere Architektur in zwei generative neuronale Netzmodelle, die für Transaktionsdaten angepasst wurden, und untersuchen, wie gut Normalverhalten modelliert wird. Wir zeigen, dass beide Architekturen normale Daten erfolgreich modellieren können, wobei die in dieser Arbeit vorgestellte neuronale Architektur die generativen Ergebnisse für ein Modell verbessert.

Da kategorische und numerische Variablen in Transaktionsdaten häufig zusammen vorkommen, viele Methoden des maschinellen Lernens jedoch nur numerische Repräsentationen verarbeiten, untersuchen wir verschiedene Techniken des Repräsentationslernens, um kategorische Transaktionsdaten in dichte numerische Vektoren zu transformieren. Wir erweitern diese, indem wir einen Diskretisierungsansatz vorschlagen, der Ausreißer berücksichtigt. Damit werden Zusammenhänge numerischer Datentypen in die Berechnung kategorischer Einbettungen einbezogen, um die Anomalieerkennung insgesamt zu verbessern.

Schließlich evaluieren wir verschiedene Szenarien für die Erkennung von Anomalien in Transaktionsdaten. Wir erweitern unsere iNALU-Architektur zu einer neuronalen Schicht, die sowohl numerische als auch nicht-numerische Abhängigkeiten modellieren kann. Wir untersuchen die Stabilität und Verallgemeinerbarkeit unseres Ansatzes und zeigen, dass er mehrere Modelle im überwachten Szenario übertrifft und im Ein-Klassen-Szenario vergleichbare Ergebnisse liefert. Mit drei Ansätzen, generative Modelle als Anomalie-Detektor zu nutzen, schließen wir schließlich unsere Untersuchung ab und liefern damit einen Beitrag zukünftig Anomalien in Transaktionsdaten besser aufzufinden.

Contents

List of Figures

List of Tables

List of Abbreviations

ACC Accuracy . . . 117
ACE Arithmetic Expression Calculation . . . 14
AD Anomaly Detection . . . 19
AE Auto-Encoder . . . 35
AI Artificial Intelligence . . . 18
AP Average Precision . . . 38
AUC Area Under the Curve . . . 37
CBOW Continuous Bag of Words . . . 113
CDF Cumulative Distribution Function . . . 99
DT Decision Tree . . . 30
EBM Energy Based Models . . . 16
ELBO Evidence Lower Bound . . . 79
EM Earth-Mover . . . 76
ERP Enterprise Resource Planning . . . 17
FID Fréchet Inception Distance . . . 86
FN False Negatives . . . 36
FP False Positives . . . 36
FPR False Positive Rate . . . 36
GAN Generative Adversarial Network . . . 6
GMM Gaussian Mixture Model . . . 92
GP Gradient Penalty . . . 76
IF Isolation Forest . . . 32
iNALU improved Neural Arithmetic Logic Unit . . . 6
ItG Inverting the Generator . . . 156
JSD Jensen-Shannon Distance
JSON JavaScript Object Notation . . . 51
KDD Knowledge Discovery in Databases . . . 21
KDE Kernel Density Estimation . . . 82
KL Kullback–Leibler . . . 79
kNN *k*-Nearest Neighbors . . . 27

LR Logistic Regression . . . 26
LSTM Long Short-Term Memory . . . 117
MCMC Markov Chain Monte Carlo . . . 17
ML-AE Mixed Layer AE . . . 151
MLE Maximum Likelihood Estimation . . . 79
ML Machine Learning . . . 4
MLP Multi-Layer Perceptron . . . 22
MSE Mean Squared Error . . . 6
NALU Neural Arithmetic Logic Unit . . . 15
NB Naïve Bayes . . . 26
NLP Natural Language Processing . . . 7
NN Neural Network . . . 13
OCC One-Class Classification . . . 35
OCL One-Class Learning . . . 22
OC-SVM One-Class SVM . . . 29
OLTP Online Transaction Processing . . . 1
OOV Out-of-vocabulary . . . 126
PCA Principal Component Analysis . . . 5
PDF Probability Density Function . . . 83
PR Precision-Recall . . . 37
PU Positive Unlabeled Learning . . . 35
QOQD Quantile Outlier Quantile Discretization . . . vii
QOUD Quantile Outlier Uniform Discretization . . . vii
RBF Radial-Basis-Function . . . 29
ReLU Rectified Linear Unit . . . 23
RF Random Forest . . . 31
RL Representation Learning . . . 7
ROC Receiver Operating Characteristic . . . 37
RQ Research Question . . . 5
SGD Stochastic Gradient Decent . . . 25
SMOTE Synthetic Minority Oversampling TEchnique . . . 34
SVM Support Vector Machine . . . 28

TN True Negatives 36
TPR True Positive Rate 36
TP True Positives 36
VAE Variational Auto-Encoder x
VGM Variational Gaussian Mixture Model 92
WGAN Wasserstein GAN ix

Chapter 1

Introduction

With advancing digitalization of our lives, data is generated everywhere. In addition to data, that is constantly and visibly being generated in computer systems as they are used, this also more and more affects traditional areas that are being digitized, such as the financial world. With credit cards being established in the 1950s, this subarea was already digitized at a very early stage [334]. However, with the convenient digital execution of payment transactions all over the world in almost real time, this area in particular has become a profitable target for criminals who abuse the system at the expense of credit institutions and customers, for example, by stealing or copying credit cards or credit card data [109, 73]. With the digital availability of credit card transaction data and the need to protect their business interests and those of their customers, a fraud detection application scenario was established as a research topic early on [109, 300, 92], which can be assigned to the topic of anomaly detection in transaction data. The application area most frequently associated with transactions is financial transactions, in which a transaction describes the exchange of financial assets or, more generally, the mutual change in the financial status of at least two parties. Transaction data hereby describes the data that is generated when a transaction is recorded.

More generally, however, a transaction can also be understood as an (atomic) action that changes data within a database and thus encompasses a number of other domains as well. Due to the requirements of parallel access, data consistency, response time and data throughput raised with the earliest systems for digitizing business processes, the term *transaction* with relation to data was coined in the 1970s by IBM's Transaction Processing Facility, a mainframe computer system used by airlines and credit card institutions [299], and later by Online Transaction Processing (OLTP) systems, which were often used in the day-to-day business of companies [100]. Therefore, the term transaction data is also used broadly in research, from financial transactions in the original sense, for example credit card transactions [261], to systems or datasets that digitize business processes, e.g. SAP S/3 [226], for "market basket data and web usage data" [348] or even "gene expression properties" [30].

In the context of this work, we consider transaction data mainly as logs of business processes, which includes credit card transactions, mobile payment logs, and SAP data, although the methods can also be applied to a broader understanding of transactions, which we investigate for example for transaction representations in Microsoft Windows audit logs.

The detection of anomalies is a topic that plays a major role with transaction data and can be understood in the following sense: Commonly one understands an *anomaly*[1] as something that deviates from the rule, is abnormal or in another way does not fit.

Anomaly detection is described in scientific literature as:

> Anomalies are patterns in data that do not conform to a well defined notion of normal behavior. [...] Anomalies might be induced in the data for a variety of reasons, such as malicious activity, for example, credit card fraud, cyber-intrusion, terrorist activity or break-down of a system, but all of the reasons have the common characteristic that they are interesting to the analyst.
>
> Chandola et al., Anomaly detection: A survey [50]

The focus is thereby on the deviation from something normal, which the anomaly is considered in relation to and antonymously characterized by. This view has a great influence on the specific identification of anomalies, as illustrated in the following examples: A person regularly traveling through the world for business will most likely require a different perspective of *normal* credit card transactions than a person who does not travel in addition to vacations. Two different products might have very different material costs and thus different definitions of anomalies for the raw material or retail price. A large number of sales before Christmas, a purposeful price drop for Black Friday or for seasonal products after the season might be considered normal from a business view while appearing anomalous from a purely data-driven perspective. When the costs to prosecute a specific potentially fraudulent anomaly is generally higher than tolerating the expected loss, from a business perspective the transaction should not be reported and blocked as potential anomaly, although from a data-driven perspective the transaction might by considered anomalous [73]. On the other hand, for a medical anomaly detection task, such a consideration is highly impractical and, in fact, unethical. These examples suggest that the definition of anomalies is domain-dependent, subjective and task-dependent.

Additionally, these examples emphasize the important aspect that such data generally contains dependencies which have to be addressed to precisely model data characteristics, which are often relatable to **numerical or arithmetical relations, or dependencies** of features. Neural networks have been shown to be universal approximators [99], in practice, however, they lack generalization and extrapolation

[1] Oxford Dictionaries: "a thing, situation, etc. that is different from what is normal or expected"

of even the most simple arithmetic relations [168, 317]. Modeling such dependencies including extrapolation beyond the training data is therefore an important and challenging aspect, which we will focus on in this thesis.

From an economic point of view, detecting anomalous transactions of various types is a highly rewarding endeavor. The definition by Chandola et al. already contains several examples for which anomaly detection is applied. The associated problems are highly relevant for the respective sectors: According to the Nilson report, card fraud globally accumulated losses of $28.58 billion in 2020, effectively ranging at 6.78 cents per $100 total volume [220]. In 2020, the Association of Certified Fraud Examiners [3] estimated the average financial loss due to fraud on companies at 5% of revenue each year, while the global cost of cyber crime in general, which in addition to online fraud also includes hacking, ransomware, and other 'cyber' related issues was estimated up to $600 billion in the cybercrime report by McAfee [204]. Besides being interesting from a definitional or academic point of view, anomalies and their detection are highly relevant from an economic perspective, as associated problems have a high economic impact on companies and individuals.

With anomaly detection and its application, several challenges arise that introduce complexity and need to be addressed. The most obvious challenge is **rarity of anomalies in practice**. While not explicitly demanded per definition, anomalies are in general implicitly required to be less frequent than non-anomalies as it otherwise didn't comply with the definition of normality or normal behavior besides a normative point of view. In practice, this class imbalance is often given for anomaly detection in general and for anomaly detection for transaction data in particular: The European Central Bank estimated that from 100.75 billion card transactions in 2019, 24.16 million were fraudulent, which approximately corresponds to rare 0.024 percent of transactions [22]. This rarity emphasizes the fact that for many applications **anomalies cannot be defined precisely**, especially with increasing complexity of the systems, as the number of aspects that can contribute to non-normality grows with the number of dimensions (related to the "curse of dimensionality") [50] and anomalies in case of criminal intent such as fraudulent activities are often intentionally obfuscated or covered [3].

However, the greatest challenge for academic research in this area is the availability of data. The **lack of labeled data** can be addressed partially with unsupervised or one-class approaches mostly by characterizing normal data or unknown data with an acceptable low contamination rate. This approach can be summarized under the term **modeling normality** or **modeling distributions and dependencies**, where a model is trained to learn these characteristics from a large set of normal, potentially unlabeled data, implicitly circumventing the two previously noted challenges of rarity and the lack of universal definitions of anomalies. Such models can be approached from a generative or discriminative perspective. Generative models have the advantage that data are modeled explicitly in a way that new samples can be generated by such a model. Discriminative models have the advantage that they are

directly trained to distinguish anomalies from normal data and therefore can focus on the important aspects for the downstream task. Therefore, they can be directly incorporated as a **methodology for anomaly detection**. Both generative and discriminative models are promising approaches for the transaction data domain for which they are studied throughout this thesis.

In contrast to the lack of labels, the lack of large real-world datasets cannot be met by technological means without compromises, such as working with synthetic data or simulations [349, 147, 190, 19]. Real-world datasets cannot be made publicly available due to their relevance as business secrets, as for example SAP data includes sensitive information about prices, suppliers, customers, and business or privacy related data in general. Especially data for which fraud is not only to be expected from unauthorized third parties, but where the analysis could also suspect employees, allow systematic conclusions to be drawn about their working quality. This setting, where discriminatory characteristics can be learned, is particularly critical from a data protection point of view. Machine Learning (ML) and automated decision making on sensitive data, for example, is under European law required to have a "human in the loop" for privacy and ethical concerns [88], such that, for example, automated decisions are being reviewed before any consequences are drawn. Therefore, the assessability by auditors has to be considered for balancing evaluation metrics with the expected workload. The synthetic generation and modeling of specifically normal data, for example by generative models, represents a promising connection between the lack of publishable data due to privacy concerns and a perspective on anomaly detection by modeling normality.

A challenge from a technical point of view is the appropriate representation of unique dataset characteristics present in transaction data. Besides numerical dependencies between features and samples, for example, with features of quantities, price, invoice totals, or amounts to be transferred, the large number of columns, and datasets consisting of mixed binary, categorical, and numerical features play a major role for the modeling of transaction data as they are inherently present and relevant for anomalies. To address these challenges, the field of **representation learning** offers solutions to represent categorical features similar to numerical features in dense vector representations, which are suitable for neural network models and have been successfully applied for natural language processing [207] and in the cybersecurity domain [251]. However, the choice and adaptation of an appropriate representation learning approach for transaction data remains an interesting question to successfully apply anomaly detection in this domain.

Thus, we identified three main aspects of anomaly detection for transaction data, which will be focused on in this thesis: (1) The ability to learn numerical dependencies along with the modeling of normality, (2) learning meaningful representations for transaction data, and (3) the methodology to detect anomalies in transaction data. In the next chapter, the research questions which motivate our work and follow through this thesis will be introduced in detail based on these three aspects.

Chapter 2

Research Questions

Companies that have been subject to fraud or security incidents embrace the idea of detecting new attacks using data mining more and more, as digitization and associated data is prevalent in all business areas today.

Since many countries have strict regulations with regard to data protection and companies want to protect their business secrets, datasets for the detection of transaction anomalies are barely available and cannot be shared with the public. One solution is the obfuscation of real data such that the interests of customers, employees, and the company itself are preserved. For example, for the credit-card fraud dataset all but two features have been transformed with Principal Component Analysis (PCA). For the IEEE-CIS dataset, customer profiles have been decoupled from transactions and several features are only included in an aggregated form. Although these anonymization and obfuscation steps allowed the publication and use of these specific datasets, they have severe drawbacks, as the meaning of features and decisions regarding preprocessing can no longer be reconstructed or revised.

Modeling Normality A different idea addressing both privacy and the lack of data is the use of neural generative machine learning models [184]. Synthesized artificial data which mimics important characteristics of real data can then be used to augment a small dataset to improve the performance of data-intensive machine-learning approaches [85]. The generated artificial data or models can be shared even if tight privacy guarantees are required if a privacy-preserving framework such as Differential Privacy [80] and appropriate training procedures [1] and models [339, 158] are employed [9]. The first Research Question (RQ) therefore focuses on neural models and their ability to precisely model real data.

RQ 1: How well do neural network models capture the characteristics of transaction data?

For this RQ, we distinguish between the ability of neural networks to model numerical dependencies, referred to as *Modeling Numeric Dependencies* (Section 6.1)

and motivating RQ 1.1, and the capability of generative models to synthesize data-distributions and correlations, which we refer to as *Modeling Distributions* (Section 6.2) and which motivates RQs 1.2 and 1.3.

Modeling Numeric Dependencies Financial data in general and transaction data in particular often contain mathematical and numerical dependencies and constraints between features, which must be taken into account by a model for precise adaptation or replication of real data. However, modeling numerical dependencies in a way that a model extrapolates and generalizes outside the training data range is considered a difficult task for feed forward neural networks [23, 181, 341]. To overcome the shortcomings of networks when it comes to modeling numerical relations and extrapolation, a neural architecture, the improved Neural Arithmetic Logic Unit (iNALU), is proposed and systematically evaluated in several experiments in this thesis to answer RQ 1.1.

RQ 1.1: To what extent can the extrapolation of numerical dependencies be improved by our neural architecture?

RQ 1.2: How well do generative neural networks model feature distributions?

RQ 1.3: How well do generative neural networks model feature correlations?

Modeling Distributions To answer RQ 1.2 and 1.3, two of the most relevant generative neural model architectures, Variational Auto-Encoder (VAE) and Generative Adversarial Networks (GANs) are adapted to synthesize transaction data. The generated samples of both models are then compared with real data on feature distributions and feature correlations.

Besides (generative) modeling of data characteristics on the feature level, meaningful representations of transactions and their similarity is an important aspect for several machine learning models from different perspectives. For example, k-Nearest Neighbors as a classifier builds on the assumption that similar samples with respect to the task to be solved are also considered to be *near* each other, often measured by a similarity metric. Another example is the similarity of feature-values, which from a domain perspective resemble each other to varying degrees, e.g. goods within the same category versus other categories or price range in an e-commerce setting. Similarity for numeric attributes in general is well defined and can be operationalized in several ways, mostly relying on the numerical difference, e.g. absolute error or Mean Squared Error (MSE), or more generally on metric spaces and norms. The differentiation for categorical features, however, is non-trivial, since categorical attributes have no inherent order. Often the similarity of categorical features is reduced to a match (e.g. Matching Coefficient or Overlap, Hamming-Distance) or is weighted

differently according to the application (e.g. Inverse Occurrence Frequency, ConDist or Eskin) [33, 252, 87, 157].

Representation Learning Depending on the method, similarity in terms of match or non-match may vary for each use case. In practice, however, it is not clear which metric is the most appropriate for the data or task and a deeper understanding of which metric deals with particular dataset charactersitics in which way is necessary [33], introducing cumbersome manual labor for feature-rich datasets. In the field of Natural Language Processing (NLP), where data is almost exclusively categorical with many possible values when it comes to words, sentences, and documents, Representation Learning (RL) approaches have proven to be useful, which generate a dense data representation instead of categorical values or the associated sparse one-hot vectors. RL approaches generate a structured latent space, which implicitly represents relations and similarities in the data. These dense representations allow similarity computation with numerical distance metrics such as cosine similarity or direct processing with neural networks, which motivates our use case of RL on transaction data.

The following RQ is defined to evaluate the benefit of RL in the context of transaction logs and anomaly detection.

RQ 2: How are transactions best represented to emphasize structural similarity?

To answer this RQ, we adopt several RL approaches such as Word2Vec or GloVe on transaction logs and evaluate their benefit on two different datasets, Windows Audit logs, which shares more structural properties with classical text-based applications, as well as an SAP transaction dataset, which resembles a tabular structure with categorical values mainly consisting of single words or even single characters. These differences motivate RQ 2.1 and 2.2, which address two main aspects of RL: Meaningful structure for human evaluation and from a machine learning perspective.

RQ 2.1: Which representation learning approach yields the most promising structure in latent space?

RQ 2.2: Which representation learning approach yields the best performance in extrinsic evaluation?

For these RQs, we evaluate ability of the models to learn meaningful representations regarding fraud and attack detection as extrinsic evaluation on the one hand, and the structure of their latent space by explorative evaluation on the other.

Representation Learning and Numerical Features In general, transaction datasets in contrast to natural language texts consist of categorical and numerical features. As RL approaches are originally proposed for categorical features only, adapting RL methodology to transaction data including numerical features is a relevant research direction with the opportunity to find more meaningful representations in this domain. As the SAP transaction dataset consists of several relevant numeric features that could enhance the representations, we propose two discretization approaches, QOQD and QOUD, specifically designed to emphasize numerical outliers to answer RQ 2.3.

RQ 2.3: To what extent does the introduction of numerical features improve representation learning in extrinsic evaluation?

Anomaly Detection in Transaction Data The third research question finally focuses on anomaly detection and its methodology for transaction data. We therefore combine the previous contributions for the use case of anomaly detection on transaction data and investigate the benefit of each method.

RQ 3: To what extent do the proposed methods improve anomaly detection for transaction data?

To approach the question, we first propose a neural architecture called Mixed Layers and evaluate our model on financial transaction datasets for fraud detection, as well as for detecting anomalies in an SAP dataset in comparison to several baseline methods to answer RQ 3.1. Secondly, we incorporate the generative models VAE and GAN for anomaly detection. We evaluate three approaches to adapt our GAN model as an anomaly detector and evaluate their performance on several financial transaction datasets to answer RQ 3.2.

RQ 3.1: Can fraud detection performance on transaction data be improved by our Mixed Layer model?

RQ 3.2: Do our generative VAE and GAN models yield competitive anomaly detection performances, and do optimal preprocessing and parameter choice differ from their generative setting?

Contributions

The contributions of this thesis can be summarized as follows: In the context of RQ 1.1 we first propose the improved Neural Arithmetic Logic Unit, a novel architecture to model and extrapolate numeric dependencies. Therefore, we discuss the shortcomings of previous models and suggest several enhancements and architectural changes to improve extrapolation stability and precision. We evaluate our model in five experiments of increasing complexity on synthetic datasets and show that our model outperforms reference models in terms of precision and stability.

Parts of these experiments have been published in
Schlör, D., Ring, M., and Hotho, A. (2020a). iNALU: Improved neural arithmetic logic unit. *Frontiers in Artificial Intelligence*, 3:71.

To answer RQ 1.2 and 3.2, we adapt two VAE and GAN-based models to synthesize transaction data. We therefore introduce architectural improvements for both models and experimentally evaluate preprocessing decisions and model parameters for their ability to generate synthetic transaction data on three datasets.

Parts of these experiments are based on our work
Ring, M., Schlör, D., Landes, D., and Hotho, A. (2019a). Flow-based network traffic generation using Generative Adversarial Networks. *Comput. Secur.*, 82:156.

Our contributions for RQ 2.1 and RQ 2.2 include a systematic evaluation of multiple representation learning techniques for two datasets of varying degree of structuring in the transaction data domain. For RQ 2.3, we introduce two novel discretization approaches to emphasize outliers in discretization. We evaluate both methods in a large-scale experiment in conjunction with dataset preparation and representation learning choices for the task of financial fraud detection.

Parts of these experiments have been published in
Ring, M., Schlör, D., Wunderlich, S., Landes, D., and Hotho, A. (2021). Malware Detection on Windows Audit Logs using LSTMs. *Computers & Security*, 109:102389.

For RQ 3.1, we propose a novel neural architecture to address numerical as well as non-numerical dependencies in fraud detection for transaction data. We show that our model outperforms neural as well as non-neural baselines and evaluate its benefit for anomaly detection in transaction data. To answer RQ 3.2, we evaluate three methods for GAN-based models in anomaly detection and discuss parameter choices and preprocessing decisions in comparison to generative applications.

Parts of these experiments have been published in
Schlör, D., Ring, M., Krause, A., and Hotho, A. (2020b). Financial Fraud Detection with Improved Neural Arithmetic Logic Units. Fifth Workshop on MIning DAta for financial applicationS, (honored with the best paper award).

Outline

The outline of this thesis can be summarized as follows. After stating the Research Questions and Contributions, we will discuss Related Work in the general field of detection anomalies in transaction data. This section will be structured along the main aspects of this thesis, numeric dependencies and generative models, representation learning as well as anomaly and fraud detection. Then the Foundations chapter focuses on the methodological background of this thesis, introducing basic background for machine learning models including the baseline models and evaluation methodology. In the Datasets chapter we will introduce the datasets used for our experiments, followed by the three chapters to answer three research questions raised in this thesis as depicted in Fig. 2.1.

In Modeling of Distributions and Dependencies we propose a neural architecture to model numerical dependencies and evaluate generative neural networks for their ability to learn data distributions and feature dependencies. In the Representation Learning chapter, we focus on two transaction-data-specific tasks of maleware and fraud detection and adapt several representation learning approaches for their ability to construct meaningful representations from an explorative, visual perspective, as well as for their benefit for machine learning models to solve the respective downstream task. Then both perspectives for modeling numeric dependencies and distributions are transferred to the tasks of Anomaly and Fraud Detection, where we evaluate our models along with several baseline approaches in a supervised and semi-supervised setting.

We then summarize our findings to answer our research questions and finally conclude this thesis with an outlook.

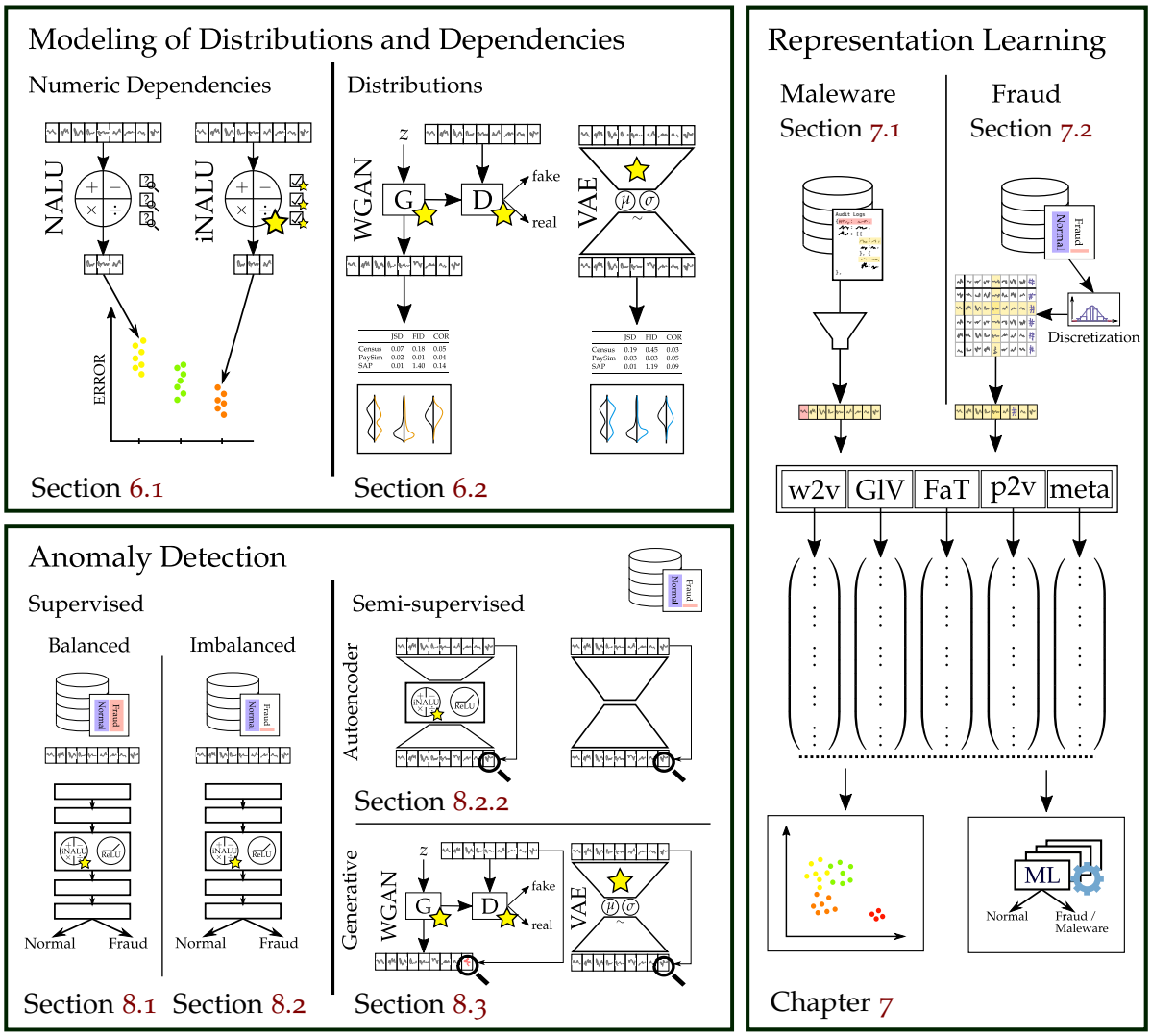


Figure 2.1: Visual outline of the thesis. This thesis is comprised of three main topics: Modeling of Distributions and Dependencies (Chapter 6), Anomaly Detection (Chapter 8) and Representation Learning (Chapter 7). We contribute by analyzing and improving a neural architecture to model numeric dependencies and adopt generative neural models for transaction data, including quantitative and qualitative analysis. We then evaluate our contributions in several anomaly detection settings. Furthermore, we study the influence of representation learning approaches for anomaly detection in transaction data exploratively as well as in extrinsic evaluation. Models which incorporate our iNALU architecture are marked with ☆.

Chapter 3

Related Work

In this section, related works from literature associated with the core topics of this thesis are introduced.

Section 3.1 outlines different approaches to model data characteristics and synthetically generate data following the characteristics of real data. We focus on two different aspects, neural modeling of dependencies within data with a focus on numerical dependencies and generative modeling of data and models synthesizing data in general.

Section 3.2 then presents other works that incorporate representation learning approaches in anomaly detection, transaction data, or related domains. Besides a short overview, we refrain from discussing representation learning in the Natural Language Processing (NLP) domain in depth, as, although some datasets have commonalities with natural language data, transaction data generally follow other characteristics, as discussed in Chapter 7.

In Section 3.3 we finally outline work related to our main aspects, anomaly and fraud detection in transaction data, with a broad outlook into similar application domains.

3.1 Modeling Data and Distributions

For machine learning in general and anomaly detection in particular, models are trained on data to yield useful information as discussed in more detail in Section 4.1, often with the intention of applying these models to previously unseen data. To generalize and provide useful information, models must focus on specific data characteristics that are helpful in terms of the respective task. In traditional ML the approach often consists of specific feature engineering steps, explicitly modeling and representing data, features, and feature combinations meaningfully to emphasize the focus on these aspects. This is often driven by expert and domain knowledge. With the raise of Neural Networks (NNs), explicit feature engineering is less common and often limited to representing, selecting, and rescaling data to fit the

computational and practical requirements to be processed by the NN effectively, while the feature extraction and combination are part of the model. For example, different layers of a convolutional neural network focus on (visual) manifestations of characteristic features and combinations of increasing semantic complexity up to solving the actual task [223, 307]. Thus, the manifestation of relevant data characteristics such as features, feature distributions or relations, and combinations of features must be constructed implicitly within the model by the training objective. When it comes to financial datasets, such feature relations are often numerical dependencies that are inherently characterized by arithmetic relations. In a simple example, anomalies in a credit payment log example (cf. Section 5.2.4) could be characterized by a closed formula, which can be explicitly modeled as a feature with traditional feature engineering. For a neural and data driven approach, however, the model itself is expected to learn this relationship from data, which makes such approaches applicable for more complex scenarios, i.e. when such an explicit formula is not available or the relationship is incomprehensible. NN have been shown to be universal approximators [99, 141, 140, 238], i.e. theoretically they can approximate arbitrary mathematical functions of varying complexity given a sufficient network size [116] implying the suitability to model such numerical dependencies and arithmetic relations. In practice, however, they lack generalization and extrapolation of even the most simple arithmetic operations such as identity [168] or basic operations [317] such as multiplying or subtracting. For challenges arising with neural learning of arithmetical relations implicitly and explicitly, we will review relevant research in the following section.

3.1.1 Modeling Numerical Dependencies

While even today, "we are still lacking a comprehensive theory that could explain how numerical concepts are learned by the human brain", as Testolin stated in 2020 [310], studies by Anderson et al. [14], worked as early as 1994 on teaching arithmetic to artificial neural networks. Anderson et al. conclude that "such a system is genuinely creative and flexible, though only in a limited domain". Anderson et al. studied the learning of simple multiplication tables and "greater than" relations, which can be associated with other tasks that explicitly model arithmetic, such as Arithmetic Expression Calculation (ACE). These tasks include discrete single operation expressions, such as binary multiplication or binary addition [370], 15-digit decimal multiplication [160], or symbolic, algorithmic and arithmetic expression calculation [355, 159, 53, 172] up to Turing Machine inspired approaches [356] with the ability to solve more general problems [120] further related to research on neural execution of algorithmic code snippets or subroutines [347, 356, 248].

Besides explicit arithmetic tasks, other approaches aim at determining governing equations or equation parameters with NNs from data. In this category, approaches focus on predicting the dependencies or making them explicit, and thereby yield

a generalizable mathematical model describing data characteristics. Examples include recovering undetermined coefficients from partial differential equations [246], using ResNets as a numerical integrator [242] or Recursive Deep NNs [364] to approximate dynamic systems. This research branch has a strong focus on dynamic systems and is driven by applications in neuroscience, biology, and climate science [49]. In contrast, from a data science perspective, the precise formulation of governing equations from data is generally neither the objective nor feasible, as the data might not be sufficient to approximate any underlying dynamical system if given at all. Instead, modeling numeric dependencies and arithmetic relations implicitly within the model is a more targeted goal to solve a downstream task which is not necessarily directly related to the numeric dependencies.

One example of such indirectly related arithmetic dependencies is the task of counting objects in pictures, where counting, and thus arithmetic dependencies, are not directly related to features but instead to abstract semantic manifestations within the feature space of pixels, e.g., counting numbers [286], people [188, 361], animals [221], or vehicles [225]. Approaches to solving counting tasks can be categorized into two groups: multistage approaches and end-to-end models. While multistage approaches generally use an auxiliary task in the first stage, which is for image-based counting mainly object density estimation [102], object recognition [103] or object tracking [61], and evaluate counts in the second step from objects, end-to-end approaches directly model the counting (regression) task on input images [286, 201].

End-to-end models are more related to general machine learning tasks that involve explicit and implicit arithmetic relations. Evaluating an MNIST image counting task, Trask et al. [317] proposed a specific neural architecture, the Neural Arithmetic Logic Unit (NALU), implicitly modeling numerical dependencies and simple arithmetic operations. NALUs are constructed task-agnostically and without the input of an explicit arithmetic relation as in contrast to Arithmetic Logic Units in processors, where an op-code specifies the operation to be executed. Instead, the NALU learns to select the operation to execute per unit directly from data with regard to the task to be executed. This task can be an explicit arithmetic operation or an arbitrary downstream task, such as anomaly detection, while for the latter, NALU can be viewed as a special-purpose neural cell targeting numerical dependencies. Several empirical [153] and theoretical [197] analyses of the NALU have been carried out mainly evaluating the stability and explicit arithmetic performance, which are extended by our findings in Section 6.1.2 and motivated architectural improvements (cf. Section 6.1.3) leading to models such as our improved Neural Arithmetic Logic Units [276], Neural Arithmetic Units [197], Neural Power Units [130], nonlinear Arithmetic Units [21], and Neural Reciprocal Units [211]. These subsequent architectures mostly focus on improving explicit arithmetic precision for the multiplication or division operation and stability by reformulating the cell architecture. In contrast, our approach preserves the original structure of the NALU and allows improvements only beneficial for explicit arithmetic calculation to be excluded for

modeling implicit numerical dependencies on a non-arithmetic downstream task as given in our anomaly detection scenario.

Besides being specialized neural cells, the NALU can also be seen in the scope of complex activation functions with linearities (summation and subtraction) and non-linearities (multiplication and division) being parametrized internally. In this vein, other works on specific parametric activation functions allow to learn numeric dependencies of various function classes, e.g. from linear to exponential [113, 318], or parametric bounded activation functions [366, 78]. More distantly related is the field of mathematical reasoning and modeling of mathematical concepts and numeric relations in natural language [311]. In this area, studies, for example, evaluate the numeracy in word-embeddings [325, 303, 249] and language models [151, 101] or question answering with mathematical reasoning [94, 210, 185, 273].

3.1.2 Generative Models and Data Synthesis

In contrast to discriminative models, which typically learn to establish a decision boundary between the classes to be distinguished, generative models estimate the (generally unobservable) distribution from which the dataset instances are drawn from. In general, this approach is not limited to labeled data, which makes it particularly suitable for unsupervised and semi-supervised tasks [115, 227] as which anomaly detection can be modeled (cf. Section 4.4). In this context, the model is learned from normal data and must capture the characteristics of these instances such as feature distributions and correlations (hence modeling distributions and dependencies). This model can then be considered a precise estimate of the underlying unobservable distribution, as which the data generation process can be understood [4, 193], making them, as Ruff et al. stated, "a clear candidate for the task of anomaly detection" [263].

Generative models can be further categorized into neural network-based and traditional approaches. Under the term *traditional*, we subsume approaches which employ generative models without a neural network formulation. This corresponds to parametric density estimation such as fitting a normal distribution, or Gaussian Mixture Models which have been described as "popular generative techniques" providing good results if the underlying assumptions of Gaussian mixtures are met [289]. Density estimation-based generative models have also been used for anomaly detection in direct comparison to neural networks very early [258]. Although these approaches yield good results on low-dimensional datasets, they suffer from the curse of dimensionality that motivates the use of deep statistical models for complex or high-dimensional datasets [263].

With Bolzmann Machines [90], Energy Based Models (EBM) can be considered the earliest generative models, and Restricted Bolzmann Machines [293] and related architectures such as Deep Belief Networks [137] or Fully Visible Belief Networks [97] have received attention in modern architectures [321] and for anomaly detec-

tion [359]. EBMs involve an energy function depending on the model parameters, which rates the parameter configuration by an energy scalar. Training corresponds to finding a specific energy function such that suitable parameter configurations correspond to smaller energy values. However, EBMs often require dedicated training schemes [227], and sampling from the model is computationally expensive [115] and typically involves Markov Chain Monte Carlo (MCMC) methods [227]. EBMs such as fully visible belief networks are described as tractable explicit models, i.e., tractable models defining an explicit probability density function in contrast to explicit models with intractable density that require approximation such as Variational Auto-Encoder (VAE) or implicit density models such as Generative Adversarial Network (GAN) [115]. Although some EBMs can also be understood as neural architectures or involve cost function-based training objectives [178], the term *neural generative model* [264] or *cost function based model* [227] is commonly used for VAE and GAN architectures, which are considered the "most established neural generative models" [264] and are the basic architectures on which we build our *Modeling Distributions* experiments (cf. Section 6.2). For a detailed description of VAE and GAN, see Section 6.2.1.

VAEs and variants have been widely studied as generative models to synthesize data, both, for representation learning and as an anomaly detection approach. Some models explicitly aim to synthesize images [346, 106, 142, 36, 143, 297, 287], music [257, 256, 131, 314, 121], or data in the biomedical domain [331]. Aside from synthetic toy datasets to evaluate architectural changes [166, 195], only few studies applied VAEs to tabular data, for example, to evaluate privacy [184], data imputation [215], for the detection of anomalies or cybersecurity threats [340, 350, 229, 328]. In contrast, in Section 6.2, we explicitly evaluate the generative performance of a VAE-based model for transaction data in comparison to a GAN-based model.

The idea of GANs was introduced by Goodfellow et al. [117] in 2014 as a novel implicit density generative neural network architecture which is trained in a min-max game theory based strategy (see Section 6.2.1 for a detailed explanation). GANs have been used successfully on various image generation tasks [367, 45, 222, 145, 15, 148, 44, 162, 163] yielding realistic high quality images [179]. Despite training difficulties [115] and issues such as mode collapse and the open question of evaluation [313, 344] (see, e.g., [272, 68] for a broader overview), GAN-based approaches have been more widely applied for the synthesization of tabular and transaction data, e.g., for tabular data [230, 343], financial time-series generation [308, 354] or financial and transaction datasets in general [253, 83, 202, 81].

With a focus on the outcome of generative models, specifically the data synthesized by a model, another branch of research can be considered distantly related. These works are more aimed at solving the aforementioned challenge of lack of (labeled) data. While some works still incorporate machine learning models such as Markov Chains to generate Enterprise Resource Planning (ERP) data [349], other works do not have a focus on machine learning models, e.g., generating synthetic

ERP data by randomly making changes to a provided database [147] or data generation by serious games [274], which put less emphasis on data quality and more on the educational objective teaching players how to use and misuse an ERP system and how to detect it.

3.2 Representation Learning

The importance of Representation Learning (RL) for Artificial Intelligence (AI) was recognized very early as Anderson et al. [14] humorously stated in 1994:

> There is a folk saying in AI that there are three important aspects to any AI system: representation, representation and representation.
>
> Anderson et al., A study in numerical perversity [14]

Although the frontiers of AI research have changed considerably since then, the importance of properly representing data for AI still remains and has become a research discipline on its own [27]. RL can have different views depending on the application domain and the task in a broader view, and generative models and neural models capable of capturing data characteristics as described in Section 3.1.1 can also be considered as RL [27]. However, in the context of this thesis, we will use the term *Representation Learning* specifically referring to the subarea of embedding techniques. Hereby an instance (or a feature) is to be represented as a dense vector in the d dimensional vector space $\mathbb{R}^d$ which is then, for example, used in downstream ML tasks such as anomaly detection. Incorporating RL typically aims to find a suitable representation to solve a machine learning task, for example, by reducing the dimensionality of the input space or to disentangle the properties and similarities between instances or features [174] with the general objective of facilitating subsequent tasks [116]. The field of RL was notably driven by the research discipline of Natural Language Processing (NLP). When working with natural language, from a data-driven perspective, types and tokens formed in sequences of tokens to meaningful text denote the data foundation to work with. This implies an exorbitantly large number of possible types when working with unconstrained vocabulary, which can be interpreted as categorical features, although novel architectures use other concepts such as byte pair encoding [46]. When interpreting tokens as atomic features, Distributed Representations [136] are constructed to generalize over semantic attributes between tokens (or more generally concepts) [116]. Mikolov et al. [207] introduced efficient neural formulations to apply the concept of distributed representations to large-scale corpora and thus prepared the ground for neural representation learning methods and their application to various fields beyond NLP, including cybersecurity [251, 239, 18, 327, 336, 189, 309, 56, 183, 337, 326, 304, 152], the financial domain [352, 345], and fraud detection [265, 155, 132, 360, 156, 332, 363, 25, 118]. While a large number of works only apply word2vec [207], we adapt the methodology of

extracting negative and positive samples to the row and column specific structure of features and samples, incorporate numerical attributes, and apply it to other established RL approaches as detailed in Sections 7.1 and 7.2.2. Mostly related to our adaptions is the table2vec [362] approach. In contrast to their approach, however, we include numeric features and infer per-sample representations instead of using table-cell embeddings for table-specific tasks such as row or column population.

3.3 Anomaly and Fraud Detection

The topic of Anomaly Detection (AD) in transaction data subsumes several application domains, task formulations, and related research topics. In this section, the various perspectives and related works will be discussed. When talking about AD, anomalies typically refer to rare events or outlier data points, which should be distinguished from normal behavior or associated data. Detecting anomalies is important because they indicate rare events with major consequences. For example, an anomaly in a credit card transaction may indicate fraudulent activity that could result in large losses for the affected person or institute. When referring to transaction data, rare and anomalous events often correspond to unexpected behavior by users or customers involved in the corresponding (business) processes. After a broader view, we therefore focus on related work on AD in the financial domain in general and fraud detection in particular.

A comprehensive review of anomaly detection methods in general is provided by Candola et al. [50]. They categorize existing approaches for anomaly detection based on their techniques and applications including fraud detection and cybersecurity. Chalapathy and Chawla [48] provide a more recent survey of deep learning for anomaly detection that, among other things, addresses the topics of fraud, cyber security intrusion, and maleware detection. The authors describe existing approaches for the detection of credit card fraud, fraud in mobile cellular networks, insurance fraud, and healthcare fraud, and categorize host- and network-based intrusion detection approaches. This survey shows that various network architectures such as Auto-Encoders, Generative Adversarial Networks, Convolutional NNs, Restricted Boltzmann Machines, or Recurrent NNs are used for anomaly detection in both domains.

Financial fraud detection is very diverse and may appear in different areas such as mobile payments, credit card misuse, or ERP systems. Often, fraud represents only a very small proportion of transactions in these areas. Consequently, many financial fraud detection methods are based on anomaly detection. While Sabau [266] and Phua et al. [236] provide an overview of traditional approaches to detect fraud, Singla et al. [161] give a specific overview on fraud detection with a dedicated focus on fraud detection based on deep learning on online transactions. An overview of fraud detection with a dedicated evaluation of applications and techniques between

2004 and 2015 is provided by Albashrawi [10], while simulation-based works are summarized by Lopez-Rojas [191]. Hilal et al. [135] investigate popular and effective semi-supervised and unsupervised learning anomaly detection techniques applied to detect financial fraud. While these works provide a broad overview over fraud detection in general, two specific applications are related more closely to AD in transaction data, namely AD in credit card transactions and in transactions recorded within ERP systems.

Many works investigate the suitability of machine learning methods for credit card misuse. Wang et al. [330] evaluate Random Forest, Support Vector Machines, and Capsule Networks for the detection of credit card fraud. They evaluate these appraoches on a private credit card transactions dataset and extract and aggregate features for each transaction. Similarly, Maes et al. [199], Shen et al. [288], and Sun and Vasarhelyi [302] evaluate different neural network architectures for credit card fraud detection.

Baader and Krcmar [20] present an approach to detect fraud on purchase-to-pay processes in ERP systems, combining red flag analysis and process mining techniques. Schreyer et al. [280] use deep Auto-Encoder to detect anomalies in journal entries of an ERP system and Schultz and Tropmann-Frick [281] also apply Auto-Encoder to detect unusual journal entries within individual financial accounts on a real-world ERP dataset.

Although incorporating neural models, these approaches in several ways follow traditional approaches using expert knowledge for feature engineering or feature selection. In contrast, our experiments avoid relying on domain experts to model the data foundation or features, as in practice adaptations to other datasets require the repetition of this costly and time-consuming step. Instead, our approach, building NNs on raw data, allows one to easily transfer approaches to other datasets and domains, as the models learn to focus on relevant features implicitly by design.

Chapter 4

Foundations

In this chapter, we will introduce the basic concepts and methodological foundations used in the different sections and experiments throughout this thesis. In Section 4.1, the methodological foundations for the Machine Learning approaches that this thesis builds on are presented. We will explain the basic concepts for neural networks in detail, which are the basis for several models developed for this thesis. Further, we will briefly introduce the baseline approaches used in our experiments. In the Anomaly Detection Section 4.4, we will discuss the relationship to the previously introduced Machine Learning approaches and the challenges that arise with AD. Then, in Section 4.5.1, the evaluation metrics used to evaluate our results are introduced, and we discuss the influence of extreme data imbalance on different metrics. Finally, in Section 4.6, we will briefly frame the setting of this thesis in the context of the Knowledge Discovery in Databases (KDD) process and discuss Data Mining as a component of the KDD process with its relation to Machine Learning and Anomaly Detection.

4.1 Machine Learning

The term Machine Learning (ML) is most generally described as modeling learning processes with computers and denotes one of the "most challenging and fascinating long-range goal" in Artificial Intelligence [206]. Alpaydin [12] describes ML as optimizing "a performance criterion using example data or past experience" for a ML model to make predictions or describe data and gain knowledge, i.e., a computer program that improves in solving a task with respect to a performance metric by processing data to learn from. Data are typically structured as samples, the individual examples to consider during learning, which themselves are composed of features, i.e., the aspects characterizing each sample. For this thesis, we denote the data of a dataset by $\mathcal{D}$ with samples $\mathbf{x} \in \mathcal{D}$. The dataset has a fixed number f of feature categories $X_i, i \in \{1, \ldots, f\}$ with feature values $x_j \in X_i$ of which each sample $\mathbf{x} = (x_1, \ldots, x_f)$ is composed. ML algorithms are considered *supervised*, if a

target feature, also called label and typically noted as $y \in Y$, is present for each sample to be trained on, whereas algorithms learning features without direct feedback from a label are called *unsupervised*. In general, supervised algorithms are trained to predict the label from data as learned during training, whereas unsupervised algorithms learn structural properties of the dataset. For supervised training, the dataset $\mathcal{D}$ is often split into disjoint parts for training and testing, for example $\mathcal{D}^{\textbf{train}}$, $\mathcal{D}^{\textbf{eval}}$ and $\mathcal{D}^{\textbf{test}}$ for a fair evaluation on $\mathcal{D}^{\textbf{test}}$, having $\mathcal{D}^{\textbf{train}}$ for training and $\mathcal{D}^{\textbf{eval}}$ for hyperparameter tuning.

In practice, however, the distinction between supervised and unsupervised is often blurred [116] with several settings such as semi-supervised learning [368] or One-Class Learning (OCL) [24] in between. Especially, OCL is highly relevant for our setting of anomaly detection: For OCL, only one class is well defined and well represented in the training data. However, the task is to find instances deviating from the *known* class without explicitly knowing how these samples can be described.

In general, typical ML tasks can be categorized into predictive categories such as classification, where y belongs to one of the k categories to be predicted for a given sample and regression, where y is a numerical variable to be predicted, but also into non-predictive categories such as clustering or generative tasks. For clustering, the cluster assignment y is not given in the training data, but must be assigned by structuring the feature space. Examples of generative tasks are sampling or data synthesis. The task of anomaly detection can be seen in between this field. It can be modeled as a classification task as applied in Section 8.1, as an OCL task with properties from regression, and clustering as applied in Section 8.2.2, as the numerical label, which describes how anomalous a sample is, is not present during training. It can also be modeled as generative task as given in Section 8.3, which can also be used to synthesize data as in Section 6.2. In this section, we will focus on specific ML-models and ML-optimization algorithms and introduce them formally.

4.2 Neural Networks

NNs have been inspired by the internal structure of the (human) brain with neurons connected to each other and the decision to activate based on the activation of connected input neurons. This concept dates back to the introduction of a basic calculus model for nerve activity by McCulloch and Pitts [205]. The Perceptron as introduced by Rosenblatt [260] can be understood as the basic structure for neurons in NNs, which has been extended to multiple layers as Multi-Layer Perceptron (MLP), different activation functions, and network architectures, eventually outperforming other 'traditional' ML-approaches in several domains, powered by larger and larger datasets and increasingly powerful CPUs and general-purpose GPUs [116].

In general, NNs approximate a function f that is applied to data samples, for example to predict a target label y such that $f(\mathbf{x}) = y$. This approximation is achieved

by parameterizing f by variables subsumed in θ, representing weights to be learned during training such that $f(\mathbf{x};\theta) = y$. The most simple manifestation of a class of functions to approximate is the linear neuron, as shown in Eq. 4.1.

$$f(\mathbf{x};\underbrace{\mathbf{W},\mathbf{b}}_{\theta}) = \mathbf{x}^\top\mathbf{W} + \mathbf{b} \tag{4.1}$$

As Eq. 4.1 maps an input vector $\mathbf{x}$ to an output vector of arbitrary size defined by the shape of $\mathbf{W}$, several functions can be chained or layered. Accordingly, Eq. 4.1 is also called a *linear layer* of a neural network that may consist of multiple layers. If a layer is not directly associated with an output variable but succeeded by another layer, it is called *hidden layer*. A neural network with several layers is commonly termed a deep neural network [116]. To harness the depth of multi-layer NNs, linear layers alone are not suitable, as the linear transformations over multiple layers can be simplified to a single affine transformation and non-linear dependencies can not be modeled. Therefore, a non-linear activation function a is introduced (see Eq. 4.2) to leverage the full capabilities of NNs as universal approximators [141].

$$f(\mathbf{x};\underbrace{\mathbf{W},\mathbf{b}}_{\theta}) = a(\mathbf{x}^\top\mathbf{W} + \mathbf{b}) \tag{4.2}$$

For the perceptron, a is typically the threshold function t (Eq. 4.3), other common non-linearities are the sigmoid function σ (Eq. 4.4) or the hyperbolic tangent tanh (Eq. 4.5). The Rectified Linear Unit (ReLU) [212] (Eq. 4.6) has developed to the default activation function, allowing fast computation and good performance [112, 338, 116]. Other variants such as leaky ReLU [196] (Eq. 4.7) have been proposed addressing practical issues of ReLUs such as "dying neurons", which refers to the observation that a ReLU neuron which has reached values $x < 0$ is unlikely to recover to another state, since the gradient signal is zero. The SoftMax (Eq. 4.8) activation function is typically applied in the output layer of a NN to produce an output that can be interpreted as a multinomial probability distribution for K categories.

$$t(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4.4}$$

$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \tag{4.5}$$

$$\text{ReLU}(x) = \max\{0, x\} \tag{4.6}$$

$$\text{LeakyReLU}(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{otherwise} \end{cases} \tag{4.7}$$

$$\text{SoftMax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{x} = (x_1, \ldots, x_K) \in \mathbb{R}^K \tag{4.8}$$

To approximate the target function, the parameters (weights) of the NN have to be learned. This is achieved by optimizing the parameters with respect to a cost function, typically called *loss*: An NN with randomly initialized parameters is trained to minimize the loss. The training (i.e. the educated change of the parameters) is typically approached with gradient decent-based optimization, iteratively minimizing the loss up to a local optimum, as the cost surface of non-linear NNs is in general non-convex and there is no analytical solution to find a global optimum (which is also not desirable as it often leads to overfitting [59]). Several loss functions have been proven to be suitable for NN training, depending on the data and the task, with cross-entropy being the most common loss for classification. Cross-Entropy (Eq. 4.9 and 4.10) is based on the maximum likelihood principle minimizing the difference between the training probabilities and the probability distribution predicted by the NN. In this notation, x and y denote output and target and with w a weighting factor can be defined. For regression tasks, the Mean Squared Error (MSE) (Eq. 4.11) is often used, minimizing the squared differences between predicted and expected values. Other loss functions such as the Jensen-Shannon divergence or the Wasserstein distance are explained in Section 6.2 in detail, which are used in Generative Adversarial Networks (GANs) and Variational Auto-Encoders (VAEs) to compare predicted probability distributions and distributions derived from data.

$$\text{CCE}(x, y; w) = -\sum_{c=1}^{C} w_c \log \frac{\exp(x_c)}{\exp(\sum_{i=1}^{C} x_i)} y_c \tag{4.9}$$

$$\text{BCE}(x, y; w) = -w\,(y \log x + (1 - y) \log(1 - x)) \tag{4.10}$$

$$\text{MSE}(x, y) = (x - y)^2 \tag{4.11}$$

Besides optimization according to the label, a regularization term is also often included as part of the loss, for example, to constrain a layer's output to standard normal distribution for VAE (see Section 6.2.1), or to decrease weights with weight decay or ℓ_2 regularization [192], or as task-specific regularization (see Section 6.1.3). In particular, the loss does not necessarily reflect the true evaluation criterion, but instead reflects a surrogate loss function, which can be optimized (more) efficiently. As the surrogate loss might yield slightly different local optima (e.g. due to regularization terms) and training is generally not stopped at an optimum automatically, *early stopping* [240] is often applied for which training is stopped when a stopping criterion based on the true loss or performance metric is reached on a validation

split unseen during training to avoid *overfitting*. Overfitting can occur, since with ML, optimization is performed on a training dataset, not on the underlying true data distribution, which is unknown. This can lead to overly adaption to the training samples up to memorization and does not generalize to other dataset splits generally assumed to be drawn from the true data distribution. Besides traditional ML approaches, especially NNs are prone to overfitting due to their large number of trainable parameters [175] motivating methods such as early stopping or dropout [298] to address this issue.

In general, for the training of NNs, a cost function J is optimized by minimizing the loss $\mathcal{L}$ on the training data with respect to the parameters of the neural network θ and the output of the network $\hat{y} = f(x;\theta)$.

$$J(\theta) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}^{\textbf{train}}} \mathcal{L}(f(x;\theta), y) \tag{4.12}$$

If the expected value is realized as a sum of all samples of $\mathcal{D}^{\textbf{train}}$, each optimization step requires processing of the entire dataset which is computationally expensive. Instead, NN training is typically performed in *minibatch training*, where the empirical cost and the resulting training step are calculated on a small subset of m training samples randomly drawn from the training dataset.

$$J(\theta) = \sum_{(x,y)\sim\mathcal{D}^{\textbf{train}}}^{m} \mathcal{L}(f(x;\theta), y) \tag{4.13}$$

As optimization algorithm, Stochastic Gradient Decent (SGD) is one of the most common algorithms used for NN training. The idea is that by following the gradient iteratively, a downward movement on the loss surface will lead to an optimum. With SGD, the gradient is estimated as average over a minibatch $(X, Y) \sim \mathcal{D}^{\text{train}}$ and the 'movement' is scaled by a parameter γ, the *learning rate*.

Algorithm 1: Stochastic Gradient Decent (SGD) algorithm

input : dataset $\mathcal{D}$, initial weights θ, number of iterations T, learning rate γ
output: updated parameters θ

for $t = 1$ **to** T **do**
- $(X, Y) \sim \mathcal{D}$ with $|(X, Y)| = m$; `// sample minibatch of size m`
- $g \leftarrow \frac{1}{m} \nabla_\theta \sum_{(x,y)\in(X,Y)} \mathcal{L}(f(x;\theta), y)$; `// compute gradient estimate`
- $\theta \leftarrow \theta - \gamma g$; `// update parameters`

return θ

The SGD algorithm [262, 116] as shown in Alg. 1 can be varied by several means, for example by varying the learning rate, incorporating early stopping instead of

fixed iterations of T or introducing momentum [305]. Another popular optimizer that combines several improvements is ADAM [165] which will be used in this thesis. While these foundations lay the common ground as NN basics, several specific foundations and models are introduced in Chapter 6, such as the VAE model, GANs, or the NALU along with our adaptations and improvements developed in this thesis.

4.3 Baseline Approaches

Besides NN models, we use several classic ML-approaches as a baseline in our experiment, which will be introduced briefly in this section.

4.3.1 Naïve Bayes

One of the most basic ML models for classification is the Naïve Bayes (NB) classifier. NB directly builds on Bayes' theorem (Eq. 4.14) which allows to calculate the posterior probability $p(y \mid \mathbf{x})$ based on the prior class distribution $p(y)$, the likelihood of the observation under the classes $p(\mathbf{x} \mid y)$ and the observations $p(\mathbf{x})$:

$$p(y \mid \mathbf{x}) = \frac{p(y)\ p(\mathbf{x} \mid y)}{p(\mathbf{x})} \tag{4.14}$$

NB postulates conditional independence, that is, all characteristics $x_i \in \mathbf{x}$ are independent of each other with respect to y, such that $p\left(x_i \mid \{x_j\}_{j \neq i}, y\right) = p(x_i \mid y)$. For K classes y is generalized to K probabilities y_k with $1 \leq k \leq K$ and the posterior and class estimate is given by Eq. 4.15. In practice $p(\mathbf{x})$ can be omitted as a constant factor independent of y for determining the most probable class.

$$p(y_k \mid x_1, \ldots, x_n) = \frac{p(y_k) \prod_{i=1}^{n} p(x_i \mid y_k)}{p(\mathbf{x})}, \quad y^{\text{pred}} = \underset{k}{\operatorname{argmax}}\, p(y_k \mid \mathbf{x}) \tag{4.15}$$

For implementation, $p(x_i \mid y)$ and $p(y)$ have to be estimated from data. While $p(y)$ can be estimated as relative class frequency from the training set, the conditional probability model is typically estimated parametrically, for example by assuming a normal distribution for Gaussian Naïve Bayes, but non-parametric approaches to density estimation have also been proposed [154].

4.3.2 Logistic Regression

Logistic Regression (LR) is a linear classification model related to NB [217], which compares the odds of possible outcomes for individual classes. These outcomes can

be described as $p(y_0 \mid \mathbf{x})$ and $p(y_1 \mid \mathbf{x}) = 1 - p(y_0 \mid \mathbf{x})$, which are estimated as linear parameters applied to the sigmoid function for LR:

$$\hat{y}(\mathbf{x};\theta) = p(y_0 \mid \mathbf{x};\theta) = \sigma(\mathbf{x}^\top \underbrace{\mathbf{W} + b}_{\theta}) = \frac{1}{1 + \exp\left(-(\mathbf{x}^\top \mathbf{W} + b)\right)} \tag{4.16}$$

LR is then trained using the maximum likelihood principle, that is, the likelihood L of the model parameters θ with respect to the data $(x_i, y_i) \sim \mathcal{D}$ is maximized, which is typically approached by gradient descent [12]. Parameters can also be optimized by minimizing the log-likelihood J, which is equivalent to the Cross Entropy loss (Eq. 4.9) resulting in the following optimization problems:

$$\max_\theta L(\theta \mid \mathcal{D}), \qquad L(\theta \mid \mathcal{D}) = \prod_i^N \hat{y}(\mathbf{x};\theta)^{y_i}(1 - \hat{y}(\mathbf{x};\theta))^{1-y_i} \tag{4.17}$$

$$\min_\theta J(\theta \mid \mathcal{D}), \qquad J(\theta \mid \mathcal{D}) = -\sum_{i=1}^N y_i \log \hat{y}(\mathbf{x};\theta) + (1 - y_i)\log(1 - \hat{y}(\mathbf{x};\theta)) \tag{4.18}$$

In practice, ℓ_1 or ℓ_2 regularization is often added to the objective, which improves the generalization performance for high-dimensional features [218, 180]. Multiple classes can be modeled using the one-versus-all scheme [12]. Note that in contrast to generative methods, such as NB, which model $p(\mathbf{x} \mid y)$ or $p(\mathbf{x}, y)$, LR is a discriminative model directly modeling $p(y \mid \mathbf{x})$ [12, 217]. It is also worth noting that, as Eq. 4.16 shows, LR can be understood as NN with one layer and sigmoid activation optimized through Cross-Entropy.

4.3.3 k-Nearest Neighbors

The k-Nearest Neighbors (kNN) approach is a distance-based classification algorithm, which decides on the class based on the majority class of the k nearest neighbors with respect to the distance metric. The model is therefore instance-based, i.e. the training instances are directly "memorized" and compared with the samples to be predicted instead of generalizing an abstract decision function. To efficiently search the space of neighbors, several data structures such as Ball Trees [224] have been adopted in practical implementations [232] and approximations have been proposed to handle large datasets [52]. As distance measure any metric can be used, which fulfills the three metric axioms identity of indiscernibles, symmetry, and the triangle inequality. While in practice, distances based on the ℓ_p norm such as the Euclidean distance are often used, the most suitable metric choice is dependent on the dataset [2]. Besides the metric, feature transformations such as min-max scaling or standardization can have a strong influence on the performance of a kNN model [315]. Formally, let $(X^{\text{train}}, Y^{\text{train}}) = \mathcal{D}^{\text{train}}$ be the set of training samples

with features X^{train} and labels Y^{train}. Given a sample $\hat{\mathbf{x}}$ to classify, let the relevant *neighborhood* $\mathcal{N}$, i.e., the subset of k nearest neighbors with respect to a distance d, be

$$\begin{aligned} \mathcal{N}_k(\hat{\mathbf{x}}; \mathcal{D}^{\text{train}}) &= (X^{\mathcal{N}}, Y^{\mathcal{N}}) \subset \mathcal{D}^{\text{train}} \\ &\text{with } |X^{\mathcal{N}}| = k \\ &\text{satisfying } \forall \mathbf{x} \in X^{\mathcal{N}},\ \forall \bar{\mathbf{x}} \in X^{\text{train}} \setminus X^{\mathcal{N}} : d(\mathbf{x}, \hat{\mathbf{x}}) \leq d(\bar{\mathbf{x}}, \hat{\mathbf{x}}). \end{aligned} \tag{4.19}$$

The predicted class for $\hat{\mathbf{x}}$ is then given by

$$\hat{y} = \underset{y}{\operatorname{argmax}} \, |\{i : y_i \in Y^{\mathcal{N}} \mid y_i = y\}| \tag{4.20}$$

4.3.4 Support Vector Machine

Support Vector Machine (SVM) is a discriminative classification approach based on the idea of finding the class boundary, also called the hyperplane, between two classes $y \in \{1, -1\}$, where $P(y = -1 \mid \mathbf{x}) = P(y = 1 \mid \mathbf{x})$. This hyperplane for a linear SVM is defined as linear equation

$$\mathbf{W}^\top \mathbf{x} + b = 0 \tag{4.21}$$

with learnable parameters $\mathbf{W}, b$. For linearly separable instances, two parallel *support hyperplanes* can be constructed such that for all $(\mathbf{x}, y)$

$$\mathbf{W}^\top \mathbf{x} + b \geq 1 \text{ for } y = 1 \tag{4.22}$$

$$\mathbf{W}^\top \mathbf{x} + b \leq -1 \text{ for } y = -1 \tag{4.23}$$

which geometrically have a distance $m = \frac{2}{\|\mathbf{W}\|}$, typically called *margin*. Therefore, the objective of the SVM is to maximize the margin, i.e. minimizing $\mathbf{W}$ with respect to Eq. 4.22, and 4.23 which can be summarized as

$$\text{minimize } \frac{1}{2}\|\mathbf{W}\|^2 \text{ subject to } y_i(\mathbf{W}^\top \mathbf{x}_i + b) \geq 1 \tag{4.24}$$

for each data point $(\mathbf{x}_i, y_i) \in \mathcal{D}^{\text{train}}$ [64], which can be solved as a standard quadratic optimization problem [12], such that there are instances on both sides of the hyperplane, which have a distance of $\frac{1}{\|\mathbf{W}\|}$. As the support hyperplanes are defined to maximize the margin between instances of different classes, some instances, called *support vectors*, are located directly on the support hyperplane, and thus $\mathbf{W}$ can be written as a linear combination of those instances with coefficients $\alpha_i \neq 0$ for support vectors $\mathbf{x}_i$ [278]:

$$\mathbf{W} = \sum_i \alpha_i \mathbf{x}_i \tag{4.25}$$

As data is generally not perfectly separable, Eq. 4.24 is typically relaxed to allow deviation from the margin by introducing *slack* variables ξ_i by $y_i(\mathbf{W}^\top \mathbf{x}_i + b) \geq 1 - \xi_i$.

As deviation from the margin by slack variables $\xi_i > 0$ must be avoided as far as possible for better generalization, the slack variables are regularized by introducing a penalty $\sum_i \xi_i$ to the objective scaled by a hyperparameter C.

$$\text{minimize } \frac{1}{2}\|\mathbf{W}\|^2 + C\sum_i \xi_i \text{ subject to } y_i(\mathbf{W}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \tag{4.26}$$

To solve non-linear problems, a non-linear transformation of the feature space X to a potentially higher-dimensional vector space $\mathcal{X}$ with a *mapping* function $\phi : X \to \mathcal{X}$ can be applied, in which the problem is linearly separable and thus solvable by the SVM. The *kernel trick* [35] avoids this explicit mapping by defining a kernel function $K : X \times X \to \mathbb{R}$, which satisfies $K(\mathbf{x}, \hat{\mathbf{x}}) = \phi(\mathbf{x})^\top \phi(\hat{\mathbf{x}})$.

$$\text{minimize } \frac{1}{2}\|\mathbf{W}\|^2 + C\sum_i \xi_i \quad \text{s.t.} \quad y_i \mathbf{W}^\top \phi(\mathbf{x}_i) + b \geq 1 - \xi_i \tag{4.27}$$

$$\overset{(4.25)}{\Leftrightarrow} \text{minimize } \frac{1}{2}\|\mathbf{W}\|^2 + C\sum_i \xi_i \quad \text{s.t.} \quad y_i \sum_j \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) + b \geq 1 - \xi_i \tag{4.28}$$

A kernel often applied with SVMs is the Radial-Basis-Function (RBF) or Gaussian kernel, which is also used for Gaussian kernel density estimation having $\gamma = \frac{1}{2\sigma^2}$, as detailed in Section 6.2.2.

$$K^{\text{RBF}}(\mathbf{x}, \hat{\mathbf{x}}; \gamma) = \exp\left(-\gamma \|\mathbf{x} - \hat{\mathbf{x}}\|^2\right) \tag{4.29}$$

4.3.5 One-Class Support Vector Machine

The One-Class SVM (OC-SVM) as proposed by Schölkopf et al. [279] varies the idea of a separating hyperplane such that the distance from the origin in feature space is maximized for given samples of the known class. The precise formulation is given as follows:

$$\text{minimize } \frac{1}{2}\|\mathbf{W}\|^2 + \frac{1}{\nu n}\sum_i \xi_i - \rho \quad \text{s.t.} \quad \begin{aligned} &\mathbf{W}^\top \phi(\mathbf{x}_i) \geq \rho - \xi_i \\ \Leftrightarrow\ &\textstyle\sum_j \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \geq \rho - \xi_i \end{aligned} \tag{4.30}$$

The smoothness hyperparameter C from Eq. 4.26 is hereby replaced by $\frac{1}{\nu n}$, with $\nu \in (0, 1)$ setting an upper bound on the fraction of training outliers and a lower bound on the number of training examples used as Support Vector [279]. The hyperplane defined by $\mathbf{W}$ and ρ is thereby constructed to have a maximum distance from the origin and to separate the samples of known class from the origin. The kernel trick can be applied analogously to Eq. 4.28 as given in Eq. 4.30 for the dual formulation.

4.3.6 Tree-based Approaches

Another class of non-parametric ML algorithms is building on tree data structures and divide the input space into local regions hierarchically.

Decision Trees

The Decision Tree (DT) is the most basic tree-based ML approach and is composed of decision nodes and leaves. At every decision node, the input space is divided by a split criterion recursively until a leaf node is reached. Leaves therefore correspond to a local region which has a class label assigned for classification DTs. The decision boundary is finally defined by the split criteria traversed from the root node to the leaf. More formally, a split criterion for node n is given by a function $f_n(\mathbf{x}) : \mathbb{R}^d \to Y$ defined as threshold function f_n^u for the univariate or f_n^m for the multivariate decision tree, which takes into account one input dimension (univariate) or all input dimensions (multivariate).

$$f_n^u(\mathbf{x}) = \begin{cases} f_{n_\mathrm{L}}^u(\mathbf{x}) & \text{if } x_i + w_n > 0 \\ f_{n_\mathrm{R}}^u(\mathbf{x}) & \text{otherwise} \end{cases} \tag{4.31}$$

$$f_n^m(\mathbf{x}) = \begin{cases} f_{n_\mathrm{L}}^m(\mathbf{x}) & \text{if } \mathbf{w}_n^\top \mathbf{x} + w_n > 0 \\ f_{n_\mathrm{R}}^m(\mathbf{x}) & \text{otherwise} \end{cases} \tag{4.32}$$

$$\mathcal{D}_{|n_\mathrm{L}} = \{(\mathbf{x}, y) \in \mathcal{D}_{|n} \mid x_i + w_n > 0\} \tag{4.33}$$

$$\mathcal{D}_{|n_\mathrm{R}} = \{(\mathbf{x}, y) \in \mathcal{D}_{|n} \mid x_i + w_n \leq 0\} \tag{4.34}$$

Here, f_{n_L} and f_{n_R} denote the subsequent decision functions for the left or right child node of f_n with the local region covering the data subset $\mathcal{D}_{|n}$, which splits into child regions and associated subsets $\mathcal{D}_{|n_\mathrm{L}}$, $\mathcal{D}_{|n_\mathrm{R}}$. As the predicted class label is only determined by leaf nodes $f_{l_i} : \mathbf{x} \mapsto \hat{y}, l_i \in \mathbb{L}$, the tree is recursively composed of decision functions along the path from the root node to the leaves:

$$f_\mathrm{root}(\mathbf{x}) = f_*(f_*(f_*(\cdots f_{l_i}(\mathbf{x}) \cdots)))$$

For a regression task, the decision function of the leaves maps $\hat{y}$ to a numeric given by the mean of y of the training data in the leaf l_i, $\hat{y} = \mathbb{E}[y : (\mathbf{x}, y) \in \mathcal{D}_{|l_i}]$. Accordingly, a label for a sample $\mathbf{x}$ can be indexed as $\hat{\mathbf{y}}_{q(\mathbf{x})}$ with respect to a given tree structure $q : \mathbb{R}^d \to \{1, \ldots, |\mathbb{L}|\}$ defining the leaf index $i : l_i$ associated with $\mathbf{x}$ of the expectation leaf vector $\hat{\mathbf{y}} = (\hat{y}_{l_i})_{l_i \in \mathbb{L}}$.

Many different trees can be constructed, which fit training data without any error. In particular the most (unnecessarily) complex tree could fit a local subspace around each training sample as leaf. However, this would result in a highly overfitted model that probably does not generalize well to the test data [39]. The problem of finding the smallest tree, however, is NP-complete [173]. To learn a decision tree in practice,

several algorithms have been proposed, which mostly greedily split according to a heuristic H, e.g. ID3 [243] by choosing the attribute which minimizes the entropy regarding the training data with improvements, such as C4.5 [244], or CART [42] which uses the Gini impurity as split criterion and prunes the tree after creation by removing unhelpful branches. Theoretical analyses suggest that the difference between both criteria is negligible [245]. For our experiments, we use the Gini criterion. For a class k in node n, p_{n_k} denotes the proportion of observations. Gini impurity and entropy are given by:

$$p_{n_k} = \frac{|\{(x,y) \in \mathcal{D}_{|n} \mid y = k\}|}{|\mathcal{D}_{|n}|} \tag{4.35}$$

$$H_{\text{Gini}}(\mathcal{D}_{|n}) = \sum_k p_{n_k}(1 - p_{n_k}) \tag{4.36}$$

$$H_{\text{Entropy}}(\mathcal{D}_{|n}) = -\sum_k p_{n_k} \log(p_{n_k}) \tag{4.37}$$

The set of classification and regression decision trees is given by

$$\text{CART} = \{f : \mathbf{x} \mapsto \hat{\mathbf{y}}_{q(\mathbf{x})}\} \text{ with } q : \mathbb{R}^d \to \{1, \ldots, |\mathbb{L}|\} \text{ and } \hat{\mathbf{y}} \in \mathbb{R}^{|\mathbb{L}|} \tag{4.38}$$

having the set of leaves noted as $\mathbb{L}$ and the tree structure determined by q for each tree.

Random Forests

Random Forest (RF) is a bootstrap aggregating (bagging) ensemble approach that is built on DTs. While the first publication building ensembles of DTs suggested sampling on feature level by randomly constructing several feature subsets and using the complete dataset to train a model each [138], Breiman [41] coined the term *random forest* and made the bagging approach for DT ensembles popular. By randomly sampling with replacement from the training dataset $\mathcal{D}$, m sub-datasets $\mathcal{D}_i$ are created and individually used to train a DT model each. Since each DT is constructed on a different dataset, each DT selects other split criteria, helping to avoid overfitting [41]. The decisions of the individual trees are combined by voting, i.e. assigning the class label predicted most often by the individual classifiers or by averaging the prediction probabilities (Eq. 4.35) of the individual trees. For our experiments we use the scikit-learn implementation [232], which combines sample-based bagging and feature-subspace sampling by restricting the subset of features to be considered by the splitting criterion.

XG-Boost

XG-Boost (named after eXtreme Gradient Boosting) is a popular gradient boosted tree algorithm, often chosen for its "superior performance over deep learning" [290]

on tabular data. Similar to RF, XG-Boost uses ensembles of decision trees combined to an additive model (Eq. 4.39) iteratively, i.e. the model includes more and more trees, which in sum improve the objective most. The selection of a tree added to the model is decided by optimizing a training objective $\mathcal{L}$ (Eq. 4.40) including a loss l (such as MSE) and a regularization term Ω. Here, $\hat{y}_i^{(t)}$ denotes the prediction of the i-th sample in the t-th iteration and a specific f_t is added, which improves the model the most. The regularization term $\Omega(f) = \gamma|\mathbb{L}| + 0.5\lambda\|w\|^2$ penalizes the complexity of the tree f_t to be added by the number of leaf nodes $|\mathbb{L}|$ and the leaf weights $w \in \mathbb{R}^{|\mathbb{L}|}$.

In practice this optimization objective is implemented by its second-order Taylor approximation $\tilde{\mathcal{L}}^{(t)}$ (Eq. 4.41). For a more detailed derivation of this approximation, see [55].

$$\hat{y}_i^{(t)} = \phi^{(t)}(\mathbf{x}_i) = \sum_{e=1}^{t} f_e, \; f_e \in \text{CART} \tag{4.39}$$

$$\mathcal{L}^{(t)} = \sum_{i=1}^{N} l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t) \tag{4.40}$$

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{N} \left[l(y_i, \hat{y}_i^{(t-1)} + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t(\mathbf{x}_i)^2 \right] + \Omega(f_t) \quad \simeq \quad \mathcal{L}^{(t)} \tag{4.41}$$

$$\text{with } g_i = \frac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}}, \; h_i = \frac{\partial^2 l(y_i, \hat{y}^{(t-1)})}{\partial (\hat{y}^{(t-1)})^2}, \; \Omega(f_t) = \gamma|\mathbb{L}| + \frac{1}{2}\lambda \sum_{j=1}^{|\mathbb{L}|} \hat{y}_j^2$$

$$\mathcal{L}_{\text{split}} = \frac{1}{2} \left[\frac{\left(\sum_{i \in \mathcal{D}_{|n_\text{L}}} g_i\right)^2}{\sum_{i \in \mathcal{D}_{|n_\text{L}}} h_i + \lambda} + \frac{\left(\sum_{i \in \mathcal{D}_{|n_\text{R}}} g_i\right)^2}{\sum_{i \in \mathcal{D}_{|n_\text{R}}} h_i + \lambda} - \frac{\left(\sum_{i \in \mathcal{D}_{|n}} g_i\right)^2}{\sum_{i \in \mathcal{D}_{|n}} h_i + \lambda} \right] - \gamma \tag{4.42}$$

In Eq. 4.41 and Eq. 4.42 the terms g_i and h_i denote the first- and second-order gradient statistics for l and Ω regularizes the number of leaves $|\mathbb{L}|$ and their scores $\hat{y}_j$.

The final model after E iterations is given by $\hat{y}_i = \hat{y}_i^{(E)}$. In practice, $\hat{\mathcal{L}}$ is often rearranged to the splitting criterion $\mathcal{L}_{\text{split}}$ [55] as given in Eq. 4.42. XG-Boost additionally incorporates shrinkage [98], i.e. rescaling weights by a given factor after each iteration and feature-subspace sampling as introduced with RF.

Isolation Forest

Isolation Forest (IF) [186] is a Random Forest-based anomaly detection approach. The key observation motivating this approach is that anomalies are infrequent and differ from normal data, making them more susceptible to isolation than normal samples [186]. When constructing decision trees with random partitioning criteria

(isolation trees) by randomly selecting a feature and split value within the value range of the selected feature, the dataset is partitioned and data instances are isolated in leafs until either the height limit is reached, each leaf contains a single instance or all remaining data points have the same value. It can be observed that the paths from the root to an anomalous sample are notably shorter than paths for normal instances [186]. For IF, t isolation trees are combined similarly to RF while the length of the path $h(x)$ defined by the nodes between the root and the leaf is averaged over the trees $\mathbb{E}[h(x)]$ as a normality score. To obtain an anomaly score $s(\mathbf{x}; n)$ for a sample $\mathbf{x}$ and $n = |\mathcal{D}|$, this average path length is normalized by $c(n)$, the path length of a unsuccessful search in Binary Search Trees [186].

$$c(n) = 2H(n-1) - (2(n-1)/n) \tag{4.43}$$

$$s(\mathbf{x}; n) = 2^{-(\mathbb{E}[h(\mathbf{x}])/c(n))} \tag{4.44}$$

Hereby $H(i)$ denotes the harmonic number estimated by $H(i) = \ln(i) + \gamma$ with Euler's constant $\gamma \approx 0.5772$. IF has two hyperparameters, the number of trees t and the size of subsampling bagging ψ defining the size of each subset sample $X' \subset \mathcal{D}$ used to create the respective tree. The height limit l of the trees is defined in relation to ψ by $l = \lceil \log_2 \psi \rceil$. Similarly to RF, IF is in addition to subsampling in sample space (bagging), in practice often implemented with feature space subsampling, introducing an additional hyperparameter φ defining the proportion of sampled features per tree [232].

4.4 Anomaly Detection

Anomaly Detection (AD) refers to finding data points "different from the remaining data" [5]. Hawkins [127] discusses two origins of such outliers: Either, data are drawn from an "outlier-prone" distribution, heavy-tailed distributions "which go to zero slowly", or data arise from two different distributions, a "basic distribution" generating normal data and a "contaminating distribution", which is outlier-prone. While the second mechanism can be understood as a special case of the first, it better reflects the definition of Chandola et al. [50] understanding anomalies as data, that does not conform to expected behavior:

> Anomalies are patterns in data that do not conform to a well defined notion of normal behavior. [...] Anomalies might be induced in the data for a variety of reasons, such as malicious activity, for example, credit card fraud, cyber-intrusion, terrorist activity or break-down of a system, but all of the reasons have the common characteristic that they are interesting to the analyst.
>
> Chandola et al., Anomaly detection: A survey [50]

From a data point of view, this can be operationalized in several ways: While reconstruction error-based models such as Auto-Encoder try to capture a compressed representation of normal data and measure abnormality by the difficulties reproducing such samples (cf. Eq. 4.45), other approaches such as Isolation Forest (cf. Section 4.3.6) are based on frequent feature combinations or try to learn a decision boundary regarding the normal class as (One-Class) Support Vector Machine (cf. Section 4.3.5) for example.

Expanding this data-driven perspective, anomalies can not only be understood as improbable outcomes of probability distributions, but the process or origin from which they came from can be considered as well to define the requirements for detecting anomalies:

Although *point anomalies* can be considered from either point of view, since the anomaly is present or absent in each data point with respect to all other data points, the broader, process or behavioral definition allows one to consider *contextual anomalies*. For these, an outcome is not conspicuous when considered in isolation, while consideration of contextual properties allows a clear distinction as anomalous with respect to that context. This context can be temporal, for example, a large number of transactions occurring outside typical working hours, spatial, for example, credit card transactions issued in another country, or contextual with regard to other attributes, for example, the price with regard to the product group. *Collective anomalies* denote anomalous behavior that manifests itself in a group or sequence of data points, for example slowly draining an account with a number of transactions, all inconspicuous for themselves.

For the majority of our experiments, we only consider point and contextual anomalies and shift sequential aspects to dataset preparation. For example, for the IEEE-CIS fraud dataset, counting features aggregate sequences, while for the SAP dataset, related transactions for purchase and sales or accounting have been aggregated by foreign key constraints. For the extrinsic evaluation for Windows Audit Logs (Section 7.1), however, a sequential model is used, as a sample of malicious behavior in this application is characterized by a sequence of audit log events as collective anomalies.

Several challenges arise with AD, namely imprecise boundaries of normal and abnormal data, which is strongly related to the difficulties encountered in precisely defining or operationalizing *non-normal behavior* for a given task, possible drifts in the characteristics of normal behavior, the incentive to obfuscate anomalies when malicious intent is involved, and the availability of data, particularly labeled training data [50]. Training data with labels for normal instances and anomalies allow the application of supervised ML approaches as for our experiments in Section 8.1. Whereas the consequent problem of class-imbalance between normal data and anomalies can be addressed in several ways, e.g. by introducing class weights or by sampling including techniques such as the Synthetic Minority Oversampling TEchnique (SMOTE) [51], the availability of labeled datasets of appropriate size to

apply deep learning based methods is usually not given [50]. Consequently, AD approaches beyond supervised ML are developed, which are trained in an unsupervised or semi-supervised setting and do not rely on labeled anomalies. Semi-supervised approaches typically incorporate few labeled and a large number of unlabeled data points to yield a better performance in comparison to training only on the labeled data with a supervised approach or discarding the labels to train with unsupervised algorithms.

Whereas for AD the term semi-supervised is also used when only one type of data (typically normal data) is available, resulting in a labeled dataset containing only one class [50, 5], for ML this scenario is generally referred to as One-Class Classification (OCC) or Positive Unlabeled Learning (PU). In practical terms, with OCC the unlabeled data is not necessarily used for training or improving the model, and especially in the AD setting, the normal class is by definition more frequent. Therefore, from two perspectives, it is not given that learning is done on few labeled and many unlabeled data as for semi-supervised learning. In the literature, there are also approaches that combine OCC and semi-supervised learning by allowing a few positive labels to be considered alongside a lot of unlabeled normal data [264] and related approaches, e.g. in online novelty detection, where new classes can be added and hence treated differently from previously unseen classes [203] or transfer learning, using labeled data from an unrelated task to learn feature representations [235]. Moreover, Aggarwal [5] argues that OCC trained on normal data is just a more precise application of unsupervised methods and coincides with OCC from a methodological perspective, as both in general aim to model normal behavior. This applies, for instance, for reconstruction-based models such as Auto-Encoder (AE), which can be trained on and applied to unlabeled and potentially contaminated data as well as benign data reflecting normal behavior. For reconstruction-based models the ℓ^2 norm is commonly used as anomaly score as [50]:

$$as(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|_2 \tag{4.45}$$

Both settings are identical up to the situation, when for a sample the precise class has to be decided based on the anomaly score given by the AE: Often threshold values are chosen to mark the border between normal data and anomalies (cf. the challenges above), which in the case of uncontaminated normal training data can be selected, for example, according to the "most abnormal" normal sample present in the held out training data, while for potentially contaminated training data, other approaches must be pursued [26, 11, 216]. However, from a non-methodological point of view, the interpretation of OCC as an unsupervised method is arguable, as Perera et al. [234] conclude that OCC should be considered "to be a supervised learning problem".

The output of an AD approach is typically either an *anomaly score* as illustrated with the AE example above, which corresponds to the degree to which an instance is to be considered anomalous, or an *anomaly label* as given for ML classification

tasks, indicating whether a sample is detected as anomaly or not. A scoring based approach allows to focus on the n most anomalous samples ordered by rank and by this to choose n appropriately for the task and application, which for example is useful for evaluation of fraudulent transactions by controlling or auditors. On the other hand, abnormal scores raise the problem of choosing n or a score threshold appropriately, as the AD system is to be deployed in practice. For anomaly labels, the number of positives including true and false positives (which corresponds to the n most anomalous samples in a scoring setting) has to be calibrated during training, e.g. by hyper-parameter choices of the model. In between are models which are classifiers but allow the evaluation of per-class scores, for example, by the decision function of an SVM. With different output formulations, different evaluation metrics are applicable.

4.5 Evaluation Metrics

In this section, first, the basic metrics for evaluating ML models quantitatively are introduced. We then discuss the impact of skewed class label distributions on different metrics, as they are commonly given for AD applications.

4.5.1 Basic Metrics

Models with an anomaly label as output, i.e. classification-based models, decide for each instance on the predicted class. Therefore, all evaluation metrics[1] commonly used in supervised classification can be applied. The most basic differentiation that is combined in most other metrics is the notion of *positives* (P) and *negatives* (N) corresponding to both classes in a binary classification setting such as anomalies and normal instances. For a predicted label, this notion expands to both aspects, the *annotated* label and the *predicted* label, introducing *True Positives* (TP) for samples, which are positive by both annotation and prediction, the *False Positives* (FP), which are predicted positively but actually labeled negatively, *True Negatives* (TN) which are predicted and labeled negatively, and *False Negatives* (FN) which are predicted negatively but actually labeled positively. Thus, a perfect model has no FN and FP samples. These terms are often summarized in a *confusion matrix*, a tabular view of the counts for the four categories, marginalized by the predicted and actual class distribution. From these counts, several other metrics can be derived.

Precision (Eq. 4.46) is given as how many of the positively predicted (TP + FP) instances are correct. In case all samples are classified as negative, we set the precision to 0. The *recall* (Eq. 4.47), also called True Positive Rate (TPR) or sensitivity, is defined as the number of positives found in relation to all positives in the dataset. The False

[1] We use the term *metric* in this case not in the mathematical meaning of *distance metric* for a metric space but in terms of a *measure* to compare two models quantitatively.

Positive Rate (FPR), given in Eq. 4.48, can be described as how many "false alarms" are indicated by the model. TPR and FPR are commonly used evaluation metrics for intrusion detection [86] as they indicate the two most important aspects: How many true anomalies are found and need to be examined and how many incidents raise an alarm. The F_1 measure denotes the harmonic mean of precision and recall by combining both aspects in one metric. In case of a recall and precision of 0, we set F_1 to 0. When anomaly score-based methods are used, the TP, FP, etc. statistics are always given with respect to a threshold t such that for a model $f(\mathbf{x}) = \hat{y}_{\text{score}}$, the set of instances predicted as an anomaly is given as $S(t) := \{(\mathbf{x}, y) : f(\mathbf{x}) \geq t\}$. Therefore, we denote, for example, TP with respect to t as TP_t.

$$\text{precision}(t) = \frac{|\text{TP}_t|}{|\text{TP}_t| + |\text{FP}_t|} = \frac{|S(t) \cap \text{P}|}{|S(t)|} \tag{4.46}$$

$$\text{recall}(t) = \text{TPR}(t) = \frac{|\text{TP}_t|}{|\text{P}|} = \frac{|S(t) \cap \text{P}|}{|P|} \tag{4.47}$$

$$\text{FPR}(t) = \frac{|\text{FP}_t|}{|\text{FP}_t| + |\text{TN}_t|} = \frac{|S(t) \setminus \text{P}|}{|\mathcal{D} \setminus \text{P}|} \tag{4.48}$$

$$\text{F}_1(t) = 2 \cdot \frac{\text{precision}(t) \cdot \text{recall}(t)}{\text{precision}(t) + \text{recall}(t)} = \frac{|\text{TP}_t|}{|\text{TP}_t| + \frac{1}{2}(|\text{FP}_t| + |\text{FN}_t|)} \tag{4.49}$$

From precision, recall and FPR, two evaluation metrics can be derived, which are very common in anomaly detection and indicate the trade-off between false alarms and missed anomalies. The Receiver Operating Characteristic (ROC) curve, given in Eq. 4.50, opposes FPR on the x axis and TPR on the y axis in a curve $\text{ROC} : (0, 1) \rightarrow (0, 1)$, while Precision-Recall (PR) curve, given in Eq. 4.51, opposes the recall TPR on the x axis and precision on the y axis, which is typically implemented by increasing t so that $S(t)$ grows each time by one instance and calculating the respective statistics each.

$$\text{ROC}(\,\cdot\,) = \{(\text{FPR}(t), \text{TPR}(t)) : t \in (-\infty, \infty)\} \tag{4.50}$$

$$\text{PR}(\,\cdot\,) = \{(\text{recall}(t), \text{precision}(t)) : t \in (-\infty, \infty)\} \tag{4.51}$$

$$\text{ROC-AUC} = \int_0^1 \text{ROC}(t)dt \tag{4.52}$$

$$\text{AP} = \sum_t (\text{recall}(t) - \text{recall}(t-1))\ \text{precision}(t) \tag{4.53}$$

$$r_{min} = |\{\mathbf{x} \in \mathcal{D} \ :\ score(\mathbf{x}) \geq \min_{\mathbf{f} \in \mathcal{D}|_F} score(\mathbf{f})\}| \tag{4.54}$$

As PR and ROC are not metrics that result in a single score, for quantitative evaluation, the Area Under the Curve (AUC) is reported. As for practical implementations ROC and PR are only given at the threshold points of each sample, an interpolation is required to draw a connected curve and to integrate over this curve to determine the AUC. While for ROC this interpolation and therefore ROC-AUC (Eq. 4.52) is

straightforward as the curve is increasing monotonically stepwise, the interpolation for PR is more complicated as the precision does not necessarily change linearly with the recall [282]. Linear approximation in this case gives overly optimistic performance estimates [71]. As a single-value quantitative proxy, we use Average Precision (AP), given in Eq. 4.53, which is calculated by averaging the precision with respect to the change in recall for each anomaly score t as a threshold. Finally, r_{min} describes the number of samples included within the score of the least anomalous sample of the anomaly class according to the anomaly-*score* (cf. Eq. 4.45) with $\mathcal{D}$ being the complete dataset and $D|_F$ the subset of anomalous samples in the dataset.

4.5.2 Evaluation Metrics and Skewed Class Distributions

As discussed in Chapter 1, AD in the domain of transaction data typically deals with very rare anomalous events, e.g., a ratio of 0.0003 between normal samples and anomalies for SAP (see Table 5.1 for an overview). This property translates to highly skewed class distributions when considering normal and non-normal as class labels, which influence the focus and expressiveness of the evaluation metrics by different means. In this section, we will discuss the consequences of this class imbalance on the evaluation metrics and show their different focus for practical applications in such an AD setting.

The F_1 score calculation has the drawback that for a highly imbalanced class distribution very few instances with wrong class assignments have a high impact on the score. Consider for example datasets with 1, 10, 100 and 1000 fraud cases and 10000 non-fraud each. In Fig. 4.1, the impact of 0 to 100 false positives is depicted for a classifier judging all instances labeled fraud correctly with an increasing number of false positives. The different curves correspond to the F1 score calculated for the different example datasets with a varying number of fraud samples: While for a larger number of fraud cases (e.g., 1000) the influence of 100 false positives with regard to the F1 score is negligible, for fewer fraud cases the score quickly deteriorates. In a real-world application considering, for example, an auditor manually inspecting the predicted fraud cases, however, a list of 110 transactions containing 10 actual fraud cases is economically feasible, which is not reflected by the differences in score well. The issues regarding the practicability of the scores become even more distinct if one compares two classifiers: Classifier A correctly classifies all fraud cases, but has 100 false positive results and classifier B only predicts 2 of 10 fraud cases correctly and has 10 false positives. Classifier A will yield an F_1 score of 0.166, although correctly predicting all fraud cases. Classifier B surpasses the performance of classifier A with an F_1-score of 0.188 although most fraud cases are overlooked. These examples suggest that binary F_1-score is not a perfect metric on its own when it comes to an imbalanced fraud detection scenario, which is also supported by the findings of Fourure et al., showing how biased evaluation protocols and the contamination rate, i.e. the rate of anomalies in the test set, can artificially increase the F_1-score [96].

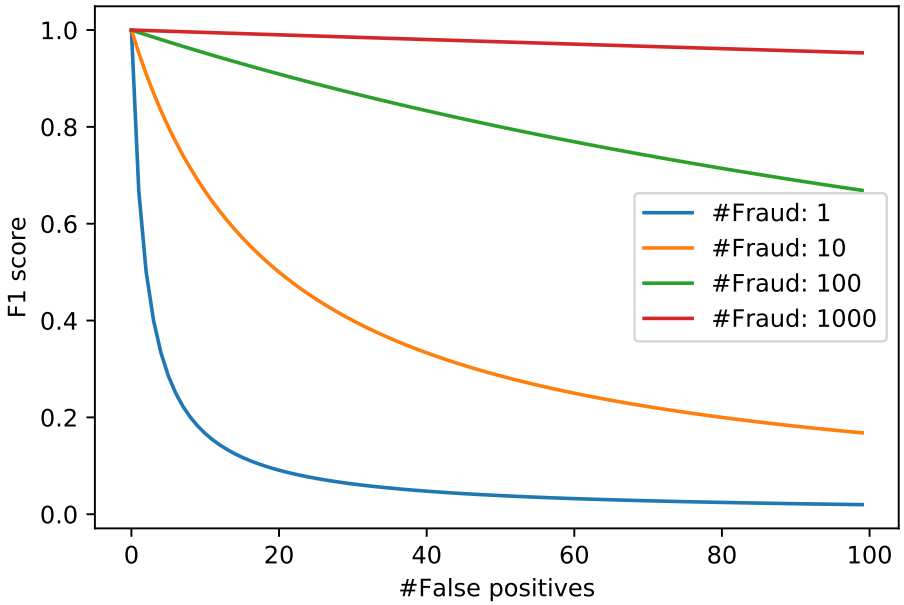


Figure 4.1: Impact of class imbalance and false positives on (binary) F1 score. For a very imbalanced dataset with 1 fraud sample and 10 000 normal samples, the F1 score quickly deteriorates for few false positives (blue curve).

As ROC judges a model's output regarding the true positive rate and false positive rate for a variable threshold, it can be used to analyze the performance under a variable number of instances to be considered, e.g. for an auditor: The user-determined threshold decides how many instances are considered positive regarding the model's output. A perfect classifier will, by increasing the positive rate, only introduce (more) true positives. However, an imperfect classifier will introduce more false positives. Therefore, it can be considered as a trade-off between acquiring all fraudulent instances and revising too many false alarms in the fraud detection application scenario. Although ROC fits our evaluation setting well, for very imbalanced class distributions as given in our SAP dataset, it is considered as "optimistic" [71], especially when it comes to a very low number of instances in the minority class [40].

The PR curve is considered to be more suitable for highly imbalanced data [295, 228, 269]. In contrast to the ROC, the PR curve can be understood as a trade-off between precision and recall. Since recall and true positive rate denote the same aspect, the difference between both evaluation metrics is between the false positive rate for ROC and the precision for PR.

$$\text{precision} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FP}|} = 1 - \frac{|\text{FP}|}{|\text{FP}| + |\text{TP}|} \tag{4.46}$$

$$\text{FPR} = \frac{|\text{FP}|}{|\text{FP}| + |\text{TN}|} \tag{4.48}$$

Recalling and rearranging Eqs. (4.46) and (4.48), it becomes visible that both metrics are related and differ (for their application in ROC respectively PR) primarily regarding the normalization by true negatives or true positives. This can explain the "optimistic" behavior for ROC, since the number of true negatives in an imbalanced fraud setting is by magnitudes larger than the number of true positives. In Fig. 4.2 a comparison between precision and false positive rate for a varying number of detected fraud cases (TP) can be interpreted in two ways: On the one hand, FPR and therefore ROC is invariant with respect to the class distribution [91]. This means that ROC scores are comparable on different datasets (splits) with a varying proportion of fraud, but have the drawback that false positives have little impact on the score [291]. On the other hand, the observation from Fig. 4.1 also applies for the PR curve, arguably overemphasizing the influence of the true positives and introducing a dependency on the class label distribution. This makes reported PR performances difficult to compare between different datasets or splits of the dataset with a varying proportion of fraud.

As discussed in this section, all metrics have individual advantages and drawbacks, and especially regarding class imbalance the question of which metric is most suitable depends on the valuation of false positives versus false negatives, which has to be rated individually according to the application: In an automated online credit card fraud detection setting, the cost of unblocking small-scale transactions

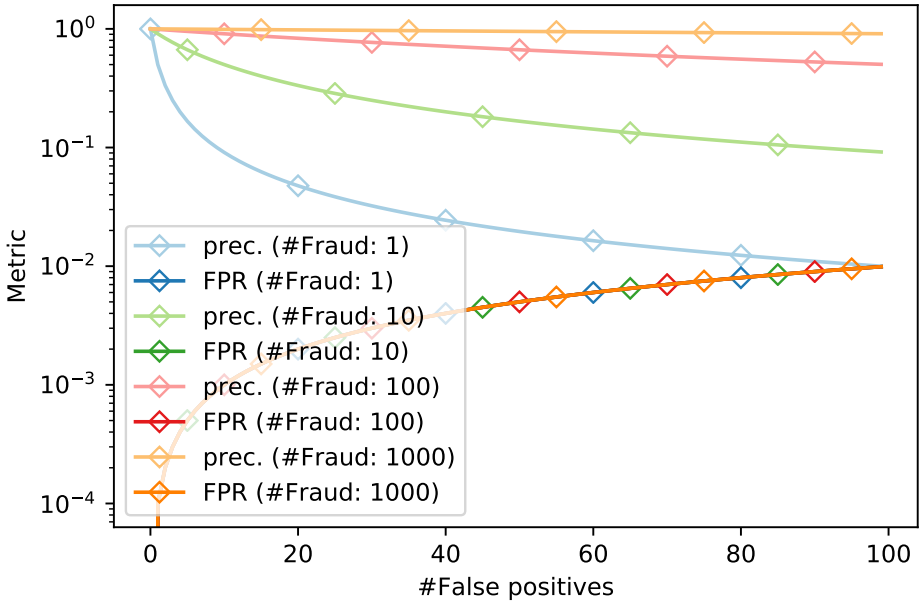


Figure 4.2: Impact of class imbalance (10 000 benign samples fixed, varying number of fraud samples) and number of false positives on precision (prec.) and FPR. The FPR curves are independent of the number of fraud samples and thus are identical, while the PR curves depend on the number of fraud samples in the dataset.

for falsely accused benign transactions will probably exceed the financial benefit of detecting *all* fraud. For few high-value transactions or an offline audit of, e.g. ERP transactions, the auditor will possibly prefer to inspect a feasible number of false alarms in favor of detecting large-scale fraud with large financial impact.

4.6 Knowledge Discovery in Databases and Data Mining

KDD is a standardized multistep process that systematizes the discovery of knowledge from data as characterized by Piatetsky-Shapiro [237]. This process defines methodological steps as illustrated in Fig. 4.3, reaching from data to the distilled knowledge, and can, in general, involve loop backs and several iterations [93].

In the following we will shortly introduce the different process steps of the KDD process and bring the contributions of this thesis in line with the process steps overall suggesting a practical implementation of AD for transaction data in the framework of the standardized KDD process. Mapping our task to the overall process, data

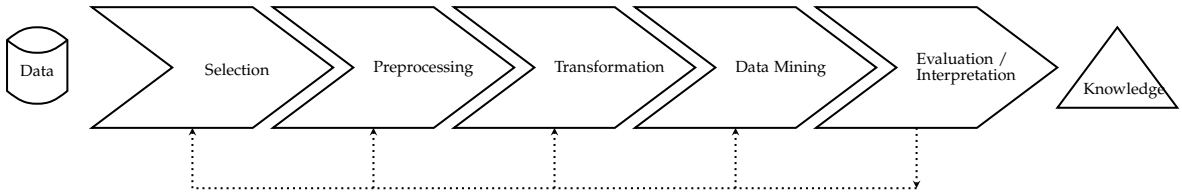


Figure 4.3: Overview over the KDD Process

refers to the transaction datasets from which useful knowledge in terms of previously unknown anomalies, such as fraudulent transactions, should be discovered.

After obtaining an understanding of the domain and data, the first step of the KDD process, *Selection*, refers to the selection of the dataset or subset from which knowledge should be discovered. For this thesis, we use several datasets that include labels to evaluate our methodological contributions. In terms of the KDD process, the knowledge which transactions are anomalous cannot be regarded as useful in terms of novelty with regard to the application of, e.g. fraud detection, however, allows the precise evaluation of the other KDD steps. Thus, to be precise, the potential application of our approaches to similar, unlabeled datasets has to be defined as Selection step, which is not part of this thesis. However, the datasets used in this thesis can be considered proxy datasets that precisely define the features and tables to be considered in the application for the selection process.

The second step, *Preprocessing*, can also involve data cleaning and handling of missing data. In this thesis, preprocessing decisions, for example, numerical scaling, are evaluated in several experiments in Chapter 6. Other data imputation and cleaning steps are dataset dependent and were not necessary for the datasets used in our experiments. *Transformation* in the KDD process denotes feature selection and projection for a task-dependent data view. While feature engineering in classical terms is generally not applied for neural networks, Chapter 7 evaluates with representation learning feature transformations. Hereby, data dimensionality reduction, emphasis of structural similarity, feature discretization (Section 7.2.1), and feature projections with PCA (Chapter 8) are evaluated.

Data Mining refers to specific methods for extracting knowledge from the data. This is the process step that this thesis primarily focuses on with our methodological contributions. Specifically, our research questions focus on the selection and improvement of the most beneficial models and approaches for anomaly detection in transaction data, which essentially outlines the choices to be made for the Data Mining process step. The execution of this step typically results in extracted patterns, predictions, or descriptions. The data, models, and results considered in the explorative evaluation in Chapter 6 and Chapter 7 can be considered as the output of the Data Mining step, as well as the prediction which transactions can be considered anomalous. Thus, the evaluation itself qualitatively and quantitatively can be subsumed in the last process step, *Evaluation / Interpretation* which generally involves

visualization of the patterns and models or data with respect to the models and their output. While the KDD process in practice finally makes use of the extracted knowledge in application, e.g. by using the knowledge directly or reporting findings, our evaluation primarily aims at answering our research questions. Thus, for a practical implementation of the KDD process, the focus has to be shifted from the selection, comparison, and improvement of approaches we focused on to a view on data and sample level, i.e. reporting fraudulent transactions in contrast to comparing models by their performance metric aggregated over dataset splits.

All in all, several results of this thesis can be mapped to respective steps in the KDD process model and thus simplify the choices that must be made in each process step for practical applications. To be precise, our evaluation regarding preprocessing and data transformation reveal promising parameters and choices for the respective KDD process steps and the ML methods and approaches we propose can be directly applied in the Data Mining step as data mining algorithms as referred to by Fayyad et al. [93].

Chapter 5

Datasets

In this chapter, the datasets used for our experiments are described. We first focus on our main datasets, studied in several experiments, and then introduce the benchmark datasets incorporated in individual experiments throughout this thesis.

5.1 Financial Fraud Datasets

One major domain where anomaly detection can be applied to transaction data is the field of financial fraud detection. In this setting, transactions logs can often be directly associated with financial transactions, i.e. the documentation of transfer of money directly or indirectly, e.g. in form of goods. Although such transaction logs are often mandatory by legal requirements and thus generated on a large scale, only few datasets are publicly available. This can be explained by several aspects: First, privacy issues might appear, since transaction data might contain sensitive information that can be used to reconstruct personal information. Second, such transaction logs can be related to business decisions and well-kept trade secrets for which the associated entities have no interest in making publicly available. Third, fraud cases are often not investigated on a per-data basis, thereby no labels are

Table 5.1: Main characteristics of the datasets.

Dataset	Features	Samples	Benign	Fraud	Fraud-ratio	Origin
PaySim	11	6 362 620	6 354 407	8 213	0.001	synth.
CCFraud	30	284 807	284 315	492	0.002	real (PCA)
IEEE-CIS	431	590 540	569 877	20 663	0.035	real
SAP $^{\text{train}}$	52	54 677	54 677	0	0	synth.
SAP $^{\text{eval}}$	52	19 714	19 696	18	0.0005	synth.
SAP $^{\text{test}}$	52	19 716	19 710	6	0.0003	synth.

obtained as a byproduct of auditing and controlling tasks, and companies in general have no financial incentive to collect such fine-grained data in a targeted manner. Nevertheless, some datasets have been collected and made publicly available.

The financial fraud datasets used in several experiments are summarized in Table 5.1. The four datasets vary in size, from the smallest dataset, SAP, with 94 107 transactions in all three splits, to PaySim with 6 362 620 samples. Although the fraud ratio of PaySim and CCFraud is of the same order of magnitude with 0.001 and 0.002, respectively, SAP contains a relatively smaller number of fraud cases with a ratio of 0.0005 for the eval and 0.0003 for the test split. IEEE-CIS contains an order of magnitude more fraud with a ratio of 0.035. The number of features varies between 11 (PaySim) and 431 (IEEE-CIS) features.

While the size and kind of features are different over the datasets, the anomalies of all datasets can be considered as point or context anomalies, i.e. the detection can be approached on per-transaction basis. Although PaySim and SAP are both synthetic datasets, their characteristics are very different: While PaySim is a simple dataset constructed by multi-agent simulation, the SAP dataset is constructed from data exported and aggregated from a genuine SAP system, which was used by real users participating in a business simulation game. Thereby the complexity of normal and non-normal behavior varies highly as detailed in the following sections.

Both real datasets have been anonymized to varying degrees: While CCFraud is obfuscated by PCA feature transformations, IEEE-CIS has been modified by disentangling user information and aggregating and obfuscating several features, consequently impeding the appropriate treatment of transaction context and features. Therefore, we focus on PaySim and SAP for our generative experiments and other experiments, where appropriate modeling of features and their dependencies is subject to research, as CCFraud and IEEE-CIS cannot be qualitatively evaluated in this context.

In the following sections, the datasets are explained in more detail.

5.1.1 PaySim

The PaySim dataset is a synthetic dataset, which was created by Lopez-Rojas et al. as part of their mobile money payment simulator, PaySim [190]. PaySim mimics real mobile money transactions as they are frequently used in African countries by applying a multi-agent based simulation. By the implementation of merchant, customer, and bank logic within this simulation, the resulting dataset can approximate a particular real dataset with regard to its statistical properties without introducing any privacy issues evolving from real data.

In this simulation, agents are associated with different roles implementing their respective behavior and they interact in this mobile money transaction simulation logic, ultimately composing the transaction dataset. The following roles are involved in this procedure:

Client is the main agent that initializes the transactions within the simulation. A client can deposit, withdraw, and transfer money and has different parameters associated with their specific customer profile. The behavior of each client is randomly assigned following statistical distributions of the real dataset, e.g., regarding their transaction type for a specific time, the transfer limits, and number of transactions per day. Clients can send money to other clients, resulting in a transaction of type *TRANSFER*.

Merchant are agents that the client can approach to buy something or request any other service from, which is then paid within the simulation. This payment is associated with the *PAYMENT* transaction type. In addition to payments, merchants offer the transaction types *CASH-IN* / *CASH-OUT* that deposit or withdraw physical money into or from the mobile money account of a client.

Bank is a specific role as a target for the *DEBIT* transaction. By this, money from the mobile money service is sent to an actual bank account.

Fraudster is associated with a malicious client that tries to withdraw money from accounts illicitly.

The PaySim dataset consists of 11 columns, specifying the hour the transaction is performed (*step*), the type of transaction recorded (*action*), the amount of money involved in the transaction (*amount*) and the origin (*nameOrig*) and destination (*nameDest*) of the transaction with their associated balances before (*oldBalanceOrig* / *oldBalanceDest*) and after (*newBalanceOrig* / *newBalanceDest*) the transaction. The dataset further includes a flag for unauthorized overdrafts (*isUnauthorizedOverdraft*) for which the transaction is stopped and a rule-based flag for transactions involving more than 200 000 of the local currency, mostly hinting at fraudulent behavior (*isFlaggedFraud*). In the PaySim dataset 6 362 620 transactions were recorded of which 6 354 407 (99.871%) are benign and 8 213 (0.129%) are labeled as fraud.

5.1.2 CCFraud

The credit card fraud (CCFraud) dataset [70] is a real-world dataset consisting of credit card transactions which were recorded in two days in September 2013. The dataset has been anonymized by applying Principal Component Analysis (PCA) to 28 of the 30 features. Two features, *time* and *amount*, have not been preprocessed through PCA and contain the number of seconds relatively to the start of the recording and the amount of money involved in the credit card transaction. For the other features which have been transformed by PCA, their original meaning remains unknown. The distribution of the label, denoting whether a transaction is considered fraudulent, is highly imbalanced: Next to 284 315 benign transactions, 492 are labeled as fraud, which means that 0.173% of the transactions are fraudulent.

5.1.3 IEEE-CIS

The IEEE-CIS dataset contains real-world e-commerce payment transactions which were provided by Vesta Corporation together with the IEEE Computational Intelligence Society as part of a challenge. The dataset is composed of two parts, one for training including labels and one unlabeled subset meant as a test dataset to be predicted as part of the competition. The features are spread over two tables, *identity*, consisting of features regarding the identity of the account holder, and *transaction*, consisting of features regarding the transaction. The meaning of most features remains undisclosed for privacy and contract reasons. The *identity* table contains information about the user's device (e.g. browser and operating system version) and network connection (e.g. IP address and Internet service provider). The *transaction* table contains relative time information from a given reference, the amount of money issued for payment, information about the product and payment method, address, as well as email domains, and several counting, matching, and timedelta features including manually engineered features which are not explained in detail. Both tables can be joint according to the transaction key and, thereby, considered as one flat dataset. The dataset consists of 432 features for 590 540 labeled samples, of which 569 877 are benign and 20 663 are labeled as fraud, leading to a fraud ratio of 3.5%.

5.1.4 SAP

The SAP dataset originates from an SAP R/3 ERP system and was extracted per table with the SAP SE16 transaction, allowing to reconstruct transactions of various levels of detail. Data within the SAP system were created by playing the business (informatics) training game ERPSim. In this game participants adopt the different roles within a manufacturing company including the purchasing of raw material, production planning, production, sales, and logistics to virtually run a production business. From simulated markets, customers and raw-material-vendors buy and sell goods depending on the prices and supply and demand, whereas the participants make business decisions on product variations, quantities, and prices. Examples are to decide which ingredients a product is composed of, which product is produced in which quantity, how much the product costs for different sales markets, and to which warehouse the stock products are transported.

The dataset focuses on the purchase-to-pay process (P2P) of this make-to-stock business process and is composed of two separate game runs, each representing one fiscal year. The first one consists of 54 677 benign transactions without explicit modeling of fraudulent behavior. The second run consists of 39 430 transactions with 19 696 benign and 18 fraud transactions for training and 19 710 benign 6 fraud transactions in the test set.

The fraud cases can be assigned to three different fraud schemes according to the Association of Certified Fraud Examiners' report to the nations (ACFE, 2020): Larceny, corporate injury, and invoice kickback. Larceny fraud can also be distinguished between legit manipulated orders (Larceny 1) and fraudulent purchase orders created manually (Larceny 2).

Larceny 1 is built on a legitimate purchase request, which is turned into a fraud by creating a purchase order with a manually increased quantity of the item. For receipt of receiving goods, the number is changed to the initial legitimate value to avoid a recording of the suspicious quantity. The physical leftovers could then be secretly taken away. The Larceny 1 fraud cases have to be considered partially unsuccessful, since a rule-based check in the ERP system stopped this suspicious process at invoice recording, the associated transactions however remain in the dataset and can be considered anomalous.

Larceny 2 on the other hand, involves a manual purchase order, which was regularly recorded and booked and fraudulently taken away. The larceny could be recognized during stocktaking at the end of the fiscal year and might be mapped to this irregular purchase order, possibly revealing the fraud scheme, however, since the process itself has no obvious flaws, the transactions and payment was not stopped by the ERP system and can thus be considered successful.

Corporate injury is a destructive fraud case possibly caused by a leaving employee who ordered such a large quantity of goods that the business would suffer significant financial damage. This was achieved by changing the quantity in several purchase orders.

Invoice kickback is a cooperative fraud scheme between a raw material supplier and a purchasing agent. For the purchase, the employee involved in the fraud buys goods such as raw materials for a deliberately increased price or increases the announced prices while creating a purchase order. The benefiting supplier then rewards the employee with a kickback.

The different fraud scenarios are divided between evaluation and test splits: Larceny 2 (4 cases) and corporate injury (14 cases) can be found in the evaluation set, while larceny 1 (2 cases) and invoice kickback fraud (4 cases) are only present in the test set. Thus, a fraud detection approach must be generalized to find previously unseen fraud cases.

5.2 Benchmark Datasets

In this section, we will shortly introduce the additional benchmark datasets we used in selected sections (see below) of this thesis.

5.2.1 Census Dataset

The Census dataset is a benchmark dataset from the UCI Machine Learning Repository [77] used in our generative modeling experiments in Section 6.2. It is a multivariate dataset containing 48 842 instances that were extracted from the 1994 Census database. The task typically associated with this dataset is to predict whether a person earns more than $50 000 per year.

The dataset contains six numerical features including, for example, age, capital-gain and loss or work hours per week, as well as 8 categorical features including workclass, education or native country.

5.2.2 Synthetic Function Learning Datasets

For our iNALU experiments in Section 6.1, we created three synthetic datasets with varying complexity to evaluate the ability to learn arithmetic tasks and thus numeric dependencies.

The **minimal arithmetic dataset** consists of two input features a, b, and one output feature y for which an operation $y = a \circ b$ with $\circ \in \{+, -\times, \div\}$ is performed. Therefore, a and b are drawn from a distribution $\mathcal{P}$ with $\mathcal{P} \in \{\mathcal{U}, \mathcal{N}, \mathcal{E}\}$ referring to uniform $(\mathcal{U})$, truncated-normal $(\mathcal{N})$, and exponential $(\mathcal{E})$ families. As the synthetic function learning datasets are designed to evaluate extrapolation performance, the training and test splits for each dataset are drawn from two different parametrizations of each probability distribution $\mathcal{P}$ indicated as $\mathcal{P}^{\text{int}}$ and $\mathcal{P}^{\text{ext}}$ for interpolation and extrapolation. Subdatasets with 64 000 samples per operation and parametrizations were created for the following parameters:

- $\mathcal{E}^{\text{int}}(0.2), \mathcal{E}^{\text{ext}}(0.5)$
- $\mathcal{E}^{\text{int}}(0.8), \mathcal{E}^{\text{ext}}(0.5)$
- $\mathcal{N}^{\text{int}}(-2, 4), \mathcal{N}^{\text{ext}}(8, 10)$
- $\mathcal{N}^{\text{int}}(-3, 3), \mathcal{N}^{\text{ext}}(8, 10)$
- $\mathcal{N}^{\text{int}}(-4, 2), \mathcal{N}^{\text{ext}}(8, 10)$
- $\mathcal{U}^{\text{int}}(-1, 1), \mathcal{U}^{\text{ext}}(-10, -5)$
- $\mathcal{U}^{\text{int}}(-5, 5), \mathcal{U}^{\text{ext}}(-10, -5)$

For the **input magnitude dataset**, a and b were drawn from a uniform random variable symmetrically around 0 from $\mathcal{U}(min, max) = \mathcal{U}(-10^{-2}, 10^{-2})$ to $\mathcal{U}(-10^4, 10^4)$. For each configuration, the extrapolation test set is drawn from $\mathcal{U}(max, 2 \cdot max)$.

The **simple arithmetic dataset** is constructed similarly to the minimal arithmetic dataset with the difference that in addition to a and b each sample consists of 8 additional variables sampled from the same distribution that do not contribute to y i.e., which have to be ignored by the model.

Finally, for the **simple function learning dataset**, a and b are not explicit features of each sample. Instead, each sample consists of 100 features, which are mutually exclusively summed to a, b or ignored for the calculation of y in a random but fixed mapping $\mathbf{m_a}, \mathbf{m_b} \in \{0,1\}^{100}$. The output y is then again calculated as

$$y = a \circ b = \mathbf{m_a}\mathbf{x} \circ \mathbf{m_b}\mathbf{x}, \quad \circ \in \{+, -\times, \div\}.$$

5.2.3 Windows Audit Log Dataset

In addition to the application domain of financial fraud detection, computer security is another domain where data considered as transaction logs occur frequently. Broadening the term *transactions*, other log data in computer systems can also be collected for anomaly detection. A typical source where various log data are accumulated is the operating system log, on UNIX based systems typically a heterogeneous collection of log files in `/var/log` for different purposes, originating from different services and applications. Logs generally contain data in various degrees of structuring and characteristics, from plain text only structured by newlines over deeply nested and linked entries to standardized key sets or headers in tabular structure. For Microsoft Windows operating systems, a large proportion of logs is unified in Windows Audit Logs. Berlin et al. [29] make a dataset available for detecting malware on Windows systems, which in this thesis is used for experiments on representation learning in Section 7.1 and unifies audit log events in nested JavaScript Object Notation (JSON) files. The dataset consists of 14 679 4-minute recordings of a system during the execution of malewares in CuckooBox and 11 123 recordings during normal system execution. The JSON files contain primarily categorical features such as process ID and name, and the associated audit log events (e.g. execution) with respective features such as filename, timestamp, event ID, action or target. The Windows Audit Log dataset is used in our representation learning experiments in Section 7.1.

5.2.4 Credit Payment

As a credit fraud benchmark dataset with a dedicated focus on numerical dependencies, for our experiments in Section 8.1 we created the synthetic credit payment dataset referred to as *Credit*, which reflects Peer-to-Peer credit fraud by incorrect interest calculation.

Each data point contains the following attributes: credit sum in month x (CS_x), interest rate (IR), payment rate (PR), credit sum in month $x+1$ (CS_{x+1}), label. The mathematical relationship between the attributes is defined as follows:

$$\mathrm{CS}_{x+1} = \mathrm{CS}_x + \frac{\mathrm{CS}_x \cdot \mathrm{IR}}{12} - \lambda \cdot \mathrm{PR} \tag{5.1}$$

With a probability of 99% the credit sum is correctly calculated ($\lambda = 1$) according to Eq. 5.1, but with a probability of 1% we simulate a fraudulent calculation of the remaining credit only by reducing the new credit sum by 95% of the paid rate ($\lambda = 0.95$). Each instance contains the columns $\mathrm{CS}_{x+1}, \mathrm{CS}_x, \mathrm{IR}, \mathrm{PR}$ and the label *isFraud*. The features are drawn randomly from a uniform probability distribution ($\mathrm{CS}_x \sim \mathcal{U}(0, 10\,000)$, $\mathrm{IR}_x \sim \mathcal{U}(0, 0.5)$, $\mathrm{PR}_x \sim \mathcal{U}(0, 5\,000)$) with the constraint that the credit is not overpaid, i.e. $\mathrm{CS}_{x+1} \geq 0$. In contrast to PaySim, fraudulent and benign transactions are modeled after the same probability distributions (or user profiles), which means, the machine learning model has to capture the mathematical relationship to predict correctly if a transaction is fraudulent. The dataset consists of 100 000 instances having 1033 fraudulent and 98 967 benign transactions.

Chapter 6

Modeling of Distributions and Dependencies for Transaction Data

Neural networks have achieved great success in various areas of machine learning applications. Different network structures have proven to be suitable for different tasks. For example, convolutional neural networks are well suited for image processing [176, 292, 169], while recurrent neural networks are well suited for handling sequential data [139, 119, 60]. For transaction data, neural networks face challenges such as processing categorical and numerical values, modeling distributions of various kinds, and recognizing specific mathematical relations.

In this section, we will therefore evaluate neural approaches to model distributions and dependencies present in various datasets to learn the characteristics of transaction data. This, on the one hand, allows the generation of realistic synthetic data. On the other hand, the precise model of normal transactions can be leveraged for anomaly detection, as evaluated in Chapter 8.

In the following sections, we first focus on numerical dependencies and propose an improved neural architecture to model arithmetic and numeric dependencies; second we adapt two generative models, propose an evaluation protocol, and evaluate the ability of both approaches to model distributions and dependencies from a quantitative and qualitative perspective.

6.1 iNALU: Modeling and Learning Numeric Dependencies

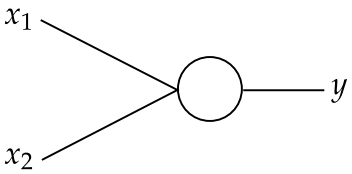
The presence of mathematical relationships between features is a well-known fact in many financial tasks [32, 190]. Other examples can be found in the intrusion detection domain. For example, some intrusion detection methods count the number of certain events [104] or consider some restrictions, such as network packets having a minimum and maximum number of transmitted bytes [254]. A model which is able

to capture these relationships explicitly in an automated way is therefore very desirable and can be incorporated in various machine learning tasks including anomaly detection in transaction data.

Problem Although neural networks are successfully applied in complex machine learning tasks, single neurons often face difficulties in calculating basic mathematical operations [317]. This fact can be explained by inspecting the structure of neurons in detail. Simplifying Eq. 4.2 of a neural network *layer* introduced in Section 4.2, the equation for a *single* neuron is given in Eq. 6.1: The output of a neuron i is the weighted sum of all input signals, an optional bias b, and an activation function

$$output_i = act\left(\left(\sum_{j=1}^{n} x_j \cdot w_j\right) + b_i\right). \tag{6.1}$$

The neuron i in Equation 6.1 receives n input signals x_j which are multiplied by the weights w_j. The parameter b_i represents an optional bias and $act(\cdot)$ is an arbitrary activation function like the identity for a linear or sigmoid for a non-linear neuron. This allows neurons to assign different weights to different input features. Furthermore, linear neurons are able to add (or subtract) different inputs by setting their corresponding weights to 1 (or -1), see tasks a) and b) in Figure 6.1. However, activation functions, weights, and bias allow neurons only to approximate the result of multiplications and divisions in their training range, since the output is the weighted sum of all inputs. Consequently, they cannot solve multiplication and division tasks for values outside the training range (see tasks c) and d) in Figure 6.1).



Tasks: a) $y = x_1 + x_2$
b) $y = x_1 - x_2$
c) $y = x_1 \times x_2$
d) $y = x_1 \div x_2$

Figure 6.1: Standard mathematical tasks.

Trask et al. [317] show empirically that artificial neurons especially have difficulties with extrapolation of mathematical operations and present the *Neural Arithmetic Logic Unit* (NALU) to address this problem. However, the NALU is only able to calculate non-negative results for multiplication and division by design. Madsen and Johansen [198] further show that the NALU is unable to learn division reliably and often fails to converge to the desired weights.

Objective Inspired by the NALU, we want to improve the architecture to address the limitations mentioned above. Our focus lies on processing negative values and improving extrapolation by forcing the internal weights to discrete values.

Contribution In this section, we propose iNALU as an improvement to the NALU architecture [317]. Our proposed architecture improves stability, enables the network to calculate with negative and positive inputs, and improves the precision of arithmetic tasks in general. Therefore, we change several technical aspects of the original NALU. To be precise, we add another path to allow multiplication and division with mixed-signed inputs. Furthermore, we propose an input-independent implementation of the gate, switching between the summative and multiplicative paths. Based on empirical observations, we add regularization to the training procedure to prevent approximation of the results due to unwanted combinations of mathematical operations. Additionally, a maximum function for the multiplicative path is introduced to avoid too large values (infinity) for deep networks of larger scale. We experimentally evaluate the improved architecture in various settings: Minimal arithmetic tasks, one-layer calculations, where among others the relevant inputs have to be recognized, and simple function learning where a combination between operations has to be learned in two layers.

Parts of this section have been published as *Schlör, D., Ring, M., and Hotho, A. (2020a). iNALU: Improved neural arithmetic logic unit.* Frontiers in Artificial Intelligence, *3:71* [276]. The iNALU code is available on github[1].

6.1.1 Improved Neural Arithmetic Logic Unit

In this section, we first describe the Neural Arithmetic Logic Unit and discuss properties and challenges. We then introduce iNALU, a new model variant, to address these limitations.

Neural Arithmetic Logic Unit

The NALU as proposed by Trask et al. [317] consists of a multiplicative and a summative path, which can be seen as a linear layer with a weight matrix constrained to $[-1, 1]$. Weights $\mathbf{W}$ are constructed as point-wise product between a matrix $\hat{\mathbf{W}}$ with activations tanh and a matrix $\hat{\mathbf{M}}$ with sigmoid (σ) activations.

$$\mathbf{W} = \tanh(\hat{\mathbf{W}}) \odot \sigma(\hat{\mathbf{M}}) \tag{6.2}$$

By matrix multiplication of inputs $\mathbf{x}$ and weights $\mathbf{W}$, the output values stay within the magnitude of the input values (since $-1 \leq \mathbf{W}_{i,j} \leq 1$) and result in the summation for values of $\mathbf{W}_{i,j} = 1$ and subtraction for values of $\mathbf{W}_{i,j} = -1$. By balancing the weights between $\{-1, 0, 1\}$ any function composed of adding, subtracting, and ignoring inputs can be learned. This summative path $\mathbf{a}$ is defined in Equation 6.3.

$$\mathbf{a} = \mathbf{x}\mathbf{W} \tag{6.3}$$

[1] https://github.com/daschloer/inalu

To multiply or divide, this calculation is performed in log-space (see Equation 6.4). The NALU encounters the problem of calculating $\log(x)$ for $x \leq 0$ by restricting the calculation to absolute input values and adding a small constant value ϵ.

$$\mathbf{m} = \exp\left(\log(|\mathbf{x}| + \epsilon)\mathbf{W}\right) \tag{6.4}$$

A gate is used to decide between the summative and the multiplicative path depending on the input vector.

$$g = \sigma(\mathbf{x}\mathbf{G}) \tag{6.5}$$

Since the gate weights $\mathbf{G}$ are multiplied by the inputs $\mathbf{x}$, each gate dimension maps to an input dimension and contains the corresponding weight to which the input should contribute to the decision between both arithmetic paths.

The output is obtained by adding the gated summative (see Equation 6.3) and multiplicative (see Equation 6.4) paths.

$$\texttt{NALU}_o\text{: } \mathbf{y}_{\text{nalu}} = g \cdot \mathbf{a} + (1 - g) \cdot \mathbf{m} \tag{6.6}$$

The NALU model can be finally implemented in two ways. One can use either a weight vector $\mathbf{G}$ and a scalar gate g, or a weight matrix $\mathbf{G}$ and a gate vector $\mathbf{g}$. Tasks for which the selection of the operation is different for each output or for which it depends on input values might benefit from the gate matrix. However, this introduces additional parameters that, for many tasks, are unnecessary. In our experiments, we used both vector-based NALU and a NALU with matrix-based gating for comparison, which we refer to as NALU (v) and NALU (m).

6.1.2 Limitations

Some of the design decisions for the NALU, however, result in limitations that we want to address in the following section.

Exploding Intermediate Results

In our experiments, we observe that training often fails due to exploding intermediate results, especially when stacking NALUs to deeper networks and having many input and output variables. For example, consider a model consisting of four NALU layers with four input and output neurons each and a simple summation task. Assuming the same magnitude for all input dimensions, the first layer could (depending on initialization) calculate x^4 for each output dimension, while the subsequent layer could calculate $(x^4)^4$, eventually leading to x^{4^l} for layer l. Therefore, the calculation can exceed the valid numeric range already in the forward pass, which ultimately causes the training to fail. As another example, in a network with three NALU layers in an MNIST classification downstream task, the NALU models failed after the first training steps (resulting in NaNs).

Multiplication / Division with Negative Result

The NALU by design is not capable of multiplying or dividing values with a negative result. In the multiplicative path, the input values are represented by their absolute value to guarantee a real-valued calculation in log space. Therefore, learning multiplication for mixed signed data with a result $y < 0$ fails. Since the NALU is expected to learn either multiplication / division or summation / subtraction in each layer, $\text{sign}(y)$ is clearly determined in the multiplicative case by the number of negative multiplicands being even or odd. Since input dimensions can be deactivated for $\mathbf{W}_{i,j} = 0$, the sign cannot be inferred counting negative input variables.

Mixed Sign Gating

Despite the fact that the summative path is capable of dealing with mixed input signs, the construction of the gating mechanism leads to problems. If the input values are constantly positive or constantly negative, Equation 6.5 leads to the desired gating behavior. However, if the input values mix negative and positive values, σ and thus the gate is dependent of the sign since $\mathbf{G}$ cannot fit the designated gate state systematically correctly.

Initialization Sensitivity

We observed that the NALU architecture is very prone to non-optimal initializations, which can lead to vanishing gradients or optimization into undesired local optima. In general, finding the optimal initialization is difficult, since it depends on the operation and the input distribution, but in a real-world scenario, both are unknown.

Leaky Gates

Another challenge we observe are variables not tied near their boundaries. Generally, in the NALU design, the variables $\mathbf{W}$ and g are intended to reach their boundaries of $[-1, 1]$ and $[0, 1]$ for maximum precision. However, during training and for interpolation, an approximation of the intended calculation, for example having gates trained to $g = 0.5$ with a specific configuration of $\mathbf{W}$, might represent a local optimum. For extrapolation, such a model fails by large margin. We suggest regularizing the trained variables to avoid this behavior.

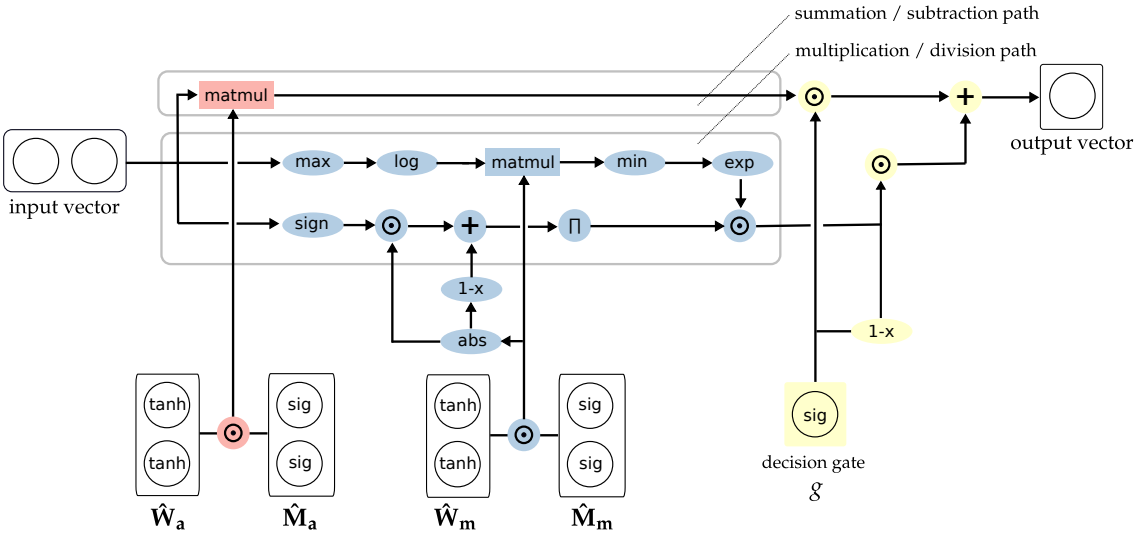


Figure 6.2: Architecture of the improved Neural Arithmetic Logic Unit (iNALU).

6.1.3 Technical Adaptations

This section describes the improvements we incorporate into our iNALU model to address the aforementioned challenges. Figure 6.2 summarizes the complete model architecture. In the following, we discuss each improvement and extension in detail.

Independent Weights

The summative and multiplicative paths share their weights $\hat{\mathbf{W}}$ and $\hat{\mathbf{M}}$ in the original NALU model. We propose using separate weights for each path for two reasons: First, the model can optimize $\mathbf{W}$ for the multiplicative and summative path without interfering with the other path. For example, given a setting with inputs $a, b < -1$ and the operation $a \times b$, the result would be a positive number greater than 1 and the optimal parameter setting would be $\mathbf{W}_a = \mathbf{W}_b = 1$ and $g = 0$. However, the only way for the summative path (see Equation 6.3) to generate positive results is to force the weights $\mathbf{W}_a$ and $\mathbf{W}_b$ towards -1. In this case, the summative and multiplicative paths force the weights into opposite directions. Using separate weights, the model can learn optimal weights for both paths and select the correct path using the gate. Second, consider that the multiplicative path yields large results, whereas the summative path represents the correct solution but yields relatively small results. In that case, the multiplicative path influences the results even if the sigmoid gate is almost closed. For example, in a setting with inputs $a, b, c > 0$ with the desired result $a + b$, the summative path yields the correct solution, and the optimal weight setting is $\mathbf{W}_a = \mathbf{W}_b = 1$, $\mathbf{W}_c = 0$, and $g = 1$. In that case, $\mathbf{W}$ may contain very small weights to omit the input c. However, small negative weights for $\mathbf{W}_c$ (e.g. $-1e-5$) lead to the situation that the multiplicative path divides the inputs a and b by values close

to 0, which results in large numbers. Consequently, the multiplicative path influences the results even if the gate (see Equation 6.5) is almost closed. In this case, the model with independent weights can optimize $\mathbf{W_m}$ to smaller values to mitigate the influence caused by the leaky gate. Our modifications are depicted in Fig. 6.2 and summarized in the following equations.

$$\mathbf{W_a} = \tanh(\hat{\mathbf{W}}_\mathbf{a}) \odot \sigma(\hat{\mathbf{M}}_\mathbf{a}) \tag{6.7}$$
$$\mathbf{W_m} = \tanh(\hat{\mathbf{W}}_\mathbf{m}) \odot \sigma(\hat{\mathbf{M}}_\mathbf{m}) \tag{6.8}$$
$$\mathbf{a} = \mathbf{x}\mathbf{W_a} \tag{6.9}$$
$$\mathbf{m} = \exp\left(\log(|\mathbf{x}| + \epsilon)\mathbf{W_m}\right) \tag{6.10}$$

Weight and Gradient Clipping

To address the challenge of exploding intermediate results in a multilayer setting, we improve the model by clipping the exploding weights in the back-transformation from log-space (see Equation 6.11). Further, we avoid imprecise calculations by incorporating ϵ and ω only if $\mathbf{x}$ values caused exploding intermediate results.

$$\mathbf{m} = \exp\left(\min\left(\log\left(\max(|\mathbf{x}|, \epsilon)\right)\mathbf{W}_\mathrm{m}, \omega\right)\right) \tag{6.11}$$

This kind of weight clipping is a simple practical solution to improve the stability of deep iNALU networks, which has, for example, been successfully applied in Wasserstein Generative Adversarial Networks [17]. The original NALU architecture did not address this problem causing practical stability issues. To validate this, we incorporated three NALU layers in a MNIST classification downstream task. Our proposed clipping mechanism resulted in a successful training that solved the task very well with an accuracy of 0.94 after 64000 steps, whereas the original NALU fails producing NaNs.

This shows the effectiveness of our proposed improvement, albeit more sophisticated solutions might be an interesting topic of future work to avoid vanishing gradients for clipped neurons. Further, we apply gradient clipping to avoid stability problems due to large gradients, which can occur when input values are close to zero. We set ϵ to 10^{-7} and ω to 20.

Sign Correction

The NALU cell by design is not capable of multiplying or dividing values with a negative result. Therefore, NALU fails in calculating the multiplication of mixed signed data. Considering the sign within the log space transformation is not trivial since $\log(x)$ is not defined for $x < 0$ in $\mathbb{R}$. Instead, inferring the correct sign

post-hoc is a more feasible solution, which follows the human intuition of multiplying or dividing numbers with mixed signs. However, multiplying over the sign $(\mathbf{x})$ vector does not provide a universal solution, since some input dimensions may be deactivated ($\mathbf{W}_{i,j} = 0$).

We propose a solution by taking into account the sign of only the relevant input values (that is, all $\mathbf{W}_{i,j} \neq 0$).

$$\mathbf{msm_1} = \text{sign}(\mathbf{x}) \odot |\mathbf{W_m}| \tag{6.12}$$

$$\mathbf{msm_2} = 1 - |\mathbf{W_m}| \tag{6.13}$$

$$\mathbf{msm} = \mathbf{msm_1} + \mathbf{msm_2} \tag{6.14}$$

$$\mathbf{msv} = \prod_i \mathbf{msm}_{ij} \tag{6.15}$$

$$\texttt{iNALU}_s\text{: } \mathbf{y}_{\text{nalu}} = g \cdot \mathbf{a} + (1-g) \cdot \mathbf{m} \odot \mathbf{msv} \tag{6.16}$$

The sign correction is independent of the operation in the multiplicative path and has to be applied for multiplication and division. Therefore, we use the absolute value of the weight matrix $\mathbf{W_m}$ to identify relevant and irrelevant input values. First, the sign function is applied to the input vector $\mathbf{x}$, which is then multiplied element-wise with the absolute value of the weight matrix $\mathbf{W_m}$, leading to +1 for positive relevant inputs, -1 for negative relevant inputs and 0 for irrelevant inputs (see Equation 6.12). The multiplication of all row elements (input dimensions) per column of $\mathbf{msm_1}$ leads to 0, if any input dimension is irrelevant ($\mathbf{W}_{i,j} = 0$). To avoid this, we represent all irrelevant inputs as +1, since +1 does not influence the result of a multiplication. We achieve this by introducing a second matrix $\mathbf{msm_2}$ (see Equation 6.13) which is +1 for irrelevant inputs and 0 for relevant inputs and add $\mathbf{msm_1}$ and $\mathbf{msm_2}$. Finally, we infer the sign vector containing the sign of the multiplicative path for each output dimension (see Equation 6.15) by multiplying over each column of $\mathbf{msm}$. This sign represents the correct solution if $\mathbf{W}$ is discrete, i.e. $\mathbf{W}_{i,j} \in \{-1, 0, 1\}$. Discrete weights are a desired property [317] to achieve generalization and interpretability and ensure that $\mathbf{msm}$ is also discrete, i.e. $\mathbf{msm}_{i,j} \in \{-1, 1\}$. By introducing regularization (see Section 6.1.3), we force the model to find discrete weights $\mathbf{W}$. Implementing both improvements enables the model to correctly calculate inputs with mixed signs for the multiplicative path.

Regularization

In general, having discrete values for $\mathbf{W}$ and g is often crucial for a model to generalize and learn a calculation correctly rather than approximating the solution. This becomes even more important for the sign-corrected multiplication. We therefore propose regularizing the weights such that $\hat{\mathbf{W}}$, $\hat{\mathbf{M}}$, and $\mathbf{G}$ do not contain values near zero by introducing a piecewise linear regularization term (see Equation 6.17) which adds to the loss until the weight has reached a discretization threshold t. We found $t = 20$ suitable since $\sigma(-20) < 10^{-9}$ and $1 - \tanh(20) < 10^{-17}$.

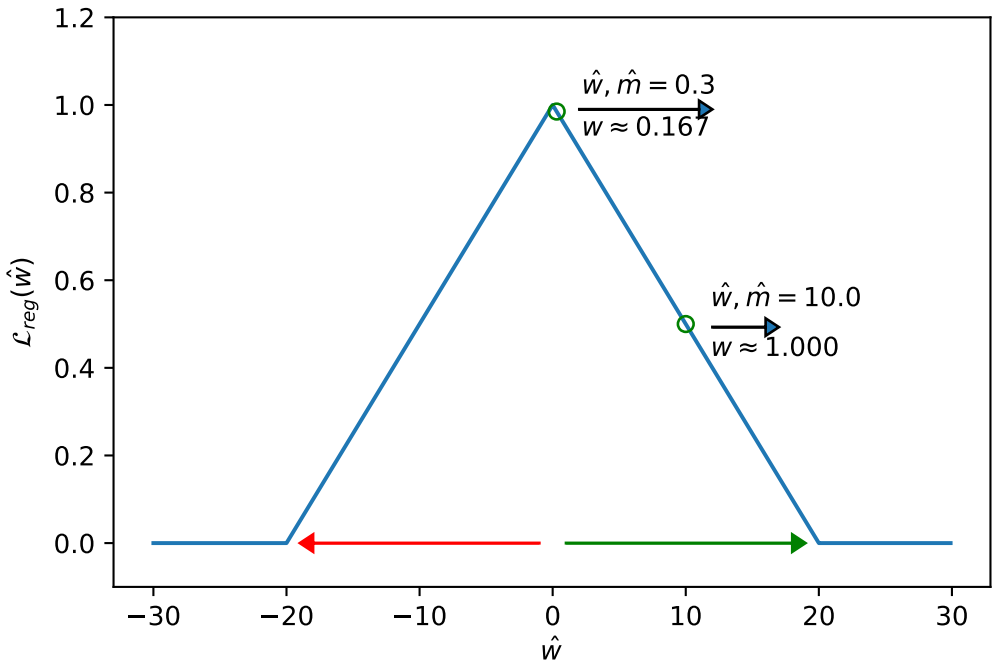


Figure 6.3: Regularization approach: Weights $0 < \hat{w}, \hat{m} < 20$ are pushed towards $t = 20$ by $\mathcal{L}_{\text{reg}}$, while weights $-20 < \hat{w}, \hat{m} < 0$ are pushed towards -20 causing discrete values $\{-1, 0, 1\}$ for $w = \tanh(\hat{w}) \cdot \sigma(\hat{m})$.

$$\mathcal{L}_{\text{reg}}(w) = \frac{1}{t} \max(\min(-w, w) + t, 0) \tag{6.17}$$

For example, for weights $-t < \hat{w}, \hat{m} < t$, e.g. $\hat{w} = \hat{m} = 0.3$, applying Equation 6.2 results in $w = \tanh(\hat{w}) \cdot \sigma(\hat{m}) \approx 0.167$. This means that the corresponding input x is scaled down for the calculation of the additive or multiplicative path as, for example, $a = x \cdot w = 0.167x < x$ in this one-dimensional example. For a higher dimensional case, matrix multiplication (e.g. $\mathbf{a} = \mathbf{x}\mathbf{W}_\mathbf{a}$ for the additive path) sums over scaled inputs instead of a summation ($w = 1$) or subtraction ($w = -1$) of the relevant inputs ($\sigma(\hat{m}) = 1$) with discrete weights w. Although calculating with scaled inputs might result in suitable approximations of the underlying training data, these solutions usually fail to generalize the function, and thus to extrapolate. By incorporating regularization loss, the model has a small gradient forcing the weights $\hat{w}$ and $\hat{m}$ towards $-t$ and t, respectively. Therefore, the model is penalized for (local) approximations with values $-t < \hat{w}, \hat{m} < t$ pushing $\tanh(\hat{w})$ towards 1 or -1 and $\sigma(\hat{m})$ towards 0 or 1 as illustrated in Fig. 6.3.

Note that regularization can cause gradient directions contradicting the gradient direction of the loss without regularization depending on the initialization. We try

to mitigate this problem by incorporating regularization only after several training steps, when the loss is below a threshold (see Section 6.1.4 for more details).

In addition, regularization is especially useful to improve extrapolation performance. For example, we evaluate regularization in the Simple Function Learning Task (see Section 6.1.9) setup for a summation task (i.e., an overdetermined task where an optimal and generalizing solution can be found even for $-1 < \mathbf{W}_{i,j} < 1$). After 10 epochs without regularization, we observed an interpolation loss of $5.95 \cdot 10^{-4}$ and an extrapolation loss of $4.46 \cdot 10^{11}$. The model has found a suitable approximation for the training range but failed to generalize. Introducing regularization after the 10th epoch and evaluating after 15 epochs we reach an interpolation loss of $2.2 \cdot 10^{-13}$ and an extrapolation loss of $2.2 \cdot 10^{-11}$, whereas without regularization we just improve the interpolation loss ($8.30 \cdot 10^{-5}$) and the extrapolation loss even impairs ($8.76 \cdot 10^{14}$).

Reinitialization

Since NALU does not recover well from local optima on its own [198], we suggest a reinitialization strategy. This strategy evaluates the loss for each m-th epoch and randomly reinitializes all weights if the loss did not improve for the last n steps and if the loss is greater than a predefined threshold (see Section 6.1.4).

Independent Gating

In the original NALU model the gate that decides between the multiplicative and the summative path is calculated by multiplying the input vector and the gate weight matrix $\mathbf{G}$ (see Equation 6.5). While this can be beneficial if the decision between operations is encoded in an op-code-like fashion, in many tasks, the decision which operation path to choose is not depending on the input values but instead fixed for the task, e.g. typical spreadsheet tasks like calculating the sum or product of different columns.

For this case, we propose a model, where the scalar gate is replaced by a vector that is, in contrast to the original NALU model, independent from the input (see Equation 6.18). Thereby, the gate weights are indirectly optimized through backpropagation during training of the network to represent the best-fitting operation, reminiscent of training bias in a linear layer.

$$\mathbf{g} = \sigma(\mathbf{G}) \tag{6.18}$$

For example, consider a NALU network with one layer, the operation $+$ and the inputs $\mathbf{x_1} = (2, 2)$ and $\mathbf{x_2} = -\mathbf{x_1} = (-2, -2)$ resulting in the calculations $2 + 2 = 4$ and $-2 + (-2) = -4$. For the original NALU the function $y = \sigma(xG) \cdot a + (1 - \sigma(xG)) \cdot m$ has to be optimized, i.e. a G has to be found which holds $\sigma(xG) \to 1$ for all x to choose the correct operation. Both inputs $\mathbf{x_1}$ and $\mathbf{x_2}$ must be calculated with

the same operation $+$, therefore, for a suitable G, $\sigma(\mathbf{x_1}G) = \sigma(\mathbf{x_2}G) \Leftrightarrow \sigma(\mathbf{x_1}G) = \sigma(-\mathbf{x_1}G)$ must hold. Since $G = 0$ is the only solution leading to $\sigma(xG) = \sigma(0) = 0.5$, there is no valid solution satisfying both constraints. With independent gating, the iNALU can optimize $\sigma(G) \to 1$ up to arbitrary precision and, therefore, learn the function correctly.

Furthermore, choosing a vector over a scalar enables our model to select the operation for each output independently, introducing the ability to calculate, for example, $y_1 = x_1 + x_2$, $y_2 = x_1 \cdot x_2$ for an input $\mathbf{x} = (x_1, x_2)$ simultaneously.

6.1.4 Design of Experiments

In this section, we perform an experimental evaluation of the proposed iNALU model to analyze its basic abilities to solve mathematical tasks compared to the original NALU. To be precise, we compare two NALU models, NALU (v) with a gate vector $\mathbf{G}$, NALU (m) with gate matrix $\mathbf{G}$ with two iNALU models, iNALU (sw) with shared weights between the additive and multiplicative path, and iNALU (iw) with independent weights for each path as proposed in our *Independent Weights* adaptation (cf. Section 6.1.3). Some technical adaptations such as the sign correction or independent gating explicitly remedy deficient design choices of the original NALU preventing correct calculation. Therefore, we refrain from reporting their individual contribution as their expectable outcomes are confirmed such as erroneous calculation of mixed signed multiplications, incoherent calculations for input dependent gating, or increased imprecision without regularization enforcing discrete, non-leaking weights. In contrast, the benefit of independent weights between multiplicative and additive paths (experiments 1-3 and 5) as well as (re)initialization (experiment 4) are evaluated, as the outcome of these experiments offers interesting insights into constructing and applying our architecture in different tasks of varying complexity examining the following research questions:

Experiment 1 examines the research question, how well each model performs in its minimal setup for different input distributions, i.e. one layer with two input and one output neurons. We show that the iNALU outperforms the NALU and reaches very low error rates for almost all distributions.

In experiment 2 we evaluate how well the models perform on different magnitudes of input data. The results show that the iNALU models can reach high precision for data of different magnitude, although the precision for multiplication impairs with increasing magnitude of input data.

Experiment 3 examines the ability of each model to ignore input dimensions. We show that the iNALU is capable of learning to ignore input dimensions well, whereas the original NALU fails for most operations and distributions.

With experiment 4 we compare different initialization strategies. The parameter study shows that initialization has a large impact on the stability of the network. We finally identify the most suitable parameter configuration for more complex tasks.

Finally, experiment 5 examines the performance of NALU and iNALU models for a function learning task involving two arithmetic operations per function using architectures with two layers and 100 input dimensions. We show that the iNALU models outperform both NALU models by a large margin and yield a very high precision for all operations except division.

Prerequisites

This section first describes the general commonalities of all experiments.

Datasets For all experiments, we evaluate on an interpolation task as well as an extrapolation task. For the interpolation task, the training and evaluation datasets are drawn from the same distribution. For the extrapolation task, the evaluation dataset is drawn from a distribution with a different value range to evaluate the ability to generalize. Each dataset contains $N = 64\,000$ samples.

Tasks For our experiments, we focus on mathematical operations since these are the building blocks of more complex tasks. All tasks involve calculating an arithmetic operation $\diamond \in \{+, -, \times, \div\}$ on input and/or hidden variables a and b by $y = a \diamond b$. Note that Trask et al. introduce additional operations such as identity, square, and the square-root but since these operation are special cases of the basic operations, their learning performance is closely correlated with the performance on the basic operations and therefore omitted for the sake of clarity. The input variables for all experiments are sampled randomly from a distribution $\mathcal{P}$ with a parameterization λ, which are defined in the following sections in more detail. Note that for $\mathcal{P} = \mathcal{N}$ the normal distribution for our experiments is truncated to $\lambda = [a, b] = [\mu - 3\sigma, \mu + 3\sigma]$ (containing $\approx 99,7\%$ probability mass) to ensure that the extrapolation task is performed outside of the test distribution range. For the exponential distribution ($\mathcal{P} = \mathcal{E}$) the extrapolation task involves no extrapolation in a literal sense but examines if generalization can be achieved for different values of λ.

Evaluation In contrast to the experiments by Trask et al., we choose a different evaluation strategy: They reported the error for each operation relatively in comparison to a randomly initialized network prior to training. Since the performance of the untrained network is constantly bad, the relative performance reported can be used to decide how well each architecture performs rank-wise, but it cannot be used to infer, to which extent the calculated result differs from the expected result. Instead, we use a more intuitive approach for evaluation and report the mean squared error (MSE) between the calculated and expected results on the complete evaluation

datasets. For all experiments, we report results for extrapolation, since this is the more difficult task.

$$\mathrm{MSE}(y^{\mathrm{pred}}, y^{\mathrm{real}}) := \frac{1}{N}\sum_{i}^{N}(y_i^{\mathrm{pred}} - y_i^{\mathrm{real}})^2 \tag{6.19}$$

The MSE comes with another advantage. Combined with a predefined threshold, the MSE can be used to assess if the model reaches the necessary precision [198]. If not stated otherwise, we understand an MSE $\leq 10^{-4}$ as successful training. We repeat each experiment ten times with different random seeds. This procedure examines whether the performance is stable or how much it scatters.

Training We use the Adam optimizer [165] in mini-batch training with a learning rate of 0.001 and a batch size of 64. Training is carried out for 100 epochs using the MSE as loss. Clipping, regularization, and random reinitialization as described in Section 6.1.3 are implemented. Regularization is activated after 10 epochs if the training loss $\mathcal{L} < 1$. Reinitialization is applied each 10th epoch if the loss has not improved over $m = 10\,000$ steps. This means that during training, reinitialization can occur up to nine times. Note that this method could lead to incompletely trained models if a reinitialization occurs late during training in favor of a fair model comparison.

6.1.5 Experiment 1 - Minimal Arithmetic Task

Experiment 1 constructs the most minimalistic task where the model has two inputs and one output and analyzes the influence of the input value distribution by sampling a and b from uniform, truncated normal, and exponentially distributed random variables in various ranges.

Results The extrapolation results of this experiment are presented in Figure 6.4.

In general, our iNALU models perform substantially better in all operations. With the exception of exponentially distributed data for $\lambda = 0.2$, for summation all and for subtraction almost all models learned the task successfully. For multiplication, iNALU with independent weights performs best with very good precision, with the exception of $\mathcal{E}(0.2)$ and $\mathcal{N}(-4, 2)$. All models yield worse results for division. In fact, for the original NALU, no tested input parameter configuration leads to acceptable MSEs (the average MSE is $4.36 \cdot 10^4$). Our models also yield mixed results, some solving the task nearly perfect after one to six reinitializations, but others failing after nine reinitializations as well.

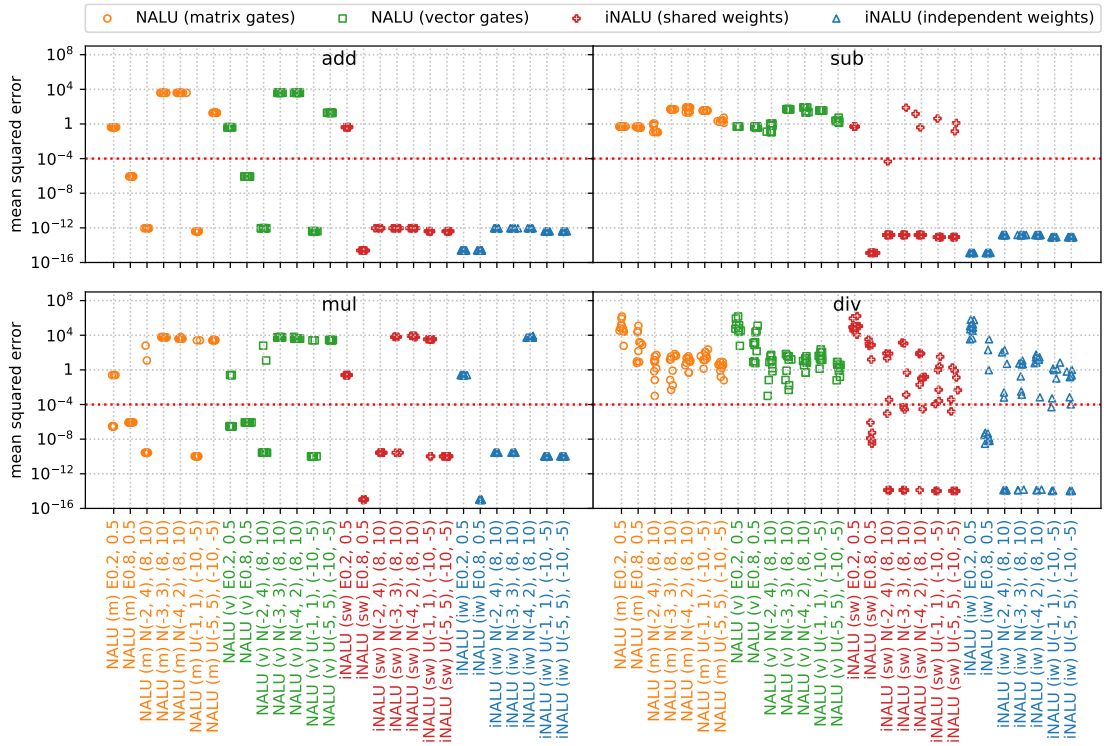


Figure 6.4: MSE for various input distributions per operation over the extrapolation test dataset of experiment 1 (minimal arithmetic task). The original NALU is colored orange and green, (m) stands for the matrix gating, and (v) for the vector gating version. Our iNALU models are depicted in red for the shared weights variant and blue for the version with independent weight matrices for the summative and multiplicative path. For truncated normal (N) as for uniform distributed data (U), the first parameter tuple represents the training data range, the second tuple represents the extrapolation range. For exponentially distributed data (E) the parameter λ is reported.

6.1.6 Experiment 2 - Input Magnitude

In this experiment, we generate data of different magnitude for the minimal arithmetic task of experiment 1 to examine the influence of the data magnitude on the model precision. We sample a and b from a uniform random variable symmetrically around 0 from $(min, max) = (-10^{-2}, 10^{-2})$ to $(-10^4, 10^4)$. For each configuration, we extrapolate to $(max, 2 \cdot max)$.

Results The results of this experiment are shown in Figure 6.5. For input data of a magnitude larger than 1, the NALU models fail to capture the underlying function precisely for all operations. In contrast, the iNALU models calculate precisely for all magnitudes for summation and subtraction. For multiplication, the influence of the input data magnitude is larger, which was to be expected since the magnitude of the results for $a = 10^x, b = 10^y, a \times b$ is 10^{x+y} and so is the magnitude of the error, which is squared in addition, as we report the mean square error. For division, independently of the data magnitude, some iNALU models capture the underlying operation very precisely, others fail. All NALU models fail to calculate division precisely.

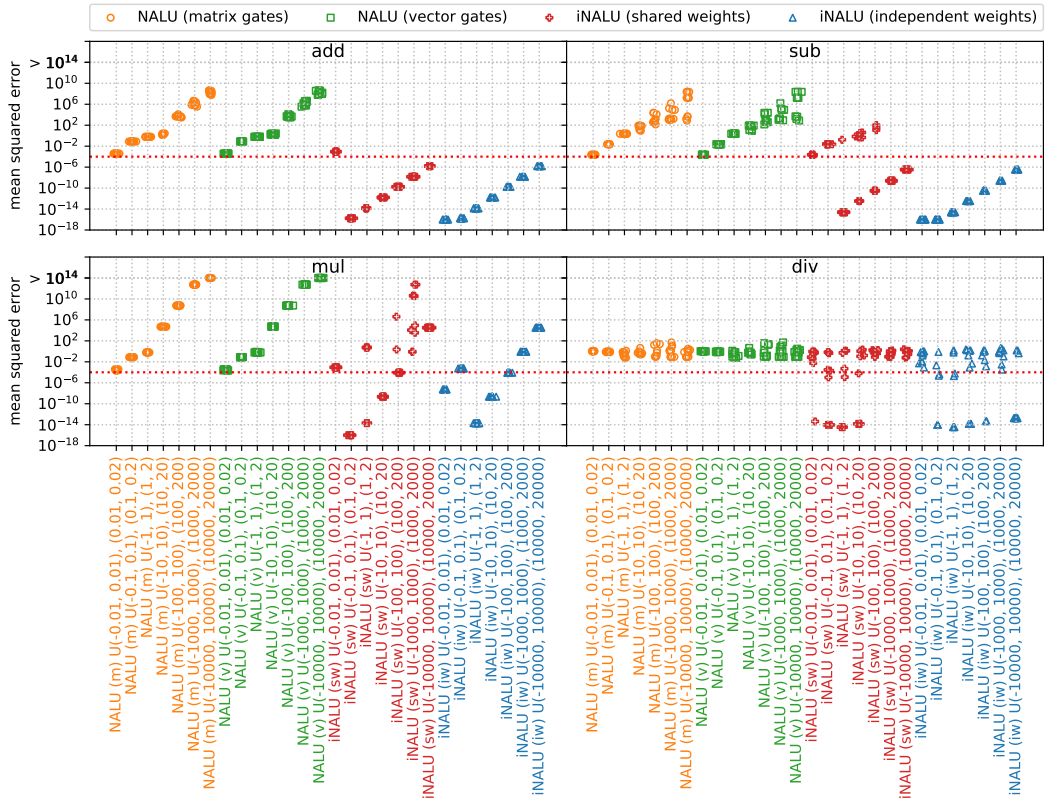


Figure 6.5: MSE for various magnitudes per operation over the extrapolation test dataset of experiment 2 (input magnitude). For a detailed description see Fig. 6.4.

6.1.7 Experiment 3 - Simple Arithmetic Task

Experiment 3 is a generalization of the minimal arithmetic task where the model has to learn to ignore irrelevant input dimensions to calculate the correct solution.

This setting is motivated by real-world tasks like spreadsheet calculations where one column is calculated by applying a simple operation to two specific columns while other columns are present but must not influence the result.

The model consists of one NALU layer with ten inputs and one output. We test the same input distributions as in the minimal arithmetic task (see 6.1.5).

Results Figure 6.6 shows the results of this experiment. Although the setting of experiment 3 is slightly more complex than experiment 1, most performance patterns repeat. In the following, we want to highlight some interesting exceptions.

For input data sampled from an exponential distribution, the results improve for the original NALU models, especially for summation and multiplication. For summation, training is unstable since some models succeed, but others fail to learn the task. In contrast to the minimal arithmetic task, iNALUs are successful for summation of exponentially distributed data with $\lambda = 0.2$ and show better results for multiplication. For division, the stability decreases as discussed before, so that only very few of our iNALU models succeed ($\approx 6.4\%$ of all experiments reach an MSE $< 10^{-5}$). The original NALU constantly failed for division. For subtraction, our model with shared weights is slightly more unstable, but our model with independent weights still yields stable results and calculates precisely.

6.1.8 Experiment 4 - Influence of Initialization

Experiment 1 suggests that training is unstable for some operations (subtraction and division). Whereas some of our improved models happen to solve the minimal task flawlessly, others fail to converge. As a consequence, a suitable initialization seems to be crucial for the successful training of more complex architectures. This observation is also confirmed by Madsen and Johansen [198].

In this experiment, we analyze the effect of different parameters for random weight initialization of the neurons. In contrast to the Minimal Arithmetic Task, the variables a and b are constructed by summing up 100 input vector entries assigned to a and b. Since Trask et al. does not specify the assignment in detail, we construct it by randomly assigning entries mutually exclusive to a and b and demand some inputs to be ignored by the model (since they neither contribute to a nor to b). We decide on the assignment once per task randomly so that the assignment is constant for all samples. Note that the assignment is not an additional input to the neural network, but instead it has to learn this assignment.

For this study, we examine the performance of our iNALU model with shared weights for standard normal distributed input values drawn from $\mathcal{N}(0, 1)$. We

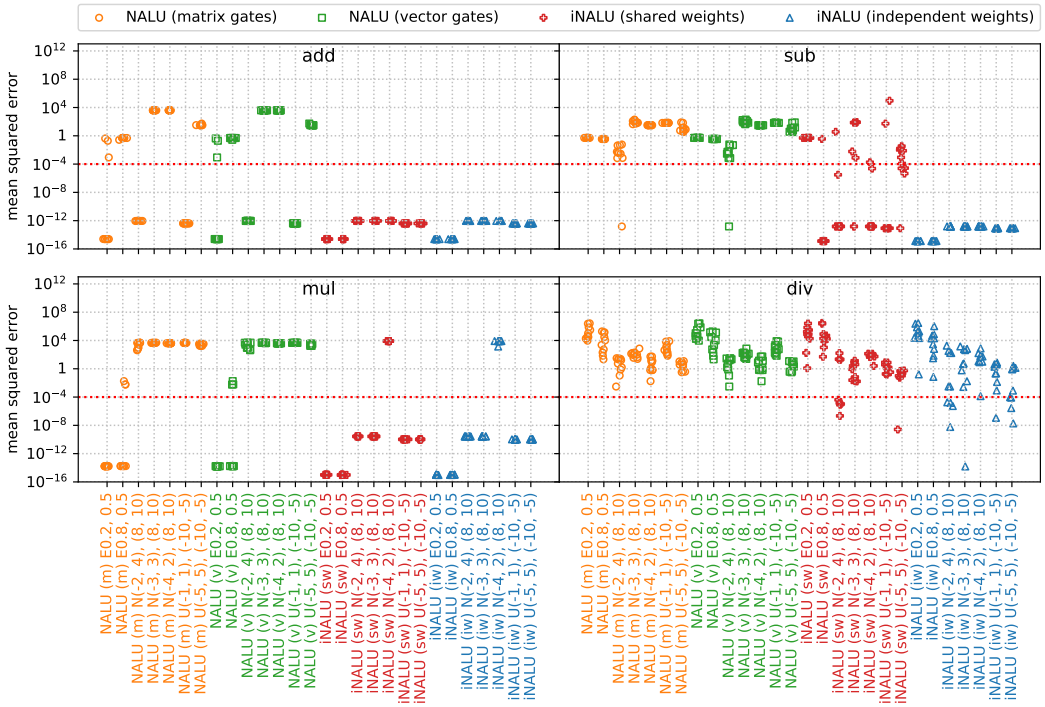


Figure 6.6: MSE for various input distributions per operation over the extrapolation test dataset of experiment 3 (simple arithmetic task). For a detailed description see Fig. 6.4.

choose to initialize the model weights following a normal distribution as well. To find suitable initialization parameters, we performed an exhaustive search for the parameters $\mu_g, \mu_{\hat{M}}, \mu_{\hat{W}} \in \{-1, 0, 1\}$ and $\sigma_g, \sigma_{\hat{M}}, \sigma_{\hat{W}} \in \{0.1, 0.5\}$. We repeat each parameter setting 20 times with different seeds to be able to assess the model stability. Note that large initializations ($\mu \neq 0$) bias the model towards specific operations, but especially sigmoid activations suffer from small random initializations ($\mu = 0$) [111] introducing conflicting requirements, which make this experiment highly relevant to find the best compromise for stability.

Results Table 6.1 shows the results of our parameter search. We consolidated the results for $\sigma = 0.1$ and $\sigma = 0.5$, since both parameters yielded similar results, and report the maximum MSE of all runs for each parameter setting. This is a very strict evaluation metric since only 1 of 20 models failing could obfuscate 19 successful runs. However, we are particularly interested in parameters that lead to stable models. The results support our finding from the arithmetic experiments that division is very unstable to learn as no model solved the problem reliably. Stable parameter configurations could be found for the remaining operations. Overall, the configuration $(\mu_g, \mu_{\hat{M}}, \mu_{\hat{W}}) = (0, -1, 1)$ is clearly most stable among all parameters tested for this task and architecture.

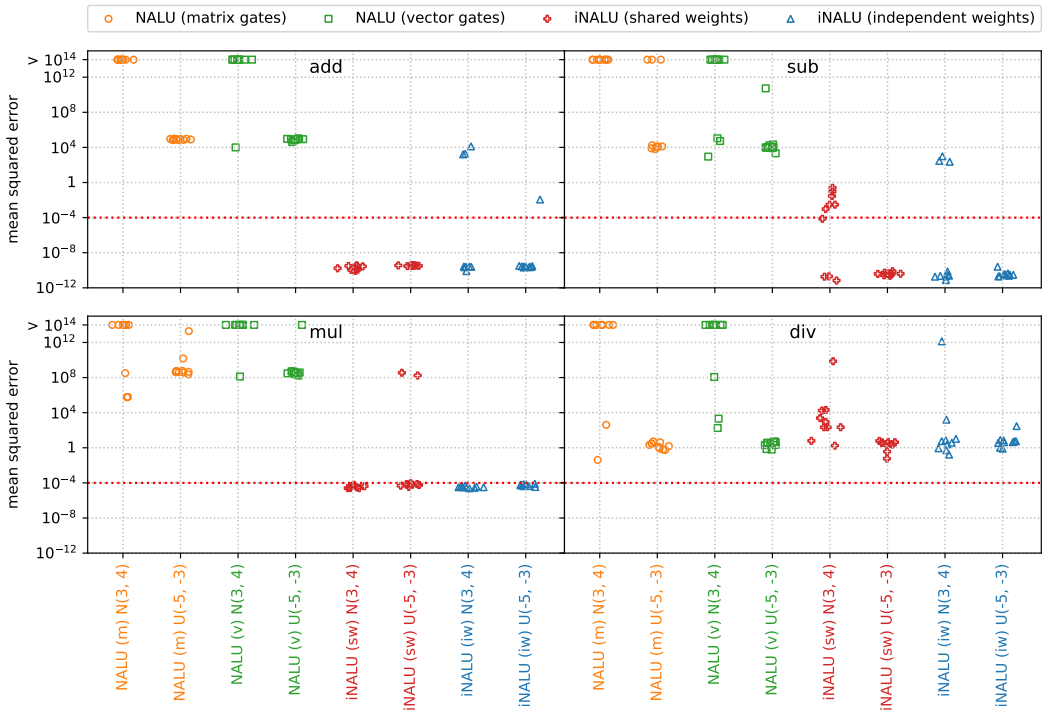


Figure 6.7: Extrapolation MSE for Experiment 5 (Simple Function Learning Task). Original NALU with gating matrix (m) and gating vector (v) are colored orange and green, our iNALU model with shared weights (sw) is colored red and with independent weights (iw) in blue.

6.1.9 Experiment 5 - Simple Function Learning Task

For the Simple Function Learning Task, we keep the setting of the previous experiment but focus on the comparison of our model using both, shared path-weights and independent path-weights, to the originally proposed NALU in both variants (see Section 6.1.1).

Since we found suitable initializations, we sample from uniform and truncated normal distributions and interpolate within the interval $[a, b] = [-3, 3]$ for both. This translates to a standard normal distribution ($\mu = 0, \sigma = 1$) for the truncated normal distribution. For the extrapolation interval, we choose $[3, 4]$ and $[-5, -3]$ to test positive as well as negative values outside the training range with different standard deviations.

Results Figure 6.7 shows that our iNALU models outperform the original NALU for summation, subtraction, and multiplication on almost all runs. Our model with independent weights is the most promising, since almost all runs succeed. However, few outliers indicate that the stability problem is not completely solved yet. This especially holds for division where all models fail to learn the operation correctly.

Table 6.1: Maximum MSE over all models in Experiment 4 for the Simple Function Learning Task (extrapolation) for weight initialization means of -1, 0, 1. Successful configurations (maximum loss < 0.001) in bold, percentage of successful repetitions in brackets.

$E[\mathbf{G}]$	$E[\hat{\mathbf{M}}]$	$E[\hat{\mathbf{W}}]$	ADD	DIV	MUL	SUB
-1	-1	-1	1E−01 (93)	7E+09 (0)	1E+07 (81)	1E−02 (95)
		0	1E−02 (95)	7E+09 (0)	1E+07 (95)	1E−03 (98)
		1	3E+00 (98)	7E+09 (0)	**1E−04** (100)	**2E−08** (100)
	0	-1	3E+07 (13)	2E+14 (0)	1E+07 (25)	1E+04 (16)
		0	1E−01 (78)	7E+09 (0)	1E+07 (95)	1E−01 (68)
		1	5E+03 (73)	1E+05 (0)	**1E−04** (100)	3E−02 (89)
	1	-1	6E+07 (0)	5E+14 (0)	1E+07 (50)	8E+03 (0)
		0	9E+14 (30)	3E+06 (0)	1E+07 (87)	9E+14 (21)
		1	1E+17 (13)	7E+09 (0)	6E+00 (94)	1E+15 (14)
0	-1	-1	2E−01 (91)	7E+09 (0)	1E+07 (53)	1E−02 (95)
		0	1E−01 (88)	1E+05 (0)	1E+07 (64)	1E−02 (94)
		1	**1E−04** (100)	4E+05 (0)	**1E−04** (100)	**1E−04** (100)
	0	-1	8E+03 (6)	3E+14 (0)	1E+07 (29)	8E+03 (7)
		0	3E−01 (68)	1E+14 (0)	1E+07 (65)	2E−01 (65)
		1	2E−01 (71)	7E+09 (0)	**2E−04** (100)	3E+00 (70)
	1	-1	8E+03 (6)	7E+14 (0)	1E+07 (27)	7E+03 (0)
		0	3E+16 (23)	2E+14 (0)	1E+07 (60)	1E+15 (10)
		1	2E+17 (21)	7E+09 (0)	1E+01 (94)	4E+15 (18)
1	-1	-1	1E−02 (92)	4E+05 (0)	1E+07 (40)	1E−02 (98)
		0	9E−03 (93)	7E+09 (0)	1E+07 (50)	5E−03 (87)
		1	**2E−04** (100)	7E+09 (0)	**1E−04** (100)	6E−03 (97)
	0	-1	8E+03 (21)	2E+14 (0)	1E+07 (29)	8E+03 (34)
		0	3E−01 (36)	7E+09 (0)	1E+07 (36)	5E−01 (26)
		1	3E+00 (80)	7E+09 (0)	**2E−04** (100)	1E−01 (72)
	1	-1	4E+05 (11)	4E+14 (0)	1E+07 (61)	8E+03 (10)
		0	7E+16 (17)	7E+09 (0)	1E+07 (28)	1E+13 (0)
		1	2E+17 (21)	2E+14 (0)	1E+01 (93)	7E+15 (21)

6.1.10 Discussion

The experiments in Section 6.1.4 analyzed the ability of the original NALU and our iNALU to solve various mathematical tasks and show that the performance of the NALU greatly depends on the distribution of the input data. The quality of the iNALU also depends on the input distribution, but is generally more stable and achieves better results. However, for larger magnitudes of input data, multiplication becomes challenging for the iNALU, compared to the NALU the input range for which the model can learn to multiply precisely is several magnitudes larger. Experiment 3 extends the arithmetic task by switching off several inputs. The results reinforce the findings of the first experiment that iNALU achieves better and more stable results than NALU. The differences between both iNALU models can be explained by the separate weighting matrix for summation/subtraction and multiplication/division. In experiment 5, which examined the performance of NALU and iNALU models for a complex function learning task, the iNALU achieves acceptable results for three of the four operations, whereas the original NALU fails for all four operations.

In general, the MSE calculated on the extrapolation datasets provides a good intuition if the NALU has learned the correct logical structure which is resilient to other value ranges. The interpolation results are very similar regarding the relative performance of all models but in general achieve a higher precision and thus a lower MSE (e.g. for summation in experiment 1 our iNALU model with independent weights yields $6.14 \cdot 10^{-15}$ for interpolation and $5.45 \cdot 10^{-13}$ for extrapolation on average MSE).

Furthermore, all experiments show that the operation division is the most challenging task for NALU and iNALU. The instabilities for division could be explained by the special case of dividing by near zero and the sampling strategy for a and b: For sampling inputs in an interval including zero, division might cause huge or very small results depending on the assignments of dividend or divisor which are represented by completely different weights. Possibly irrelevant input variables might therefore influence the result by such magnitude that there is no clear gradient signal for the assignment.

Another observation is that the optimal initialization is dependent on many factors such as task, model size, and value range. We want to emphasize that our parameter study is not intended to raise a claim for generally finding the optimal parameters, but rather to find initialization parameters for this specific task to allow for a model comparison. Our study suggests the parameter configuration $(\mu_g, \mu_{\hat{M}}, \mu_{\hat{W}}) = (0, -1, 1)$ which seems to be reasonable, since it treats the summative/subtraction path and multiplicative/division path equally at the beginning and assigns small activation weights to all inputs. We believe that the problem of generally finding optimal or near-optimal initializations is an interesting and theoretically challenging task for future work.

6.1.11 Conclusion

Recently, the NALU architecture was proposed to learn mathematical relationships, which are necessary to solve various machine learning tasks. In this work, we proposed an improved version of this architecture, called iNALU. The original NALU is only able to calculate non-negative results for multiplication and division by design and often fails to converge to the desired weights. We solved the issues of multiplying and dividing with mixed-signed results and proposed architectural variants for shared and independent weights with input independent gating. Further, we introduced a regularization term and a new reinitialization strategy which help to overcome the problem of unstable training.

We evaluated the improvements in five large-scale experiments which examine the influence of different input distributions and task-unrelated inputs. The first two experiments analyze the basic capabilities of NALU and iNALU. Furthermore, the parameter study for the Simple Function Learning Task shows that the choice of weight initializations has a huge impact on model stability. The parameter study revealed suitable initialization parameters. We showed that our proposed architectures can learn simple mathematical functions and outperform the reference models in terms of precision and stability.

Consequently, we will evaluate our iNALU architecture as an additional component in various architectures and downstream tasks such as data synthesis (Section 6.2.3) and anomaly detection (Sections 8.1 to 8.3) in the course of this thesis.

6.2 Modeling Distributions with GANs and VAEs

An important question for modeling anomalies with statistical approaches as well as neural networks is how well the model is able to capture non-anomalous (in the following *regular*) data. Several parametric [351, 125] and non-parametric [353] approaches find anomalies by the unlikeliness of data points regarding a ground-truth probability distribution [50], which is implicitly or explicitly learned from data.

Consider, for example, an anomaly detection task where for a given sample $\mathbf{x} \in \mathcal{D}$ the label $y \in \{\text{anomalous}, \text{regular}\} = Y$ has to be predicted. In this setting, a generative approach will model the joint distribution $P(\mathcal{D}, Y)$ as sample distribution together with the label distribution. In contrast to a discriminative model, a generative model will approach the question whether a given sample is anomalous by deciding from which underlying (learned) distribution the sample under observation is more likely to be chosen. Besides the obvious advantage of generating new samples from a trained model by sampling from the joint probability, and thus being able to create synthetic data following the learned distributions of the underlying model, a one-class learning setup can be constructed by forcing a model to learn an implicitly compressed representation. This compressed representation will use the dependencies and redundancy within the data, to model the data distribution by approximation. The model most likely fails to approximate samples which do not follow these dependencies seen during training, and which can thereby be considered anomalous. A typical parametric generative model that explicitly models these distributions is the Variational Auto-Encoder (VAE). An example for a non-parametric model which implicitly models the distributions is the Generative Adversarial Network (GAN).

This section elaborates the research question, how well both neural architectures are able to follow data characteristics regarding probability distributions and feature correlations of transaction data typically used in an anomaly detection setting with a focus on the generative performance. In Section 8.3, we will then evaluate the most promising models in an anomaly detection scenario.

6.2.1 Model Architectures

In this section, we will first introduce Generative Adversarial Networks, followed by the Variational Auto-Encoder including their specific architectures and adaptions for our experiments in detail.

Generative Adversarial Networks

GANs have been proposed by Goodfellow et al. [117] as a generative neural model that implicitly models a probability density function to generate new samples based on the learned characteristics of the real underlying data. This is achieved by a

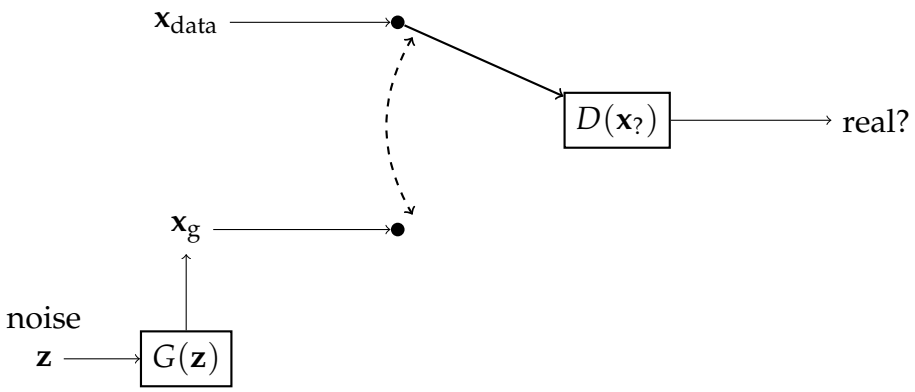


Figure 6.8: General GAN architecture: From a noise vector $\mathbf{z}$, the generator constructs fake samples x_g. Alternately, the discriminator D has real samples $\mathbf{x}_{\text{data}}$ or fake samples x_g as input and is trained to distinguish real from fake data. Iteratively, the generator learns to produce more realistic samples while the discriminator improves distinguishing fake from real samples.

game-theory-inspired setup having two networks competing against each other: A generator, which takes noise as input and shapes this noise to mimic realistic samples, and a discriminator, which takes generated samples or real samples as input and tries to differentiate real from fake data. The discriminator is trained in a supervised manner, whereas the generator is trained to deceive the discriminator as the only training objective. Thereby the quality of the network depends on learning both models in balance, i.e. the discriminator not outstripping the generator but providing enough information to let the generator improve. More formally, the interplay between discriminator and generator can be understood as a min-max game, where the discriminator D maximizes the probability of recognizing fake samples correctly, whereas the generator G minimizes inverse probability:

$$J = \min_G \max_D \mathop{\mathbb{E}}_{\mathbf{x}\sim P_{\text{data}}} [\log(D(\mathbf{x}))] + \mathop{\mathbb{E}}_{\mathbf{x_g}\sim P_{\text{G}}} \left[\log(1 - D(\mathbf{x_g}))\right]$$

The probability P_{G} is implicitly given by the generator model G such that $\mathbf{x_g} = G(z)$ having $\mathbf{z} \sim P(z)$, i.e. the noise sample z drawn from a noise distribution P(z) which the generator transforms to mimic a data sample. P_{data} denotes the real data distribution, that is, $\mathbf{x} \sim P_{\text{data}}$ are samples of the data. Practically, the generator is often trained to maximize $D(\mathbf{x_g})$ instead, which provides better gradients in early training [117].

As Goodfellow [115] discusses, this approach has several advantages in comparison to other generative models, such as parallel data sampling, no restrictions regarding the generator model choice, and better generated samples, i.e. when it comes to sharpness of images or distributions in general. However, the learning

procedure, which can be formalized as Nash Equilibrium, makes training challenging due to training instabilities such as failing convergence, vanishing gradients, the discriminator overpowering the generator, and the lack of diversity for generated samples, referred to as mode collapse [16, 187, 272]. Another drawback of the GAN, as proposed by Goodfellow et al., is the inability to generate discrete variables [115]. The Wasserstein GAN (WGAN) [17] and the advances built on WGAN, such as the gradient penalty instead of the gradient clipping [123] or the two time-scaled update rule [133], address these stability issues [17] and allow the modeling of discrete distributions over a continuous latent space [123] and faster convergence [124]. As these advancements promise better stability, we build the model for our experiments based on the WGAN architecture. Changes from the basic GAN architecture to WGAN will be detailed in the following.

WGAN is built on the Earth-Mover (EM) or Wasserstein distance as the objective, given as

$$W(P_{\text{data}}, P_{\text{G}}) = \inf_{\gamma \in \Pi(P_{\text{data}}, P_{\text{G}})} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\sim\gamma} [\|\mathbf{x} - \mathbf{y}\|] \tag{6.20}$$

with $\Pi(P_{\text{data}}, P_G)$ denoting the set of joint distributions $\gamma(x, y)$ with marginals P_{data} and P_G, respectively. Here, $\gamma(x, y)$ can be interpreted as "how much mass must be transported from x to y" [17] to transform P_{data} into P_G. With the Infimum the EM distance is defined by the best way to transform the distribution, i.e., the one with the least transported mass.

For the WGAN, the dual form of EM is used having

$$W(P_{\text{data}}, P_{\text{G}}) = \sup_{\|f\|_L \leq 1} \mathop{\mathbb{E}}_{\mathbf{x}\sim P_{\text{data}}} [f(\mathbf{x})] - \mathop{\mathbb{E}}_{\mathbf{x_g}\sim P_{\text{G}}} \left[f(\mathbf{x_g})\right]$$

with the Supremum of 1-Lipschitz functions representing the best score function [324, 17]. For the WGAN, this function f is learned by a neural network D that replaces the discriminator from the original GAN. D is called *critic* in this setting, since it is not trained to classify between real and fake [123], but instead the score which indicates how *real* the data are considered regarding the EM distance.

By this, the GAN objective changes to

$$J_{\text{WGAN}} = \min_G \max_D \mathop{\mathbb{E}}_{\mathbf{x}\sim P_{\text{data}}} [D(\mathbf{x})] - \mathop{\mathbb{E}}_{\mathbf{x_g}\sim P_{\text{G}}} \left[D(\mathbf{x_g})\right]$$

with the assumption that D is 1-Lipschitz. Arjovsky proposed to enforce 1-Lipschitz by clamping the gradients to a compact space $[-c, c]$, however, they commented themselves that "*weight clipping is a clearly terrible way to enforce a Lipschitz constraint*" [17]. A more theoretical analysis is given by Gulrajani et al. [123], who proposed ensuring 1-Lipschitz by introducing a Gradient Penalty (GP) in the loss.

$$J_{\text{WGAN-GP}} = \min_G \max_D \mathop{\mathbb{E}}_{\mathbf{x}\sim P_{\text{data}}} [D(\mathbf{x})] - \mathop{\mathbb{E}}_{\mathbf{x_g}\sim P_{\text{G}}} \left[D(\mathbf{x_g})\right] + \lambda \mathop{\mathbb{E}}_{\hat{\mathbf{x}}\sim P_{\hat{x}}} \left[(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2\right] \tag{6.21}$$

Here, $P_{\hat{x}}$ is defined by uniform sampling of the linear space between pairs of real and generated points, sampled from P_{data} and P_G, for which the gradient norm (given as $\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2$ in Eq. 6.21) is thereby constrained and λ is a weighting coefficient. Gulrajani et al. found $\lambda = 10$ to work well for several experiments and architectures that outperformed weight clipping by a large margin. Consequently, we follow Gulrajani et al. and build our models based on the WGAN-GP architecture.

For categorical features, GANs have inherent difficulties in modeling discrete distributions [115]. If categorical features are modeled as one-hot vectors, the min-max game becomes heavily biased in favor of the discriminator: The generator has to precisely generate discrete values of 0 and 1, whereas the task of the discriminator simplifies to detecting discrete values versus continuous values near the desired discrete values at best.

This issue can be addressed by different means, where two approaches are predominantly used in literature, using embeddings [241, 58, 253] and approximating discrete distributions with gumbel-softmax [171, 47, 85]. In [253] we evaluated three approaches to represent categorical features with a large number of possible values and found that learned dense representations, i.e., embeddings of categorical attributes, perform well.

The use of embeddings however has the disadvantage that a mapping from embedding space back to the data space is necessary when it comes to synthesizing data or inspecting generated samples on a feature-level view (i.e., for the generator), which is not straightforward when the latent embedding space representation is generated synthetically. Using gumbel-softmax allows to omit this transformation into a latent space and reverse staying within the feature space.

The gumbel distribution can be used to sample from a categorical distribution with class probabilities π_i [149] and samples $g_i \sim \text{Gumbel}(0, 1)$ by

$$s = \mathbb{1}\text{-hot}\left(\arg\max_i [g_i + \log \pi_i]\right).$$

Combining gumble and softmax, a differential approximation for arg max, gumble-softmax, is proposed by [149] as

$$y_i = \frac{\exp\left((\log(\pi_i) + g_i)/\tau\right)}{\sum_{j=1}^{k} \exp\left((\log(\pi_j) + g_j/\tau)\right)}$$

for $i \in \{1, \ldots, k\}$ and a temperature coefficient τ generating a k-dimensional sample vector. For $\tau \to 0$ gumble-softmax becomes identical to the respective categorical distribution $P(z)$ and samples y_i resemble one-hot vectors. For an increasing temperature $\tau \gg 0$, the samples resemble a uniform distribution over the categories. Hereby, the imprecision with regard to perfect one-hot vectors from data versus

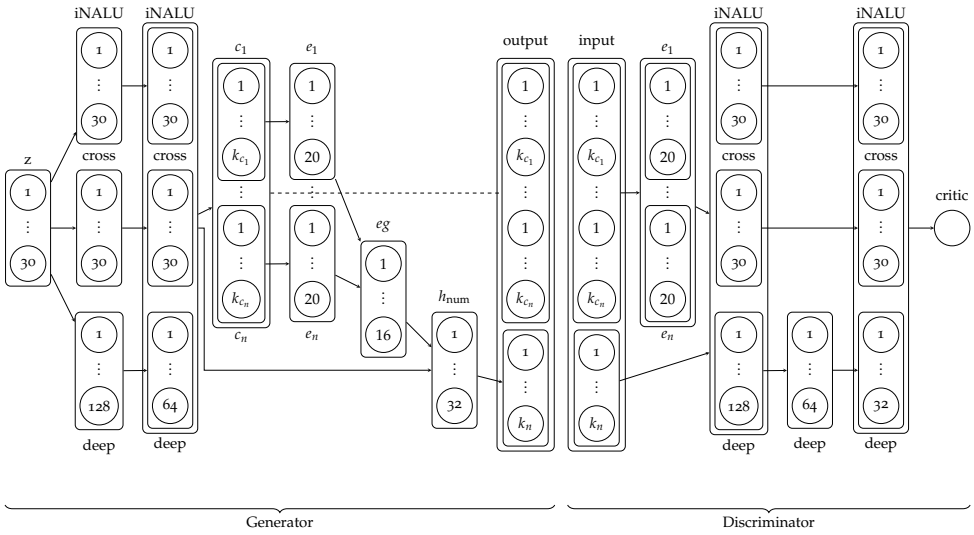


Figure 6.9: WGAN architecture with two cross- and iNALU layers and a specific generation of categorical and numeric features. The input vector z is sampled from noise followed by two deep cross-layers. Layers c_n represent the one-hot representation of the n-th categorical feature with k_{c_n} unique values. To allow numeric features depending on categorical features, embedding layers e_n are trained from one-hot representations and reduced to a single vector e_g concatenated with the last deep cross iNALU layer and used as input for the hidden layer h_{num} to generate the k_n numeric values. The categorical output values are directly mapped to the categorical layer prior embeddings denoted by the dashed line, arrows denote connections between all neurons within layer-boxes, i.e. fully connected layers. As the activation function, deep, e_g, and h_{num} layers use LeakyReLU. The critic and e_n layers have linear activations with (critic) and without (e_n) bias.

imperfect generated vectors can be balanced by replacing categorical values with gumbel-softmax approximations.

The approach proposed by Engelmann et al. [85] combines both a gumbel-softmax-based generator and an embedding layer for the discriminator input, which we adopt for our experiments. As shown in Fig. 6.9, the categorical layers c_i use the gumbel-softmax activation function, whereas the embedding layers e_i learn a latent representation of the categories to condition the numeric variables in the generator or a representation for the categorical features in the discriminator.

We also evaluate the benefit of feature crossings [329], which is a neural architecture similar to our Mixed Layers (proposed in Section 8.1.1). Both architectures rely on unspecific feed forward layers such as ReLUs and task-specific layers which are combined to a larger network: As task-specific layers, iNALU layers for Mixed

Layers support precise numeric calculation, cross layers for feature crossings emphasize feature combinations of varying degrees. This is achieved by re-introducing the input feature vector of the network x_0 and the output of the previous layer x_l to deeper layers as

$$x_{l+1} = x_0 x_l^\top w_l + b_l + x_l,$$

similar to residual connections in ResNet [128] with w_l, b_l denoting the weights and biases of the feature crossing layer l. We introduce iNALU layers as third component in addition to deep and cross layers to emphasize the ability to represent numeric dependencies.

The complete WGAN architecture used in our experiments is shown in Fig. 6.9. To evaluate the benefit of iNALU layers, we conduct our first experiment to find the optimal parameters without iNALU layers (i.e., having the concatenation layer only consisting of deep and cross layers) and include the iNALU layers in our second experiment measuring the improvement over the best model without iNALU.

Variational Auto-Encoder

Creating a (generative) model according to data-driven observations can also be understood from a statistical point of view in the framework of Maximum Likelihood Estimation (MLE). In this setting, the parameters of a model represented, for example, by a probability distribution, are learned by maximizing the likelihood of the data with respect to the parameter choice. For a generative task with N variables or features $X_i \in \mathbf{X}$, this is typically done by maximizing the marginal log-likelihood

$$\max_\theta \log p_\theta(\mathbf{X}) = \max_\theta \sum_{i=1}^{N} \log p_\theta(X_i),$$

which is in general computationally intractable, e.g., for a non-linear neural network [167]. To overcome this problem Kingma and Welling [167] proposed the Variational Auto-Encoder introducing a parametric inference model $q_{\theta^e}(z \mid x)$ to approximate the true posterior $p(z \mid x)$ and optimizing the variational lower bound

$$\log p_\theta(x) \geq \mathop{\mathbb{E}}_{q_{\theta^e}(z|x)} \left[\log p_{\theta^d}(x, z) - \log q_{\theta^e}(z \mid x)\right]$$

such that the inference model q approximates p best, i.e.,

$$\hat{q}_{\theta^e} = \arg\min_{\theta^e} \mathrm{KL}\left(q_{\theta^e}(z \mid x) \parallel p(z \mid x)\right).$$

Here, $\mathrm{KL}(P \parallel Q) = \sum_{x \in X} P(x) \log \frac{P(x)}{Q(x)}$ denotes the Kullback–Leibler (KL) divergence between two probability distributions. The *reconstruction* of x, $\mathcal{L}_{rec}$, as well as the regularization regarding the variational latent variable z, $\mathcal{L}_{reg}$, are summarized in the Evidence Lower Bound (ELBO),

$$ELBO(\theta) = \overbrace{\mathop{\mathbb{E}}_{q_{\theta^e}(z|x)}\left[\log p_{\theta^d}(x \mid z)\right]}^{\mathcal{L}_{rec}} - \overbrace{\mathrm{KL}\left(q_{\theta^e}(z \mid x) \parallel p(z)\right)}^{\mathcal{L}_{reg}},$$

which is used to optimize the parameters θ of the encoder (q) and decoder (p) neural network [167]. The idea of this regularization of the encoder regarding a desired probability distribution is to encourage the model to form the latent space, i.e., the posterior $q_{\theta^e}(z \mid x)$, similar to $p(z)$ according to KL, while on the other hand optimizing the precise reconstruction. This enables the model to learn a smooth meaningful latent space, suitable for interpolation [333] and as a generative model [114, 38], contrary to classical Auto-Encoder solely based on reconstruction error, which is "well known [to be] [...] not sufficient for learning useful representations" [164]. VAE allows, in addition to the reconstruction error, to consider the reconstruction probability when it comes to applications such as outlier or anomaly detection [13]. As practical estimator for *ELBO*, Kingma and Welling introduce the *reparametrization trick*, for which $z \sim q_{\theta_e}(z \mid x)$ can be reparameterized with a differentiable transformation $g_{\theta^e}(\epsilon, x)$ and independent marginal $p(\epsilon)$. Typically the VAE is reparameterized using Gaussian distributions $z \sim \mathcal{N}(\mu, \sigma^2)$ as $z = \mu + \sigma\epsilon$ with noise $\epsilon \sim \mathcal{N}(0, 1)$. The encoder thereby encodes data samples from input space into the parameter space of the variational (Gaussian) distribution. The decoder decodes samples drawn from this latent variational distribution back to the input-space reconstructing the original data. With Gaussian reparametrization, the KL divergence between the (multivariant) normal distribution with mean(s) $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ derived from the encoder and the standard-normal Gauss distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$ can be explicitly notated in closed form, which is often used as a practical implementation:

$$\begin{aligned}
\overbrace{\mathrm{KL}\left(q_{\theta^e}(z \mid x) \parallel p(z)\right)}^{\mathcal{L}_{reg}} &= \mathrm{KL}\left(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I})\right) \\
&= \frac{1}{2}\left(\boldsymbol{\mu}^T\boldsymbol{\mu} + \mathrm{tr}(\boldsymbol{\Sigma}) - k - \log(\det\boldsymbol{\Sigma})\right) \\
&\overset{(*)}{=} \frac{1}{2}\sum_{i=1}^{k}\left(\mu_i^2 + \sigma_i^2 - 1 - \log\left(\sigma_i^2\right)\right),
\end{aligned}$$

while $(*)$ holds for independent $X \sim \mathcal{N}_k(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with $\boldsymbol{\sigma} = \mathrm{diag}\,\boldsymbol{\Sigma}$.

When it comes to practically applying VAE to transaction data, heterogeneous feature types have to be taken into account. To represent categorical and other non-continuous attributes properly, different approaches have been suggested, e.g., choosing different likelihood models and priors [215] or approximating categorical variables by continuous variables introducing noise [89, 301]. For our experiments, we follow [82] and model categorical attributes by parametrizing each categorical

feature i as $p_{\theta^d}(x_i \mid z) = \text{softmax}(a_i^d(z))$. Therefore, $a_i^d(z)$ is an unnormalized vector of probabilities for all values, learned as feed-forward neural network. The input encoding from one-hot representations to a dense representation is given as $q_{\theta^e}(z \mid a^e(x))$ with a^e being composed of embedding layers for each categorical feature and direct input mappings for numerical features as depicted in Fig. 6.10. Additionally, for our later experiments, we adapt the model by including iNALU layers to evaluate the influence on model performance regarding numeric dependencies. As depicted in Fig. 6.11, we specifically add iNALU layers for numeric attributes in parallel to embedding layers for categorical attributes.

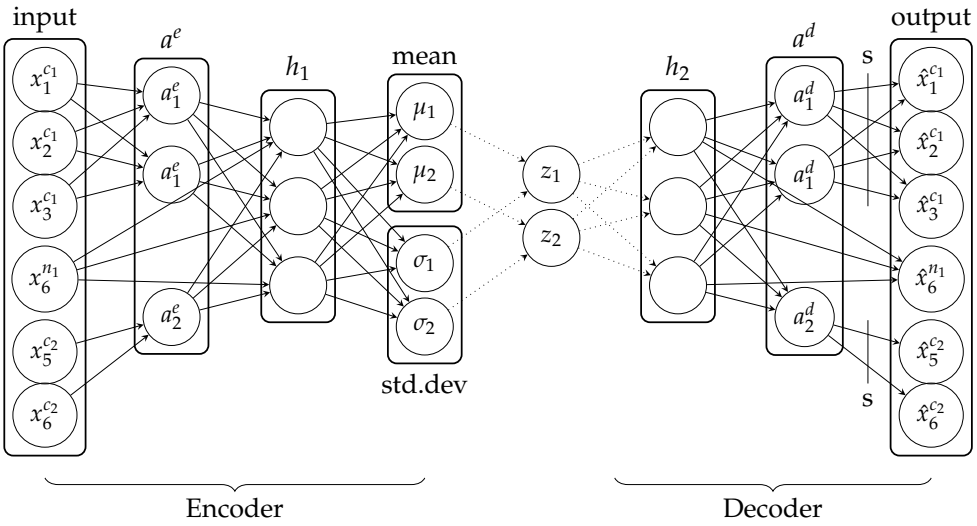


Figure 6.10: VAE for one-hot encoded categorical input for features c_1, c_2 and numerical feature values n_1, with embedding layers a^e, a^d per categorical value and $\overset{\text{s}}{\nrightarrow}$ denoting the softmax activation function for categorical variables, hidden layers h_1, h_2 and variational parameterization layers μ and σ. Dotted arrows correspond to sampling and reparametrization for training.

6.2.2 Evaluation

The evaluation of generative models is generally not straightforward as there is no specific sample-based ground truth regarding the generated data compared to a classification setting, where the predicted class and the actual class can be compared per sample [335]. Besides this, the desired application of the generative model is important, when it comes to defining criteria to evaluate generated samples and thus generative models: For example, generative models for image creation may be used to create visually appealing images from a human perception or may be used to improve the robustness of classifiers, e.g., against adversarial attacks. These appli-

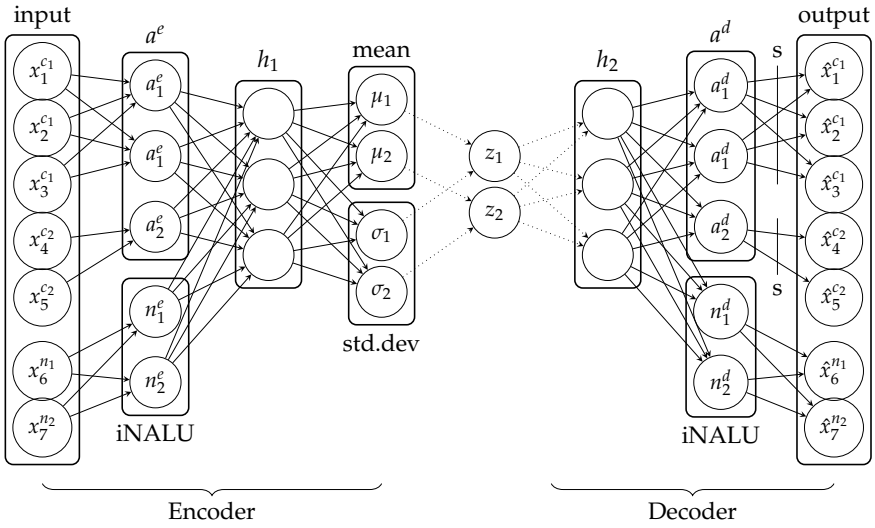


Figure 6.11: VAE with iNALU for one-hot encoded categorical input for features c_1, c_2 and numerical feature values n_1, with embedding layers a^e, a^d per categorical value and $\overset{s}{\nrightarrow}$ denoting the softmax activation function for categorical variables, hidden layers h_1, h_2 and variational parameterization layers μ and σ. Dotted arrows correspond to sampling and reparametrization for training.

cations can require completely different samples and therefore need their generative models to be evaluated regarding different objectives [313].

Although logarithmic likelihood is considered the standard approach to evaluate generative models, its applicability in high-dimensional settings and regarding the computational intractability for implicit models, such as GANs, as well as its expressiveness regarding sample quality are questioned [313, 335]. If likelihood calculation is intractable, approaches to estimate density from samples such as Parzen window-based log-likelihood estimates [231], often referred to as Kernel Density Estimation (KDE), have been used [43, 117] and refined as a metric [335, 316, 365] despite questions of validity as a generalizable evaluation function to quantitatively compare generative models [313].

Some evaluation approaches are domain-specific, such as Domain Knowledge Checks [253] for network traffic generation, Inception Score [271] or the Fréchet Inception Distance [134] for image data, while other metrics are built upon test statistics [122, 250]. For an overview of several evaluation measures, see [34]. To summarize, a vast amount of evaluation measures have been proposed with differing focus on key aspects of the generated samples, yet "there is no one-fits-all [metric] [...] but a proper assessment of model performance is only possible in the the context of an application" [313]. For our experiments we therefore consolidate different empirical metrics applicable to transaction data and discuss their implications side by side.

Expanding simple feature statistics such as mean and variance of the real and generated data, we compare the likelihood of generated data on per-feature basis with real data using kernel density estimates giving an in-depth view on feature modes. This can, for example, reveal whether properties such as work days or working hours for time-related features are correctly learned (as in [253]), as well as numerical modes, e.g., for prices or quantities over different products.

To compare generated and real data quantitatively, approaches to compare probability distributions can be adopted: For our experiments, a quantitative evaluation is used based on Kullback-Leibler divergence by its symmetric variant, Jensen-Shannon divergence (detailed in Eq. 6.30). Although the Earth-Mover distance could also be incorporated as an evaluation metric, we refrain from explicitly evaluating with EM distance for a fair comparison, since it is explicitly used as training objective in the WGAN model (see Eq. 6.20). The Fréchet Inception Distance adaption we evaluate implicitly uses the Wasserstein-2 distance, however, as a relaxed parametric measure which therefore does not hinder a fair comparison.

While the aforementioned views are limited to inter-feature dependencies between real and generated data, intra-feature dependencies are considered by a feature correlation analysis. In the following, we will summarize the evaluation approaches and discuss the implications of each choice.

Kernel Density Estimation

Technically a density estimator aims to estimate the Probability Density Function (PDF) on which the data is based on. The most basic density estimator, the histogram, consolidates data points within equidistant ranges to the respective bin and can be formulated as follows:

For equidistant bin edges $\{b_k\}$ that span the entire value space with a bandwidth $h = b_{k+1} - b_k$, the histogram density can be defined as

$$\mathrm{KDE}(x) = \frac{v_k}{nh} \quad \text{for} \quad b_k < x < b_{k+1} \,, \tag{6.22}$$

with $v_k = |\{x \in \mathcal{X} : b_k < x < b_{k+1}\}|$, i.e. the number of data points that fall into the k-th bin. Intuitively, this can be seen as stacked boxes with a width of h around the center of each bin. Per observation, the height of $1/nh$ adds to the height of the bin, ultimately giving the height of the bin v_k/nh for the respective stack of boxes. This intuition can be generalized from boxes to any *Kernel* function K that satisfies the property

$$\int_{-\infty}^{\infty} K(x)dx = 1 \,. \tag{6.23}$$

Instead of boxes, for each observation the associated kernel can be stacked or, in this case, summed to form the probability mass by its kernel according to the

contributing observations. The density estimate for x with samples $\{x_1, ..., x_n\}$ is then given by

$$\mathrm{KDE}(x; K, h) = \frac{1}{nh} \sum_{i=0}^{n} K\left(\frac{x - x_i}{h}; h\right) \tag{6.24}$$

$$K^{\mathrm{Gauss}}(x; h) \propto \exp\left(-\frac{x^2}{2h^2}\right) \tag{6.25}$$

The most frequently used kernel is the Gaussian kernel K^{Gauss}, which is also applied in this thesis. Bandwidth h is a parameter that can be chosen to adjust *smoothness* of the resulting density function and results in a smooth distribution for a large bandwidth. As a basic rule of thumb to choose h, Scott suggested for a Gaussian kernel [285]

$$h = 1.06 \sigma n^{-1/5} ,$$

with n samples and their standard deviation σ. For a thorough review of density estimation and a mathematical introduction of Scott's rule of thumb, see [284].

Evaluation Metrics for Generative Models

While KDE allows an explorative evaluation, for a quantitative view, additional metrics have to be defined, several of which have also been introduced as training objectives for generative models. The Kullback-Leibler divergence [170] allows the comparison of two probability distributions P and Q and is also incorporated as part of the training objective in the VAE. A KL divergence of zero denotes identical distributions.

$$\underset{X}{\mathrm{KL}}(P \parallel Q) = \begin{cases} \sum\limits_{x \in X} P(x) \log \frac{P(x)}{Q(x)} & \text{for discrete } P, Q \\ \int\limits_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx & \text{for continuous } P, Q \text{ with PDFs } p, q \end{cases} \tag{6.26}$$

KL has the disadvantage of being unsymmetrical and having an undefined maximum bound. Jensen–Shannon divergence (JSDiv) is a symmetrical variant derived from KL divergence with bounds in $[0, 1]$, which can be used as metric (in a mathematical sense). The Jensen-Shannon *distance* (JSD) is defined as square root over JSDiv. With probability distributions P, Q and $M = \frac{1}{2}(P + Q)$, it is defined by:

$$\underset{X}{\mathrm{JSDiv}}(P \parallel Q) = \frac{1}{2}\left(\underset{X}{\mathrm{KL}}(P \parallel M) + \underset{X}{\mathrm{KL}}(Q \parallel M)\right) \tag{6.27}$$

$$\underset{X}{\mathrm{JSD}}(P \parallel Q) = \sqrt{\underset{X}{\mathrm{JSDiv}}(P \parallel Q)} \tag{6.28}$$

In general, for KL and JSD, X is implicitly defined as the common probability space of P and Q and therefore is omitted in notation. We introduce X in notation where we explicitly want to emphasize the probability space considered.

To evaluate a learned model by its generated data in comparison to real data, P and Q are distributions that correspond to the same feature of both datasets. Therefore, a model that learned the feature distribution P similar to the real feature distribution Q will yield a JSD $\lim_{P \to Q} = 0$, while a model that failed to learn the correct feature distribution will yield a JSD $\lim_{P \not\to Q} = 1$. However, the probability distributions are modeled implicitly by the model, and in general, the real probability distribution from which the data are 'drawn' is unknown as well. To compare real and generated samples by JSD, first numeric features are pairwise min-max normalized over their joint feature-value space to ensure a common probability space for both real and generated data. Then we estimate their PDF using KDE for numeric features and the Empirical Distribution $P_E(X = i) = c_i / \sum_i c_i$ for counts c_i of feature-value i as estimate for categorical features. The JSD per feature is calculated over their joint probability space as given in Eq. 6.29. From all estimators, we then predict the joint probability space of P and Q and evaluate pairwise with JSD per feature $X_i^{\text{syn}}, X_i^{\text{real}}$. We then report the JSD averaged over all features, as defined in Eq. 6.30.

$$\hat{X}^{\text{syn}} = \frac{x - \min(X_{g+r})}{\max(X_{g+r}) - \min(X_{g+r})} \quad \forall x \in X^{\text{syn}}$$

$$\hat{X}^{\text{real}} = \frac{x - \min(X_{g+r})}{\max(X_{g+r}) - \min(X_{g+r})} \quad \forall x \in X^{\text{real}}$$

$$\hat{X}_{g+r} = \hat{X}^{\text{syn}} \cup \hat{X}^{\text{real}}$$

$$\hat{P}_X = \begin{cases} \mathrm{KDE}(X; K, h) & \text{for a numeric feature X} \\ P_E(X) & \text{for a categorical feature X} \end{cases}$$

$$\mathrm{JSD}(X^{\text{syn}}, X^{\text{real}}) = \underset{\hat{X}_{g+r}}{\mathrm{JSD}}(\hat{P}_{\hat{X}^{\text{syn}}} \parallel \hat{P}_{\hat{X}^{\text{real}}}) \tag{6.29}$$

$$\mathrm{JSD}(\mathcal{D}^{\text{real}}, \mathcal{D}^{\text{syn}}) = \frac{1}{n}\sum_{i}^{n} \mathrm{JSD}(X_i^{\text{syn}}, X_i^{\text{real}}) \tag{6.30}$$

For KDE we use a bandwidth of $h = 0.01$ and a linear kernel $K(x; h) \propto 1 - x/h$. Bandwidth and kernel can be tuned for various applications and dataset conditions (see, for example, [129] for a detailed discussion). For our application, however, a

precise estimate over the whole numeric scale is not important as long as the estimate does not oversmooth the differences between both distributions.

Besides JSD the Earth-Mover (EM) or Wasserstein distance is another metric to compare distributions. Intuitively, it can be understood as the best way to transform one distribution into the other, i.e., the 'plan' with the least transported mass. For two discrete variables, this can be visualized as the most effective way to jointly increase and decrease (normalized) histogram bins to transform one distribution into the other. More formally, the n-th Wasserstein distance is defined as

$$\mathrm{W}_n(P,Q) = \left(\inf_{\gamma \in \Pi(P,Q)} \mathop{\mathbb{E}}_{(x,y)\sim\gamma} [\|x-y\|^n] \right)^{\frac{1}{n}}, \tag{6.31}$$

with $\Pi(P,Q)$ denoting the set of all possible joint distributions $\gamma(x,y)$ with marginals P and Q. Here, $\gamma(x,y)$ can be interpreted as "how much mass must be transported from x to y" [17] to transform P into Q, and the infimum corresponds to the least costly way γ over all possible instances $(x,y) \sim \gamma$ having $x \in P$ and $y \in Q$.

The 2-Wasserstein distance is used in Fréchet Inception Distance (FID) [134], for which P and Q are assumed to be Gaussian with $P = \mathcal{N}(\mu_P, \Sigma_P)$ and $Q = \mathcal{N}(\mu_Q, \Sigma_Q)$. This simplifies $\mathrm{W}_2(P,Q)$ to

$$\mathrm{W}_2(P,Q)^2 = \|\mu_P - \mu_Q\|_2^2 + \mathrm{tr}\big(\Sigma_P + \Sigma_Q - 2\big(\Sigma_P\Sigma_Q\big)^{\frac{1}{2}}\big).$$

For typical FID evaluations in the image domain, P and Q are the activations of a pretrained Inception v3 ImageNet model [306] fed with real and generated samples. The idea of FID as evaluation measurement is to compare generated and real data not at the feature distribution level, since feature distributions (i.e., pixel distributions) are not expressive for image quality assessment. Instead, meaningful image representations, extracted by pretrained 'image-feature-extraction' layers of an Inception v3 model are compared.

For our application domain of transaction data, the choice of an auxiliary model to use as a feature extractor is not straightforward. As the most general model, a pretrained Auto-Encoder could be used, mapped to learn low-dimensional representations which reproduce the original sample best. However, such an evaluation, similar to modeling features of visually appealing images, is not guaranteed to emphasize the appropriate aspects of transaction data as well: In general, reconstruction from a lower-dimensional space itself is prone to inaccuracies in feature space and might obfuscate problems within the generated data. On the other hand, an auxiliary classification task is task- and dataset-dependent and might emphasize representations which are beneficial to solve the given task. However, surface-level features that are relevant for the auxiliary task might be overestimated and other characteristics of the real data, which are important to generate realistic data, might be overlooked. For our evaluation, we therefore evaluate FID directly on feature

level, which corresponds to comparing Gaussian distributions regarding their mean and standard deviation, which were fitted to the generated and real data. This might obscure different modes of normal distributions or other non-normal distributions in general, but ensures that the mean and standard deviation of generated data generally fits real data.

Correlation Analysis

Inter-feature relations are another important aspect a generative model has to capture in order to model and synthesize data in a realistic manner: If two features within the real data are highly correlated, this correlation also has to be considered by a generative model and can be examined in the generated data. For the evaluation of correlations between numeric features, two correlation coefficients are typically used: Pearson's and Spearman's correlation coefficient. Both are applied pairwise between features and range from -1 for an inversely proportional correlation to 1 for a perfect correlation. A value of 0 expresses no correlation with respect to the coefficient. Pearson's correlation coefficient is directly motivated by covariance, which models a monotonic relationship between two random variables.

$$\operatorname{cov}(X, Y) = \mathbb{E}\left[(X - \mathbb{E}(X))(Y - \mathbb{E}(Y))\right] \tag{6.32}$$

The covariance is defined as the product of the differences between each random variable and its mean, i.e., is positive if both random variables differ from their expected value in the same direction or negative for opposite directions.

Pearson's correlation coefficient is calculated by standardizing both random variables

$$\rho_p(X, Y) = \frac{\operatorname{cov}(X, Y)}{\sigma_X \sigma_Y} \tag{6.33}$$

with σ_X denoting the standard-deviation of the random variable X and can be used to find linear relationships.

Spearman's correlation coefficient can be considered a special case of Pearson's correlation coefficient for which the rank of two random variables is compared instead of their values.

$$\rho_s(X, Y) = \frac{\operatorname{cov}(\mathrm{rg}_X, \mathrm{rg}_Y)}{\sigma_{\mathrm{rg}_X} \sigma_{\mathrm{rg}_Y}} \tag{6.34}$$

Here, rg_X denotes the ranking of a list of feature values X, i.e., the smallest value in the list is replaced by 0, the second smallest value by 1 and so on.

For the cost of losing the scaling of differences between values, Spearman's correlation coefficient is more robust against outliers and captures non-linear relationships better in comparison to Pearson's [247].

Since both coefficients are pairwise, the correlation of all feature combinations is calculated and a correlation matrix typically depicted as heat map is composed. Such heat maps can be easily compared to evaluate if a synthetic dataset resembles the inter-feature relationships as they are present in real data.

When it comes to categorical features, the question of feature correlation becomes more complex: While for ordinal features the values can be ordered and thus a ranking-based correlation can be calculated, categorical features do not have an inherent order and Pearson's and Spearman's correlation coefficient cannot be directly applied. For example, the categorical attribute *transaction type* may consist of categories such as CASH-IN or CASH-OUT. However, there is no relation defining if CASH-IN < CASH-OUT or CASH-IN > CASH-OUT. When represented in an ordinal way, one could define any arbitrary order for the features, which implies that there is no inverse correlation. Assume a numerical feature, the amount of money added to the balance, is positive for the transaction type CASH-IN but negative for CASH-OUT. If we represent CASH-IN by 0 and CASH-OUT by 1, we can calculate the correlation coefficient and will find a strong positive correlation between the amount and transaction type. Since the encoding has an arbitrary order, we could also encode CASH-IN by 1 and CASH-OUT by 0, leading to a strong negative correlation in numbers. This is typically addressed by calculating the correlation ratio [95], which measures the dispersion for different categorical values. For a numerical feature Y and a categorical feature X let $S_x^X := \{i \mid \mathcal{D}_{X_i} = x\}$ be the set of indices for all samples with a particular value $x \in X$. With the mean per category $\bar{y}_x = |S_x^X|^{-1} \sum_{i \in S_x^X} y_i$ and the mean over all Y $\bar{y} = \mathbb{E}[Y]$ the correlation ratio is calculated by

$$\eta^2(X, Y) = \frac{\sum_x |S_x^X|(\bar{y}_x - \bar{y})^2}{\sum_y (y - \bar{y})^2}. \tag{6.35}$$

In the case of a dichotomous categorical variable X, i.e., a variable with only two possible values, the correlation analysis can be simplified further by using the point-biserial correlation coefficient ρ_{pb} as follows [110]:

$$\rho_{pb}(X, Y) = \frac{\bar{y}_1 - \bar{y}_0}{\sigma_Y} \sqrt{\frac{|S_1^X||S_0^X|}{|\mathcal{D}|^2}} \tag{6.36}$$

Here, the differences of the means $\bar{y}_i$ for all samples that are associated with the respective category i are normalized by Y's standard deviation σ_Y and scaled with $|S_i|$ as the number of samples that belong to the respective category and $|\mathcal{D}|$, the total number of samples. For typical one-hot encoded categorical variables, we use this simplified calculation. Note that for this case, it is $\rho_p(X, Y)^2 = \eta^2 = \rho_{pb}^2$.

For two categorical variables, the correlation (or in this case often association) can be calculated by *Cramér's V* [65, 28]. It is based on χ^2 statistics and can be used with variables with more than two categories. For two discrete random variables X, Y with k and l possible values $x \in \{C_{X_1}, \cdots, C_{X_k}\}$, $y \in \{C_{Y_1}, \cdots, C_{Y_l}\}$, let

S_x^X, S_y^Y be sets of indices for all samples with a particular value $x \in X$, $y \in Y$, and $S_{x,y}^{X,Y} := \{i \mid \mathcal{D}_{X_i,Y_i} = (x,y)\}$ be the set of indices for all samples that have both specific values $x \in X$ and $y \in Y$. Then the χ^2 statistics can be determined by

$$\chi^2(X,Y) = \sum_{i=1}^{k}\sum_{j=1}^{l} \frac{(|S_{i,j}^{X,Y}| - \mathbb{E}[i,j])^2}{\mathbb{E}[i,j]}$$

with $\mathbb{E}[i,j] = \frac{|S_i^X||S_j^Y|}{|\mathcal{D}|}$ assuming statistical independence. Cramer's V is then calculated by normalizing χ^2 by the number of samples in the dataset and the number of categories:

$$V(X,Y) = \sqrt{\frac{\chi^2(X,Y)/|\mathcal{D}|}{\min(k-1, l-1)}}$$

Cramer's V ranges from 0 (no association between two features) to 1 (both features are completely dependent on each other).

If bot variables are dichotomous, Cramér's V simplifies to Pearson's phi coefficient since $\min(k-1, l-1) = 1$, which can also be formulated as Pearson's correlation coefficient between two binary variables [63]

$$\phi(X,Y) = \sqrt{\frac{\chi^2(X,Y)}{|\mathcal{D}|}} = \frac{|S_{0,0}^{X,Y}||S_{1,1}^{X,Y}| - |S_{0,1}^{X,Y}||S_{1,0}^{X,Y}|}{\sqrt{|S_0^X||S_1^X||S_0^Y||S_1^Y|}},$$

since χ^2 simplifies for two dichotomous variables to

$$\chi^2(X,Y) = \frac{|\mathcal{D}|\left(|S_{0,0}^{X,Y}||S_{1,1}^{X,Y}| - |S_{0,1}^{X,Y}||S_{1,0}^{X,Y}|\right)^2}{|S_0^X||S_1^X||S_0^Y||S_1^Y|}.$$

Besides the advantage of integrating well within Peason's correlation framework, it has the disadvantage of judging two categorical variables symmetrically. In practice, the association between two variables, however, can be asymmetric, e.g., when observing that one variable completely determines the value of variable but not vice versa. To consider this asymmetry in evaluation, the Uncertainty Coefficient $U(X,Y)$ (often attributed as Theil's U [312]) can be used, which builds upon entropy $H(X)$ and conditional entropy $H(X \mid Y)$:

$$
\begin{aligned}
H(X) &= -\sum_{x \in X} P(x) \log P(x) \\
H(X \mid Y) &= -\sum_{x \in X, y \in Y} P(x,y) \log \frac{P(x,y)}{P(x)} \\
U(X,Y) &= 1 - \frac{H(X \mid Y)}{H(X)}
\end{aligned}
$$

for outcomes x, y of categorical random variables X and Y. As the conditional entropy is not symmetric, U is neither and allows a more precise evaluation of asymmetric associations. $U(X,Y)$ ranges from 0 (no association) to 1 (feature Y fully determines feature X).

Consider, for example, the case for the columns *tax code* and *payment block* from the SAP dataset:

		payment block	
	counts	empty	XI
tax code	empty	52 883	1 650
	R	0	144

Whereas the *tax code* value *R* (144 observations) is always associated with the value *XI* for *payment block*, *XI* is associated with 1 650 observation of empty values for *tax code*. This results in a Cramer's V of

$$V(\textit{payment block}, \textit{tax code}) = V(\textit{tax code}, \textit{payment block}) = 0.278$$

and the Uncertainty Coefficient yields

$$U(\textit{tax code}, \textit{payment block}) = 0.063$$

but on the other hand

$$U(\textit{payment block}, \textit{tax code}) = 0.498\ ,$$

which means *tax code* provides way more information about *payment block* than vice versa. A model not considering this asymmetric relationship in data would not be penalized with Cramer's V as evaluation metric, which rationalizes the choice of the Uncertainty Coefficient over Cramer's V.

In summary, we calculate the correlation between two numerical variables using Pearson's correlation coefficient ρ_p, between categorical and numerical variables using the Correlation Ratio η^2 and the association between categorical variables using the Uncertainty Coefficient U, as formalized in Eq. 6.37. With this theoretical

framework, we can compare the correlation of categorical and numerical features in various combinations between the real dataset and the synthesized samples and evaluate if the relations and dependencies within the features are correctly modeled by the generative process.

$$\text{corr}(X, Y) = \begin{cases} \rho_p(X, Y) & \text{if } X \text{ and } Y \text{ numerical} \\ \eta^2(X, Y) & \text{if } X \text{ categorical and } Y \text{ numerical} \\ U(X, Y) & \text{if } X \text{ and } Y \text{ categorical} \end{cases} \tag{6.37}$$

Explorative Evaluation To exploratively evaluate the correlation, we calculate the pairwise correlation between all features X_i of a dataset $\mathcal{D}$ and analyze the resulting correlation matrix as heat map, color-coding the strength of feature correlation of real and synthetic data.

$$\text{corr}(\mathcal{D}) = \begin{pmatrix} \text{corr}(X_1, X_1) & \text{corr}(X_1, X_2) & \cdots & \text{corr}(X_1, X_i) \\ \text{corr}(X_2, X_1) & \text{corr}(X_2, X_2) & \cdots & \text{corr}(X_2, X_i) \\ \vdots & \vdots & \ddots & \vdots \\ \text{corr}(X_i, X_1) & \text{corr}(X_i, X_2) & \cdots & \text{corr}(X_i, X_i) \end{pmatrix} \tag{6.38}$$

Quantitative Evaluation A quantitative evaluation is approached by calculating the mean absolute error between the correlation matrix of two datasets, $\mathcal{D}^{\text{syn}}$ and $\mathcal{D}^{\text{real}}$, with an identical number of features n.

$$\overline{\text{corr}}(\mathcal{D}^{\text{syn}}, \mathcal{D}^{\text{real}}) = \frac{1}{n^2} \sum_{i}^{n} \sum_{j}^{n} \left| \text{corr}(\mathcal{D}^{\text{syn}}_{X_i}, \mathcal{D}^{\text{syn}}_{X_j}) - \text{corr}(\mathcal{D}^{\text{real}}_{X_i}, \mathcal{D}^{\text{real}}_{X_j}) \right| \tag{6.39}$$

In our experiments, $\mathcal{D}^{\text{syn}}$ corresponds to the dataset synthesized by a NN which is expected to mimic the real dataset $\mathcal{D}^{\text{real}}$. Therefore, the NN is expected to generate data with pairwise feature correlations $\text{corr}(\mathcal{D}^{\text{syn}}_{X_i}, \mathcal{D}^{\text{syn}}_{X_j})$ similar to the correlations $\text{corr}(\mathcal{D}^{\text{real}}_{X_i}, \mathcal{D}^{\text{real}}_{X_j})$ present in the real dataset.

6.2.3 Experiments

The experiments in this section are constructed to answer RQ 1.2 and RQ 1.3 by specifically evaluating both generative neural network models, VAE and GAN, including the adaptations we proposed to model transaction data at best. For our experiments in this section, we focus on two transaction datasets, PaySim and SAP, as well as a benchmark dataset from the UCI-Machine Learning Repository, the "Census Income" dataset, which have been introduced in Chapter 5 in more detail. While SAP offers a clean train split (i.e., without fraud) and defined validation and test splits, PaySim and Census are split randomly in 80% train, 10% validation, and 10% test subsets. We repeat each experiment 5 times for different random splits and report the mean and standard deviation of all results.

Experiment 1: Preprocessing

Both models have a number of parameters, which are depending on dataset, task, and preprocessing. While the task in this section is to generate realistic synthetic data (the AD task is evaluated in Section 8.3), our first experiment focuses on preprocessing for each dataset.

For numerical features, literature relies on different preprocessing choices. While [82] uses z-score normalization for numeric attributes, [85, 230, 57] use min-max normalization and [342] learns a Variational Gaussian Mixture Model to represent each value by a one-hot vector for the mode and a scalar for the value within each mode. As a first part of this experiment, we evaluate the number of modes for Variational Gaussian Mixture Model (VGM) and Gaussian Mixture Model (GMM) models, as a second part of this experiment, we compare the best Mixture Model to min-max and z-score normalization. We conduct each experiment for the VAE as well as the WGAN model.

For our experiments, we use a learning rate of 10^{-4} and train for up to 300 epochs. We use early stopping by comparing our model with JSD to a validation split to select the best model.

VGM/GMM Modes Our first experiment evaluates the number of mixtures considered for the (Variational) Gaussian Mixture Model (VGM / GMM) for each dataset and generative NN model. In this experiment, all categorical variables are encoded with one hot encoding, and the number of m GMM modes is varied by $m \in \{2, 3, 4, 5, 10, 15, 20\}$. We compare the distribution of the generated data to the three real-data splits, train, eval, and test, by JSD. Note that reducing the number of modes to one essentially results in fitting a single Gaussian distribution to the data and evaluating the difference to its mean, scaled by its standard-deviation, which is closely related to z-standardization and will be evaluated alongside the best VGM and GMM models and min-max scaling in the next experiment. Models with a larger number of modes generally yielded inferior results and were therefore omitted.

Table 6.2: JSD (see Eq. 6.30) between generated and real data with VGM and GMM preprocessing and a varying number of modes averaged over 5 runs ± standard deviation for the VAE. Smaller JSD values correspond to better models. Highlighted entries denote the most suitable number of modes for each dataset, which we select for further experiments. For equal averaged results, we select the best individual model.

		VGM			GMM		
Dataset	Modes	↓ JSD_{train}	↓ JSD_{eval}	↓ JSD_{test}	↓ JSD_{train}	↓ JSD_{eval}	↓ JSD_{test}
Census	2	0.18 ± 0.02	0.19 ± 0.02	0.20 ± 0.01	0.22 ± 0.02	0.22 ± 0.02	0.23 ± 0.02
	3	0.19 ± 0.01	0.19 ± 0.00	0.19 ± 0.01	0.23 ± 0.02	0.23 ± 0.01	0.24 ± 0.00
	4	0.17 ± 0.01	0.19 ± 0.00	0.19 ± 0.00	0.24 ± 0.01	0.24 ± 0.01	0.25 ± 0.01
	5	0.19 ± 0.02	0.19 ± 0.02	0.20 ± 0.01	0.25 ± 0.01	0.26 ± 0.00	0.27 ± 0.01
	10	0.20 ± 0.02	0.22 ± 0.02	0.21 ± 0.01	0.28 ± 0.01	0.28 ± 0.02	0.29 ± 0.02
	15	0.22 ± 0.01	0.22 ± 0.01	0.22 ± 0.01	0.29 ± 0.01	0.29 ± 0.02	0.29 ± 0.01
	20	0.23 ± 0.01	0.23 ± 0.01	0.24 ± 0.01	0.30 ± 0.01	0.30 ± 0.02	0.30 ± 0.02
PaySim	2	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00
	3	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00
	4	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.01	0.04 ± 0.01	0.04 ± 0.01
	5	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00
	10	0.04 ± 0.01	0.04 ± 0.01	0.04 ± 0.01	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00
	15	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00
	20	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.01	0.04 ± 0.01	0.04 ± 0.01
SAP	2	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.09 ± 0.01	0.01 ± 0.00	0.06 ± 0.00
	3	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.05 ± 0.01	0.01 ± 0.00	0.02 ± 0.00
	4	0.01 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00
	5	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00
	10	0.01 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00
	15	0.01 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00
	20	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00

↓: lower value is better

Results for the VAE Table 6.2 shows the results of this parameter study for the VAE and reports the mean and standard deviation over 5 runs per parameter configuration. For all three datasets, the train, eval, and test splits yield very similar results, indicating that the models did not overfit to the training data but capture the behavior present in all three splits equally well. To decide on the best model, we average on all splits. For the Census dataset, a VGM model with 4 modes shows the best results over all three splits, while a GMM model with only 2 modes outperforms all other configurations. For PaySim, 3 modes for VGM and 10 modes for GMM yielded the best results. While models with two modes performed slightly worse, all larger models yielded very similar results for VGM. For GMM, all models performed comparably. The SAP dataset also shows comparable performances for all VGM models. For GMM, the results indicate a preference for larger models, as 2- and 3-mode models yield notably worse results in comparison. However, the differences between larger models are negligible. Overall, the VGM models perform comparably or better for all datasets compared to the best GMM models. The dif-

ferences in the preferred number of modes is a result of the different procedures to fit each mixture, i.e., depending on the positioning and scaling of each mode, the generative model can utilize a different number of modes best. Nevertheless, as discussed, the tendency for more or fewer modes can be considered consistent.

Results for the WGAN Table 6.3 shows the results of this parameter study for the WGAN model. All in all, the observations from the VAE models repeat for our WGAN experiments, as a small number of modes for Census, a small to medium number of modes for PaySim and a larger number of modes for GMM yield the best results and VGM generally outperforms GMM. In detail, for the Census dataset, again 2 modes yield the best results for GMM preprocessing, while for VGM the preprocessing with 3 modes performs best. For PaySim, there is a more notable tendency towards a smaller number of modes from 2 to 5 modes for VGM and GMM all within the same performance range, with 3 modes for VGM and 4 modes for GMM performing best. The SAP dataset confirms the findings for the VAE model, showing a preference for 3 or more modes for GMM, while for VGM all models

Table 6.3: JSD between generated and real data with VGM and GMM preprocessing and a varying number of modes averaged over 5 runs ± standard deviation for the WGAN model. Highlighted entries denote the most suitable number of modes for each dataset.

		VGM			GMM		
Dataset	Modes	↓ JSD_{train}	↓ JSD_{eval}	↓ JSD_{test}	↓ JSD_{train}	↓ JSD_{eval}	↓ JSD_{test}
Census	2	0.15 ± 0.02	0.17 ± 0.03	0.16 ± 0.03	0.17 ± 0.02	0.18 ± 0.02	0.18 ± 0.03
	3	0.15 ± 0.02	0.16 ± 0.02	0.16 ± 0.02	0.17 ± 0.01	0.19 ± 0.03	0.18 ± 0.03
	4	0.15 ± 0.02	0.17 ± 0.01	0.16 ± 0.03	0.19 ± 0.02	0.20 ± 0.04	0.21 ± 0.03
	5	0.18 ± 0.02	0.18 ± 0.02	0.20 ± 0.04	0.18 ± 0.01	0.20 ± 0.02	0.20 ± 0.01
	10	0.15 ± 0.03	0.16 ± 0.02	0.17 ± 0.02	0.21 ± 0.02	0.21 ± 0.02	0.22 ± 0.02
	15	0.22 ± 0.04	0.23 ± 0.03	0.23 ± 0.04	0.25 ± 0.01	0.25 ± 0.01	0.25 ± 0.02
	20	0.25 ± 0.02	0.26 ± 0.04	0.26 ± 0.03	0.30 ± 0.01	0.30 ± 0.02	0.30 ± 0.02
PaySim	2	0.03 ± 0.01	0.04 ± 0.01	0.04 ± 0.01	0.04 ± 0.00	0.04 ± 0.00	0.04 ± 0.00
	3	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.04 ± 0.01	0.04 ± 0.01	0.04 ± 0.01
	4	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00
	5	0.03 ± 0.00	0.03 ± 0.01	0.03 ± 0.01	0.03 ± 0.00	0.04 ± 0.00	0.04 ± 0.00
	10	0.15 ± 0.06	0.15 ± 0.06	0.15 ± 0.06	0.06 ± 0.02	0.06 ± 0.02	0.06 ± 0.01
	15	0.11 ± 0.06	0.11 ± 0.07	0.11 ± 0.07	0.11 ± 0.04	0.11 ± 0.04	0.11 ± 0.04
	20	0.13 ± 0.03	0.13 ± 0.03	0.13 ± 0.04	0.10 ± 0.02	0.10 ± 0.02	0.10 ± 0.03
SAP	2	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.04 ± 0.02	0.01 ± 0.00	0.03 ± 0.01
	3	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.02 ± 0.01	0.01 ± 0.00	0.02 ± 0.01
	4	0.01 ± 0.01	0.01 ± 0.00	0.01 ± 0.01	0.02 ± 0.01	0.01 ± 0.00	0.02 ± 0.01
	5	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.03 ± 0.01	0.01 ± 0.00	0.03 ± 0.01
	10	0.01 ± 0.01	0.00 ± 0.00	0.01 ± 0.01	0.02 ± 0.01	0.01 ± 0.00	0.02 ± 0.01
	15	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.02 ± 0.01	0.00 ± 0.00	0.02 ± 0.01
	20	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.02 ± 0.00	0.01 ± 0.00	0.01 ± 0.00

↓: lower value is better

Table 6.4: Overview of the number of input dimensions after preprocessing for each dataset. *Num.* denotes the numerical preprocessor with the number of modes listed for VGM/GMM according to the best mode choice per model (cf. Tables 6.2 and 6.3).

Dataset	Num.	Modes (Model)	Input Dim.
Census	VGM	4 (VAE)	132
	GMM	2 (VAE)	120
	VGM	3 (WGAN)	126
	GMM	2 (WGAN)	120
	minmax	-	108
	z-score	-	108
PaySim	VGM	3 (VAE)	37
	GMM	10 (VAE)	93
	VGM	3 (WGAN)	37
	GMM	4 (WGAN)	45
	minmax	-	13
	z-score	-	13
SAP	VGM	10 (VAE)	296
	GMM	20 (VAE)	396
	VGM	15 (WGAN)	346
	GMM	20 (WGAN)	396
	minmax	-	196
	z-score	-	196

perform comparable. In general, the WGAN model yields better results than the VAE models, while the largest improvement is shown for the Census dataset.

Embeddings and scaling Besides numerical scaling, other preprocessing decisions in literature differ in several aspects and often lack evaluation: While in general, GANs and VAEs on tabular data both have the use of embedding layers in common, their approach of deciding on embedding layer size varies and needs further justification and evaluation for our datasets. For example, [85] incorporates embedding layers of size $d = \min(\lceil \frac{k}{3} \rceil, 20)$ for each categorical attribute with k unique feature values in their GAN model, while [82] assigns a fixed embedding dimensionality of 50 for each categorical attribute in their VAE experiments. Note that for several features in our datasets, an embedding size of 50 per feature effectively means an increase in dimensionality, since the number of unique values for some categorical features is very low, which might contradict the purpose of embedding as dense dimension reductions. For datasets with a reasonable number of categorical feature values per feature, directly representing each feature by its one-hot encoding becomes feasible. Therefore, we include the use of one-hot encoding without any embedding as an additional preprocessing variant.

Table 6.5: Comparison between generated and real data with all preprocessing combinations and a varying embedding size d_{emb} averaged over 5 runs ± standard deviation for the VAE. Highlighted entries denote the most suitable number of modes for each dataset..

		Census			PaySim			SAP		
Num	d_{emb}	↓ JSD	↓ FID	↑ Corr	↓ JSD	↓ FID	↑ Corr	↓ JSD	↓ FID	↑ Corr
GMM	10	0.22 ± 0.02	0.49 ± 0.05	0.03 ± 0.01	0.03 ± 0.00	0.02 ± 0.00	0.06 ± 0.01	0.01 ± 0.00	1.28 ± 0.05	0.10 ± 0.00
	20	0.23 ± 0.01	0.42 ± 0.02	0.03 ± 0.00	0.03 ± 0.00	0.02 ± 0.01	0.06 ± 0.00	0.01 ± 0.00	1.23 ± 0.10	0.10 ± 0.01
	50	0.22 ± 0.02	0.42 ± 0.04	0.03 ± 0.00	0.03 ± 0.00	0.02 ± 0.01	0.06 ± 0.00	0.01 ± 0.00	1.28 ± 0.12	0.11 ± 0.00
	None	0.22 ± 0.02	0.43 ± 0.02	0.03 ± 0.00	0.03 ± 0.00	0.02 ± 0.01	0.06 ± 0.00	0.01 ± 0.00	1.24 ± 0.09	0.11 ± 0.00
	auto	0.23 ± 0.02	0.54 ± 0.02	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.01	0.06 ± 0.01	0.01 ± 0.00	1.25 ± 0.09	0.10 ± 0.00
minmax	10	0.33 ± 0.03	0.56 ± 0.06	0.10 ± 0.01	0.17 ± 0.01	0.03 ± 0.00	0.12 ± 0.01	0.21 ± 0.04	1.03 ± 0.15	0.09 ± 0.01
	20	0.33 ± 0.02	0.54 ± 0.02	0.10 ± 0.00	0.17 ± 0.01	0.04 ± 0.01	0.12 ± 0.01	0.25 ± 0.03	1.05 ± 0.08	0.09 ± 0.01
	50	0.33 ± 0.02	0.50 ± 0.02	0.10 ± 0.00	0.17 ± 0.00	0.04 ± 0.01	0.13 ± 0.00	0.25 ± 0.03	1.02 ± 0.10	0.09 ± 0.01
	None	0.33 ± 0.02	0.53 ± 0.03	0.10 ± 0.01	0.18 ± 0.01	0.04 ± 0.00	0.12 ± 0.00	0.22 ± 0.02	0.82 ± 0.07	0.08 ± 0.01
	auto	0.33 ± 0.02	0.69 ± 0.03	0.11 ± 0.01	0.17 ± 0.01	0.04 ± 0.00	0.12 ± 0.01	0.22 ± 0.02	1.04 ± 0.10	0.09 ± 0.01
z-score	10	0.21 ± 0.01	0.45 ± 0.03	0.04 ± 0.00	0.05 ± 0.01	0.01 ± 0.00	0.05 ± 0.00	0.10 ± 0.00	1.08 ± 0.09	0.08 ± 0.00
	20	0.20 ± 0.01	0.40 ± 0.04	0.04 ± 0.00	0.05 ± 0.01	0.02 ± 0.01	0.05 ± 0.00	0.10 ± 0.01	1.19 ± 0.13	0.08 ± 0.00
	50	0.21 ± 0.01	0.38 ± 0.04	0.04 ± 0.00	0.05 ± 0.00	0.01 ± 0.00	0.05 ± 0.00	0.10 ± 0.00	1.04 ± 0.08	0.08 ± 0.00
	None	0.20 ± 0.02	0.41 ± 0.01	0.04 ± 0.00	0.05 ± 0.01	0.01 ± 0.00	0.05 ± 0.00	0.10 ± 0.00	0.96 ± 0.07	0.08 ± 0.00
	auto	0.21 ± 0.01	0.57 ± 0.07	0.04 ± 0.00	0.05 ± 0.01	0.02 ± 0.00	0.05 ± 0.01	0.10 ± 0.01	1.11 ± 0.09	0.08 ± 0.00
VGM	10	0.19 ± 0.01	0.50 ± 0.05	0.04 ± 0.00	0.04 ± 0.00	0.02 ± 0.00	0.05 ± 0.01	0.00 ± 0.00	1.32 ± 0.08	0.10 ± 0.00
	20	0.19 ± 0.01	0.48 ± 0.03	0.03 ± 0.00	0.03 ± 0.00	0.02 ± 0.00	0.05 ± 0.00	0.00 ± 0.00	1.26 ± 0.11	0.10 ± 0.00
	50	0.20 ± 0.01	0.44 ± 0.04	0.03 ± 0.00	0.04 ± 0.00	0.02 ± 0.00	0.05 ± 0.00	0.00 ± 0.00	1.21 ± 0.07	0.10 ± 0.00
	None	0.19 ± 0.01	0.46 ± 0.04	0.03 ± 0.00	0.03 ± 0.00	0.02 ± 0.00	0.05 ± 0.00	0.00 ± 0.00	1.20 ± 0.05	0.10 ± 0.00
	auto	0.19 ± 0.01	0.63 ± 0.05	0.04 ± 0.00	0.04 ± 0.00	0.03 ± 0.01	0.05 ± 0.00	0.00 ± 0.00	1.35 ± 0.18	0.10 ± 0.01

↑ / ↓: higher / lower value is better

In the following experiment, we evaluate all preprocessing choices in combination with both models on all three datasets for categorical and numeric features, i.e., one-hot encoded embeddings with a fixed dimensionality $d \in \{10, 20, 50\}$ and a variable dimensionality per feature by $d = \min(\lceil \frac{k}{3} \rceil, 20)$ for k unique feature values referred to as *auto* and min-max normalization, z-score scaling, and VGM/GMM as evaluated in Table 6.2 for numerical features. The input size after each preprocessing step is shown in Table 6.4.

Results for the VAE Table 6.5 shows the results of this experiment for the VAE. Again, each result reported is the mean and standard deviation over 5 repetitions. When comparing different numeric preprocessing approaches, for all datasets, VGM yields the most promising results with varying performance gap across the datasets: For Census, min-max is the worst performing approach with a JSD of 0.33 compared to the best (VGM) of 0.19, while for PaySim, GMM and VGM yield the best performance with a JSD of 0.03 compared to the worst (min-max) of 0.17. For SAP, the results are similar to Census, VGM performs best with a JSD of 0.0, however, the second-best model, GMM performs only slightly worse (0.01). The standard deviation in general is low, which means that the models perform consistently over several iterations with the largest scattering for min-max (0.04 at maximum). Over-

all, the performance differences between all numeric preprocessing approaches are statistically significant[2].

When it comes to embedding dimensionality d_{emb}, the results differ only slightly, and the one-hot encoding (i.e., $d_{\text{emb}} = \text{None}$) performs comparably. The difference between the different fixed dimensionalities is neglectable and does not show a clear pattern significantly favoring one size over the others. For Census and PaySim, VGM without embedding and for SAP, VGM with an embedding size of 50 yield the best results, which we select for our next experiment respectively.

Note that the minimal difference between one-hot encoding and all embedding variants suggests that learning per-feature independent embeddings from scratch alongside the task is not beneficial for VAE. In Chapter 7, we therefore focus on approaches to learn meaningful representations for categorical features from a different perspective by formulating a context-based prediction task known from the NLP domain as word embeddings, such as Word2Vec [208] and GloVe [233].

In Table 6.5, besides JSD, FID and Corr evaluation measurements are also included. For Census and PaySim, the results for JSD, FID, and Corr correspond quite well with z-score, GMM and VGM yielding good results and min-max scaling performing worst. For SAP, however, the results are mixed when looking at the influence of different embedding sizes: For VGM the best FID is achieved without embeddings while for z-score normalization an embedding size of 10 performs comparable. The correlation does not show a clear pattern over different peprocessing and embedding parameters with several configurations that achieve equally good correlation errors of ≈ 0.09.

Results for the WGAN The results for the WGAN model regarding the scaling of numerical features confirm the findings from the VAE model, as min-max scaling yields the worst results for all datasets. GMM and VGM perform comparable while the best results for PaySim and SAP are achieved with VGM and a dimensionality of $d = \min(\lceil \frac{k}{3} \rceil, 20)$ referred to as *auto*. For Census, z-score normalization with an embedding size of 10 performs best. However, with respect to the benefit and optimal size of the embeddings, the results for WGAN differ from the VAE results, as using embeddings in contrast to one-hot encoding categorical attributes yield slightly better results with a tendency of smaller embedding sizes over all models and datasets.

While the correlation evaluation suggests only minor differences for Census over the scaling and embedding choices, for PaySim min-max yields notably worse results for the precise modeling of feature correlations. For SAP VGM, GMM and z-score normalization capture correlations comparably well.

For our next experiments, we select the best parameter configuration of this experiment, i.e, with the GAN model z-score normalization and embeddings with

[2]Mann-Whitney U Test, $p \leq 0.01$

Table 6.6: Comparison between generated and real data with all preprocessing combinations and a varying embedding size d_{emb} averaged over 5 runs ± standard deviation for the WGAN model. Highlighted entries denote the most suitable number of modes for each dataset.

		Census			PaySim			SAP		
Num	d_{emb}	↓ JSD	↓ FID	↑ Corr	↓ JSD	↓ FID	↑ Corr	↓ JSD	↓ FID	↑ Corr
GMM	10	0.17 ± 0.02	0.17 ± 0.01	0.02 ± 0.00	0.03 ± 0.00	0.01 ± 0.00	0.03 ± 0.00	0.02 ± 0.01	1.19 ± 0.23	0.10 ± 0.01
	20	0.17 ± 0.02	0.21 ± 0.05	0.02 ± 0.00	0.03 ± 0.00	0.01 ± 0.00	0.03 ± 0.00	0.01 ± 0.00	1.14 ± 0.14	0.09 ± 0.01
	50	0.18 ± 0.02	0.17 ± 0.02	0.02 ± 0.00	0.04 ± 0.00	0.01 ± 0.00	0.03 ± 0.00	0.01 ± 0.01	1.14 ± 0.08	0.10 ± 0.01
	None	0.18 ± 0.02	0.22 ± 0.02	0.03 ± 0.01	0.03 ± 0.00	0.01 ± 0.00	0.04 ± 0.01	0.01 ± 0.01	1.13 ± 0.18	0.10 ± 0.01
	auto	0.18 ± 0.03	0.33 ± 0.03	0.03 ± 0.00	0.04 ± 0.00	0.02 ± 0.00	0.04 ± 0.00	0.01 ± 0.00	1.18 ± 0.18	0.10 ± 0.01
minmax	10	0.21 ± 0.08	0.17 ± 0.04	0.05 ± 0.02	0.20 ± 0.20	0.21 ± 0.59	0.17 ± 0.10	0.14 ± 0.17	1.25 ± 0.17	0.13 ± 0.02
	20	0.32 ± 0.05	0.24 ± 0.05	0.03 ± 0.00	0.22 ± 0.04	0.07 ± 0.05	0.16 ± 0.05	0.40 ± 0.04	1.26 ± 0.17	0.11 ± 0.01
	50	0.32 ± 0.05	0.26 ± 0.03	0.03 ± 0.00	0.33 ± 0.14	0.30 ± 0.53	0.19 ± 0.06	0.42 ± 0.06	1.26 ± 0.22	0.12 ± 0.00
	None	0.31 ± 0.05	0.20 ± 0.02	0.03 ± 0.00	0.30 ± 0.12	0.23 ± 0.46	0.17 ± 0.06	0.40 ± 0.04	1.45 ± 0.27	0.12 ± 0.01
	auto	0.30 ± 0.04	0.33 ± 0.04	0.03 ± 0.01	0.34 ± 0.21	0.53 ± 0.69	0.27 ± 0.10	0.31 ± 0.03	1.61 ± 0.66	0.12 ± 0.02
z-score	10	0.14 ± 0.02	0.18 ± 0.02	0.02 ± 0.00	0.04 ± 0.01	0.01 ± 0.01	0.02 ± 0.01	0.06 ± 0.01	1.06 ± 0.24	0.09 ± 0.02
	20	0.17 ± 0.01	0.24 ± 0.04	0.02 ± 0.01	0.04 ± 0.01	0.01 ± 0.01	0.02 ± 0.01	0.08 ± 0.01	1.12 ± 0.17	0.09 ± 0.01
	50	0.16 ± 0.02	0.22 ± 0.03	0.02 ± 0.00	0.04 ± 0.01	0.01 ± 0.00	0.02 ± 0.01	0.09 ± 0.03	1.23 ± 0.10	0.09 ± 0.01
	None	0.19 ± 0.04	0.20 ± 0.02	0.02 ± 0.00	0.04 ± 0.01	0.01 ± 0.01	0.02 ± 0.01	0.09 ± 0.02	1.28 ± 0.15	0.10 ± 0.02
	auto	0.17 ± 0.03	0.33 ± 0.07	0.03 ± 0.00	0.04 ± 0.01	0.01 ± 0.00	0.02 ± 0.01	0.05 ± 0.01	1.31 ± 0.12	0.11 ± 0.01
VGM	10	0.15 ± 0.03	0.19 ± 0.03	0.03 ± 0.01	0.03 ± 0.01	0.01 ± 0.00	0.02 ± 0.00	0.01 ± 0.00	1.10 ± 0.14	0.10 ± 0.01
	20	0.16 ± 0.02	0.18 ± 0.01	0.02 ± 0.00	0.03 ± 0.00	0.01 ± 0.00	0.02 ± 0.00	0.01 ± 0.00	1.10 ± 0.14	0.10 ± 0.02
	50	0.17 ± 0.00	0.20 ± 0.02	0.03 ± 0.01	0.03 ± 0.00	0.01 ± 0.00	0.03 ± 0.00	0.01 ± 0.00	1.16 ± 0.21	0.10 ± 0.02
	None	0.15 ± 0.02	0.17 ± 0.02	0.03 ± 0.00	0.03 ± 0.00	0.01 ± 0.00	0.03 ± 0.00	0.01 ± 0.00	1.04 ± 0.13	0.11 ± 0.01
	auto	0.16 ± 0.01	0.31 ± 0.03	0.03 ± 0.00	0.03 ± 0.00	0.01 ± 0.00	0.02 ± 0.00	0.01 ± 0.00	1.22 ± 0.26	0.09 ± 0.01

↑ / ↓: higher / lower value is better

$d_{\textbf{emb}} = 10$ for the Census dataset and VGM with $d_{\textbf{emb}} = \textit{auto}$ for PaySim and SAP. The best VAE and WGAN models are compared in the following experiment (cf. Table 6.7) along with the evaluation of additional iNALU and Cross Layers.

Experiment 2: Evaluation of iNALU and Cross Layers

In this experiment, we additionally introduce Cross layers and iNALU layers and vary the respective number of layers to examine whether a specific representation of dependencies has a beneficial influence on generative modeling quality in general, including our contributions from RQ 1.1. We therefore select the best models for VAE and WGAN respectively based on the results of the previous experiments and introduce our architectural changes regarding iNALU and Cross layers to each model.

Results for the VAE In Table 6.7 the results for including an iNALU, a Cross layer, or both are depicted. For all datasets, the result does not benefit from introducing additional layers. While the influence of Cross layer alone is not significant[3], incorporating an iNALU layer impairs the results significantly[4]. For further iNALU layers or Cross layers, the model has stability issues and does not converge consistently.

[3]Mann-Whitney U Test, $p = 0.425$
[4]Mann-Whitney U Test, $p \leq 0.01$

Table 6.7: Comparison between generated and real data for both models (best configuration from Experiment 1 per dataset) with iNALU and Cross layers.

			Census			PaySim			SAP		
Model	Cross	iNALU	↓ JSD	↓ FID	↑ Corr	↓ JSD	↓ FID	↑ Corr	↓ JSD	↓ FID	↑ Corr
VAE	0	0	0.19 ± 0.01	0.45 ± 0.06	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.01	0.05 ± 0.00	0.01 ± 0.00	1.19 ± 0.09	0.09 ± 0.00
	0	1	0.34 ± 0.01	0.66 ± 0.08	0.06 ± 0.01	0.10 ± 0.02	0.11 ± 0.05	0.08 ± 0.02	0.01 ± 0.00	1.70 ± 0.12	0.11 ± 0.00
	1	0	0.19 ± 0.03	0.46 ± 0.12	0.04 ± 0.01	0.04 ± 0.01	0.02 ± 0.01	0.04 ± 0.00	0.01 ± 0.00	1.14 ± 0.12	0.09 ± 0.00
	1	1	0.34 ± 0.01	1.72 ± 2.05	0.10 ± 0.08	0.10 ± 0.03	0.08 ± 0.03	0.10 ± 0.05	0.01 ± 0.00	1.50 ± 0.13	0.11 ± 0.01
WGAN	0	0	0.14 ± 0.02	0.18 ± 0.02	0.02 ± 0.00	0.03 ± 0.00	0.01 ± 0.00	0.02 ± 0.00	0.01 ± 0.00	1.22 ± 0.26	0.09 ± 0.01
	0	1	0.07 ± 0.01	0.18 ± 0.02	0.05 ± 0.03	0.02 ± 0.00	0.01 ± 0.01	0.04 ± 0.02	0.01 ± 0.01	1.40 ± 0.38	0.14 ± 0.02
	0	2	0.08 ± 0.03	0.18 ± 0.01	0.04 ± 0.01	0.12 ± 0.14	0.60 ± 0.82	0.30 ± 0.39	0.04 ± 0.06	10.44 ± 20.38	0.15 ± 0.08
	1	0	0.12 ± 0.01	0.19 ± 0.01	0.02 ± 0.00	0.02 ± 0.00	0.00 ± 0.00	0.02 ± 0.00	0.17 ± 0.03	1.91 ± 0.38	0.12 ± 0.03
	1	1	0.10 ± 0.03	0.26 ± 0.04	0.03 ± 0.01	0.02 ± 0.01	0.01 ± 0.01	0.04 ± 0.01	0.01 ± 0.01	1.49 ± 0.20	0.15 ± 0.01
	1	2	0.09 ± 0.01	0.38 ± 0.21	0.06 ± 0.03	0.08 ± 0.08	0.78 ± 0.93	0.38 ± 0.41	0.07 ± 0.13	10.92 ± 21.35	0.16 ± 0.08
	2	0	0.12 ± 0.01	0.19 ± 0.02	0.02 ± 0.00	0.02 ± 0.00	0.00 ± 0.00	0.02 ± 0.00	0.12 ± 0.02	1.47 ± 0.28	0.12 ± 0.03
	2	1	0.10 ± 0.02	0.26 ± 0.05	0.05 ± 0.02	0.02 ± 0.01	0.00 ± 0.00	0.12 ± 0.17	0.02 ± 0.04	1.41 ± 0.23	0.14 ± 0.01
	2	2	0.07 ± 0.05	0.27 ± 0.17	0.07 ± 0.06	0.03 ± 0.03	0.29 ± 0.50	0.28 ± 0.43	0.01 ± 0.01	1.40 ± 0.26	0.13 ± 0.01

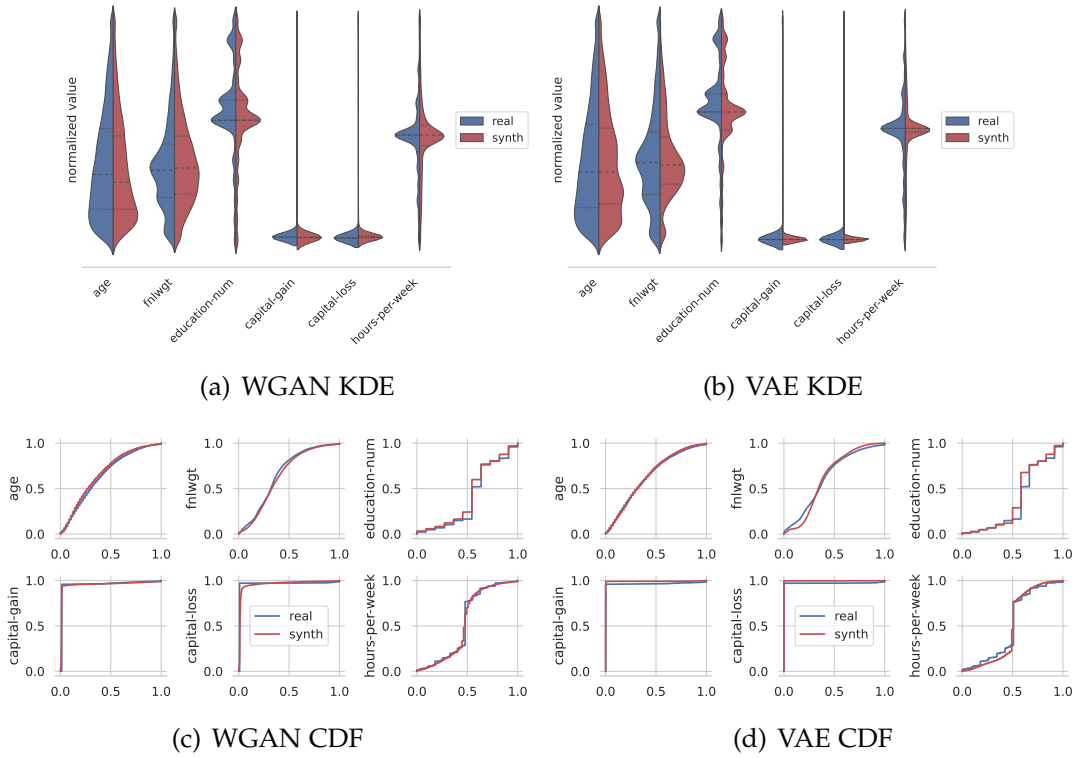
↑ / ↓: higher / lower value is better

Results for the WGAN Table 6.7 summarizes the results for a number of Cross and iNALU layers between 0 and 2 for the WGAN model. For the Census dataset, introducing additional specialized layers improves the JSD by large margin[4]. Without iNALU layers, the results for 1 and 2 Cross layers improve slightly, while adding one iNALU layer yields a large performance gain achieving the best JSD without Cross layers and with one iNALU layer. While the FID supports this result, the feature correlation error Corr slightly impairs. For PaySim, introducing a Cross and iNALU layer both slightly improves the performance, while two iNALU layers impair the performance notably. The best performance is achieved using only one or two Cross layers, which perform equally well. For the SAP dataset, introducing additional layers does not further improve the performance. While configurations with one iNALU layer and one Cross layer or two iNALU and two Cross layers perform similar for JSD, FID and Corr metrics show the best results without any additional iNALU and Cross layers. These results suggest that, depending on the dataset, specialized layers for numeric dependencies or feature crossings can support the performance of the WGAN model. However, this does not hold for all applications where such dependencies are expected in the dataset, as shown for the SAP dataset.

Generative Performance Evaluation

As we evaluated the best preprocessing parameters and specific architectural choices in the previous experiments, we now focus on a thorough evaluation and comparison of the generative performance of the best VAE and WGAN models for the three datasets. For RQ 1.2, the question of modeling distributions, we exploratively evaluate data generated by our models side by side using KDE plots and Cumulative Distribution Functions (CDFs). To address RQ 1.3, we compare feature correlation plots for both models in parallel to answer which model is able to reproduce feature correlations from the original data best.

Census Fig. 6.12 visualizes the numeric features of the Census dataset. Both models, WGAN and VAE, model the characteristics of numerical attributes quite well. While the VAE models the *age* feature sightly more precisely, the *fnlwgt* feature shows a notable underestimation in the low value range of the distribution. The discrete features *age* and *education-num* are modeled by both approaches discretely (visible as steps in the CDF plots). The WGAN approach tends to underestimate the sharpness of long-tailed distributions such as *capital-gain/loss*, whereas the prediction of the VAE model mostly collapses to the high density value range. Note that for this evaluation the most extreme 1% quantiles were already clipped during normalization, i.e., the infrequent values in the 'tail' account for more than 1% of the data. The *hours-per-week* feature, which contains notable steps but is not as discrete as *education-num*, is not well modeled by both approaches. While the WGAN overall fits the distribution better than VAE, the steps are mostly smoothed out for both models.

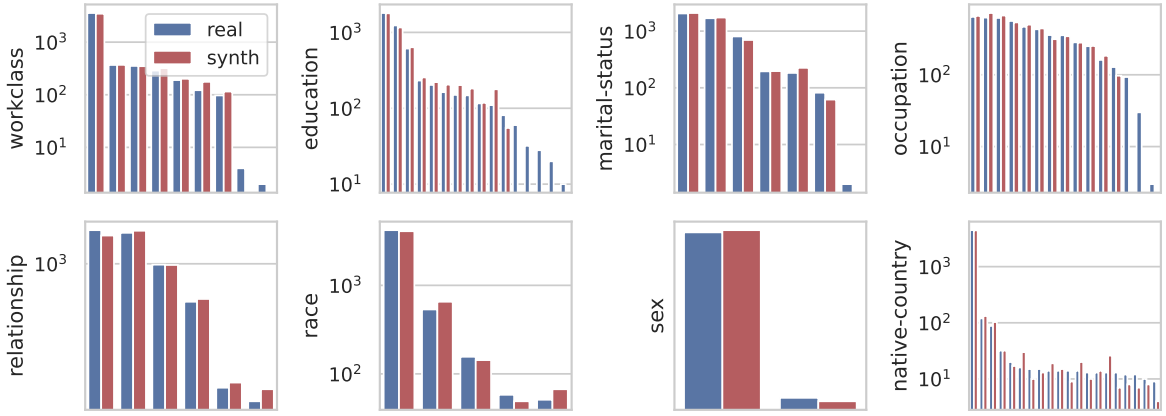


(a) WGAN KDE (b) VAE KDE

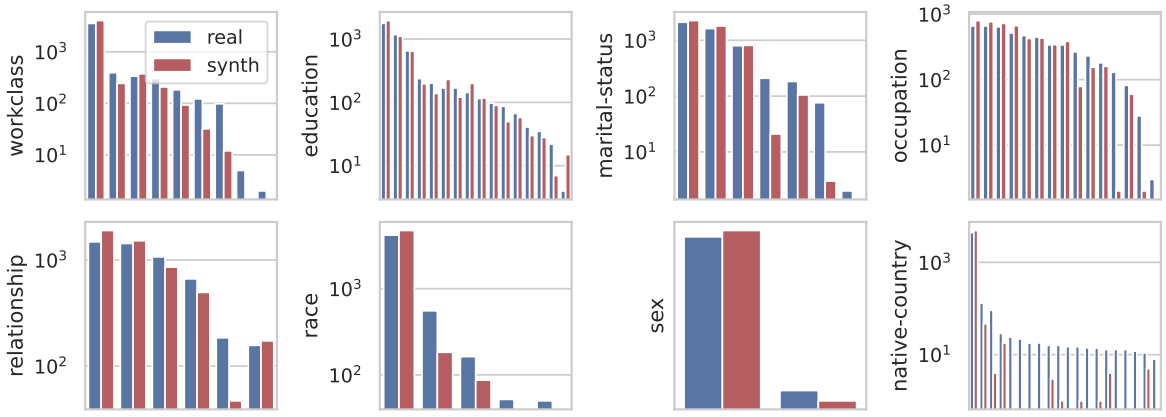
(c) WGAN CDF (d) VAE CDF

Figure 6.12: Numeric features of the Census dataset (blue) in comparison to generated synthetic data from WGAN (a), (c) and VAE (b), (d) by kernel density estimation (a), (b) and CDF (c), (d).

In Fig. 6.13 the frequency of categorical feature values for real and generated data is depicted in log scale. Overall, the WGAN mimics the real frequency more precisely up to a certain frequency threshold for feature values, which then do not occur in the generated data, visible in the upper row of features. In contrast, VAE has a larger discrepancy for frequent feature values and generates the most infrequent values for some features (see, for example, *education*), however, fails for others (*race*, *native-country*).

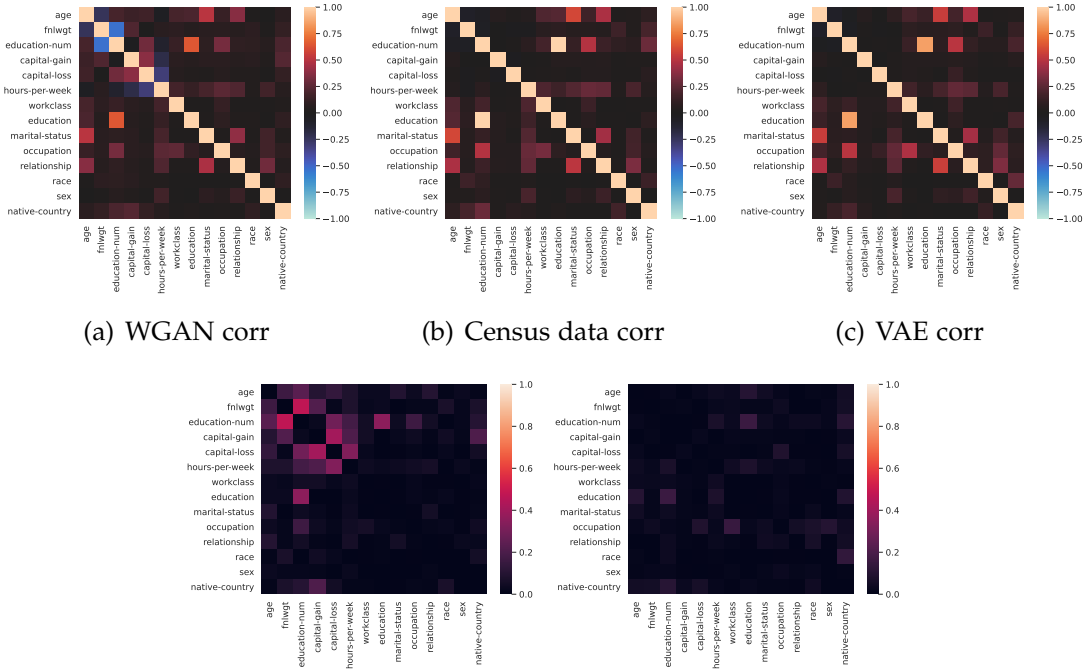


(a) WGAN



(b) VAE

Figure 6.13: Categorical features of Census dataset (blue) in comparison to generated synthetic data from WGAN (a) and VAE (b) in logarithmic scale.



(a) WGAN corr (b) Census data corr (c) VAE corr

(d) WGAN $\overline{\text{corr}}$ (e) VAE $\overline{\text{corr}}$

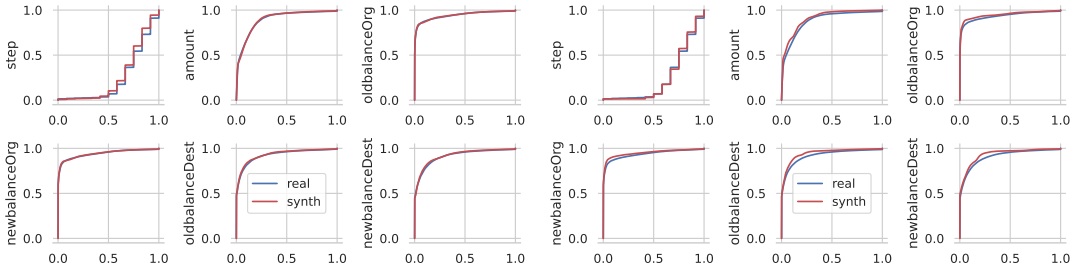
Figure 6.14: Feature correlations for WGAN (a) and VAE (c) generated and real (b) Census data in comparison. Absolute differences $\overline{\text{corr}}$ between data correlations and WGAN (d) and VAE (e) generated correlations.

Fig. 6.14 shows the feature correlations for the data generated by WGAN and VAE compared to the correlations of features of the real dataset. Both approaches model feature correlations well, which are present in the real dataset. The VAE almost perfectly reproduces the correlations with only small differences in the most correlated features. The WGAN model has difficulties with the numeric attributes *fnlwgt* and *education-num*, which are associated with a strong negative correlation, although no correlation is present in the real dataset. This also becomes apparent for the plot of absolute differences between generated and real data in the lower row, where the VAE is almost without differences, whereas WGAN has notable errors primarily among the numeric attributes.

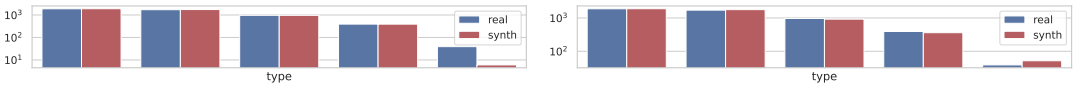
To summarize, for the Census dataset both models model numeric and categorical feature distributions as well as feature correlations well. While WGAN is more precise regarding numerical features and infrequent categorical features, VAE is more precise replicating feature correlations.

PaySim The results for PaySim are similar to Census, as both approaches represent the distributions and dependencies well. Fig. 6.15 shows the results for the features for WGAN and VAE. VAE more accurately models the discrete *step* feature, whereas the other numeric features are modeled by WGAN more precisely. The categorical feature counts of the only categorical variable *type* (see Fig. 6.15(c) and (d)) are well modeled for the most frequent values, while both models struggle to generate the most infrequent value according to the real frequency.

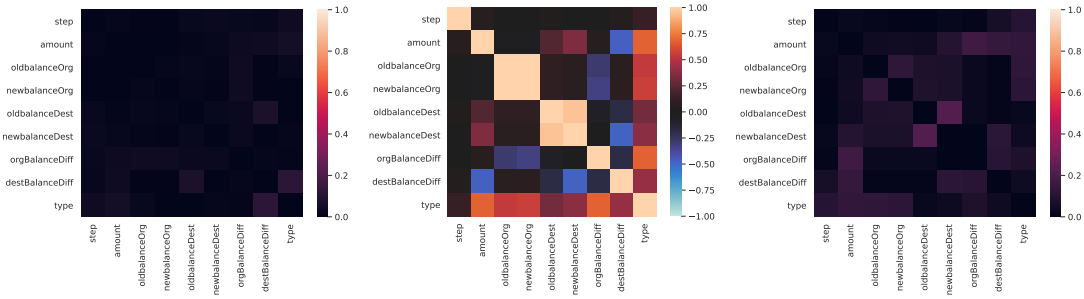
The feature correlations present in real data are accounted for by both models. Fig. 6.15(f) shows the correlations between all the features of the real data, while (e) and (g) depict the absolute error between the correlations of the generated data and the correlations of the dataset. The WGAN model is slightly more precise for correlations within the synthesized PaySim data.



(a) WGAN CDF (b) VAE CDF



(c) WGAN log-counts (d) VAE log-counts

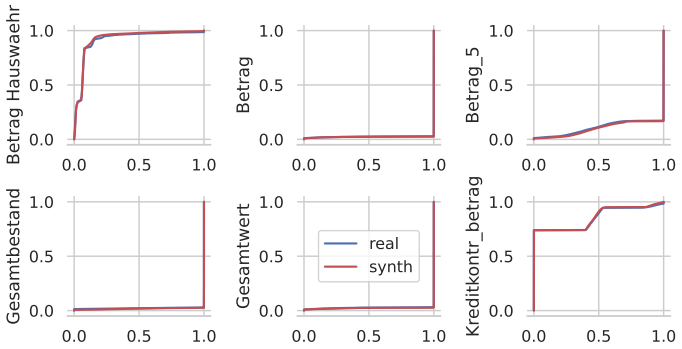


(e) WGAN $\overline{\text{corr}}$ (f) PaySim data corr (g) VAE $\overline{\text{corr}}$

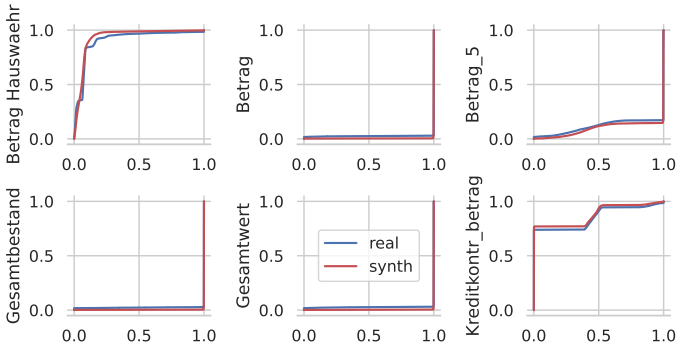
Figure 6.15: Features of PaySim dataset (blue) in comparison to generated synthetic data from WGAN (a,c) and VAE (b,d) by CDF (a), (b), log-counts (c), (d) and abs. correlation error $\overline{\text{corr}}$ (e), (g).

SAP For SAP, both models perform comparably well. Figs. 6.16 and 6.17 depict a selection of the 10 numeric and 42 categorical features of the generated data by both models in relation to real data. Features that are not depicted show similar characteristics. While the numeric features are modeled very accurately by WGAN, VAE does not predict values as precise outside the high density areas (i.e., 'long-tail' values) as shown for *Betrag Hauswaehr* (see Fig. 6.16 (b)). For categorical features, both models yield good results for frequent feature values, while underestimating infrequent feature values up to lost values for very infrequent values as for WGAN and *Alternative Kontonummer* (see Fig. 6.17(a)). As the feature value counts are log-scaled, some values in the real dataset are very rare (up to one occurrence) in comparison to frequent values (>1 000 occurrences) in the dataset.

The analysis of feature correlations gives a differentiated view: As WGAN models the majority of all features very well, some features (represented by rows and columns in the plot) have high absolute errors for correlations with other features. In contrast, VAE has overall higher modeling errors but does not show consistently very high errors for single features.

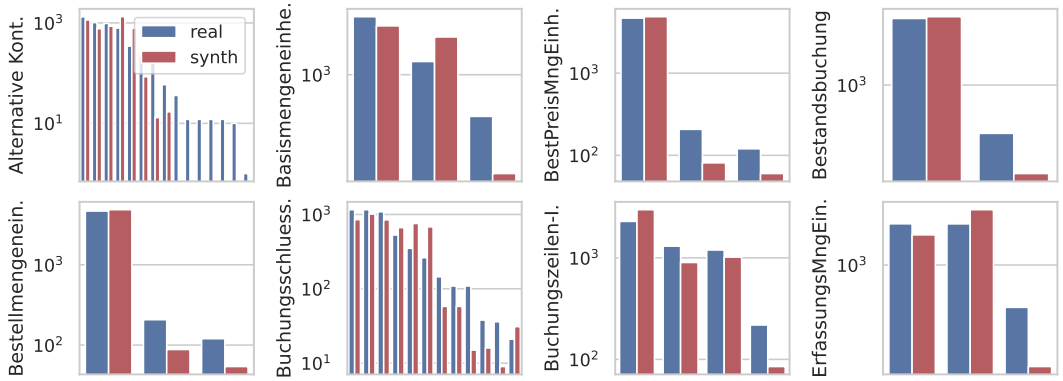


(a) WGAN CDF

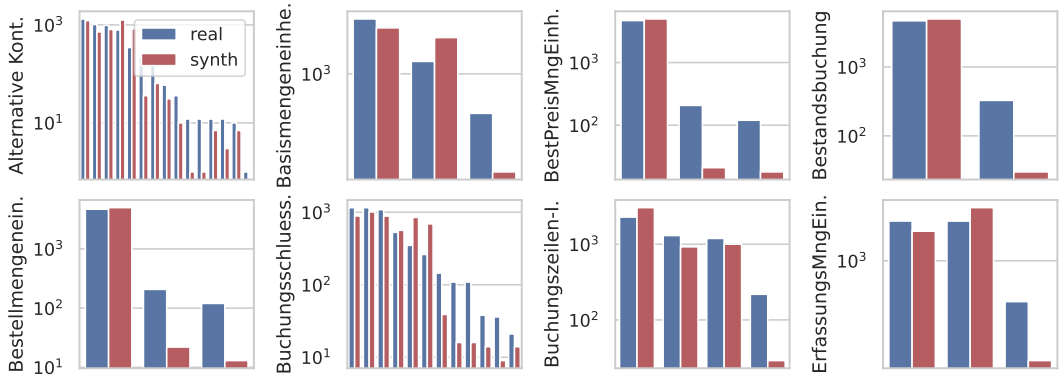


(b) VAE CDF

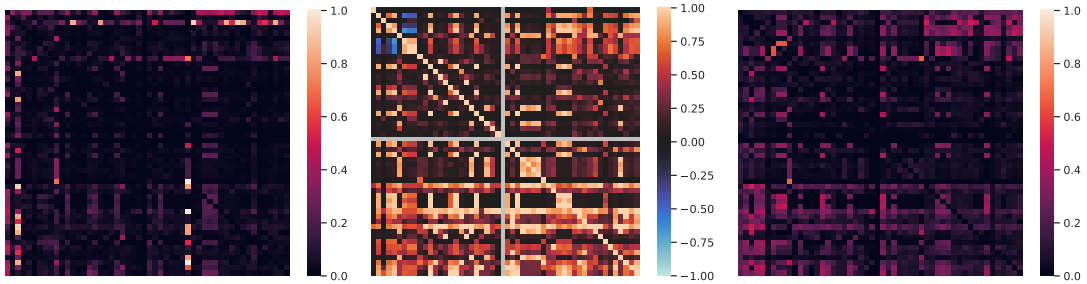
Figure 6.16: Selected features of SAP dataset (blue) in comparison to generated synthetic data from WGAN and VAE by CDF.



(a) WGAN log-counts



(b) VAE log-counts



(c) WGAN $\overline{\text{corr}}$ (d) SAP data corr (e) VAE $\overline{\text{corr}}$

Figure 6.17: Selected features of SAP dataset (blue) in comparison to generated synthetic data from WGAN and VAE by log-counts (a, b) and abs. correlation error $\overline{\text{corr}}$ (c, e).

6.2.4 Conclusion

In this section, we adapted two generative neural network models, the Variational Auto-Encoder and Wasserstein Generative Adversarial Network, modeling distributions to synthesize transaction data. We first evaluated several choices to represent numerical and categorical features in favor of precisely learning feature distributions and correlations by different scaling approaches for numeric and embedding sizes for categorical features. We found Variational Gaussian Mixture models beneficial for both models and most datasets with the exception of VAE for Census, which yielded best results with z-score normalization, and WGAN for PaySim, for which Gaussian Mixture Models performed slightly better. Contrarily to scaling, the embedding parameters did not show a large influence on generative performance for both approaches. No parameter configuration consistently outperformed the others.

We incorporated iNALU and Cross Layers to explicitly model categorical and numerical feature dependencies. For the VAE, Cross Layers had no considerable influence, while iNALU layers impaired the performance. For the WGAN model, incorporating an iNALU layer notably improved the results for Census while performing comparably on the other datasets. Cross Layers improved the results only slightly for the WGAN model with no additional benefit when mixed with iNALU layers.

In the qualitative analysis of the best models, we found that both VAE and WGAN effectively capture dataset distributions and dependencies and are able to synthesize a dataset following the characteristics of the underlying real dataset trained on. For both datasets with low feature count, Census and PaySim, the models accurately account for characteristics of numeric and categorical variables as well as feature correlations. For SAP, the large number of categorical features are modeled less precisely by both approaches while still preserving high accuracy for numerical variables. To summarize, the WGAN model tends to account for numeric distributions more precisely, while the VAE is less prone to missing feature values, typically referred to as mode collapse, which is often discussed as disadvantage of GAN-based models in general. Including specific layers for modeling categorical and numerical dependencies can be beneficial depending on the dataset and model.

6.3 Summary

In this chapter, we focused on modeling of distributions and dependencies for transaction data with NNs. We first proposed a specialized neural architecture, the improved Neural Arithmetic Logic Unit to model mathematical relations and dependencies more precisely. We evaluated this architecture on several synthetic datasets and showed its ability to model mathematical relations of various complexity implicitly. Therefore, we can answer the research question RQ 1.1, to what extent extrapolation of numerical dependencies can be improved by our neural architecture as follows:

> Although not all stability issues and the precise modeling of the division operator have been solved, our proposed iNALU model generally improved the extrapolation of numerical dependencies regarding precision and stability in comparison to the previous state of the art.

Secondly, we adapted two generative neural architectures based on Generative Adversarial Networks and Variational Auto-Encoder to evaluate their ability of modeling distributions within data precisely. In this chapter, we focused on generating synthetic data with real-world transaction data characteristics to evaluate the capabilities of the model, whereas an evaluation as Anomaly Detectors will be presented in Section 8.3. We incorporated our iNALU architecture in both models and found that WGAN benefits from our specialized architecture. Therefore, we can answer the research questions RQ 1.2 and RQ 1.3 on how well generative neural networks model feature distributions and feature correlations:

> For modeling distributions and correlations, both models studied are able to capture both aspects very precisely on datasets we evaluated.

In this chapter, we therefore proposed and evaluated several neural network models and architectures that are able to capture the characteristics of transaction data well, which will be further investigated in the context of Anomaly Detection in Chapter 8.

Chapter 7

Representation Learning for Transaction Data

Transactions such as journal entries in ERP systems or system logs used for computer security contain comprehensive information, which can be used to detect fraudulent actions or attacks within the system. These data sources typically follow various levels of structuring ranging from relational databases to semi-structured text files and introduce various data types, often mixed on per-sample basis, such as timestamps, amounts and counts, categorical values, keys, indices, and text.

To build a Machine Learning system that is capable of classifying transactions as benign or attack, one major challenge is to choose the way *how* heterogeneous data can be appropriately represented when it comes to different data types and levels of structuring. Many Machine Learning approaches and especially Neural Networks demand numeric values as the basic input representation. Among possible data types some of them such as amounts and counts are naturally numeric values, which are suitable to serve as input for a Neural Network. Other data types can be transformed into a numeric representation in several ways, possibly highlighting different aspects of a data point. Consider for example, a feature containing date and time of a transaction. The feature can be represented, e.g., as (Unix) timestamp, i.e. the seconds counted since January 1st, 1970. This representation meets the requirements to be a numeric value and accurately represents the actual date and time of the data point. However, humanly derivable information from the originating data point might become less obvious, for example, the time of the day, which might be useful for the task when precisely searching for transactions that took place outside typical office hours. In *classical* Data Mining approaches, this is reflected in the feature engineering step, where raw data are prepared and augmented to derive meaningful features, often based on domain expert knowledge and domain and task understanding. With the rise of Neural Networks in a majority of Machine Learning application domains, this part of the Data Mining process has become less important. Instead, the Neural Network is expected to distill and focus on relevant

features implicitly. Therefore, several approaches to represent categorical and textual features have been studied with **one-hot encoding** as the most basic approach. For one-hot encoding, each feature is represented by a vector for which each possible attribute or value of a categorical variable becomes a vector dimension. This dimension in the one-hot encoding is set to 1 for the specific attribute and 0 for all other attribute possibilities, thereby creating vectors consisting of all 0 apart from a single one in one dimension. As each previously unseen attribute value results in an additional vector dimension, this sparse representation becomes very large for variables with many possible attribute values. Another aspect is the context in which certain features occur: For example, consider a certain category of products that is always associated with specific features of a product. In a traditional sense, this can be modeled by expert knowledge as a derived feature, however, when using one-hot encoding, this contextuality is not reflected within the feature representation.

As both aspects, feature sparsity and contextuality, can be associated with natural language and texts by the large number of different tokens (sparsity) and grammatical and semantic sense (contextuality), several alternative approaches for encoding data less sparsely have been studied in the field of Natural Language Processing, such as Word2Vec [208, 177], GloVe [233] and FastText [31]. These approaches are referred to as **word embeddings** and make use of text corpora to learn dense real-valued vector representations for words. The word context and the co-occurrence information seen during training introduce distributional semantics [267] within this dense vector space, also called *latent space*. The encoding of an attribute value within this latent space is consequently called *latent representation*.

In this section, several approaches for encoding and learning representations are adapted to the application domain of transaction logs to answer the research question RQ 2, which approaches are most beneficial for different data types, levels of data structuring and application domain. Besides one-hot coding, we focus on Word2Vec, GloVe and FastText, which have already been successfully applied to computer security related domains, such as IP-Addresses [251] and system calls [336, 337].

Originally, representation learning approaches were developed for natural language texts. In this section, we present two adaptations for transaction data. Our first adaptation is proposed for semi-structured Windows Audit Logs data (see Section 5.2.3). We then propose an approach to cope with mixed data types by including numeric features in the representation learning process and thereby adapt representation learning to SAP transaction data.

Parts of this section have been published as *Ring, M., Schlör, D., Wunderlich, S., Landes, D., and Hotho, A. (2021). Malware Detection on Windows Audit Logs using LSTMs.* Computers & Security, *109:102389* [255].

7.1 Representation Learning for Windows Audit Logs

For the first study, we focus on the domain of computer security and the Windows Audit Log dataset (see Section 5.2.3) in particular. For this dataset, we identified the categorical features *action*, *name*, and *target*, for which meaningful representations should be learned. The inherent structure of each feature value can offer valuable information, which when incorporated in the representation semantically, can improve the downstream task of maleware detection. For example, the features *name* and *target* can be considered as special data type of file-path within the Microsoft Windows operating system, implicitly containing aspects of the file and folder structure of Windows, which can be used as domain knowledge to assess the semantic quality of the latent representation. For example, files with file name endings *.exe* or *.dll* in general are executables, which can be benign application software but also executable maleware. By this, representation learning approaches can implicitly make use of this semantics introduced on substring level. Additionally, the folder hierarchy typically follows a defined structure with folders like the system folder or the folder for temporary files serving a dedicated purpose and an implicit best practice where which type of files should be saved. In this case, 'best practice' is understood as where a user typically expects a certain file to be located and includes 'best' practice used by attackers to hide a malicious file. In the following, we therefore introduce the embedding methods briefly, discuss our adaptation for this dataset, and present an explorative analysis of the latent space.

One-hot vector

One-hot vectors consider the different values of a feature. As a feature in the Windows Audit Log dataset, consider for example *action*, which has 10 different values specifying which file or process event is recorded: *close*, *create*, *delete*, *execute*, *modify*, *permissions*, *read*, *spawn*, *write* and *write_and_createReg*. In that case, the one-hot vector contains 10 components where each component represents a possible value of the feature *action*. The components are listed in a predefined order, i.e., the order of the listed values of the feature *action* given above. For one-hot vectors, the component that represents the current value is 1, while all other components are 0. Consequently, the value *close* is represented by the vector $\vec{a}_{close} = (1,0,0,0,0,0,0,0,0,0)^T$, the value *create* by the vector $\vec{a}_{create} = (0,1,0,0,0,0,0,0,0,0)^T$ and so on.

Word2vec

Methodically, Word2Vec [208] is based on a neural network where the hidden layer contains fewer neurons than the input and output layer. Using the skip-gram approach of Word2Vec, the neural network is trained with an input word and should predict the surrounding words. After training, the weights of the hidden layer are

used as vector representations for words. Therefore, words that often appear in similar contexts have similar vector representations, whereas words that do not appear in similar contexts have different vector representations.

From an implementation view, the skip-gram model can be understood as a neural network with three layers, the input layer, a hidden layer with the designated *latent* dimensionality N for the extracted word vectors, and an output layer. The input and output layers have a dimensionality of the size of the vocabulary V such that each dimension corresponds to one word in the vocabulary. Since skip-gram predicts the surrounding context words for a single input word, the input resembles a one-hot encoding of the corresponding input word, whereas the output vector contains the probability of a specific word belonging to the context of the input word.

This intuition can be formalized for a sequence of words

$$w_1, w_2, \cdots, \underbrace{w_{i-c}, \ldots, w_i, \ldots, w_{i+c}}_{\text{context including word } w_i}, \cdots, w_T$$

to the training objective of maximizing J, the average log-transformed conditional probability for the surrounding words $w_o, o \in C_i$ given $C_i = \{i-c, \cdots, i+c\} \setminus i$ for a context size of $2c$ words and the input word w_i over a text sequence containing T words, i.e.

$$J = \frac{1}{T} \sum_{i=1}^{T} \sum_{o \in C_i} \log p(w_o \mid w_i). \tag{7.1}$$

Because the computation of $p(w_o \mid w_i)$ is impractical for large vocabulary sizes, Mikolov et al. [208] introduced Negative Sampling. Intuitively, instead of considering all words in the vocabulary for each word for the optimization to infer $p(w_o \mid w_i)$ precisely, only a number of words which are not the input word (thereby negative samples) are sampled. For the word w_o, let $\mathbf{v}_{w_o}$ be the output vector of the neural network and $\mathbf{h}_{w_i}$ the vector of the hidden layer associated with w_i, all summarized in the parameters θ of the neural network, for a fixed parameter k denoting the number of negative samples drawn from the probability distribution $P_{\text{neg}}(w)$, $\log p(w_o \mid w_i)$ is approximated by $\text{neg}_\theta(w_o, w_i)$.

$$\text{neg}_\theta(w_o, w_i) := \log \sigma\left(\mathbf{v}_{w_o}^\top \mathbf{h}_{w_i}\right) + \sum_{w_n \sim P_{\text{neg}}(w)}^{k} \log \sigma\left(-\mathbf{v}_{w_n}^\top \mathbf{h}_{w_i}\right) \tag{7.2}$$

Mikolov et al. empirically found that $k \in [5, 20]$ for small and $k \in [2, 5]$ for large datasets and choosing $P_{\text{neg}}(w) := \hat{U}(w)^{3/4}$ for the Unigram probability distribution $\hat{U}(w_i) = \frac{\text{freq}(w_i)}{\sum_j \text{freq}(w_j)}$ performed well, decreasing the probability for frequent words and increasing the probability for less frequently words, compared to $\hat{U}(w)$.

During training, the weights of the neural network, i.e. the two weight matrices W^I of size $V \times N$, mapping from the input layer to the hidden layer, and W^O

of size $N \times V$, summarized in the parameter $\theta = [W^I, W^O]$, are updated via back-propagation. When using negative sampling, in addition to the input word w_i, only the drawn negative samples w_o are considered when minimizing $\hat{J}$ (Eq. 7.3) with Stochastic Gradient Decent. Thereby, only the respective rows of the weight matrices are changed [259].

$$\hat{J}(w_i; \theta) = -\sum_{o \in C_i} \text{neg}_\theta(w_o, w_i) \tag{7.3}$$

In addition to the skipgram where the surrounding context is predicted for an input word, Mikolov et al. also proposed the Continuous Bag of Words (CBOW) as an alternative approach, which predicts an output word based on the surrounding context and is defined analogously [208, 259].

FastText

FastText [31] builds on the basic idea of Word2Vec but additionally represents each word as a bag of character n-grams. The vector representation of a word is then the sum of the representation of all character n-grams. This approach allows sharing of subword representations between words, which improves the representations of infrequent words. More precisely, Bojanowski et al. generalize the idea of the skip-gram approach for Word2Vec such that $\mathbf{v}_{w_o}^\top \mathbf{h}_{w_i}$, or $\mathbf{v}_{w_n}^\top \mathbf{h}_{w_i}$ respectively, is replaced by a scoring function $s : w_i, w_o \mapsto \mathbb{R}$,

$$s(w_i, w_o) = \sum_{g \in \mathcal{G}_{w_i}} \mathbf{v}_{w_o}^\top \mathbf{z}_g \tag{7.4}$$

for a set of n-grams $\mathcal{G}_{w_i}$ appearing in and including word w and their respective latent vector representation $\mathbf{z}_g$. The resulting objective is then trained as described above for W2V.

GloVe

GloVe [233] is a co-occurrence-based approach, which uses local context window information like Word2Vec but also incorporates global information similar to matrix factorization to obtain meaningful word representations. In contrast to Word2Vec and FastText, GloVe is not prediction-based, but rather a counting-based model. It benefits from training on non-zero entries in the co-occurrence matrix to learn latent word representations in a way that they follow the probability of co-occurrence of the respective words.

In detail, let X be the matrix of co-occurrences between words, i.e. $X_{i,j}$ denotes how often a word j occurs in the context of word i and $P_{i,j} = P(j \mid i)$ the probability of j occurring in the context of i. The idea is to learn vector representations that resemble the ratio of co-occurrence probabilities $\frac{P_{i,k}}{P_{j,k}}$. Therefore, Pennington et

al. reformulate this problem as a weighted least squares regression model with a weighting function $f(X_{i,j})$ and cost function

$$J = \sum_{i,j=1}^{V} f(X_{i,j})(v_i^\top \bar{v}_j + b_i + \bar{b}_j - \log X_{i,j})^2 \tag{7.5}$$

for vocabulary V, word vectors v and context word-vectors $\bar{v}$ with $v, \bar{v} \in \mathbb{R}^d$, i.e. the prediction of the co-occurrence matrix by the word-vector weights and biases b_i and $\bar{b}_j$ as trainable parameters and

$$f(X_{i,j}) = \begin{cases} (X_{i,j}/x_{\max})^\alpha & \text{for } X_{i,j} < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

as weighting function avoiding over-weighting of frequent and rare words. Pennington et al. propose $x_{\max} = 100$ and $\alpha = 0.75$, which we adopt for our experiments. The model creates two sets of word vectors w and $\bar{w}$, but since context and non-context words are generally exchangeable, the difference of both (a result of random initialization) is mitigated by summing $\hat{v}_i = v_i + \bar{v}_i$ to finally obtain the word embedding $\hat{v}_i$ for word i.

Adaptation

In the following, we exemplarily use the feature *action* and explain how we represent our data as word embeddings. The same procedure is applied for the features *name* and *target*.

To train the models, an analogy to sentences is needed. Therefore, we consider the sequence of events as a sentence where the *action* represents the words. Each file of the dataset represents one sentence consisting of all events in the order of occurrence. Consequently, we extract sentences like "*read execute spawn modify write*" from Windows Audit Logs. Table 7.1 illustrates the generation of training samples based on this sentence for Word2Vec.

At first, Word2Vec selects a so called *input word* from the training sentence. Then, words from the surrounding window (we refer to them as *context words*) are used to build training samples. Table 7.1 uses a window size of $c = 2$.

During training, the neural network is fed with the *input word* and tries to predict a *context word*. For each training sample, the expected output value for the corresponding *context word* is 1 and 0 for all other words. An example is shown in Table 7.1: For the first sample, *read* is the *input word*. In that case, the words *execute* and *spawn* are in the context which should be predicted by the network for the input *read*. After the training phase, the weights of the hidden layer are used as vector representations for the values of the feature *action*. The adaptation for GloVe is applied analogously.

Table 7.1: Sample generation for the parameter *action* for word2vec.

#							**input word**	**context word**
1	read	execute	spawn	modify	write	→	read	execute
							read	spawn
2	read	execute	spawn	modify	write	→	execute	read
							execute	spawn
							execute	modify
3	read	execute	spawn	modify	write	→	spawn	read
							spawn	execute
							spawn	modify
							spawn	write
4	read	execute	spawn	modify	write	→	modify	execute
							modify	spawn
							modify	write
5	read	execute	spawn	modify	write	→	write	spawn
							write	modify

FastText expands this approach by also incorporating sub-strings, i.e. *execute* is represented by the character n-grams <*ex*, *exe*, *xec*, *ecu*, *cut*, *ute*, *te*> and <*execute*>, with < and > representing word boundary tokens. The benefit of this approach becomes more visible for the features *name* and *target*, where sub-strings of for example the file name carry certain semantics, such as the prefix *[system]* for the windows system root folder or a suffix such as *.exe* indicating an executable file, even if the full path name was unique and never seen during training.

Extrinsic Evaluation

With this experiment, we analyze the effect of different representations for detecting malicious behavior in Windows Audit Log events. The dataset (cf. Section 5.2.3) consists of recorded files containing approximately four minutes of Windows audit log events each, including a label for each file, which describes if the file includes events generated during the execution of maleware or only events during normal system execution. As audit logs are sequential data for which malicious behavior has thereby to be identified over a number of samples as collective anomaly (cf. Section 4.4), we use Long Short-Term Memory-models (LSTM) on the sequences of event logs per file and predict the label, i.e., whether the file contains a malicious execution. As input for the LSTM model, we evaluate one-hot encoding and embedding representations for categorical audit-log event features concatenated per sample.

Due to long-term dependencies given in Windows Audit Logs, we stack LSTM layers, as stacking LSTM layers can improve the handling of long-term dependencies

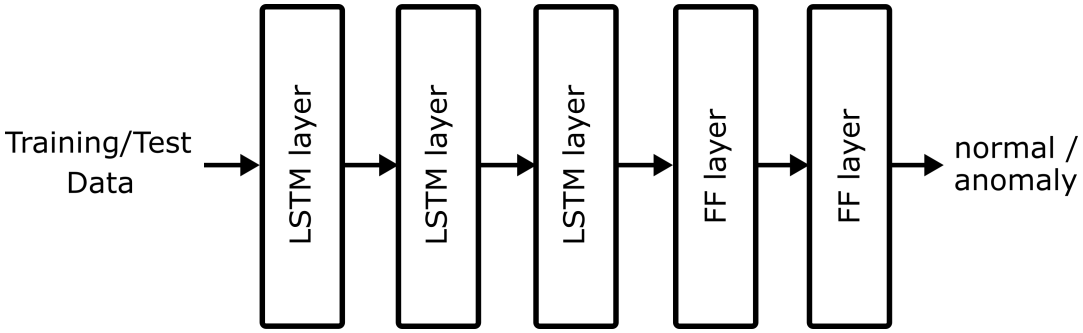


Figure 7.1: LSTM model for extrinsic evaluation.

in sequences [270]. A preliminary study showed that stacking three LSTM layers works well for our data. The model containing the LSTM layers followed by two fully connected feed-forward layers is shown in Figure 7.1.

Preprocessing and incorporating one-hot or embedding approaches lead to a varying number of features, for which we adapt the number of neurons per layer depending on the number of input features n. The first layer uses $min(max(\frac{n}{2}, 4), 128)$ neurons, the second LSTM layer uses $min(max(\frac{n}{4}, 3), 64)$, the third LSTM layer uses $min(max(\frac{n}{8}, 2), 32)$. The first fully connected layer uses 4 and the last layer 1 neuron predicting a value of 0 for normal and 1 for malicious sequences. For one-hot encoding, small threshold values *min_tar* and *min_proc* highly increase the number of input features. The maximum number of neurons in the first three layers are therefore limited to 128, 64 and 32, respectively, to avoid network architectures which are too large. The model is trained optimizing the mean absolute error using ADAM optimizer with a learning rate of 10^{-4} for 10 epochs, a batch size of 64 and a sequence length of 32.

In the context of Windows audit logs, some features are given which contain file paths within the file system tree. This path can refer to executables or data, as well as temporary files, and hence contain user-specific and random file names. While several feature values reflect typical paths and files within a Windows system, a large number of paths are extremely rare up to unique for a given application or event situation. This study aims at representing features in a semantically meaningful context, which for frequent values is typically implicitly defined by the context in which they occur. However, for rare and unique values, the single context does not allow to span a semantically meaningful space and increase the complexity and computational demands for the training of the representation and downstream task [146]. Additionally, previously unseen values during testing in a realistic experimental setup lack an embedding representation and require specific training with out-of-vocabulary tokens, which can be introduced by the masking of infrequent values allowing both, a more efficient computation and a representation of novel values.

For this experiment, we introduce the thresholds *min_proc* and *min_tar* to identify frequent and non-frequent values and thus limit the number of possible manifestations for the features *process* and *target* in the dataset. For example, all *process* values which occur less frequently than *min_proc* in the dataset are thereby replaced by a special out-of-vocabulary token. This token is also used for values previously unseen during training to represent potentially novel processes. We evaluate various threshold choices along with the different embedding variants, as these parameters influence the model size, computation time, and detection performance and potentially depend on the representation learning approach:

While low thresholds lead to very large vectors for one-hot encoding, increasing the computation time and networks architecture, the influence on the model for representation learning approaches is less obvious. Especially FastText has the potential to benefit from substring representation as infrequent strings can contain meaningful (frequent) substrings. We therefore vary *min_proc* and *min_tar* from 0.001% to 3% of 0.01% to 3% respectively, for one-hot encoding as the vectors and models become impractically large for a lower threshold. For FastText, we include the thresholds 0% (i.e., no filtering) to evaluate the benefit of substring encoding for infrequent samples.

As baseline, SVM with RBF kernel as well as a Long Short-Term Memory (LSTM) model without process and target paths are included, and we report all metrics with mean and standard deviation averaged over 5-fold cross-validation splits.

Results Table 7.2 shows the results of this experiment with respect to True Positive Rate (TPR), False Positive Rate (FPR), and Accuracy (ACC). The non-sequential SVM model is not able to capture the sequential dependencies of the dataset and shows a low performance on all metrics. The LSTM baseline performs better with regard to FPR and ACC but cannot predict malicious behavior well without a proper representation of the process and target path. Including both features in our LSTM model increases the performance on all metrics by a large margin.

Surprisingly, one-hot encoding with a *min_proc* and *min_tar* threshold of 0.1% yields the best results. However, the differences between all approaches are very low. Results for all models include runs with 0.99 TPR fluctuating within the magnitude of performance variations for different runs. Thus, the differences in performance of different approaches have to be considered as insignificant.

Different choices of the threshold have a more notable influence on the performance. One-hot encodings achieve their best results for medium-high thresholds $min\_proc = 0.1\%$ and $min\_tar = 0.1\%$, while lower thresholds lead to only slightly worse results with respect to FPR and ACC. The three embedding variants yield very similar results and perform similarly with regard to the thresholds used. The representation learning approaches, especially the models with GloVe embeddings, also impair notably for the 3% threshold. In general, lower threshold values lead to

Table 7.2: Extrinsic evaluation results of representation learning approaches and filtering for malicious event detection in Windows Audit Logs. The best results for each encoding are shown in italics, the overall best results in bold.

Encoding	min_proc	min_tar	↑ TPR	↓ FPR	↑ ACC
Baseline (SVM_{RBF})			0.9539 ± 0.0113	0.2868 ± 0.0163	0.7889 ± 0.01215
Baseline ($LSTM_{action}$)			0.9075 ± 0.0124	0.2572 ± 0.0156	0.7948 ± 0.0081
one-hot	3%	3%	0.9908 ± 0.0035	0.1634 ± 0.0086	0.8869 ± 0.0046
one-hot	1%	1%	0.9772 ± 0.0086	0.1527 ± 0.0087	0.8897 ± 0.0061
one-hot	0.5%	0.5%	0.9941 ± 0.0012	0.1349 ± 0.0102	0.9073 ± 0.0051
one-hot	0.1%	0.1%	***0.9944 ± 0.0013***	***0.1292 ± 0.0064***	***0.9112 ± 0.0031***
one-hot	0.01%	0.01%	0.9921 ± 0.0031	0.1361 ± 0.0049	0.9049 ± 0.0028
FastText	3%	3%	0.9814 ± 0.0079	0.1658 ± 0.0089	0.8823 ± 0.0076
FastText	1%	1%	0.9684 ± 0.0201	0.1557 ± 0.0102	0.8848 ± 0.0108
FastText	0.5%	0.5%	0.9913 ± 0.0034	0.1453 ± 0.0134	0.8993 ± 0.007
FastText	0.1%	0.1%	0.9928 ± 0.0012	0.1374 ± 0.0097	0.9052 ± 0.0053
FastText	0.01%	0.01%	0.9903 ± 0.0045	*0.1314 ± 0.0083*	*0.9084 ± 0.0054*
FastText	0.001%	0.001%	0.9911 ± 0.0042	0.1334 ± 0.0104	0.9073 ± 0.0068
FastText	0%	0%	*0.9930 ± 0.0033*	0.1398 ± 0.0113	0.9036 ± 0.0059
Word2Vec	3%	3%	0.9751 ± 0.0102	0.1707 ± 0.0102	0.8769 ± 0.0088
Word2Vec	1%	1%	0.9796 ± 0.0132	0.1556 ± 0.0129	0.8886 ± 0.0081
Word2Vec	0.5%	0.5%	*0.9935 ± 0.0017*	0.1438 ± 0.0118	0.9011 ± 0.0064
Word2Vec	0.1%	0.1%	0.9925 ± 0.0022	0.1354 ± 0.0097	0.9064 ± 0.0046
Word2Vec	0.01%	0.01%	0.9900 ± 0.0050	*0.1314 ± 0.0087*	*0.9082 ± 0.0060*
Word2Vec	0.001%	0.001%	0.9893 ± 0.0054	0.1314 ± 0.0097	0.9080 ± 0.0065
GloVe	3%	3%	0.9500 ± 0.0172	0.2156 ± 0.0216	0.8385 ± 0.0150
GloVe	1%	1%	0.9641 ± 0.0207	0.2125 ± 0.0203	0.8452 ± 0.0154
GloVe	0.5%	0.5%	0.9839 ± 0.0108	0.1768 ± 0.0202	0.8757 ± 0.0114
GloVe	0.1%	0.1%	0.9820 ± 0.0088	0.1626 ± 0.0211	0.8846 ± 0.0122
GloVe	0.01%	0.01%	*0.9925 ± 0.0036*	0.1591 ± 0.0232	0.8904 ± 0.0145
GloVe	0.001%	0.001%	0.9889 ± 0.0058	*0.1361 ± 0.0100*	*0.9047 ± 0.0068*

↑ / ↓: higher / lower value is better

better results in all evaluation measures, and FastText yields the best result without filtering. However, between 0% and 0.1% the results differ only slightly compared to the significantly higher computational demands. All in all, the extrinsic evaluation shows no clear pattern favoring one approach over the others, as all approaches perform similarly, but it underlines the benefit of including both textual features in a model which is able to capture the sequential dependencies of the dataset. For all representation learning approaches, filtering infrequent values below 1% does not impair performance, while reducing computational needs and allowing to process previously unseen values in practice.

Explorative Analysis

In this section, we pair the latent representations with domain knowledge to reveal further insights which aspects are captured well within the latent space. We exemplarily analyze the latent space for the attribute *target* which by itself is the most meaningful single parameter from a domain perspective as it reflects accessed or written files and paths to started processes. The embeddings are projected into a two-dimensional space using t-SNE [322]. To visualize the homogeneity of the latent space with respect to the specificity of attack or normal behavior, we apply the following coloring scheme: For all files within the dataset, we count how often each *target* occurs in an attack and normal context, respectively. We then weight the counts according to the total proportion of attack vs. normal sequences. A *target* only occurring in attacks has a score of 1.0 (red), whereas a *target* only present in normal behavior has a score of 0.0 (blue). A *target* which is present in both attack and normal behavior equally (relative to the overall proportion) has a score of 0.5 (gray).

Figure 7.2 shows the representations for *target* with a threshold of $min\_tar = 0.001\%$ for one-hot encoding, Word2Vec, GloVe, and FastText embeddings. The latent space differs considerably between the representation variants. One-hot encodings expectedly do not yield a meaningful structure, while GloVe and Word2Vec reveal latent structures to some extent. In contrast to that, the ability of FastText to consider substrings seems to be beneficial to create semantically meaningful clusters, which reveal their preference for attack or normal behavior even in a very low-dimensional space. When inspecting the different clusters closely, folder structure such as common sub-folders, file name structure and file endings become visible, as well as semantic or contextual similarity such as temporary files of various types created as web-browser cache or system file clusters. This is also reflected in the attack score (color), where several very homogeneous attack and normal behavior clusters appear.

Although the quantitative evaluation could not leverage the latent structure, the explorative analysis clearly revealed interesting patterns for all RL approaches favoring FastText as approach, which is able to incorporate substring information. Further investigations suggest that the data structure for collective anomalies defined over a stream of audit logs of arbitrary length (including a large amount of benign log entries) mitigates the benefit from meaningful individual representations. We therefore adapt this setting in the following section to point anomalies in SAP transaction data.

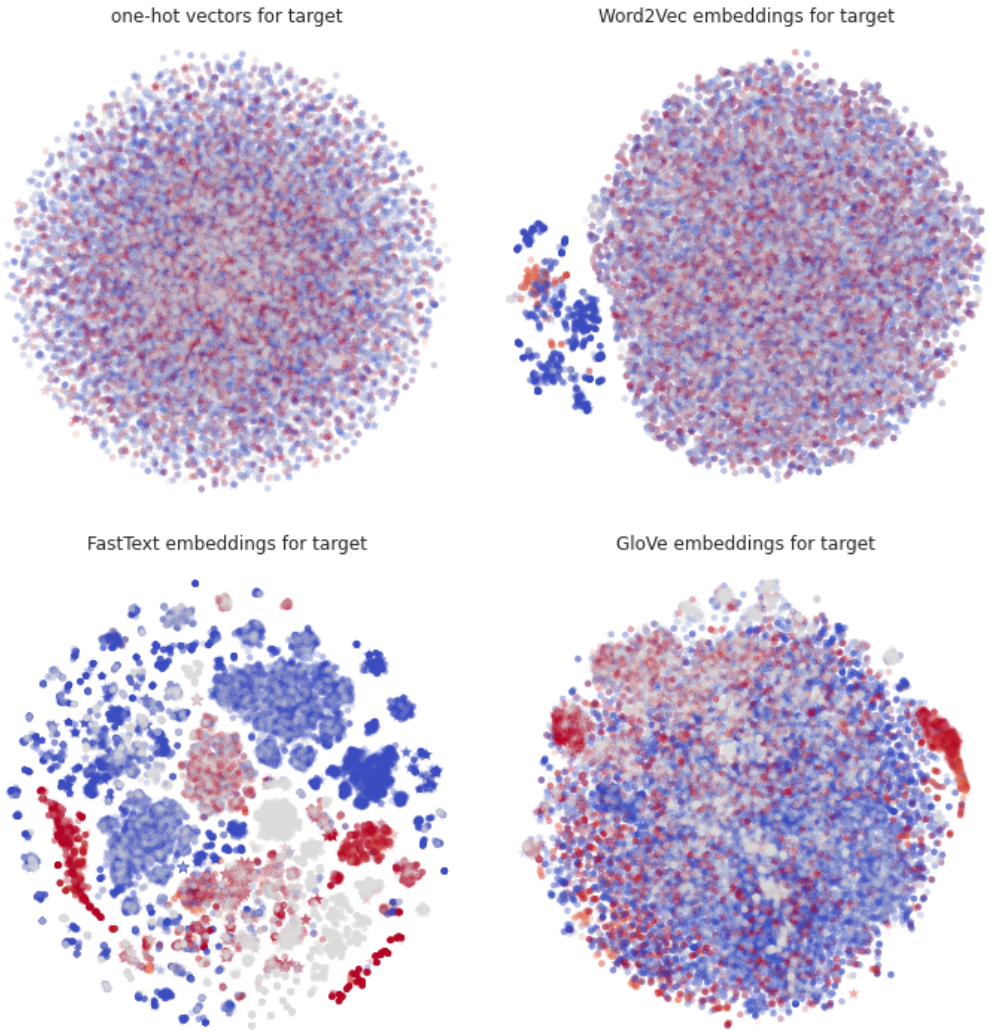


Figure 7.2: t-SNE visualization of the latent space for *target*

7.2 Representation Learning for SAP Transactions

Besides textual and categorical features and their representation using word embedding approaches, transaction logs often contain several other heterogeneous data types, which contain potentially useful information about the transaction. Consider, for example, transaction logs from an SAP system. In this database or table-like structure, features can be identified column-wise, for example, as numeric attributes, such as the quantity or price of goods or materials, but also as columns with categorical features, such as the posting key, the identifier or name of the good or material.

Since representation learning approaches have been developed for textual data as their name **Word**2Vec or Fast**Text** implies, their architecture was not developed with other data types in mind, such as numerical features. Although models trained on large-scale text datasets show limited numeracy within a certain value range, they are generally not able to represent numbers precisely [325], and specifically token-based models suffer from the sparsity of large numbers, as each number token is represented individually and might not occur frequently enough to be captured by sampling-based training. Nevertheless, for categorical or textual features in a mixed-type dataset, their numeric-feature context must be considered to learn meaningful representations while reducing the complexity of predicting specific numbers digit by digit.

Approaches for time-series data, for example, accomplish this by clustering the datapoints and representing them by their cluster [214], while others use logarithmic scaling and prototype numbers to aggregate numeric features [150]. Such methods have in common that they represent values within the most fitting prototype or cluster regardless how well they are actually represented since the objective is to fit most of the data best. However, this objective contradicts the main task of detecting anomalies as potential outlier values are represented together with normal data. To overcome this problem and address outliers in the margin of feature distributions, we choose a different approach, which explicitly takes them into account. In the following, we therefore propose an outlier-aware discretization approach to include numerical features and finally formulate an RL task for structured SAP transaction data including numeric features and outliers.

7.2.1 Outlier Aware Discretization

One typical preprocessing approach for machine learning models which require categorical input data is to discretize these attributes. Therefore, the continuous value range is typically divided into a finite number of non-overlapping intervals. The continuous values are then replaced by the (ordinal) identifier of their associated interval. As for the discretization technique, one can differentiate between different properties and assumptions to choose the best fitting discretization algorithm.

A taxonomy proposed by García et al. [105] suggests classification criteria such as supervised and unsupervised, univariate and multivariate or static and dynamic and summarizes several approaches. Supervised methods hereby use the class label in relation to the attributes to discretize, whereas unsupervised methods do not rely on a label.

To learn meaningful representations of labeled and unlabeled datasets, we are restricted to unsupervised methods. Multivariate methods focus on all attributes jointly to find a discretization, whereas univariate approaches choose a discretization on a per-attribute basis. Continuous attributes in transaction data generally are independent of their distribution range and semantic as, for example, order quantities and prices per item can differ in magnitudes and are not directly related. Since features are not comparable only by their value, univariate approaches are more promising for our application setting. A discretization is considered dynamic when the discretization is created coupled with the learning algorithm during the training procedure. Static discretization, on the other hand, can be understood as a preprocessing step independent of the subsequent learning algorithm. Since we want to create representations that are independent of the specific downstream task, static discretization on a global dataset level fits our requirements best.

Among unsupervised discretization methods, parametric approaches such as uniform, also called equal-width discretization, quantile, also called equal-frequency discretization, and clustering-based approach, e.g., based on k-means, are commonly used for a variety of data mining problems [84]. Since these are parametric methods, they demand the number of intervals (or clusters) as a predefined parameter, which has a direct influence on the discretization quality. To set this parameter, several heuristics have been discussed [283] and adapted from histogram density estimators [76] to search algorithms based on class labels [37].

For the application to anomaly detection, the frequency of observed features is inherently important, since anomalies by definition are infrequent derivations in feature space regarding the majority of normal data. In the most basic case, this is manifested in one feature where the anomalies can be understood as instances from the lowest regions of the Probability Density Function (PDF). If we choose quantile-base discretization, the PDF is separated into bins creating equal-area chunks of the PDF, i.e. such that the integral over the PDF bound to the bin limits is equal for every bin. This results in the same number of instances in every bin regardless of their explicit difference in value. As a consequence for a decent number of bins, outliers, as they are infrequent for their own, share a bin with more frequent instances which can be assigned to non-anomalous regions, and thus possible anomalous values of large value will be masked irretrievably. On the other hand, discretization by width has the downside that outliers of large value distort the potentially useful range of the bins, which can lead to a large number of unused bins and thus fewer bins to differentiate between values in the actual range of normal data.

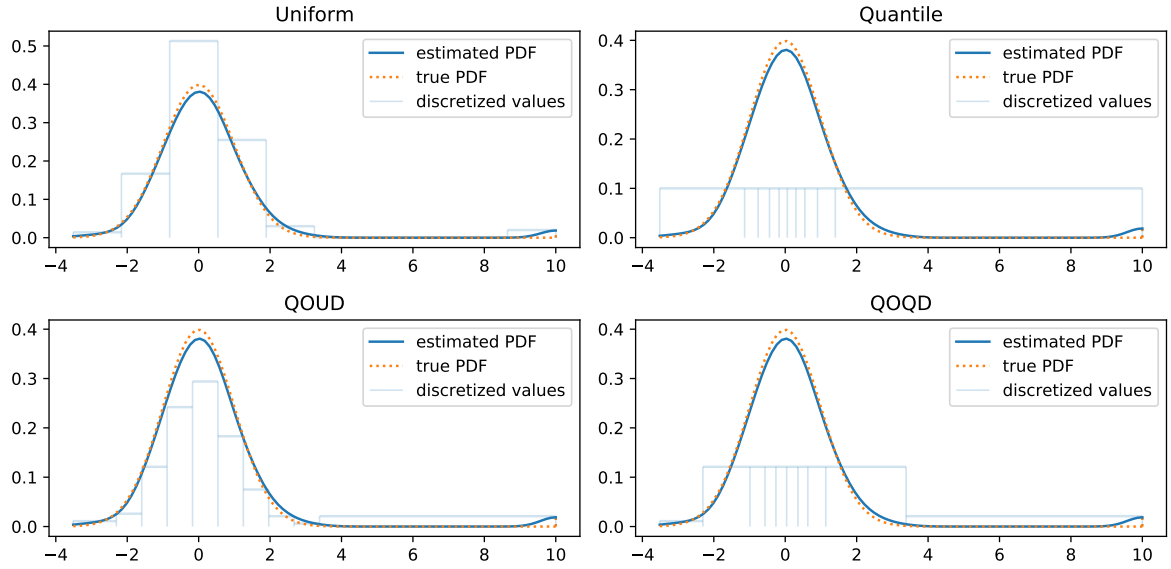


Figure 7.3: Synthetic standard normal data with 0.02% outliers with 10 bins and different discretization strategies. The outlier aware discretization strategies (QOUD, QOQD) reflect outliers in specific discretization bins, while learning meaningful bins for the majority of data-points contrarily to uniform discretization with several empty bins and a coarse resolution for the actual distribution or skewed margin-bins obfuscating the outlier characteristics.

To overcome this issue, we propose two different approaches:

1. Quantile-Outlier-Quantile discretization (QOQD)

2. Quantile-Outlier-Uniform discretization (QOUD)

The main idea of all approaches is to explicitly introduce bins for outliers. The bounds of the bins are determined by a high quantile. For example, for the 99%-quantile, the contiguous range including 99% of the values are taken as normal values and the remaining 1% is discretized into explicit anomaly bins. The normal values are then again discretized by frequency or width, depending on the variant.

Both variants have their pros and cons, which are shown in Fig. 7.3: Quantile discretization constructs each bin so that it has the same number of samples within the training data, i.e., each bin appears with equal probability. This can be beneficial for training, since there therefore are no infrequent feature values that could not be well utilized by representation learning techniques [268]. On the other hand, numerical dependencies are not reflected and especially the outer bins include a wide range from in-distribution values to outliers, which cannot be distinguished anymore. Uniform quantization, in contrast, has the advantage of taking numerical

dependencies into account, i.e. the bin number is linearly correlated with the actual value under consideration of the feature distribution. However, this comes at the expense of introducing a large number of empty bins and an unusable bin width, if outliers exist, which are way out of the distribution of normal data. With the QO*D approaches, the limitations aforementioned are mitigated to some extent: Outliers at the distribution value-edges are explicitly modeled as possibly infrequent, but large bins, whereas non-outliers, i.e. values from the actual distribution, are represented well by equal width (QOUD) or equal frequency (QOQD).

This approach can be extended by introducing special bins for semantically meaningful numeric values. In the context of tabular data, for example, missing values must be represented appropriately, which can be accomplished with an additional bin. Features which are exactly zero might also be of special interest and can be modeled in an explicit one-value bin.

7.2.2 Representations for SAP Transactions

Applying outlier-aware discretization approaches, representations for SAP transaction data can be learned, for which numeric and categorical features are taken into account. In this experiment, we answer the question which discretization and which representation learning method is most suitable to ensure data characteristics being represented appropriately for financial fraud detection. From a domain-driven perspective, several dependencies and categorization schemes are derived to analyze their influence on the structure of the latent space. Besides this explorative analysis, we construct an extrinsic evaluation by directly predicting whether transactions are fraudulent or non-fraudulent solely based on the latent representation of the transaction.

Adaptation of Representation Learning Approaches to SAP Transactions

Before learning representations of transactions, numeric features are discretized first, which allows their integration into RL approaches and thereby increases the amount of information the methods can utilize to learn meaningful representations for categorical features as discussed before.

For this, we select the numerical feature columns and apply the discretization techniques as discussed in Section 7.2.1 per feature. Over all, we create eight discretized datasets with 5 or 10 bins, respectively, per feature for uniform, quantile, QOUD and QOQD discretization and apply label encoding, i.e. the numeric value is replaced by the label of the associated discretization-bin. Categorical columns remain without transformation or encoding, so that the resulting datasets contain only categorical values for each feature.

This encoding of a transaction, consisting of several categorical features, shows similarities with sentences, paragraphs, or ultimately documents consisting of words

which define their meaning. To learn latent representations of words, several word embedding approaches have been shown to provide meaningful representations, as discussed in Section 7.1, which will also be compared in this context. Most of these word-embedding approaches build upon a prediction task, inferring context words from an input word or vice versa. In the case of SAP transactions, this context can be naturally defined by the other features present in a transaction, which we refer to as sample-context. Following [108], we additionally construct a feature-context, considering the context of neighboring transactions regarding each feature. Ghasemi-Gol et al. argue that tabular data are homogeneous along rows or column. In abstraction to transaction data, columns translate to homogeneous feature categories and rows to homogeneous transactions. Including the feature-context also introduces sequential information on a feature-level to some extent, which can be useful for SAP transactions, since, for example, placing an order, posting an invoice, or other business cases can initiate several consecutive entries in transaction logs. To model distinguishable features for the sample-context as well, we also include the feature name along with each value to be able to distinguish the meaning of, e.g., "X" for the presence of two different features such as *Line item display possible=X* or *Indicator: Item cannot be copied=X*.

Modeling features like this, we obtain a common embedding space encoding several realizations of different features all together. Motivated by the finding that the latent vector spaces allow arithmetic operations to combine word vectors [209] semantically meaningful, several studies employed averaged embedding vectors of the consisting words for phrases or sentences [294, 144, 69, 54]. Coates and Bollegala even showed that averaging of word vectors from *different* embedding spaces yields a surprisingly well-performing meta-model [62]. We follow this approach and represent a transaction by the average of its consisting feature vectors. Additionally, we apply Paragraph2Vec [177] learning paragraph vectors of transactions alongside the word-embeddings and create meta-embeddings by averaging all other approaches as suggested by [62] for comparison.

As the size of the embeddings, representation learning methodology and data foundation, discretization strategy, and number of bins are spanning a very large parameter space, we first conduct an extrinsic evaluation with the downstream task of fraud detection before an explorative analysis of the latent space is presented.

Extrinsic Evaluation

For extrinsic evaluation, we use the task of fraud detection on the SAP dataset (see Section 5.1.4 for a detailed description) to find the representation learning approach that best fits the data and the task. Therefore, we rely only on the learned transaction embeddings according to which a classifier has to decide whether a transaction represented by this embedding is considered fraud. Different approaches are compared using the performance metrics F1, ROC-AUC and AP (cf. Section 4.5.1) and

a discussion of the confusion matrix, since all metrics have their individual focus (as discussed in Section 4.5.2). Overall, we report the results of different parameter, model and representation learning choices ranked by F1 and ROC-AUC.

We create several representations of the dataset varying between uniform, quantile, QOUD, and QOQD discretization for 5 and 10 bins, each. As the dataset is divided into benign training data, training data with benign and fraudulent samples, and test data containing benign and fraudulent samples, we consider several combinations of these dataset splits for representation learning. Note that all representation learning approaches are unsupervised. Therefore, the complete dataset, train (T), validation (E) and test (S) splits, can be used to learn the latent space describing the respective data characteristics. We evaluate building "sentence"-sequences over features spanning over all features within a transaction (Tr, Er, Sr), representing a *row* in the dataset table. Column-based "sentence"-sequences are constructed from the sequence of transactions per feature (Tc, Ec, Sc), i.e., the *columns* in the dataset table. For examples, a dataset descriptor *TrErSc* denotes row-based sentences for both train and validation splits, as well as column-based sentences for the test split to learn representations from.

Regarding the benefit of each split and their associated characteristics for the task, respectively, there is no obvious best choice: It is conceivable that learning on benign data only emphasizes the unexpected characteristics of fraudulent transactions, especially in a one-class setting. On the other hand, it is conceivable that more data for learning characteristic representations might be beneficial.

For feature values only present in the test set, considering all splits avoids Out-of-vocabulary (OOV) words, which are treated differently by different representation learning approaches: While FastText is assigning substring representations which could be learned during training and therefore has no OOV by construction, for GloVe and Word2Vec OOV tokens have to be explicitly introduced in training, and there is no distinction between different OOVs. Since we aggregate all feature embeddings to a transaction embedding, we omit OOV feature values for the respective dataset variants. For building the local context or, in case of GloVe, the document corpus, we evaluate combinations of including and omitting transaction- and feature-context. Note that there are no OOVs as long as any of transaction- or feature-contexts of all splits are included for representation learning. As representation learning approaches, we include Word2Vec, FastText, GloVe, Paragraph2Vec, and the aforementioned meta-model averaging all approaches and evaluating $1, 2, 4, 8$ and 16 dimensions. The number of dimensions is much smaller than usual for pre-trained word interpretation models. This adaptation is necessary, since the vocabulary of the dataset is way more limited than vocabulary in typical natural language contexts: With 186 unique values (4.4 per feature on average), the number of unique categorical feature-values in the SAP domain is magnitudes lower in comparison to natural language, and the numeric attributes do neither artificially introduce a large

vocabulary, since the number of numeric feature-values is limited by the discretization approach.

Therefore, each transaction is represented by a vector between 1 and 16 dimensions to be evaluated in the downstream task. For classification, we evaluate different machine learning algorithms. As a one-class classifier, we include Isolation Forest solely trained on benign data and evaluate on the test set. As supervised approaches, kNN (k=3), Linear Support Vector Machine and Random Forest are included. They are trained on the second training split including fraudulent data that are oversampled for an equal class distribution using the Synthetic Minority Oversampling TEchnique (SMOTE) [51]. Each experiment is repeated five times with different random seeds to mitigate statistical fluctuation, and average performance over these runs is discussed.

Results The results ranked per F1 and ROC-AUC are depicted in Table 7.3. We include the top 5 ranks per metric and additionally report interesting results, which we discuss in the following in more detail.

The best results with a F1 value of 0.667 were achieved using QOQD with 5 and 10 discretization bins, respectively, and GloVe with 8 and 16 latent dimensions, respectively, learned on the complete dataset using transaction-context only with kNN. The model predicted 3 of the 6 fraud cases and all benign cases correctly. In comparison to the simple baseline of applying kNN directly to the data without further preprocessing and representation learning, only GloVe-based models with representations learned on all three splits outperformed the baseline. Except one uniform discretization configuration, all other outperforming runs were discretized via QOQD. Note that the best model according to F1 (and AP) only detected 3 of 6 fraud cases correctly, whereas the baseline model classified 4 of 6 fraud cases correctly at the expense of slightly more false positives.

For the ROC-AUC score, QOUD with 10 bins, GloVe with 16 dimensions trained with transaction-context on the benign and mixed training splits performed best. This kNN model predicted all fraud cases correctly and produced 100.2 false positives. Among the five best models according to each evaluation metric, all used GloVe as representation learning approach. QOQD is the discretization of the five best models regarding F1 and AP producing the least number of false positives to the expense of missing some fraud cases. It is worth mentioning that only GloVe-based models (8 and 16 dimensions) outperformed (according to F1) a baseline model (rank 10) directly trained on the discretized raw data without representation learning to reduce the dimensionality, which yielded a 0.5 F1 score.

The top 5 approaches regarding ROC-AUC use QOUD, QOQD and for one instance quantile discretization and predict all fraud cases correctly with 100.2 to 408.6 false positives on average. Most models benefit from transaction- and feature-context of the training split (TrTc) or the whole dataset (TrErSrTcEcSc). In the supervised

Table 7.3: Best models regarding F1 and ROC-AUC.

rank by		disc	bucks	dataset	class	rep	dims	↑ F1	↑ ROC-AUC	↑ AP	↑ TP	↓ FP
F1	1	QOQD	5	TrErSr	kNN	GLV	8	0.667 ± 0.000	0.750 ± 0.000	0.500 ± 0.000	3.0	0.0
	2	QOQD	10	TrErSr	kNN	GLV	16	0.667 ± 0.000	0.750 ± 0.000	0.500 ± 0.000	3.0	0.0
	3	QOQD	5	TrErSc	kNN	GLV	8	0.607 ± 0.068	0.750 ± 0.000	0.400 ± 0.105	3.0	1.0
	4	uniform	10	TrErSc	SVC	GLV	8	0.569 ± 0.102	0.833 ± 0.000	0.343 ± 0.117	4.0	4.4
	5	QOQD	10	TrErSr	kNN	GLV	8	0.567 ± 0.091	0.700 ± 0.046	0.400 ± 0.091	2.4	0.0
							...					
	10	quantile	5	TrErTcEc	kNN	none	0	0.500 ± 0.000	0.833 ± 0.000	0.267 ± 0.000	4.0	6.0
							...					
ROC-AUC	1	QOUD	10	TrEr	kNN	GLV	16	0.107 ± 0.008	0.997 ± 0.000	0.057 ± 0.005	6.0	100.2
	2	QOQD	5	TrErSrTcEcSc	SVC	GLV	4	0.082 ± 0.090	0.993 ± 0.004	0.045 ± 0.052	6.0	261.2
	3	quantile	10	TrTc	kNN	GLV	4	0.038 ± 0.001	0.992 ± 0.000	0.019 ± 0.001	6.0	302.8
	4	QOUD	5	TrErSrTcEcSc	RF	GLV	2	0.035 ± 0.001	0.992 ± 0.000	0.018 ± 0.001	6.0	331.6
	5	QOQD	5	TrEr	SVC	GLV	8	0.030 ± 0.010	0.990 ± 0.002	0.015 ± 0.005	6.0	408.6
	6	QOQD	10	TrErSc	SVC	FaT	2	0.029 ± 0.001	0.990 ± 0.000	0.014 ± 0.001	6.0	409.2
							...					
	10	quantile	5	TrErTcEc	SVC	W2V-cb	2	0.028 ± 0.001	0.989 ± 0.000	0.014 ± 0.000	6.0	424.6
	11	uniform	5	TrErSr	SVC	W2V-sg	2	0.027 ± 0.001	0.989 ± 0.000	0.014 ± 0.000	6.0	434.4
							...					
	26	quantile	5	TrErTcEc	SVC	P2V	2	0.026 ± 0.002	0.989 ± 0.001	0.013 ± 0.001	6.0	452.4
							...					
	93	uniform	10	TrEcSc	ISO	P2V	16	0.019 ± 0.001	0.984 ± 0.001	0.010 ± 0.001	6.0	621.0
							...					
	983	quantile	5	TrErSrTcEcSc	kNN	none	0	0.500 ± 0.000	0.833 ± 0.000	0.267 ± 0.000	4.0	6.0
							...					

↑ / ↓: *higher / lower value is better*

setting, kNN is the most suitable model on average for all evaluation metrics. The best results in the one-class setting using Isolation Forest all rely on Paragraph2Vec with 16 dimensions, trained with transaction-context for the benign training set and feature-context for the other splits. However, the one-class models perform notably worse than the supervised approaches (with 621 false positives and rank 93 at best compared to 100.2 false positives for the best supervised model). Generally, models using GloVe and QOQD yield notably better F1 and AP scores compared to all other models. Regarding ROC-AUC, GloVe models also yield the best results on average and occupy the top five ranks with different discretization approaches. One FastText model is directly following on rank 6, Word2Vec models are on ranks 10 and 11 and the best Paragraph2Vec-based model on rank 26. The meta-model does not outperform the individual models. Given the computational effort, the meta-model is not a reasonable choice for this dataset and task and will therefore not be analyzed any further in the exploratory evaluation.

Explorative Analysis

In this section, the best parameter configurations of each representation learning method are analyzed exploratively. We therefore show the latent space of each model on transaction level and discuss data characteristics, which can be recognized within the latent space. Fig. 7.4 shows the latent representations of all test

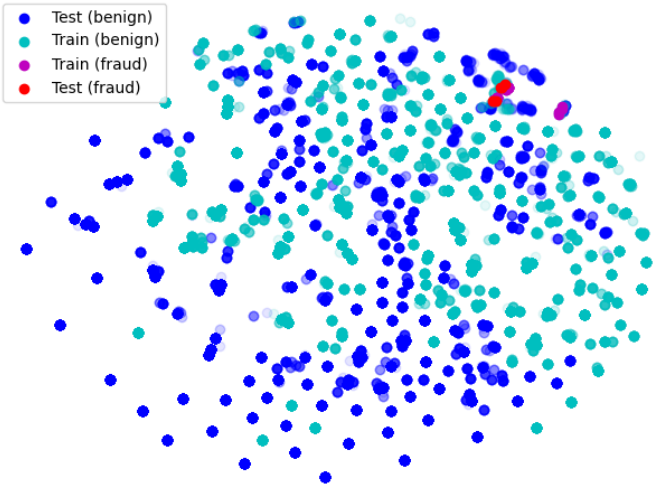


Figure 7.4: t-SNE visualization of 8-dimensional GloVe representations, trained with 5 bucket QOQD discretization and transaction context on the complete dataset (best model regarding F1). The benign transactions from train (cyan) and test (blue) splits form mostly homogeneous clusters. However, the anomalous fraud samples map well between train and test splits.

and train transactions of the best model according to the extrinsic evaluation with F1 score, i.e., QOQD with 5 buckets, 8 dimensional representations learned with GloVe on the complete dataset and transaction context. Benign train (cyan) and test (blue) transactions form mostly homogeneous clusters, which are not well separated from each other. However, some anomalous fraud cases from train and test splits are placed in direct neighborhood, which explains the good performance in extrinsic classification. Regarding the ROC-AUC score, besides GloVe models giving the best results on average, other representation learning approaches outperform the baseline model as well and therefore might be worth to consider for the explorative analysis. Fig. 7.5 shows the latent representations of the best FastText (a), Paragraph2Vec (b), Word2Vec with CBOW (c) and Word2Vec with SkipGram (d) models. Interestingly, the FastText and W2V-SkipGram models yield very similar latent spaces, although the FastText Model could make use of character n-grams, which encode the feature name along with its respective ordinal bucket number. For both models, the train and test datasets are well separated, but similarly structured, with the fraud samples mostly placed within one cluster. The Paragraph2Vec representation is much more scattered and lacks a competitively distinct structure. Especially the fraud cases of the training data are not visually separable from the most of the data well, whereas the most of the test-fraud cases are placed in a less dense area, which is better separable. This structure might explain why the one-class approach works best with Paragraph2Vec embeddings. The latent space of Word2Vec with SkipGram shows

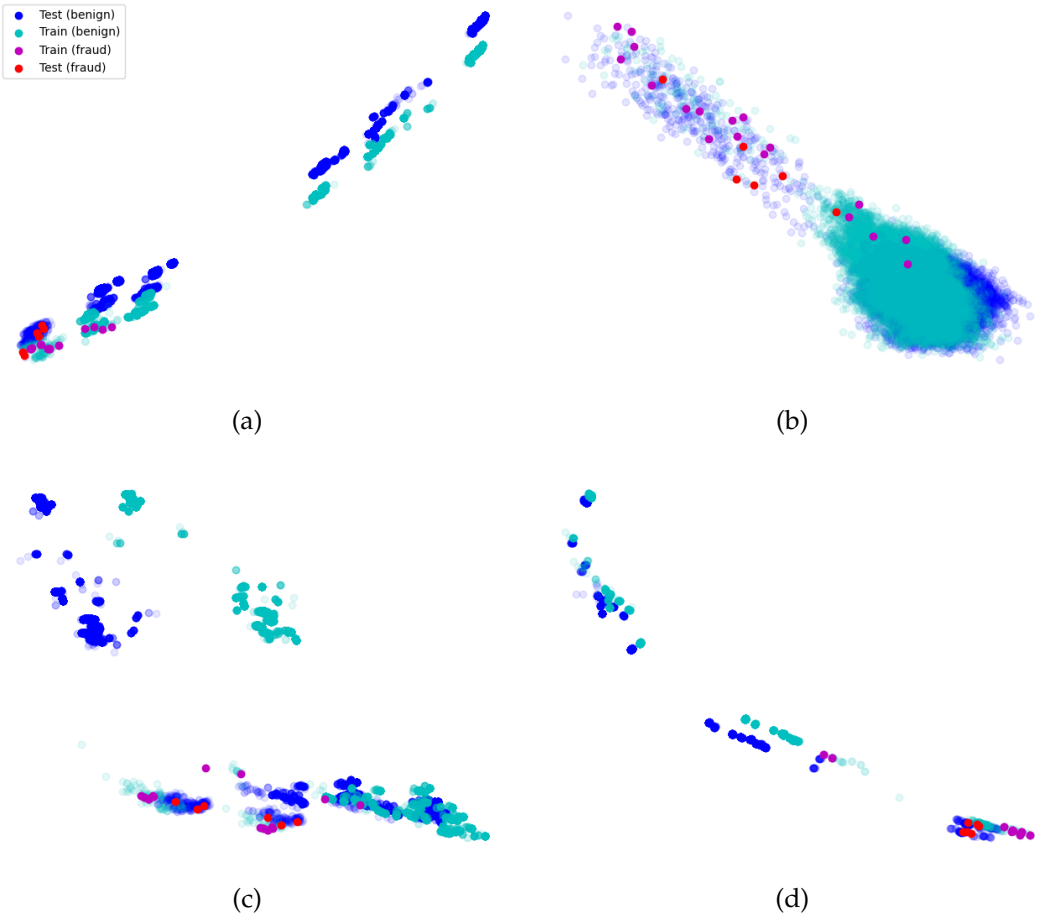


Figure 7.5: Visualization of two-dimensional (a) FastText representations with 10 bucket-QOQD, (b) Paragraph2Vec with 5-bucket-quantile, (c) W2V (CBOW) with 5-bucket-quantile, (d) W2V (skipgram) with 5-bucket-uniform (best models regarding ROC-AUC for each representation learning approach), for benign and fraud samples on both data splits.

similarities with its CBOW variant that isolates training and test data well. Especially the fraud cases of both datasets seem to be clustered very compactly for each dataset split. Overall, the visual inspection of the latent spaces regarding fraudulent and benign samples does not clearly reflect the advantage of the GloVe model (cf. Fig. 7.4) shown by quantitative evaluation, which suggests that the t-SNE projection conceals specific characteristics supporting the distinction of fraud and benign samples in higher dimensional space.

Fig. 7.6 shows the latent spaces of all transactions in the train and test dataset colored by transaction type. Transaction type can be considered one of the most relevant distinctions for different business cases and associated transaction logs such as incoming and outgoing invoices and credit notes, material receipt and issues and general ledger account postings. The transaction type is not directly present as a feature, but rather identified as a combination of features and their values, which has been annotated by domain experts to allow for reliable and fast identification of flows of goods or money for further analysis. Transaction type as explicit feature is thereby not included in the dataset, i.e. the structure and semantics emerging from the latent space, which corresponds to the transaction type quite well, is extracted by the representation learning approach from the raw features.

The parallel structure of training and test splits, which has already been discussed for the distinction of benign and fraudulent transactions, is also reflected for transaction type. FastText and Word2Vec reveal homogeneous clusters for each transaction type next to each other for training and test, with small overlaps for G/L postings, material issues and material receipts, which can be considered semantically similar from a domain perspective. Paragraph2Vec and GloVe representations are less separated in their latent space and while GloVe representations with 8 or 16 dimensions still form mostly disjunctive subspaces in their two-dimensional t-SNE projection, the larger proportion of Paragraph2Vec encoded transactions are clouded in indistinguishable overlapping structures. This supports the finding of non-competitively performing models with Paragraph2Vec (best model on rank 983, see Table 7.3). Overall, the FastText and Word2Vec models show distinct and semantically useful clusters. The quantitatively outstanding GloVe models show a similar semantic structure, however, they are not visually well-separable possibly due to their two-dimensional t-SNE projection.

7.2.3 Discussion

In contrast to our representation learning experiment on Windows audit logs, where FastText showed the most meaningful latent structure regarding the distinction of attack and non-attack, for SAP transactions, FastText could not outperform the other models. This finding is not surprising, as FastText's most prominent advantage is the use of subword strings, which in case of path names for Windows audit logs

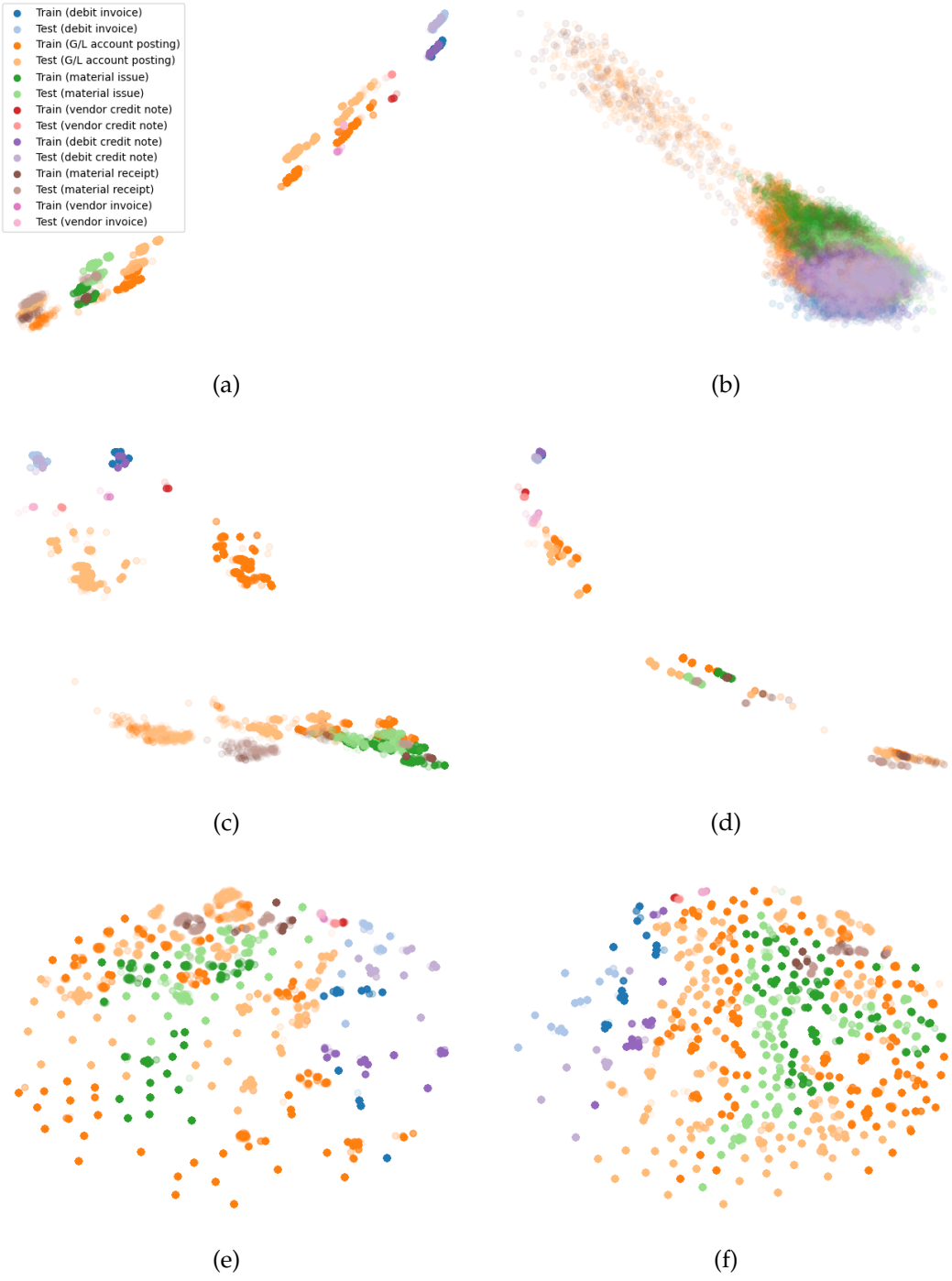


Figure 7.6: Visualization of (a) FastText representations with 10 bucket-QOQD, (b) Paragraph2Vec with 5-bucket-quantile, (c) W2V (CBOW) with 5-bucket-quantile, (d) W2V (skipgram) with 5-bucket-uniform, (e) GloVe with 10-bucket-QOUD (best ROC-AUC), (f) GloVe with 5-bucket-QOQD (best F1) colored according to transaction type on both data splits.

contain an inherently meaningful structure of sub-paths, file endings, and naming conventions.

In the case of SAP transaction logs, no features with comparable inherent structures could be identified. In fact, categorical values are often single characters, denoting abbreviations or the presence or applicability of a feature without any feature-context-free meaning. For this, we included the feature name along with each value to be able to distinguish the meaning of, e.g., "X" for the presence of two different features such as *Line item display possible=X* or *Indicator: Item cannot be copied=X*. For the training of token-based embeddings (i.e. all approaches except FastText), the nominally equivalent value for categorical features is distinguishable. For FastText, subtokens can be mapped, which might be useful if two features have semantically related feature names or values, but could also be obstructive if unrelated feature sub-words are unified. Our analysis suggests that incorporating subword tokens seems to be slightly beneficial regarding the extrinsic evaluation with FastText outperforming both W2V approaches, and negligible impact regarding the exploratory analysis, both yielding a very similar structure of the latent space.

7.2.4 Conclusion

Overall, our experiments suggest that by using word-embedding techniques, meaningful representations for transaction data can be learned. For this, we evaluated two different application settings, Windows audit logs and SAP transaction logs, discussed their respective requirements, and proposed adaptations to learn representations for each application domain. In an explorative analysis of the latent spaces, we focused on the semantic structure, which is revealed by the embedding spaces for malicious executions on the one hand and fraud/non-fraud as well as transaction types on the other hand. While the extrinsic evaluation in the Windows audit log domain could not show a strong benefit of one representation technique in favor of the others regarding their detection capabilities, for SAP transaction logs GloVe yielded the best results providing a strong baseline for fraud detection even with simple classification approaches such as kNN.

Chapter 8

Anomaly Detection and Applications in Transaction Fraud Detection

In this section, we will evaluate different anomaly detection approaches based on our findings from previous chapters in various scenarios for fraud detection. Our focus hereby is on the three possible formulations of fraud detection as supervised, semi-supervised, and unsupervised machine learning problems. We evaluate our model in scenarios of different complexities and examine the advantage or disadvantage of few but labeled training data over a large amount of (partially) unlabeled or contaminated training data.

First, we propose a neural network layer based on our iNALU architecture and evaluate on several financial fraud detection datasets in a supervised scenario with balanced class distributions. Second, we adapt our architecture to detection anomalies on the feature-rich SAP dataset (see Section 5.1.4) and evaluate the influence of different preprocessing decisions. We then approach anomaly detection as an unsupervised task by incorporating our network layer in an Auto-Encoder-alike architecture and evaluate its performance based on an anomaly score. We further investigate several approaches to use our generative models proposed in Section 6.2 for anomaly detection and evaluate their benefit for detection fraud in transaction data.

Parts of this chapter have been published as *Schlör, D., Ring, M., Krause, A., and Hotho, A. (2020b). Financial Fraud Detection with Improved Neural Arithmetic Logic Units. Fifth Workshop on MIning DAta for financial applicationS* [277].

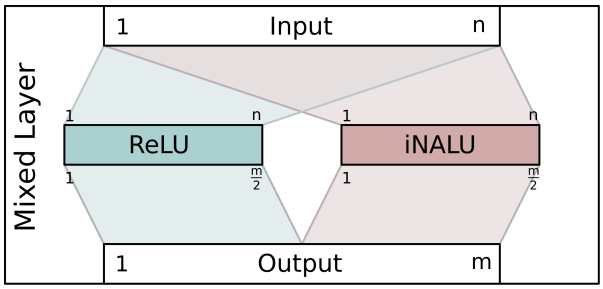


Figure 8.1: Our proposed mixed layer consisting of ReLU and iNALU neurons

8.1 iNALU Driven Mixed Layer Architecture for Fraud Detection

The presence of mathematical relationships between features is a well-known fact in many financial settings [32, 190]. For example, PaySim [190] is a mobile money simulator for fraud detection that generates bank transfers including the transmitted amount, the old account balance, and the new account balance as features. In this setting, these three features are closely related to each other in a mathematical sense. Although neural networks are well suited for many complex data mining tasks, they often have problems with the calculation of even basic mathematical operations [317].

Although such relationships are not always directly related to the downstream task to which the machine learning model is applied, a neural network architecture capable of capturing such relations is able to model the data inherently better [317]. In Section 6.1 we introduced the iNALU as a neuron specifically designed to inherently model mathematical operations within neural networks. In this section, we examine the research question if the introduction of iNALUs can improve the performance of neural networks in the task of detecting financial fraud.

Since financial fraud detection is more complex than modeling mathematical relationships in the data, we therefore propose a novel *Mixed Layer* architecture (see Fig. 8.1) that incorporates ReLUs as general-purpose neurons and iNALU neurons to capture arithmetic relationships within the data. For our experiments, we focus on two synthetic datasets, Credit and PaySim, and two real-world datasets, CCFraud and IEEE-CIS summarized in Table 8.1 and described in more detail in Section 5.1. In the first experiment, we evaluate variants of our proposed model in a supervised, class-balanced scenario each in comparison to vanilla feed-forward networks with comparable network structure. In the second experiment, we compare the best models with several standard classifiers commonly used.

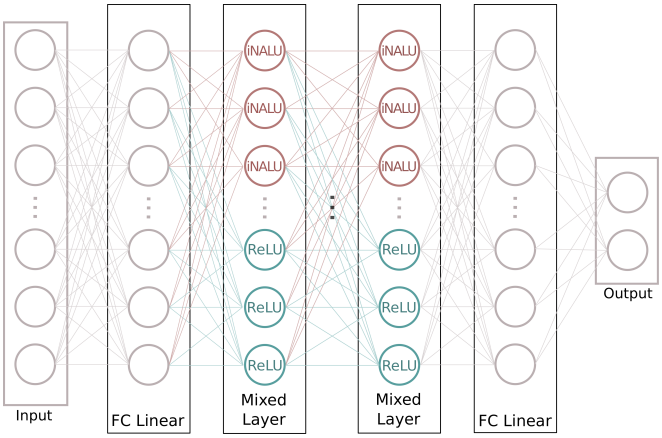


Figure 8.2: Mixed Layer network model with fully connected linear input and output layers and 1 to k Mixed Layers as used in our experiments

8.1.1 Mixed Layer model

The improved Neural Arithmetic Logic Unit (iNALU) as introduced in Section 6.1 is a neuron that by design can model simple arithmetic relationships. More complex relationships can be learned by stacking multiple layers of iNALUs for a deeper network. However, even in the financial domain, real-world datasets generally contain more than mathematical relationships. Therefore, we propose a model with each layer containing 50% general purpose non-linear hidden units (ReLUs, to be precise) and 50% iNALUs. In particular, the iNALU part and the ReLU part of each layer have the full input dimension n as input and contribute with an output dimension of $\frac{m}{2}$ concatenated to an output dimension of m for the complete layer (see Fig. 8.1). In combination with a linear layer as input and output layer, the network can thereby "route" and combine any input dimension to every part of each network layer by learning the weights accordingly. Therefore, the model is able to represent arithmetic and non-arithmetic feature relationships of varying complexity. We refer to the resulting model as shown in Fig. 8.2 as a neural network with Mixed Layers (*Mixed Layers model* for short).

Table 8.1: Main characteristics of the datasets

Dataset	Features	Samples	Benign	Fraud	Fraud-ratio	Origin
Credit	4	100 000	98 967	1033	0.010	synth.
PaySim	11	6 362 620	6 354 407	8 213	0.001	synth.
CCFraud	30	284 807	284 315	492	0.002	real
IEEE-CIS	431	590 540	569 877	20 663	0.035	real

8.1.2 Experimental setup

Architecture All experiments involve supervised training of an MLP as basic neural network architecture with linear input, non-linear dense layers with ReLUs and a linear output layer. The number of neurons in the hidden layers varies during the experiments. To investigate the research question RQ 3.1 if introducing iNALUs can improve the performance of neural networks on the task of financial fraud detection, we use the same architecture and replace the non-linear dense layers with Mixed Layers.

Train-Test Split For all experiments, we use the same strategy to generate the train-test split: For training, we randomly choose only few instances of the fraud class in order to keep the majority of fraudulent instances for evaluation. This approach reflects the class imbalance of the available data and emphasizes the requirement for a model to generalize from very few fraudulent samples to find new fraudulent cases when applied in a real-world scenario. In a preliminary study, we found that under-sampling the majority class to some extent did not affect the performance negatively but reduced training time by large margin. Therefore, only a random subset of the instances is used for training: For Credit, PaySim and CCFraud we use 2 000 instances with a fraud proportion of 1%, for IEEE-CIS we use 5 000 instances with a fraud proportion of 4%. We then use the Synthetic Minority Oversampling TEchnique (SMOTE) [51] to synthesize a balanced training dataset.

For the test dataset, we create a balanced split of 50% fraud and 50% benign samples, exclusively containing instances which haven't been used for training. This is motivated by the objective of studying the ability to capture mathematical relations rather than investigating the effects of predicting strongly unbalanced data. To represent the variety of benign instances and avoid skewed results due to random fluctuations, we repeat this process 5 times with different random seeds and report the average F1 score for the fraud class and for experiment 2 the ROC-AUC additionally.

Preprocessing Each dataset is preprocessed by one-hot encoding categorical values. To ensure comparability between all datasets, we follow the preprocessing strategy of the CCFraud dataset and apply PCA to all other datasets as well as min-max scaling to all datasets ensuring a valid train-test split by only fitting on training data. In a preliminary study, we verified that applying PCA to the datasets does not negatively impact the performance of neural networks with and without Mixed Layers. Applying PCA can also mitigate privacy issues that could possibly prevent making a real dataset publicly available.

Training Procedure For neural network training, we use ADAM [165] as optimizer with a learning rate of 10^{-3}, a weight decay of 10^{-4} and Cross Entropy loss. The

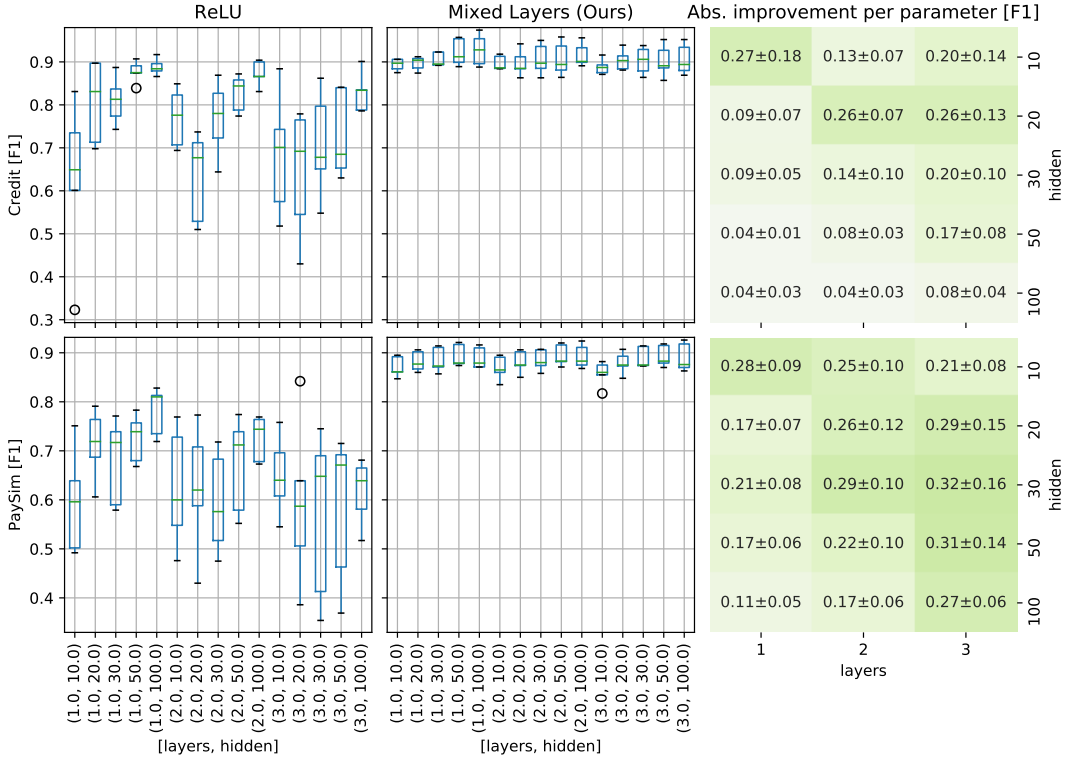


Figure 8.3: F1 scores of experiment 1 on the synthetic Credit Payment and PaySim datasets. Mixed Layers describes the neural network structure as described in Section 8.1.1 and ReLU shows the results for the same model architecture with respect to number of layers and hidden neurons having Mixed Layers replaced by layers with ReLU activations. The heatmaps show the absolute improvement of Mixed Layers compared to the respective ReLU architecture for each parameter configuration averaged over all random seeds and their standard deviations.

batch size is set to 200 and all models are trained for 200 epochs, which have been validated as suitable training parameters in preliminary experiments.

8.1.3 Experiment 1

In the first experiment, we explore the influence of Mixed Layers in neural network architectures. Therefore, we construct neural networks containing Mixed Layer as well as neural networks of identical architecture exclusively with dense layers and ReLU activations. For each dataset all possible combinations between the number of input neurons (chosen from 10, 20, 30, 50 and 100) and the number of layers (chosen from 1 to 3) are evaluated to assess the performance and stability for different neural network capacities.

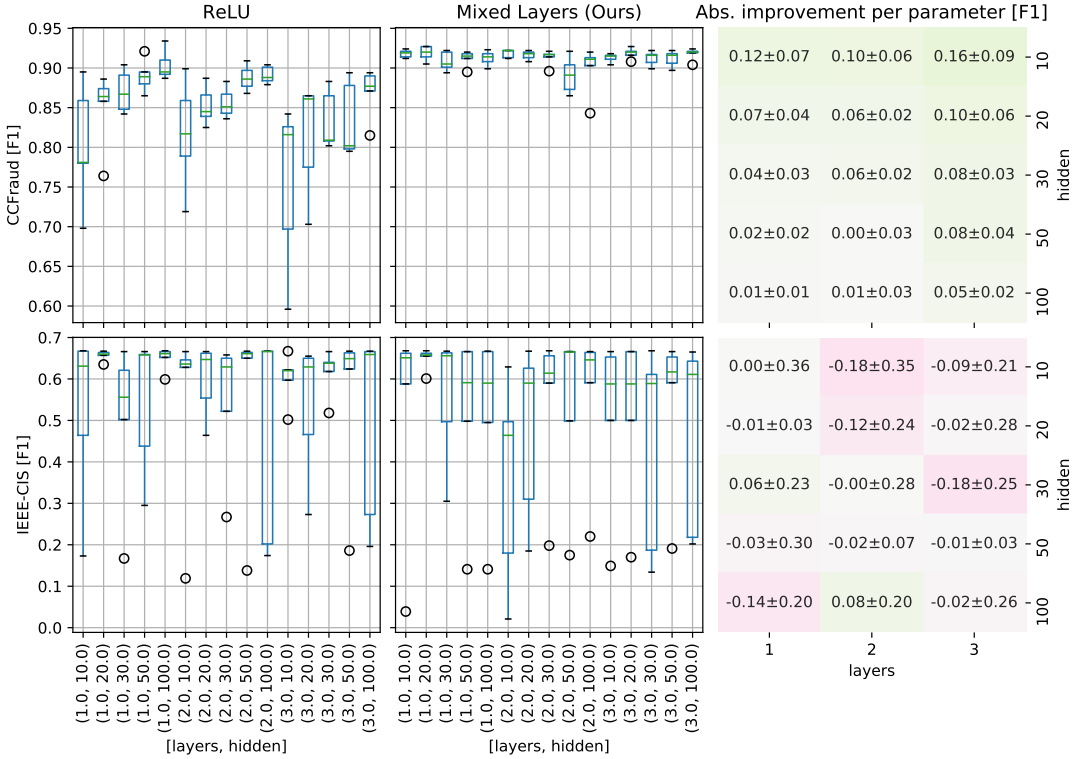


Figure 8.4: F1 scores of experiment 1 on the real CCFraud and IEEE-CIS datasets. Mixed Layers describes the neural network structure as described in Section 8.1.1 and ReLU shows the results for the same model architecture having Mixed Layers replaced by dense layers with ReLU activations. The heatmaps show the absolute improvement of our architecture in comparison to the respective ReLU architecture for each parameter configuration averaged over all random seeds and their standard deviations.

Results The results are depicted in Fig. 8.3 and Fig. 8.4. The boxplots show the F1 scores on our test datasets for both neural network architectures for a different number of layers and a varying number of hidden neurons in each layer. Each box summarizes the results of all runs for a certain parameter configuration with different random seeds. Note that both architectures share the same train- and test-splits for each run and parameter configuration. This ensures the comparability of the underlying performances and boxes for each parameter. For all dataset except IEEE-CIS, our proposed model yields very good results with F1 scores around 0.9. The performance on the IEEE-CIS dataset is notably worse for both architectures, and the results vary highly for different random seeds. As this dataset is much more complex regarding the number and kind of features, the task might require a different training strategy to achieve better results.

For all other datasets, the performance of our model is very stable over all parameter configurations, which means that even a very small neural network consisting of one Mixed Layer and 10 hidden neurons (along with a linear input and output layer) solves the task sufficiently well. In comparison, the same architecture with one dense layer and ReLU activations performs notably worse. To assess the actual performance improvement for each possible parameter configuration, we aligned the parameters and seeds of both architectures pairwise and report the average absolute improvement per parameter configuration in Fig. 8.3 and Fig. 8.4. A positive improvement value describes a performance gain of Mixed Layers over the respective ReLU model, whereas a negative improvement means that the ReLU model performs better. The Mixed Layer model outperforms the ReLU model for all datasets except IEEE-CIS. For Credit and PaySim, this observation holds for all network configurations, whereas for the CCFraud dataset large models (50 and 100 neurons) with one or two layers perform equally well.

8.1.4 Experiment 2

In the second experiment, we want to compare our model with several commonly used supervised classification algorithms. To be precise, we evaluate linear Support Vector Machine (SVM), Support Vector Machine (SVM)-RBF, k-Nearest Neighbors (kNN)[1], Decision Tree (DT), Random Forest (RF), Naïve Bayes (NB), Logistic Regression (LR) and XG-Boost. With IF, we also include an anomaly detection method for comparison.

For the ReLU model, we use the most promising parameter configuration of experiment 1, which is one layer with 100 neurons. For the Mixed Layers model, we use one layer with 20 neurons. We want to emphasize that due to the good stability of Mixed Layers over different parameter configurations, the parameter choice for Mixed Layer in this experiment is arbitrary, and other parameter configurations perform comparably.

Results The results of the second experiment are presented in Table 8.2. Our model performs the best on average regarding F1 and is among the best four classifiers for all datasets individually. All classifiers perform well on each dataset except IEEE-CIS, on which the best models except IF only achieve F1 scores of 0.65. IF performs best on this dataset with an F1 score of 0.74, suggesting that methods specifically tailored to anomaly detection can capture the characteristics of this dataset better. Comparing the F1 score averaged over all datasets (see Table 8.2, column Avg.), the Mixed Layer architecture yields significantly[2] better results compared to the best ReLU architecture. The results evaluated with the ROC-AUC metric support our

[1] $k = 3$

[2] $p = 0.00932$, Wilcoxon Signed-Rank Test over all datasets and repetitions

findings with the exception of IEEE-CIS, where our Mixed Layers performed worse than the ReLU layer. An in-depth analysis showed that two of the five repetitions with different random seeds yield notably worse results (0.41 and 0.44) which leads to a performance drop for the mean and the high standard deviation.

Table 8.2: Average and standard deviation F1 score and ROC-AUC for several supervised classifiers compared to our model aggregated over different random seeds. For ReLU and our model, we conducted this experiment using the most promising parameter configuration from experiment 1, one layer with 100 neurons for ReLU and one layer with 20 neurons for our Mixed Layer model. The last column shows the average for each classifier over all datasets. The best results per dataset are printed in bold.

	Method	Credit	PaySim	CCFraud	IEEE-CIS	Avg.
F1	SVM	0.88 ± 0.01	**0.89** ± 0.02	0.90 ± 0.01	0.41 ± 0.27	0.77
	SVM-RBF	**0.92** ± 0.03	0.85 ± 0.02	0.85 ± 0.07	0.43 ± 0.23	0.76
	kNN	0.89 ± 0.03	0.81 ± 0.01	0.91 ± 0.01	0.65 ± 0.03	0.81
	DT	0.87 ± 0.02	0.82 ± 0.03	0.80 ± 0.08	0.49 ± 0.04	0.74
	RF	0.89 ± 0.03	0.83 ± 0.03	0.88 ± 0.02	0.57 ± 0.05	0.80
	NB	0.85 ± 0.03	0.76 ± 0.05	0.91 ± 0.01	0.65 ± 0.02	0.80
	LR	0.88 ± 0.01	0.87 ± 0.02	**0.92** ± 0.01	0.55 ± 0.15	0.81
	XG-Boost	0.91 ± 0.02	0.84 ± 0.03	0.89 ± 0.01	0.62 ± 0.01	0.82
	IF	0.82 ± 0.01	0.81 ± 0.01	0.88 ± 0.01	**0.74** ± 0.01	0.81
	ReLU	0.89 ± 0.02	0.78 ± 0.04	0.90 ± 0.02	0.65 ± 0.03	0.81
	Mixed Layers	0.90 ± 0.01	0.88 ± 0.02	**0.92** ± 0.01	0.65 ± 0.02	**0.84**
ROC-AUC	SVM	0.95 ± 0.01	0.97 ± 0.01	0.95 ± 0.01	0.49 ± 0.12	0.84
	SVM-RBF	**0.98** ± 0.01	0.96 ± 0.01	0.93 ± 0.05	0.59 ± 0.09	0.86
	kNN	0.91 ± 0.02	0.86 ± 0.02	0.92 ± 0.01	0.54 ± 0.05	0.81
	DT	0.88 ± 0.02	0.85 ± 0.02	0.83 ± 0.05	0.52 ± 0.08	0.77
	RF	**0.98** ± 0.01	0.95 ± 0.01	**0.97** ± 0.00	0.48 ± 0.04	0.85
	NB	0.93 ± 0.02	0.92 ± 0.01	0.94 ± 0.03	0.50 ± 0.01	0.82
	LR	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.45 ± 0.09	0.83
	XG-Boost	**0.98** ± 0.02	**0.98** ± 0.01	**0.97** ± 0.00	0.54 ± 0.01	0.87
	IF	0.95 ± 0.01	0.90 ± 0.01	0.95 ± 0.00	**0.74** ± 0.01	**0.89**
	ReLU	0.96 ± 0.01	0.88 ± 0.02	0.94 ± 0.01	0.64 ± 0.02	0.85
	Mixed Layers	0.97 ± 0.00	0.96 ± 0.01	0.96 ± 0.01	0.57 ± 0.12	0.87

8.1.5 Discussion

In our experiments, we show that neural networks benefit from including Mixed Layers when applied to the task of fraud detection on four different datasets. This finding suggests that Mixed Layers improve the ability of neural networks to model mathematical relationships within the data. The neural network architectures containing our mixed layers have good stability over different numbers of hidden neurons and layers. Even small networks yield competitive results with several well-established supervised classification algorithms over different synthetic and real-world datasets. Overall, the ReLU architecture seems much less stable with respect to different random seeds and parameter configurations.

Although our model was among the best classifiers for the IEEE-CIS dataset in experiment 2, all methods performed notably worse compared to the other datasets. This shows that the dataset is hard to predict in our evaluation setting and suggests that the unstable performance observed in experiment 1 is presumably not related to our model but rather to the dataset itself and might be explained by different aspects: On the one hand, the dataset includes many features, which might require intensive prepossessing and feature engineering. On the other hand, the dataset is larger than all other datasets with respect to the number of features and fraud cases. This might require a different training procedure than the other datasets, for example regarding the train-test split, the network architecture, or the number of epochs for training, as recent works suggest yielding better results incorporating feature engineering [107, 74] or recurrent neural network architectures [213]. The observation that Isolation Forest yields notably better results on IEEE-CIS also suggests that for more complex datasets methods adapted for anomaly detection should be used instead of standard classifiers. Since this study is not primarily conducted to achieve best performances on each dataset but rather to examine the research question, if neural network architectures benefit from iNALU neurons applied to the financial domain, our experiments focused on a fair comparison instead of thorough hyper-parameter tuning on individual datasets.

On our synthetic Credit Payment dataset, some supervised classifiers performed surprisingly well, which by design of our dataset we did not expect. Since solving the task correctly is fully dependent on capturing the correct mathematical relationship, we expected, for example, kNN to perform worse. We suspect that the good performance is a result of applying PCA as a preprocessing step, which might contribute to modeling the correct relation within the data.

Both experiments show that our model outperforms the respective ReLU baseline models. However, Mixed Layers with iNALUs contain more trainable parameters than linear neurons with ReLU activations. One Mixed Layer in our experiments contains 50% ReLUs and 50% iNALUs and an iNALU has 4 times more trainable parameters. For a comparison of two architectures with an equal number of trainable parameters, an architecture with Mixed Layers with a hidden dimension of 20

can be compared with an architecture with ReLU Layers with a hidden dimension of 50. In this comparison (see Fig. 8.3 and Fig. 8.4) the Mixed Layers model still outperforms the ReLU-based model with an F1 score of 0.837 over 0.759, averaged over all datasets.

8.1.6 Conclusion

These experiments examined the question whether a neural network architecture that includes hidden units specifically tailored to capture mathematical relations is beneficial for supervised classification tasks on datasets in the financial fraud domain.

We designed a novel Mixed Layer for neural networks that incorporates iNALUs and ReLUs. We evaluated Mixed Layer-based neural networks on two real-world and two synthetic datasets and compared them to neural networks with ReLU activations. The experiments show that Mixed Layers are able to improve the performance of neural networks on financial fraud datasets. We compared our proposed model with several well-established classification approaches in a supervised evaluation setting and found that it belonged to the best approaches for each dataset.

Since we designed our baseline architecture as well as our proposed model to solve a supervised task, we constructed our experiments accordingly and evaluated them on a balanced dataset. However, in a real-world setting, fraud and benign transactions will not occur equally frequent, and the actual financial prejudice of having more false positives or more false negatives will be task-dependent and require a more detailed evaluation including metrics to reflect these circumstances. Moreover, many approaches applied to recognize financial fraud rely on anomaly-detection or novelty-detection techniques, which often use one-class or even unsupervised approaches. As we generally showed the benefit of our proposed model in the supervised setting, in the next sections we introduce it in other evaluation settings and compare it to unsupervised approaches, as well as approaches specifically designed for anomaly or novelty-detection.

8.2 Anomaly Detection with Mixed Layers on SAP Data

In previous experiments, we evaluated the benefit of introducing iNALUs in a supervised setting with a balanced test set class distribution. The prerequisite for this evaluation setting is a large dataset including a sufficient number of fraud cases, which in general is not given for real-world applications. In this experiment, we focus on an anomaly detection evaluation setup where fraud cases are as rare as 0.3‰ for the SAP dataset, which translates to 19710 benign and 6 fraud cases. Besides the model performance, this skewed class distribution is also challenging with respect to the interpretation of the evaluation metrics.

As discussed in Section 4.5.2, all metrics have individual advantages and drawbacks, and especially with respect to class imbalance the question which metric is most suitable depends on the application-specific valuation of false positives versus false negatives. We therefore report all metrics, precision, recall, F1, AP and ROC-AUC and discuss the confusion matrices for selected models in detail.

As first experiment, we evaluate the Mixed Layer model in a supervised scenario, i.e., by classifying fraud and non-fraud to confirm our findings from Section 8.1 on the feature-rich and imbalanced SAP dataset. The second experiment then focuses on unsupervised training and incorporates our Mixed Layers in an Auto-Encoder architecture for reconstruction error-based anomaly detection evaluating a real-world scenario without labeled anomalies as training data.

8.2.1 Supervised Anomaly Detection with Mixed Layers

For the experimental setup and training procedure, we follow Section 8.1.4. We evaluate the same supervised baseline methods and with One-Class SVM (OC-SVM), we include besides Isolation Forest (IF) a second anomaly detection approach trained on benign data. Due to the very few cases of fraud in comparison to benign data, we refrain from a cross-validation-based evaluation and use the three subsets of the SAP dataset as follows: The benign subset is used as a training dataset for unsupervised approaches. The first subset including 18 fraud cases is used as training data for the supervised approaches and the second subset including 6 fraud cases is used as test dataset for all approaches, i.e. the supervised as well as the one-class approaches. We use the hyperparameters from Section 8.1.4 for a fair comparison between anomaly detection and supervised approaches, since we do not have sufficient fraud cases for a validation split. For all supervised approaches, we apply SMOTE [51] to synthesize a balanced training dataset.

For preprocessing, we evaluate different approaches: In addition to min-max scaling and PCA processed data (as given in Section 8.1 with the CCFraud dataset for anonymization), we also conduct our experiments without PCA and with z-score normalization and discretization of numeric attributes with QOQD (see Section 7.2.1) to examine the impact of different preprocessing decisions across various approaches for the SAP dataset.

Results

Tables 8.3 and 8.4 show the results for all preprocessing variants for numeric data with and without PCA. Regarding the metrics, there are some interesting observations: Consider, for example, Table 8.3 and XG-Boost with PCA. Despite the model failing completely and classifying all samples as non-fraud, the ROC-AUC (0.990) does not reflect this bad performance since classifying the 6 fraud samples incorrectly is not significant in contrast to the 19 710 correctly predicted non-fraud

Table 8.3: Models with min-max and z-score normalization with and without PCA evaluated with different metrics on the anomaly detection task of the SAP fraud dataset. Best results per metric are written in bold. The highlighted rows denote well-performing models over all metrics which are discussed in detail. Mixed Layers perform among the best models and notably better than the ReLU model.

		PCA					no PCA				
	Method	↑ Prec.	↑ Rec.	↑ F1	↑ AP	↑ ROC-AUC	↑ Prec.	↑ Rec.	↑ F1	↑ AP	↑ ROC-AUC
z-score	SVM	0.000	0.000	0.000	0.052	0.333	0.000	0.000	0.000	0.333	0.333
	SVM-RBF	0.000	0.000	0.000	0.008	0.980	0.000	0.000	0.000	0.007	0.979
	kNN	0.000	0.000	0.000	0.000	0.499	0.000	0.000	0.000	0.000	0.499
	DT	0.000	0.000	0.000	0.000	0.500	0.000	0.000	0.000	0.000	0.500
	RF	0.000	0.000	0.000	0.337	0.914	0.000	0.000	0.000	0.285	0.915
	NB	0.061	0.667	0.111	0.086	0.672	0.020	**1.000**	0.039	0.510	0.992
	LR	0.000	0.000	0.000	0.004	0.332	0.000	0.000	0.000	0.004	0.333
	XG Boost	0.000	0.000	0.000	0.025	0.990	0.000	0.000	0.000	0.118	0.991
	OC-SVM	0.001	0.667	0.001	0.667	0.673	0.001	**1.000**	0.001	0.605	0.994
	IF	0.143	**1.000**	0.25	0.712	0.999	0.005	**1.000**	0.009	0.023	0.994
	ReLU	0.167	0.667	0.267	0.667	0.825	0.167	0.667	0.267	0.671	0.994
	Mixed Layers	0.400	**1.000**	0.571	**1.000**	**1.000**	0.067	0.667	0.121	0.683	0.998
MinMax	SVM	0.000	0.000	0.000	0.000	0.000	0.011	0.333	0.020	0.106	0.665
	SVM-RBF	0.000	0.000	0.000	0.000	0.002	0.014	0.667	0.028	0.009	0.826
	kNN	0.000	0.000	0.000	0.000	0.497	**1.000**	0.833	0.909	0.840	0.916
	DT	0.012	0.167	0.023	0.173	0.663	0.000	0.000	0.000	0.000	0.496
	RF	0.000	0.000	0.000	0.014	0.989	0.000	0.000	0.000	0.016	0.910
	NB	0.000	0.000	0.000	0.000	0.001	0.010	0.333	0.019	0.172	0.661
	LR	0.000	0.000	0.000	0.500	0.500	0.017	0.667	0.034	0.108	0.952
	XG Boost	0.000	0.000	0.000	0.011	0.988	0.000	0.000	0.000	0.008	0.982
	OC-SVM	0.000	**1.000**	0.001	0.565	0.999	0.001	**1.000**	0.001	0.008	0.982
	IF	0.130	**1.000**	0.231	0.974	**1.000**	0.004	**1.000**	0.008	0.008	0.983
	ReLU	0.000	0.000	0.000	0.000	0.001	0.000	0.500	0.001	0.000	0.544
	Mixed Layers	0.000	0.000	0.000	0.167	0.168	0.011	0.333	0.02	0.175	0.979

↑: higher value is better

samples. This also shows another aspect of supervised AD and its metrics as the methods have to precisely model the decision boundary, indicating fraud or non-fraud when evaluating precision, recall and F1 whereas for AP and ROC-AUC, the boundary is not important as long as the fraud samples are scored higher than non-fraud samples with regard to class probability. This is, for example, the case for Mixed Layers with z-score normalization where all fraud samples are scored at top positions yielding an AP and ROC-AUC score of 1.0 while the decision boundary allows 9 false positives. For an application scenario that values fewer false positives over detecting all fraud cases, kNN with discretization is the best approach, detecting 5 of the 6 fraud cases correctly without false positives, overall yielding the best F1 score.

Table 8.4: Models with discretization and without scaling with and without PCA evaluated with different metrics on the anomaly detection task of the SAP fraud dataset. Best results per metric are written in bold. The highlighted rows denote well-performing models over all metrics which are discussed in detail.

		PCA					no PCA				
	Method	↑ Prec.	↑ Rec.	↑ F1	↑ AP	↑ ROC-AUC	↑ Prec.	↑ Rec.	↑ F1	↑ AP	↑ ROC-AUC
discretization	SVM	0.333	0.333	0.333	0.333	0.463	0.333	0.333	0.333	0.334	0.663
	SVM-RBF	0.000	0.000	0.000	0.377	0.675	0.000	0.000	0.000	0.373	0.673
	kNN	**1.000**	0.833	**0.909**	0.865	0.917	**1.000**	0.833	0.909	0.865	0.917
	DT	0.000	0.000	0.000	0.000	0.500	0.500	0.333	0.400	0.417	0.667
	RF	0.000	0.000	0.000	0.488	0.999	0.000	0.000	0.000	0.494	0.916
	NB	0.000	0.000	0.000	0.000	0.137	0.000	0.000	0.000	0.000	0.500
	LG	0.250	0.333	0.286	0.358	0.772	0.250	0.333	0.286	0.358	0.772
	XG Boost	0.000	0.000	0.000	0.036	0.995	0.333	0.333	0.333	0.346	0.667
	OC-SVM	0.001	**1.000**	0.001	0.129	0.997	0.000	**1.000**	0.001	0.02	0.989
	IF	0.034	**1.000**	0.066	0.467	**1.000**	0.005	**1.000**	0.011	0.02	0.993
	ReLU	0.001	**1.000**	0.001	0.021	0.983	0.001	**1.000**	0.003	0.009	0.927
	Mixed Layers	0.333	0.333	0.333	0.336	0.818	0.333	0.333	0.333	0.264	0.767
no scaling	SVM-RBF	0.019	0.667	0.036	0.009	0.828	0.000	0.000	0.000	0.008	0.665
	kNN	0.059	0.833	0.110	0.028	0.913	0.014	0.667	0.028	0.015	0.910
	DT	0.000	0.000	0.000	0.000	0.496	**1.000**	0.833	0.909	0.843	0.917
	RF	0.012	0.167	0.023	0.016	0.991	0.000	0.000	0.000	0.000	0.496
	SVM	0.011	0.333	0.020	0.186	0.649	0.000	0.000	0.000	0.017	0.91
	NB	0.000	0.000	0.000	0.000	0.234	0.013	0.333	0.025	0.173	0.663
	LR	0.017	0.667	0.034	0.190	0.952	0.016	0.667	0.031	0.012	0.986
	XG Boost	0.000	0.000	0.000	0.010	0.986	0.000	0.000	0.000	0.013	0.989
	OC-SVM	0.001	**1.000**	0.001	0.050	0.986	0.000	**1.000**	0.001	0.008	0.982
	IF	0.009	**1.000**	0.017	0.006	0.976	0.004	**1.000**	0.009	0.009	0.984
	ReLU	0.013	**1.000**	0.026	0.011	0.988	0.001	0.833	0.001	0.012	0.903
	Mixed Layers	0.011	0.333	0.020	0.010	0.981	0.000	0.000	0.000	0.009	0.824

↑: higher value is better

The anomaly detection approaches, IF and OC-SVM, reliably detect all fraud cases across most preprocessing variants. However, they are susceptible to false positives: In its best scenario, PCA with min-max scaling of numeric attributes, IF produces 40 false positives, classifying all 6 fraud cases as well as 19 670 non-fraud cases correctly. Without PCA, while stably predicting all frauds correctly, both yield more false positives. This also applies to Mixed Layers, which performs notably worse with z-score in comparison to the experiment with PCA. A notable exception is kNN, which performs well for min-max scaling, no scaling, and discretization.

Overall, the results of our experiment suggest that IF is a reliable choice for detecting all fraud cases over various preprocessing choices, however, this comes to the expense of more false positives in contrast to other approaches. As the model is purely trained on benign data, it can be applied in a real-world scenario with-

out known class labels easily to find anomalous behavior with respect to benign behavior of the past. However, this introduces the risk of accepting undiscovered anomalous behavior and less precise classification if contamination with anomalous samples cannot be completely ruled out for a real-world scenario. For example, the best isolation forest with 40 false positives and no false negatives (PCA and min-max) degrades to 278 false positives when trained on the supervised dataset which includes 18 fraud cases.

To summarize, our Mixed Layers model performed well with appropriate preprocessing choices of z-score normalization and PCA, ranking all 6 fraud cases on the top positions while the decision boundary can be learned sufficiently well, classifying all fraud cases correctly with 9 false positives. The ReLU network only detects 4 of the 6 fraud cases and yields 20 false positives. This shows the benefit of including the iNALU in this architecture for fraud detection even in an anomaly-detection evaluation setting without any tuning of the architecture for this specifically imbalanced task.

Second, kNN has proven useful with min-max scaling, and discretization, however, can be computationally expensive for larger datasets. Another drawback is the dependence on the metric, which is used to decide on the distance of two potential neighbors. The performance of kNN generally depends on the selection of an appropriate distance metric for a given dataset that must be subject to the optimization procedure [2]. For the application of kNN in this setting and on this dataset, the Euclidean distance works well and is thus a reasonable choice, however, euclidean distance is prone to the "curse of dimensionality" for a dataset with more features [6] and thus has to be reevaluated along other metric choices for the adaptation to other datasets.

Comparison of discretization strategies

The surprisingly good performance of kNN with discretization as preprocessing raises the question of to what extent these results depend on our quantile outlier QOQD discretization schema introduced in Section 7.2.1. To evaluate the influence of the discretization strategy, in this experiment, uniform and quantile discretization with and without the special encoding of outliers was applied and a kNN model was trained, which showed the most promising performances in the previous experiment.

Table 8.5 shows the results for the different discretization approaches. Including outlier bins improves the results by large margin in comparison to equal frequency quantile binning (quantile) and yields the best result for QOQD with no false positives and only one false negative. Binning by value (uniform) proves to be unsuitable with kNN missing all fraud cases. Including a larger number of bins (10) impairs the performance of the kNN classifier for all discretization strategies and generally introduces more false positives except for QOUD, which yields better results.

Table 8.5: Comparison of discretization strategies with and without PCA for 5 and 10 buckets for the kNN model. The quantile-outlier-quantile discretization (QOQD) strategy outperforms quantile and uniform by large margin. Quantile-outlier-uniform discretization (QOUD) performs slightly better than quantile and uniform discretization. PCA transformation has no notable influence.

	Discretization strategy	↑ Prec.	↑ Rec.	↑ F1	↑ AP	↑ ROC-AUC
PCA	QOQD 5 buckets	1.000	0.833	0.909	0.865	0.917
	QOQD 10 buckets	0.263	0.833	0.400	0.565	0.916
	quantile 5 buckets	0.106	0.833	0.189	0.231	0.915
	quantile 10 buckets	0.059	0.833	0.110	0.466	0.915
	QOUD 5 buckets	0.000	0.000	0.000	0.000	0.499
	QOUD 10 buckets	0.185	0.833	0.303	0.560	0.916
	uniform 5 buckets	0.000	0.000	0.000	0.000	0.499
	uniform 10 buckets	0.000	0.000	0.000	0.000	0.498
no PCA	QOQD 5 buckets	1.000	0.833	0.909	0.865	0.917
	QOQD 10 buckets	0.263	0.833	0.400	0.566	0.916
	quantile 5 buckets	0.111	0.833	0.196	0.234	0.915
	quantile 10 buckets	0.064	0.833	0.119	0.476	0.915
	QOUD 5 buckets	0.000	0.000	0.000	0.000	0.500
	QOUD 10 buckets	0.167	0.833	0.278	0.560	0.916
	uniform 5 buckets	0.000	0.000	0.000	0.000	0.499
	uniform 10 buckets	0.000	0.000	0.000	0.000	0.498

↑: higher value is better

The results of this experiment suggest that considering point anomalies for feature distributions in the discretization strategy helps to generalize over different datasets and helps to improve the classification performance of comparison-based approaches such as kNN. The difference between QOQD and QOUD however suggests that representing feature outliers by frequency alone isn't the key aspect for performance, as outliers are explicitly modeled by bins consistently for both approaches while the representation of non-outliers varies over both discretization schemes.

8.2.2 Autoencoding Mixed Layers

In the previous sections, we evaluated Mixed Layers in a supervised classification setting (Section 8.1.3) and in a supervised anomaly detection setting (Section 8.2). In this section, we formulate fraud detection without explicitly labeled positive samples

for training. This setting can be interpreted as semi-supervised anomaly detection, one-class, or unsupervised approach depending on the dataset and perspective, as discussed in Section 4.4 in depth. More precisely, we create Auto-Encoder-based models with and without Mixed Layers, which are trained to reconstruct the data. The reconstruction of samples is inherently given by the architecture as input of the encoder and output of the decoder. Input and output remark the same feature space and the model is precisely trained to reproduce the input at the output with a reconstruction objective, commonly ℓ^2 norm (see Eq. 4.45).

To avoid overfitting and learning the identity function, the number of parameters is limited in a bottleneck layer. By this, the NNs are encouraged to focus on characteristics which summarize a large number of instances on a low-dimensional manifold, since the precise reconstruction of a large number of instances will reduce the reconstruction error from a global perspective. Consequently, the precise reconstruction of samples that do not share the characteristics with the majority of *normal* data will fail. This reconstruction error, i.e., the difference between the input sample and the output of the AE, is used as anomaly score. As discussed in Section 4.4, it allows an evaluation ordered from the worst reconstructed (i.e., expectedly most anomalous) to the best reconstructed (i.e., expectedly most common) sample in an unsupervised manner without discretizing this continuum into anomalies and normal instances, for example by a threshold. Therefore, PR and ROC curves can be used as evaluation measures, while precision, recall or F_1 are unsuitable without specific task-dependent thresholds or subsequent supervised learning.

Auto-Encoder Models

AEs are NN models and consist of encoder and decoder layers. The bottleneck layer is, as the term suggests, the smallest layer, which is located at the end of the encoder. Deep AEs consist of several hidden layers on both sides of the bottle neck layer, which typically decrease in the encoder and increase in the decoder in terms of the number of neurons per layer, as already introduced in Section 6.2.1 for the special case of *Variational* Auto-Encoders. Fig. 8.6 shows the basic structure of a Mixed Layer AE while, in general, the number of layers and neurons per layer can vary. For more hidden layers, our AE model is constructed by reducing the number of neurons to 50% of the previous layer up to the bottleneck layer and symmetrically constructing the decoder with increasing layer sizes. We tested other architectural choices in preliminary experiments including Mixed Layers at different stages of the AE, e.g. after input layer, before output layer, at different hidden layer positions. However, the best performance was achieved by AEs using Mixed Layers as the bottleneck, which we therefore evaluate in detail.

Experimental Setup

For this experiment, we evaluate a basic AE and a Mixed Layer AE (ML-AE) model along with OC-SVM and IF on the SAP fraud dataset. This dataset contains an unlabeled benign subset of data and two labeled subsets including fraud data. As the ML approaches in these experiments are trained on benign data only, we use the unlabeled benign subset as training data and have two additional datasets for validation and testing, which allows us to find the optimal hyperparameters for each model on the validation split and evaluate on the test split. We therefore performed a grid-search over the hyperparameter space of each model. For OC-SVM we evaluated the linear and RBF kernel for parameters $\nu \in \{0.05, 0.2, 0.4, 0.6, 0.8, 0.95\}$ and for RBF $\gamma \in \{10^{-4,-3,\cdots,4}\}$. For ML-AE and AE we varied the number of layers for encoder and decoder from 1 to 5, the number of neurons in the bottleneck layer $m \in \{10, 20, \cdots, 100\}$, the learning rate between $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$ and the batch size between $\{8, 16, 32\}$. For IF, we evaluated $t \in \{2^{4,\cdots,10}\}$ trees and subsampling parameters $\varphi, \psi \in \{0.2, 0.4, 0.6, 0.8, 1\}$ and selected the best hyperparameter configuration based on the *eval* dataset on the r_{min} metric (cf. Eq. 8.1) detailed in the following paragraph.

The one-class experimental setup including the hyperparameter study allows evaluation with ROC-AUC and AP, however, F1 evaluation requires the definition of a threshold to assign class labels from the anomaly score. Since fraudulent samples are very rare, further splitting the labeled datasets to validate a threshold is not a feasible option. Instead of F1 score, we report an additional metric, the maximum rank of fraudulent samples, which is motivated by practical means: In a real-world application, ML and automated decision-making on sensible data is required to have the "human in the loop" for privacy and ethical concerns [88], such that, for example, automated decisions are being reviewed before any consequences are drawn. Therefore, the maximum rank of fraudulent samples (i.e. the rank of the least suspicious fraud) corresponds to the number of instances a human reviewer has to examine and therefore the workload to find all fraud cases. For the subset of fraudulent samples $\mathcal{D}|_F = \{(\mathbf{x}, y) \in \mathcal{D} \ : \ y = \text{fraud}\}$ the rank of the least suspicious fraud is given by

$$r_{min} = |\{\mathbf{x} \in \mathcal{D} \ : \ as(\mathbf{x}) \geq \min_{\mathbf{f} \in \mathcal{D}|_F} as(\mathbf{f})\}| \tag{8.1}$$

for an anomaly score *as* with increasing values corresponding to more anomalous samples.

Results

The results of the hyperparameter study with regard to AP as evaluation metric are depicted in Fig. 8.5. While IF overall performs very well on the *eval* dataset, it achieves lower results on the *test* dataset compared to the other models which may indicate overfitting of the hyperparameters. Overall, the range of results is very

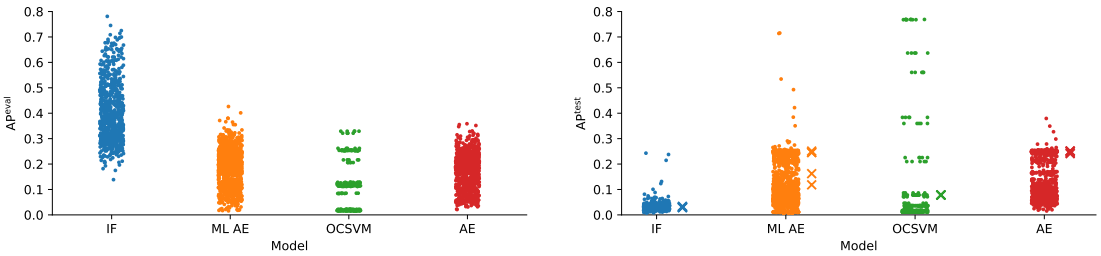


Figure 8.5: AP results for the hyperparameter study for *eval* and *test*. The results of the models selected according to best *eval* results are marked with $\times$.

large, ranging from 0.78 to 0.13 AP on *eval*, while on *test* the span is notably smaller with a best performance of 0.24. Contrarily, the OC-SVM yields inferior results on the *eval* dataset while achieving the overall best AP of 0.78 on the *test* dataset. An in-depth view on the hyperparameters suggests that they do not generalize well between both datasets, as the main difference is the scaling parameter for the RBF kernel γ, which highly influences the results achieving 0.07 on *test* for the best model with $\gamma = 1$ according to *eval*, while the best model according to *test* with $\gamma = 0.1$ reaches 0.77 AP. The ML-AE and AE models perform comparably on *eval* and *test*. Although some ML-AE hyperparameters yield very good results of 0.72 on *test*, the best models according to *eval* perform mixed between 0.12 and 0.25. The AE on the other hand does not perform comparably for the best models according to *test*, however, it yields stable results comparable to ML-AE when evaluating the best model selected from *eval*. Focusing on the hyperparameter choices for the best ML-AE and AE models, both have identical hyperparameters as listed in Table 8.6 besides relatively similar architectures with one hidden layer and a bottleneck of 80 and 90 dimensions, respectively.

Table 8.6 summarizes the results for all the best models according to *eval*. Note that all models list all fraud cases for *eval* among the first 57 most suspicious samples for all models, with IF performing best listing all among the first 44 on average. The other models follow closely, showing the same characteristics already discussed for the overview of all hyperparameter runs in Fig. 8.5. However, on the *test* dataset the differences become larger, as IF is outperformed by the other models by large margin. AE, ML-AE and OC-SVM retrieve all fraudulent samples among 46 and 61 candidates, which roughly corresponds to their performance on *eval* and can be handled by an auditor for manual inspection. Regarding preprocessing, IF benefits from z-score normalization while all other approaches yield best results with QOQD quantization.

Conclusion

In this experiment, we evaluated ML-AE models in comparison to AE, OC-SVM and IF in a semi-supervised one-class setting training on benign data and evaluating on fraudulent data. We performed an extensive hyperparameter study on the *eval* dataset for all models and reported the best models for the *test* dataset. Our findings suggest that ML-AE and AE generalize well between both datasets yielding similar performances, while other models perform better on each dataset individually, however, fail to generalize when evaluated in a fair comparison, i.e. selecting hyperparameters independently from a hold-out test dataset. Compared to our results from supervised training without PCA (see Section 8.2.1) both baseline methods, OC-SVM and IF, improve, while Mixed Layers performed slightly worse with respect to the AP metric in the semi-supervised AE-based architecture.

8.3 Modeling Distributions and Dependencies for Fraud Detection

In Section 6.2, we introduced WGAN and VAE architectures to model distributions and dependencies of transaction data as generative models and evaluated their performance with respect to data synthesis. As discussed in Section 4.4, the task of anomaly detection is often approached with the same objective, namely, learning the characteristics of normal data, for which generative models can be employed. In this section, we therefore address the task of fraud detection from a generative modeling perspective. As an intuition, for models trained to generate normal data, this corresponds to evaluating how likely it is that the model will produce or how well the model can reproduce the queried sample, which will be detailed in Section 8.3.1.

In this experiment we focus on both anomaly detection datasets already evaluated in the generative setting in Section 6.2, PaySim, and SAP. As Census has no fraud detection objective to evaluate and does not contain strongly imbalanced classes which can be modeled as such, instead of Census, we include the CCFraud dataset from Section 8.1. CCFraud could not be included as a generative evaluation dataset in

Table 8.6: Best performing hyperparameter configurations of each approach.
† Non-deterministic algorithm: Results averaged over 5 random seeds.

Approach	↓ r_{min}^{eval}	↓ r_{min}^{test}	↑ AP^{eval}	↑ AP^{test}	↑ ROC-AUC^{eval}	↑ ROC-AUC^{test}	Preproc.	Hyperparameters
OC-SVM	51	61	0.25	0.08	1.00	1.00	QOQD	{$\nu = 0.2$, $\gamma = 1.0$}
ML-AE†	55.20 ±1.64	47.40 ±3.29	0.25 ±0.01	0.20 ±0.06	1.00 ±0.00	1.00 ±0.00	QOQD	{neurons=(290, 90, 290), batch_size=16, learning_rate=1e-4}
AE†	56.67 ±2.50	46.17 ±6.46	0.24 ±0.02	0.25 ±0.00	1.00 ±0.00	1.00 ±0.00	QOQD	{neurons=(120,60,120), batch_size=16, learning_rate=1e-4}
IF†	43.80 ±1.92	151.00 ±17.65	0.31 ±0.01	0.03 ±0.00	1.00 ±0.00	0.99 ±0.00	z-score	{t=512, φ = 0.2, ψ=1}

↑ / ↓: *higher / lower value is better*

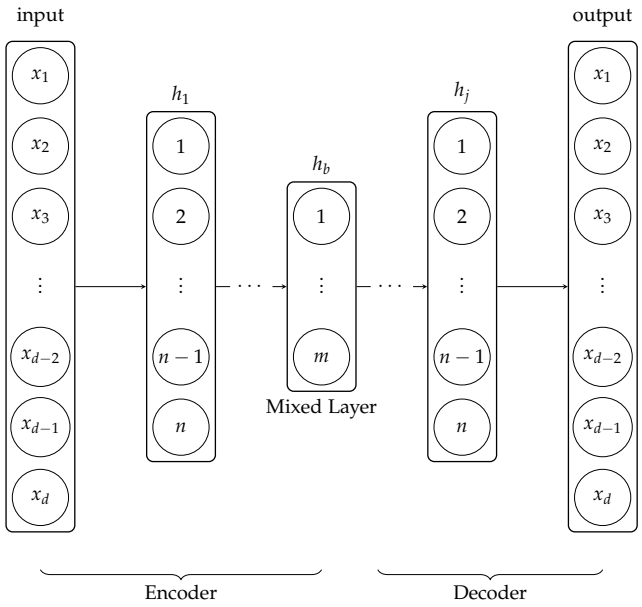


Figure 8.6: AE with Mixed Layer of dimensionality m and linear hidden layers h_1, h_j of dimensionality n. Arrows are drawn between fully connected layers. AE without Mixed Layer is structured accordingly with ReLU activations.

Section 6.2 since all features (except time and amount) are PCA transformed and therefore cannot be compared with generated data from a qualitative perspective. For fraud detection, however, the generative precision in terms of explicitly modeling feature distributions and dependencies is less important since the ability to precisely identify anomalous samples can be evaluated independently. Nevertheless, the models have to implicitly model normal data to be beneficial for detecting anomalous samples, as the evaluation typically relates to the ability to model normal data well while failing to reproduce or generate anomalous samples. In the next section, we will focus on this aspect, i.e. how the trained generative models are used to evaluate the normality of a given sample.

8.3.1 Anomaly Detection Methods

In this section, the AD methods for both models, VAE and GAN, are detailed. While AD for VAE is generally realized straight forward and analogously to training by reconstruction error, for GAN, multiple options are feasible, which are discussed in more depth.

AE Reconstruction As discussed in Section 4.4, AD often involves learning a model for normal data and identifying anomalies by reconstructing a sample through the model from which the reconstruction error as AD score is derived. For VAE models,

the reconstruction error can be incorporated analogously to AE models, as detailed in Section 8.2.2.

Discriminator-based AD With GANs, on the other hand, a reconstruction-based anomaly score is not inherently applicable, since the generator maps from random noise space z to feature space, i.e. z can not be interpreted as specific latent representation of a previously encoded sample from feature space such as the bottleneck layer of an AE. Since the discriminator maps from feature space to a score distinguishing real samples from generated, it is theoretically applicable to anomaly detection. However, as the generator learns to mimic real data better and better, the discriminator has to focus on specific artifacts of the synthesized data which are most certainly not present in real anomalies [219] therefore showing inferior results compared to a classifier dedicated for the downstream task [200]. Additionally, the training loop between generator and discriminator has to be balanced for successful training. A perfect generator will thereby stop the discriminator from learning and vice versa constraining the AD performance of the discriminator to the abilities of the generator to "resemble the anomalies you expect to need to detect", as Goodfellow stated [126]. Although first approaches reusing the discriminator are currently being studied [319, 320, 200, 219], such approaches modify the training objective to generate data outside the normal data distribution [219] or introduce changes in the objectives and training protocol [319, 320] with unclear adaptability for AD specific tasks and missing performance improvements over models trained outside the GAN framework [200]. Therefore, we refrain from using discriminator-based AD for fraud detection.

Complex Architectures for AD Besides reusing the discriminator, other approaches have been proposed for GAN-based AD, which rely on augmented architectures such as BiGAN [75, 182, 358, 357] or combining GANs or adversarial mechanisms in general with AEs [79, 194, 8, 369, 296]. However, all of these methods imply large architectural changes with potential consequences for the balancing, training process, and stability to which GAN models are known to be prone [115]. Therefore, we follow another methodology that allows us to use unaltered GAN architectures already proven in the generative course. This methodology has been successfully applied for GAN-based AD and is known as *inverting the generator* technique.

Generator-based AD As GANs are trained to generate normal data, the generator is expected to reproduce normal samples well while struggling to produce specific anomalies which have not been part of the training data. Formally, we expect a z from random latent space Z to exist for which the output of the generator $G : z \mapsto \hat{x}$ is close to the associated real sample x, as the interplay between discriminator and generator encourages the generator to generate samples similar to real

samples presented to the discriminator. Note that this assumption only holds when mode collapse is avoided, i.e. the generator produces samples over the complete sample probability space as encouraged by the WGAN-GP architecture we used in our experiments. In a reconstructive setting, the generator is expected to produce a certain $\hat{x}$ similar to a given x, which demands an inverse mapping of the generator $G^{-1} : x \mapsto z$ hence the term *inverting the generator* techniques. If this G^{-1} is given, the generator can be used similarly to AEs by mapping a real sample x to the most representative latent variable z (encoding) and generating the synthetic sample $\hat{x}$ with $G(z)$ (decoding). Then reconstructive anomaly scoring as used in AEs can be applied directly. Note that determining G^{-1} is not trivial in practice as in general the generator is a deep neural network comprised of several non-linear layers. While several complex model extensions (BiGAN [358], additional encoder [7, 8], etc.) aim to approximate G^{-1} by additional GAN components and thus could be considered as 'inverting the generator' in a broader sense, we only consider direct approaches on the trained generator. For this purpose, three related methods have been presented in literature, which we briefly review before evaluating their performance in the fraud detection domain.

Inverting the Generator (ItG) The first method, hereafter referred to as ItG, was developed by Creswell and Bharath [66, 67]. They propose inverting the generator by starting with a randomly initialized $\hat{z} \sim P(Z)$ which is optimized $\hat{z} \rightarrow z^*$ towards the specific z for which $G(z) = x$.

$$z^* = \min_z (- \mathop{\mathbb{E}}_x [\log(G(z)])) \tag{8.2}$$

$$\mathcal{L}_{ItG}(z; x) = - (x \log(G(z)) + (1 - x) \log(1 - G(z))) \tag{8.3}$$

With the resulting loss function $\mathcal{L}_{ItG}(z; x)$, $\hat{z}$ is optimized by gradient descent leaving G itself unchanged. In practice, we adopt the seeding strategy from [72] by performing this procedure for a set of seeds z simultaneously to increase the robustness, motivated by the observation that several z map to a given x. The final anomaly score is then the average over the loss of 64 seed-z after their last update step. We follow the authors with respect to the choice of parameters for optimization of z with $lr = 0.0002$ and $k = 200$ steps. As these steps have to be performed for each sample x, this approach is quite computationally intensive.

ADGAN Deecke et al. [72] proposed ADGAN, another method to invert the generator. The underlying principle is similar to ItG. However, in addition to optimizing z, the generator is re-trained at each step, which is expected to increase the flexibility to better reconstruct x. The generator is reset to its post-training state for each sample. ADGAN therefore corresponds to the ItG approach in Eq. 8.2 with the exception that the parameters of G are not fixed and optimized alongside z. Since inverting the

generator is a slow method, the authors suggested using $lr_z = 0.25$ and $k = 5$ steps to update z and $lr_g = 5 \cdot 10^{-5}$ for the generator, which we adopt for our experiments.

ANOGAN The third method we compare is ANOGAN by Schlegl et al. [275]. Instead of retraining the generator, this method adds the feature matching [271] discriminator loss $\mathcal{L}_D$ to the optimization problem for $\hat{z}$. The loss function for optimizing the noise vectors z is thus given as:

$$\mathcal{L}_R(\mathbf{z}; \mathbf{x}) = \|\mathbf{x} - G(\mathbf{z})\|_2^2 \tag{8.4}$$

$$\mathcal{L}_D(\mathbf{z}; \mathbf{x}) = \|f(\mathbf{x}) - f(G(\mathbf{z}))\|_2^2 \tag{8.5}$$

$$\mathcal{L}(\mathbf{z}; \mathbf{x}) = (1 - \lambda) \cdot \mathcal{L}_R(\mathbf{z}; \mathbf{x}) + \lambda \cdot \mathcal{L}_D(\mathbf{z}; \mathbf{x}) \tag{8.6}$$

Schlegl et al. use the last discriminator layer as f to extract meaningful features for $\mathcal{L}_D$ instead of the decision scalar. We follow this idea, use the last intermediate discriminator layer as f and set $\lambda = 0.1$, $lr = 0.001$ and $k = 100$ steps. The anomaly score is determined by the loss after the last update iteration.

8.3.2 Experiment 1: Preprocessing

In Section 6.2 we evaluated VAE and WGAN with several preprocessing and parameter choices with respect to their generative performance to synthesize data similar to the original dataset. On the other hand, in this experiment, we replicate this study for both models with a focus on AD instead of generative performance. By this we evaluate if different objectives for the same datasets and models benefit from different preprocessing choices. As Section 6.2.3 showed negligible influence of the embedding size on the generative performance, we evaluate *auto* with $d = \min(\lceil \frac{k}{3} \rceil, 20)$ for k unique feature values, no embeddings *(None)*, 50 as embedding dimensionality and min-max scaling, z-score standardization and VGM for each model in this experiment. For CCFraud, all features are PCA-encoded or numerical attributes, therefore, no embedding is evaluated for this dataset. For WGAN in the first preprocessing evaluation experiment, we fix ItG as anomaly detection method, as the other methods are based on ItG. For VAE we evaluate by reconstruction error as discussed in the last section.

Results Table 8.7 summarizes the results. Overall, it can be observed that similar to our findings from Section 6.2.3, the embedding size d_{emb} has little influence on the results with the exception of ROC-AUC for WGAN on PaySim, where *auto* performed best for all numerical preprocessors and AP for VAE on SAP, where $d_{\text{emb}} = 50$ performed best.

The VAE shows mixed results with respect to numerical encoding in different datasets. While z-score standardization performed best for both metrics for PaySim

Table 8.7: Discriminative preprocessing results for VAE and WGAN on PaySim, SAP, and CCFraud eval datasets averaged over 5 iterations. The best results are written in bold.

			VAE		WGAN	
Dataset	Num	d_{emb}	↑ ROC-AUC	↑ AP	↑ ROC-AUC	↑ AP
PaySim	minmax	auto	0.78 ± 0.03	0.04 ± 0.03	0.80 ± 0.04	0.02 ± 0.03
		50	0.78 ± 0.03	0.03 ± 0.04	0.76 ± 0.07	0.01 ± 0.02
		None	0.80 ± 0.07	0.04 ± 0.03	**0.83 ± 0.05**	0.02 ± 0.03
	z-score	auto	**0.89 ± 0.03**	0.04 ± 0.04	0.75 ± 0.05	**0.03 ± 0.04**
		50	**0.89 ± 0.03**	**0.05 ± 0.04**	0.73 ± 0.05	**0.03 ± 0.04**
		None	**0.89 ± 0.03**	0.04 ± 0.04	0.73 ± 0.04	**0.03 ± 0.04**
	VGM	auto	0.83 ± 0.06	0.04 ± 0.04	0.60 ± 0.04	**0.03 ± 0.04**
		50	0.83 ± 0.05	0.03 ± 0.04	0.55 ± 0.04	**0.03 ± 0.04**
		None	0.82 ± 0.04	0.03 ± 0.04	0.56 ± 0.08	**0.03 ± 0.04**
SAP	minmax	auto	**1.0 ± 0.0**	0.86 ± 0.11	**1.0 ± 0.00**	**0.71 ± 0.04**
		50	**1.0 ± 0.0**	**0.97 ± 0.03**	**1.0 ± 0.00**	**0.71 ± 0.07**
		None	**1.0 ± 0.0**	0.93 ± 0.06	**1.0 ± 0.00**	0.70 ± 0.08
	z-score	auto	**1.0 ± 0.0**	0.77 ± 0.02	0.99 ± 0.00	0.67 ± 0.00
		50	**1.0 ± 0.0**	0.77 ± 0.03	0.99 ± 0.00	0.67 ± 0.00
		None	**1.0 ± 0.0**	0.79 ± 0.03	**1.0 ± 0.02**	0.67 ± 0.00
	VGM	auto	**1.0 ± 0.0**	0.87 ± 0.17	**1.0 ± 0.00**	0.02 ± 0.01
		50	**1.0 ± 0.0**	0.90 ± 0.16	**1.0 ± 0.00**	0.01 ± 0.01
		None	**1.0 ± 0.0**	0.77 ± 0.26	**1.0 ± 0.00**	0.02 ± 0.01
CCFraud	minmax	None	0.93 ± 0.03	0.54 ± 0.04	**0.96 ± 0.01**	**0.16 ± 0.03**
	z-score	None	0.94 ± 0.01	**0.57 ± 0.06**	**0.96 ± 0.01**	**0.16 ± 0.03**
	VGM	None	**0.95 ± 0.02**	0.53 ± 0.07	**0.96 ± 0.01**	**0.16 ± 0.03**

↑: higher value is better

and on CCFraud best for AP, for SAP, minmax scaling yielded the best while standardization yielded the worst results for AP by large margin. For CCfraud and SAP, the ROC-AUC metric does not reflect the differences well, as all models perform very similarly. For VAE in the following experiments, we use z-score scaling for PaySim and CCFraud and minmax scaling for SAP. In comparison to the best hyperparameters for the generative task, the discriminative task benefits from different preprocessing choices depending on the dataset. For PaySim, the generative task performed comparable for z-score scaling, which was the best result for the discriminative task. For SAP in the generative task VGM clearly outperformed minmax

scaling, while for the discriminative task, for VAE minmax scaling yielded the best results.

For the WGAN model, min-max scaling yields the best results overall. For PaySim, the results regarding AP, for SAP ROC-AUC and for CCFraud both metrics vary only slightly between different preprocessing choices. In contrast, min-max clearly outperforms the other approaches for ROC-AUC on PaySim and AP on SAP. Therefore, we evaluate the minmax scaling for the next WGAN experiments with $d_{\text{emb}} =$ auto for PaySim and SAP. The preprocessing choices for WGAN differ between the discriminative and generative task as well. For the generative task, while VGM modeled the distributions according to JSD best by large margin, minmax outperformed VGM on the discriminative task, especially for SAP.

To summarize, our experiments suggest that generative and discriminative objectives for the same models and datasets require different preprocessing choices for AD. This is not completely surprising as the generative evaluation only judges the precise modeling of normal data, while the treatment of anomalies has almost no influence due to their rarity alone. On the other hand, the discriminative objective does not require the models to match the real distributions precisely as long as anomalous samples are made distinguishable, which ultimately is the only aspect that is assessed.

8.3.3 Experiment 2: GAN Anomaly Detection Methods

In Experiment 1, we evaluated the preprocessing parameters using the ItG technique. With ADGAN and ANOGAN, two additional variants have been proposed, which extend the basic idea of ItG. In this experiment, we compare the three approaches to evaluate whether both extensions, which have been originally proposed in the image domain, also improve AD in the domain of financial transactions. As the influence of preprocessing on the relative performance between ItG, ADGAN, and ANOGAN is negligible since all methods are based on the ItG principle, we use the preprocessing parameters according to Experiment 1 (cf. Section 8.3.2).

Results We conducted an experiment evaluating ItG, ADGAN and ANOGAN with the WGAN model as introduced in Section 6.2 on the three datasets, PaySim, SAP and CCFraud, and report ROC-AUC and AP as evaluation metrics. Table 8.8 summarizes the results averaged over 5 runs. For our financial fraud detection setting, ItG outperformed the other approaches on PaySim and SAP. For CCFraud, ANOGAN slightly outperformed ItG with respect to AP, while performing equally well for ROC-AUC. These results suggest that for ADGAN, either retraining the generator does not improve AD in this setting or the training parameters for z are not chosen beneficial in this context.

Table 8.8: AD methods for WGAN averaged over 5 random seeds and their standard deviation with the best results written in bold.

	ItG		ADGAN		ANOGAN	
Dataset	↑ ROC-AUC	↑ AP	↑ ROC-AUC	↑ AP	↑ ROC-AUC	↑ AP
$\text{PaySim}^{\text{eval}}$	0.821 ± 0.075	0.019 ± 0.028	0.769 ± 0.072	0.017 ± 0.028	**0.825 ± 0.064**	**0.019 ± 0.028**
SAP^{eval}	**0.988 ± 0.006**	**0.388 ± 0.067**	0.961 ± 0.022	0.362 ± 0.015	0.980 ± 0.010	0.347 ± 0.055
$\text{CCFraud}^{\text{eval}}$	0.951 ± 0.022	0.176 ± 0.036	0.925 ± 0.022	0.156 ± 0.040	**0.958 ± 0.019**	**0.214 ± 0.060**
$\text{PaySim}^{\text{test}}$	**0.797 ± 0.045**	**0.020 ± 0.032**	0.736 ± 0.121	0.018 ± 0.031	0.794 ± 0.069	0.012 ± 0.016
SAP^{test}	**0.998 ± 0.001**	**0.695 ± 0.040**	0.997 ± 0.001	0.569 ± 0.116	0.996 ± 0.002	0.424 ± 0.265
$\text{CCFraud}^{\text{test}}$	0.954 ± 0.007	0.164 ± 0.031	0.925 ± 0.022	0.155 ± 0.033	**0.956 ± 0.009**	**0.210 ± 0.055**

↑: higher value is better

Table 8.9: Comparison AD threshold selection strategies for PaySim, SAP and CCFraud eval datasets.

		max normal				min anomaly				Contamination rate			
Model	Dataset	↑ F1	↑ TP	↓ FN	↓ FP	↑ F1	↑ TP	↓ FN	↓ FP	↑ F1	↑ TP	↓ FN	↓ FP
VAE	PaySim	0.052	0.4	14.4	0.4	0.005	13.0	1.8	5769	**0.084**	1.2	13.6	12.4
	SAP	**0.728**	6.0	0.0	5.2	0.069	6.0	0.0	206	0.5	6.0	0.0	12.0
	CCFraud	0.231	7.4	37.2	1.2	0.007	43.8	0.8	19708	**0.528**	25.0	19.6	25.0
WGAN	PaySim	0.0	0.0	14.8	0.2	0.002	13.6	1.2	14101	**0.014**	0.2	14.6	13.4
	SAP	**0.747**	3.6	2.4	0.0	0.011	6.0	0.0	1370	0.367	4.4	1.6	13.6
	CCFraud	0.0	0.0	44.6	0.4	0.005	43.8	0.8	17639	**0.226**	10.6	34.0	39.4

↑ / ↓: higher / lower value is better

Given that ADGAN with per-sample retraining of the generator (and thus back-propagation for a large number of parameters) is very computationally intensive, the retraining is not sustainable as the models are reset after AD calculation and the overall performance is notably worse, we refrain from extensive hyperparameter studies and further evaluate with ItG, the overall best approach, for the next experiments.

8.3.4 Experiment 3: Fraud Detection

For Fraud Detection in an automated approach, the anomaly score with its ROC-AUC and AP metrics is not a sufficient output as for credit card transactions, for example, a decision for each sample is required whether it is to be treated as fraud (e.g. by blocking the transaction or requiring further authorization) or whether it is benign and therefore executed as desired. As discussed in Section 8.2.2, such automated decisions are not always in compliance with regularities. However, they allow a timely response (in comparison to auditor-operated fraud detection), which for example for deployment in micro-payment systems (PaySim) or credit-card pay-

ment fulfillment (CCFraud) is required to prevent fraud. We therefore evaluate three threshold-based approaches to decide on the actual class label driven by the anomaly score of VAE and WGAN.

Without class labels, the decision function can not be trained end-to-end as given for WGAN and VAE in our scenario as both approaches are trained to model normal data precisely and thus judge anomalies by their deviation from the expected normal behavior. Two options are feasible to derive the class label from anomaly score post hoc: If the fraction of expected anomalies can be estimated, for example, from prior audits or expert knowledge, the *contamination rate* can be used as decision threshold. The contamination rate denotes the percentage of samples that are expected to be anomalies. The class label *anomaly* is then assigned to the respective proportion of the samples with the highest anomaly score evaluated. This method, however, is difficult to apply in an online or streaming approach (as given for real-time application in payment fraud detection), since the dataset size is unknown. Also, the trained model might suffer from data drift as the data characteristics may change over time, so previously estimated thresholds according to the contamination rate might not be suitable any more.

Another option applicable only in the one-class setting where the training data is known to contain only normal data, is to use the maximum of the anomaly scores seen during training for normal data, i.e. the "most abnormal" normal (max normal) sample (cf. Section 4.4). However, as soon as the training data is contaminated with anomalies, this approach will most likely miss similarly and less anomalous anomalies present in the test dataset.

In a semi-supervised or one-class scenario, where a large number of training data are unlabeled or from one class only, while an evaluation or validation dataset (besides the actual test data) is available containing both class labels, this dataset can be used to decide on a threshold. The decision can then be based on the *least suspicious anomaly* in this evaluation dataset (min anomaly). In this experiment, we evaluate all three approaches to decide on the class label with regard to the AD score.

Results

In Table 8.9 all three threshold selection strategies are compared on the PaySim, SAP, and CCFraud datasets. We report TP, FN, and FP averaged over 5 seeds and the resulting F1 score for VAE and WGAN. The contamination rate and min anomaly threshold are both estimated from the eval dataset, while the max normal threshold is estimated from the training set. The results show a differentiated pattern mainly depending on the dataset: For both models, SAP yields the best F1 score for max normal while for PaySim and CCFraud, the contamination rate yields the best F1 score. For SAP, WGAN slightly outperforms VAE, while for all other datasets, VAE yields notably higher F1 scores. Min anomaly overall yields the worst results regarding F1 driven by high FP rates, although this strategy marks the most true anomalies

as such. The good performance of contamination rate for PaySim and CCFraud in contrast to the worse results for SAP can be explained by the dataset characteristics: While for PaySim and CCFraud the eval (and test) dataset was randomly sampled from the full dataset and thus is expected to follow the same dataset characteristics and fraud proportion, for SAP training and eval are different datasets remarking different fiscal years and thus different data characteristics, fraud cases, and fraud proportions.

To summarize, our experiments suggest that for class labels to be inferred from anomaly scores, the characteristics of the dataset and anomalies are important. If the training dataset is anomaly-free and validation and test datasets do not originate from the same population (i.e. the same generative process), the most suspicious normal sample defines a suitable threshold. However, when the datasets originate from the same population and the contamination rate can thus be estimated reliably, contamination rate is a precise threshold strategy without too many false positives. When the focus is not on avoiding false alarms but on finding most anomalies, for example, when the cost of a missed fraud exceeds the cost of a false positive by magnitudes, the min anomaly strategy might also be a suitable choice for some applications.

The results of this experiment comparing WGAN and VAE with respect to ROC-AUC, AP and F1 metrics are summarized in Table 8.10. In direct comparison, VAE mostly outperforms WGAN for our AD task. Regarding ROC-AUC, the WGAN model slightly outperforms VAE on CCFraud and regarding F1 on SAP. The respective other model performs only insignificantly[3] worse, all within one standard-deviation. Contrarily, for all other datasets and metrics with the exception of PaySim and AP, VAE significantly[4] outperforms the respective WGAN.

8.3.5 Conclusion

Recalling our best results of Section 8.2 on SAP, which was achieved by IF with min-max normalization and PCA encoding, our best VAE model performed equally well on ROC-AUC (both 1.0), almost equally on AP (IF 0.974, VAE 0.969) and notably better regarding F1 (IF 0.231, VAE 0.728). Overall, both generative models, VAE and WGAN, yield considerably good results. Although outperforming VAE in several generative scenarios (cf. Section 6.2), the WGAN anomaly detection approaches evaluated in this thesis, which only rely on inverting the generator without architectural changes, cannot compete with reconstruction-based AD using VAEs.

[3]Mann-Whitney U Test, $p > 0.26$
[4]Mann-Whitney U Test, $p \leq 0.05$

Table 8.10: Final anomaly detection results for the Paysim, SAP and CCFraud test datasets. Best model per metric in bold, significantly* different pairs are underlined.

Dataset	Model	↑ ROC-AUC	↑ AP	↑ F1
PaySim	WGAN	0.797 ± 0.045	0.020 ± 0.032	0.014 ± 0.029
	VAE	**0.892** ± **0.036**	**0.047** ± **0.043**	**0.084** ± **0.051**
SAP	WGAN	0.998 ± 0.001	0.695 ± 0.040	**0.747** ± **0.065**
	VAE	**1.0** ± **0.0**	**0.969** ± **0.025**	0.728 ± 0.146
CCFraud	WGAN	**0.956** ± **0.009**	0.210 ± 0.055	0.226 ± 0.056
	VAE	0.948 ± 0.015	**0.550** ± **0.072**	**0.528** ± **0.088**

↑: higher value is better

* *Mann-Whitney U Test*, $p \leq 0.05$

Chapter 9

Conclusion and Outlook

In this thesis, we approached anomaly detection on transaction data from different perspectives. As capturing normal data characteristics in comparison to anomalies is an important task-dependent and domain-dependent aspect, we studied several architectures to emphasize learning of numeric dependencies and feature distributions and proposed specific models for transaction data. We evaluated several representation learning approaches and finally evaluated different scenarios for detecting anomalies in transaction data in supervised and unsupervised, as well as balanced and imbalanced settings for several synthetic and real financial fraud detection datasets. In the following we will summarize the contributions of this thesis as well as revisit and answer our research questions.

In the first chapter, we focused on modeling distributions and dependencies specifically for transaction data. We introduced an improved neural arithmetic model identifying and discussing several shortcomings of the preceding state-of-the-art model. We showed the benefit of our proposed approach for modeling arithmetical and numerical dependencies in several experiments. This allows us to answer the first research question regarding the modeling of numerical dependencies:

RQ 1: How well do neural network models capture the characteristics of transaction data?

RQ 1.1: To what extent can the extrapolation of numerical dependencies be improved by a neural architecture?

Answer: Although not all stability issues and the precise modeling of the division operator have been solved, our proposed iNALU model generally improved the extrapolation of numerical dependencies regarding precision and stability in comparison to the previous state-of-the-art.

We then adapted two generative neural models, VAE and WGAN, for modeling transaction data. We systematically evaluated a large number of preprocessing choices with the aim to improve the generative performance for distributions and

feature correlations as present in transaction data. We incorporated our iNALU architecture in both models and found that the generative performance for WGAN could be significantly improved by incorporating our iNALU architecture for some datasets, while performing equally well on the others. We then evaluated the generative performance of VAE and WGAN exploratively and found that both models capture dataset distributions and feature correlations well, with WGAN modeling numerical variables and VAE modeling categorical variables slightly better than the respective other model. This finding answers both research questions:

RQ 1.2: How well do generative neural networks model feature distributions?

RQ 1.3: How well do generative neural networks model feature correlations?

Answer: For modeling distributions and correlations, both models studied are able to capture both aspects very precisely on datasets we evaluated.

In the second chapter, we focused on categorical features and their representation. First, we used Windows Audit Logs as transaction dataset, which resembles the characteristics of natural language best. We evaluated three word embedding approaches in comparison to one-hot encoding and found that one-hot vectors quantitatively still outperformed the word embedding approaches, while FastText embeddings were able to structure the relevant aspect of malicious and non-malicious executions very well from an explorative auditor-driven perspective. We then introduced outlier-aware discretization schemes to adapt this experiment to financial transactions, encoding categorical and numerical features within the same embedding space. We then evaluated the different discretization schemes and word embedding approaches in a classification task, finding that GloVe-based embeddings trained with our proposed discretization scheme outperform the baseline classifier, although FastText and Word2Vec, which were quantitatively outperformed by the baseline, yielded the visually most promising results. Relating to our research question, our findings can be summarized as follows:

RQ 2: How are transactions best represented to emphasize structural similarity?

RQ 2.1: Which representation learning approach yields the most promising structure in latent space?

RQ 2.2: Which representation learning approach yields the best performance in extrinsic evaluation?

RQ 2.3: To what extent does the introduction of numerical features improve representation learning in extrinsic evaluation?

Answer: We have to distinguish between exploratory analysis (RQ 2.1) as, for example, conducted by an auditor, and quantitative performance (RQ 2.2). From

a quantitative point of view, outlier-aware discretization of numerical features when incorporated in the representation learning process improved the results in conjunction with GloVe on the SAP dataset. However, the representation learning approaches studied could not outperform the baseline by large margin and even impaired the performance for our experiment on Windows Audit Logs. From an exploratory point of view, we found that FastText embeddings yielded the most promising results for both datasets. QOQD and QOUD improved representation learning in extrinsic evaluation, as the best results were obtained for models including one of our approaches.

In the last chapter, we finally approached anomaly detection from various perspectives. We showed that our Mixed Layer model incorporating iNALU neurons improved the anomaly detection performance in a supervised and balanced scenario. For an imbalanced experiment, while still outperforming the respective ReLU reference model, our model could not outperform all other approaches we examined. Further investigation showed that the performance of the overall best approach relied on our proposed outlier-aware discretization schema.

We then evaluated our model in a semi-supervised one-class scenario by proposing a Mixed Layer based Auto-Encoder. In comparison to an Auto-Encoder without Mixed Layers, our model performed equally well while outperforming IF and OC-SVM regarding their generalizability over datasets with different characteristics. Finally, we studied several approaches to incorporate WGAN and VAE for anomaly detection. We found that the differences between the three techniques for inverting the generator were only slightly relevant in our experiments, while overall, VAE outperformed WGAN in terms of anomaly detection performance on most metrics. Regarding RQ 3, the answer is thereby mixed.

RQ 3: To what extent do the proposed methods improve anomaly detection for transaction data?

RQ 3.1: Can fraud detection performance on transaction data be improved by our Mixed Layer model?

Answer: Our experiment in the supervised scenario suggests that our Mixed Layer model can improve fraud detection on transaction data, although in an imbalanced setting, a simple kNN model with our outlier-aware discretization scheme outperformed the Mixed Layer-based approach. In a semi-supervised setting, our Mixed Layer showed to generalize well, however, an Auto-Encoder reference model performed equally well. In summary, this suggests that our architecture can improve fraud detection under the given circumstances.

RQ 3.2: Do our generative VAE and GAN models yield competitive anomaly detection performances, and do optimal preprocessing and parameter choice differ from their generative setting?

Answer: We found that VAE yields competitive performance outperforming IF on one metric by large margin while performing equally well on the others. We found that the parameter choice for the generative setting differs from the best parameters for anomaly detection, which showed to be more dataset-dependent in comparison to the generative setting.

While overall the results of our experiments are promising, they come with some limitations: As discussed in the introduction, the availability of data is a predominant problem in the domains associated with transaction data, which also affects the research presented in this thesis. Our experiments are mainly founded on synthetic datasets with the exception of credit-card fraud, which has been anonymized in such a way that the evaluation on a per-feature basis including the study of preprocessing or representation learning approaches is not applicable. This implies that some of the drawn conclusions may not be directly generalizable for real-world applications, as characteristics of real-world datasets presumably increase the variability and complexity for normal data as well as for fraud. However, the research methodology applied in this thesis can be directly transferred and re-evaluated as such datasets might become available in the future.

Outlook From a methodological perspective, new research directions proven in related domains, such as large transformer-based neural architectures [323] as approved in the field of Natural Language Processing for its outstanding performance, could prove to be beneficial for transaction data for representation learning or as an end-to-end anomaly detection model as larger datasets become available. While we focused on generative models which are known for their ability to synthesize data in literature and then evaluated their performance for anomaly detection, future research could also turn the tables and focus on generative models that are specifically designed for anomaly detection and evaluate their abilities as data generators.

Furthermore, the mechanisms modeling numerical or arithmetical dependencies could be further improved, as novel approaches mainly constructed for solving arithmetic tasks might also be useful to learn implicit relations within the given anomaly detection tasks. For such a research direction, eXplainable Artificial Intelligence might give further interesting insights either to make the black-box decision process of neural models transparent or to study the direct impact of numerical dependencies on the task. It can be applied as a supportive tool for auditors and with regard to the high regulatory requirements for deploying an anomaly detection tool in this privacy-critical domain of transaction data.

Finally, different related application domains could be addressed with the findings of this thesis as, for example, transaction data in tabular form or similarly structured log data occur in several closely or more distantly related fields. Potential application examples range from SAP systems in different parts of a company to data arising with the digitization of processes in e-government or hospitals.

Bibliography

[1] Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. (2016). Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318.

[2] Abu Alfeilat, H. A., Hassanat, A. B., Lasassmeh, O., Tarawneh, A. S., Alhasanat, M. B., Eyal Salman, H. S., and Prasath, V. S. (2019). Effects of distance measure choice on k-nearest neighbor classifier performance: a review. *Big data*, 7(4):221–248.

[3] ACFE (2020). 2020 Global Occupational Fraud Study. *Report To the Nations*. [Online; accessed 12. Nov. 2020].

[4] Adel, T., Ghahramani, Z., and Weller, A. (2018). Discovering interpretable representations for both deep generative and discriminative models. In *International Conference on Machine Learning*, pages 50–59. PMLR.

[5] Aggarwal, C. C. (2016). *Outlier Analysis*. Springer.

[6] Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer.

[7] Akcay, S., Atapour-Abarghouei, A., and Breckon, T. P. (2018). Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Asian conference on computer vision*, pages 622–637. Springer.

[8] Akçay, S., Atapour-Abarghouei, A., and Breckon, T. P. (2019). Skip-ganomaly: Skip connected and adversarially trained encoder-decoder anomaly detection. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

[9] Al-Rubaie, M. and Chang, J. M. (2019). Privacy-preserving machine learning: Threats and solutions. *IEEE Security & Privacy*, 17(2):49–58.

[10] Albashrawi, M. (2016). Detecting financial fraud using data mining techniques: A decade review from 2004 to 2015. *Journal of Data Science*, 14(3):553–569.

[11] Ali, M. Q., Khan, H., Sajjad, A., and Khayam, S. A. (2009). On achieving good operating points on an roc plane using stochastic anomaly score prediction. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, page 314–323, New York, NY, USA. Association for Computing Machinery.

[12] Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.

[13] An, J. and Cho, S. (2015). Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18.

[14] Anderson, J. A., Spoehr, K. T., and Bennett, D. J. (1994). A study in numerical perversity: Teaching arithmetic to a neural network. *Neural networks for knowledge representation and inference*, 311:335.

[15] Antipov, G., Baccouche, M., and Dugelay, J.-L. (2017). Face aging with conditional generative adversarial networks. In *2017 IEEE international conference on image processing (ICIP)*, pages 2089–2093. IEEE.

[16] Arjovsky, M. and Bottou, L. (2017). Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*.

[17] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR.

[18] Awad, Y., Nassar, M., and Safa, H. (2018). Modeling malware as a language. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6.

[19] Baader, G. and Krcmar, H. (2018a). Reducing false positives in fraud detection: Combining the red flag approach with process mining. *Int. Journal of Accounting Information Systems*, 31:1–16.

[20] Baader, G. and Krcmar, H. (2018b). Reducing false positives in fraud detection: Combining the red flag approach with process mining. *Int. Journal of Accounting Information Systems*, 31:1–16.

[21] Baccelli, G., Stathis, D., Hemani, A., and Martina, M. (2020). Nacu: A non-linear arithmetic unit for neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6.

[22] Bank, E. (2019). Fifth report on card fraud. *European Central Bank: Frankfurt am Main, Germany*.

[23] Barnard, E. and Wessels, L. (1992). Extrapolation and interpolation in neural network classifiers. *IEEE Control Systems Magazine*, 12(5):50–53.

[24] Bartkowiak, A. M. (2011). Anomaly, novelty, one-class classification: a comprehensive introduction. *International Journal of Computer Information Systems and Industrial Management Applications*, 3(1):61–71.

[25] Bautista, Y. J. P., Aló, R., and Wang, N. (2020). Deep learning, cloud computing for credit/debit industry analysis of consumer behavior. In *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pages 1–7. IEEE.

[26] Beggel, L., Pfeiffer, M., and Bischl, B. (2019). Robust anomaly detection in images using adversarial autoencoders. *arXiv preprint arXiv:1901.06355*.

[27] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

[28] Bergsma, W. (2013). A bias-correction for cramér's v and tschuprow's t. *Journal of the Korean Statistical Society*, 42(3):323–328.

[29] Berlin, K., Slater, D., and Saxe, J. (2015). Malicious behavior detection using windows audit logs. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, AISec '15, page 35–44, New York, NY, USA. Association for Computing Machinery.

[30] Besson, J., Robardet, C., and Boulicaut, J.-F. (2004). Constraint-based mining of formal concepts in transactional data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 615–624. Springer.

[31] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *TACL*, 5:135–146.

[32] Bolton, R. J. and Hand, D. J. (2002). Statistical Fraud Detection: A Review. *Statistical Science*, pages 235–249.

[33] Boriah, S., Chandola, V., and Kumar, V. (2008). Similarity measures for categorical data: A comparative evaluation. In *Proceedings of the 2008 SIAM International Conference on Data Mining (SDM)*, pages 243–254.

[34] Borji, A. (2018). Pros and cons of gan evaluation measures.

[35] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152.

[36] Bouchacourt, D., Tomioka, R., and Nowozin, S. (2018). Multi-level variational autoencoder: Learning disentangled representations from grouped observations. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[37] Boulle, M. (2005). Optimal bin number for equal frequency discretizations in supervized learning. *Intelligent Data Analysis*, 9(2):175–188.

[38] Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2015). Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*.

[39] Bramer, M. (2007). Avoiding overfitting of decision trees. *Principles of data mining*, pages 119–134.

[40] Branco, P., Torgo, L., and Ribeiro, R. (2015). A survey of predictive modelling under imbalanced distributions.

[41] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[42] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). Classification and regression trees.

[43] Breuleux, O., Bengio, Y., and Vincent, P. (2011). Quickly generating representative samples from an rbm-derived process. *Neural computation*, 23(8):2058–2073.

[44] Brock, A., Donahue, J., and Simonyan, K. (2018). Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*.

[45] Brock, A., Lim, T., Ritchie, J. M., and Weston, N. (2017). Neural photo editing with introspective adversarial networks.

[46] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

[47] Camino, R., Hammerschmidt, C., and State, R. (2018). Generating multi-categorical samples with generative adversarial networks.

[48] Chalapathy, R. and Chawla, S. (2019). Deep Learning for Anomaly Detection: A Survey. *arXiv preprint arXiv:1901.03407*.

[49] Champion, K., Lusch, B., Kutz, J. N., and Brunton, S. L. (2019). Data-driven discovery of coordinates and governing equations.

[50] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58.

[51] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

[52] Chen, H., Chillotti, I., Dong, Y., Poburinnaya, O., Razenshteyn, I., and Riazi, M. S. (2020). Sanns: Scaling up secure approximate k-nearest neighbors search. In *USENIX Security*.

[53] Chen, K., Dong, Y., Qiu, X., and Chen, Z. (2018). Neural Arithmetic Expression Calculator. *arXiv preprint arXiv:1809.08590*.

[54] Chen, M. (2017). Efficient vector representation for documents through corruption.

[55] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

[56] Chen, T., Mao, Q., Lv, M., Cheng, H., and and, Y. L. (2019). Droidvecdeep: Android malware detection based on word2vec and deep belief network. *KSII Transactions on Internet and Information Systems*, 13(4):2180–2197.

[57] Choi, E., Biswal, S., Malin, B., Duke, J., Stewart, W. F., and Sun, J. (2017). Generating multi-label discrete patient records using generative adversarial networks. In *Machine learning for healthcare conference*, pages 286–305. PMLR.

[58] Choi, E., Biswal, S., Malin, B., Duke, J., Stewart, W. F., and Sun, J. (2018). Generating multi-label discrete patient records using generative adversarial networks.

[59] Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015). The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. PMLR.

[60] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[61] Ciaparrone, G., Sánchez, F. L., Tabik, S., Troiano, L., Tagliaferri, R., and Herrera, F. (2020). Deep learning in video multi-object tracking: A survey. *Neurocomputing*, 381:61–88.

[62] Coates, J. and Bollegala, D. (2018). Frustratingly easy meta-embedding–computing meta-embeddings by averaging source word embeddings. *arXiv preprint arXiv:1804.05262*.

[63] Cohen, P., West, S. G., and Aiken, L. S. (2014). *Applied multiple regression/correlation analysis for the behavioral sciences*. Psychology press.

[64] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[65] Cramér, H. (1946). *Mathematical Methods of Statistics*. Princeton university press.

[66] Creswell, A. and Bharath, A. A. (2016). Inverting the generator of A generative adversarial network. *CoRR*, abs/1611.05644.

[67] Creswell, A. and Bharath, A. A. (2018). Inverting the generator of a generative adversarial network. *IEEE transactions on neural networks and learning systems*, 30(7):1967–1974.

[68] Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., and Bharath, A. A. (2018). Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65.

[69] Dai, A. M., Olah, C., and Le, Q. V. (2015). Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*.

[70] Dal Pozzolo, A., Caelen, O., Johnson, R. A., and Bontempi, G. (2015). Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166. IEEE.

[71] Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240.

[72] Deecke, L., Vandermeulen, R. A., Ruff, L., Mandt, S., and Kloft, M. (2018). Image anomaly detection with generative adversarial networks. In *ECML/PKDD (1)*, pages 3–17.

[73] Delamaire, L., Abdou, H., and Pointon, J. (2009). Credit card fraud and detection techniques: a review. *Banks and Bank systems*, 4(2):57–68.

[74] Deng, W., Huang, Z., Zhang, J., and Xu, J. (2021). A data mining based system for transaction fraud detection. In *2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, pages 542–545. IEEE.

[75] Donahue, J., Krähenbühl, P., and Darrell, T. (2016). Adversarial feature learning. *arXiv preprint arXiv:1605.09782*.

[76] Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *Machine learning proceedings 1995*, pages 194–202. Elsevier.

[77] Dua, D. and Graff, C. (2017). UCI machine learning repository.

[78] Duggal, R. and Gupta, A. (2017). P-telu: parametric tan hyperbolic linear unit activation for deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 974–978.

[79] Dumoulin, V., Belghazi, I., Poole, B., Mastropietro, O., Lamb, A., Arjovsky, M., and Courville, A. (2016). Adversarially learned inference. *arXiv preprint arXiv:1606.00704*.

[80] Dwork, C. (2006). Differential privacy. In *International Colloquium on Automata, Languages, and Programming*, pages 1–12. Springer.

[81] Eckerli, F. (2021). Generative adversarial networks in finance: an overview. *Available at SSRN 3864965*.

[82] Eduardo, S., Nazábal, A., Williams, C. K., and Sutton, C. (2020). Robust variational autoencoders for outlier detection and repair of mixed-type data. In *International Conference on Artificial Intelligence and Statistics*, pages 4056–4066. PMLR.

[83] Efimov, D., Xu, D., Kong, L., Nefedov, A., and Anandakrishnan, A. (2020). Using generative adversarial networks to synthesize artificial financial datasets.

[84] Elhilbawi, H., Eldawlatly, S., and Mahdi, H. (2019). A taxonomy of discretization techniques based on class labels and attributes' relationship. In *2019 14th International Conference on Computer Engineering and Systems (ICCES)*, pages 316–321.

[85] Engelmann, J. and Lessmann, S. (2020). Conditional wasserstein gan-based oversampling of tabular data for imbalanced learning.

[86] Ernst, J., Hamed, T., and Kremer, S. (2018). A survey and comparison of performance evaluation in intrusion detection systems. In *Computer and network security essentials*, pages 555–568. Springer.

[87] Eskin, E., Arnold, A., Prerau, M., Portnoy, L., and Stolfo, S. (2002). A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*, pages 77–101. Springer.

[88] European Commission (2016). Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance).

[89] Everitt, B. (1988). A finite mixture model for the clustering of mixed-mode data. *Statistics & Probability Letters*, 6(5):305–309.

[90] Fahlman, S. E., Hinton, G. E., and Sejnowski, T. J. (1983). Massively parallel architectures for al: Netl, thistle, and boltzmann machines. In *National Conference on Artificial Intelligence, AAAI*.

[91] Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874.

[92] Fawcett, T. and Provost, F. (1997). Adaptive fraud detection. *Data mining and knowledge discovery*, 1(3):291–316.

[93] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37–37.

[94] Feng, W., Liu, B., Xu, D., Zheng, Q., and Xu, Y. (2021). Graphmr: Graph neural network for mathematical reasoning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3395–3404.

[95] Fisher, R. A. (1992). Statistical methods for research workers. In *Breakthroughs in statistics*, pages 66–70. Springer.

[96] Fourure, D., Javaid, M. U., Posocco, N., and Tihon, S. (2021). Anomaly detection: How to artificially increase your f1-score with a biased evaluation protocol. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 3–18. Springer.

[97] Frey, B. J., Hinton, G. E., Dayan, P., et al. (1996). Does the wake-sleep algorithm produce good density estimators? In *Advances in neural information processing systems*, pages 661–670. Citeseer.

[98] Friedman, J. H. (2002). Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378.

[99] Funahashi, K.-I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192.

[100] Gabriel, R., Gluchowski, P., and Pastwa, A. (2009). *Data warehouse & data mining*. W3l GmbH.

[101] Ganguli, I., Bhowmick, R. S., and Sil, J. (2020). Performance analysis of sota transformer network in numerical expression calculation. In *2020 IEEE 17th India Council International Conference (INDICON)*, pages 1–6. IEEE.

[102] Gao, G., Gao, J., Liu, Q., Wang, Q., and Wang, Y. (2020). Cnn-based density estimation and crowd counting: A survey. *arXiv preprint arXiv:2003.12783*.

[103] Gao, M., Jiang, J., Zou, G., John, V., and Liu, Z. (2019). Rgb-d-based object recognition using multimodal convolutional neural networks: A survey. *IEEE access*, 7:43110–43136.

[104] Garcia, S., Grill, M., Stiborek, J., and Zunino, A. (2014). An empirical comparison of botnet detection methods. *Computers & Security*, 45:100–123.

[105] García, S., Luengo, J., Sáez, J. A., López, V., and Herrera, F. (2013). A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):734–750.

[106] Gauerhof, L. and Gu, N. (2020). Reverse variational autoencoder for visual attribute manipulation and anomaly detection. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 2103–2112. IEEE.

[107] Ge, D., Gu, J., Chang, S., and Cai, J. (2020). Credit card fraud detection using lightgbm model. In *2020 international conference on E-commerce and internet technology (ECIT)*, pages 232–236. IEEE.

[108] Ghasemi-Gol, M., Pujara, J., and Szekely, P. (2019). Tabular cell classification using pre-trained cell embeddings. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 230–239. IEEE.

[109] Ghosh, S. and Reilly, D. L. (1994). Credit card fraud detection with a neural-network. In *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, volume 3, pages 621–630. IEEE.

[110] Glass, G. V. and Hopkins, K. D. (1996). *Statistical methods in education and psychology*, pages 133–134. Allyn and Bacon, Boston [u.a.].

[111] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256.

[112] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings.

[113] Godfrey, L. B. and Gashler, M. S. (2015). A continuum among logarithmic, linear, and exponential functions, and its potential to improve generalization in neural networks. In *2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*, volume 1, pages 481–486. IEEE.

[114] Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276.

[115] Goodfellow, I. (2016). Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

[116] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[117] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

[118] Gramopadhye, M., Singh, S., Agarwal, K., Srivasatava, N., Singh, A. M., Asthana, S., and Arora, A. (2021). Curl: Coupled representation learning of cards and merchants to detect transaction frauds. In *International Conference on Artificial Neural Networks*, pages 16–29. Springer.

[119] Graves, A. (2012). Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, pages 5–13. Springer.

[120] Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

[121] Grekow, J. and Dimitrova-Grekow, T. (2021). Monophonic music generation with a given emotion using conditional variational autoencoder. *IEEE Access*, 9:129088–129101.

[122] Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773.

[123] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5769–5779.

[124] Guo, X., Hong, J., Lin, T., and Yang, N. (2021). Relaxed wasserstein with applications to gans. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3325–3329. IEEE.

[125] Gwadera, R., Atallah, M. J., and Szpankowski, W. (2005). Reliable detection of episodes in event sequences. *Knowledge and Information Systems*, 7(4):415–437.

[126] Haloui, I., Gupta, J. S., and Feuillard, V. (2018). Anomaly detection with wasserstein gan. *arXiv preprint arXiv:1812.02463*.

[127] Hawkins, D. M. (1980). *Identification of outliers*, volume 11. Springer.

[128] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

[129] Heidenreich, N.-B., Schindler, A., and Sperlich, S. (2013). Bandwidth selection for kernel density estimation: a review of fully automatic selectors. *AStA Advances in Statistical Analysis*, 97(4):403–433.

[130] Heim, N., Pevnỳ, T., and Šmídl, V. (2020). Neural power units. *arXiv preprint arXiv:2006.01681*.

[131] Hennig, J. A., Umakantha, A., and Williamson, R. C. (2017). A classifying variational autoencoder with application to polyphonic music generation. *arXiv preprint arXiv:1711.07050*.

[132] Herrera, J. L. L., Figueroa, H. V. R., and Ramírez, E. J. R. (2018). Deep fraud. a fraud intention recognition framework in public transport context using a deep-learning approach. In *2018 International Conference on Electronics, Communications and Computers (CONIELECOMP)*, pages 118–125. IEEE.

[133] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30.

[134] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2018). Gans trained by a two time-scale update rule converge to a local nash equilibrium.

[135] Hilal, W., Andrew Gadsden, S., and Yawney, J. (2021). A review of anomaly detection techniques and applications in financial fraud. *Expert Systems with Applications*, page 116429.

[136] Hinton, G. E. et al. (1986). Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA.

[137] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

[138] Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE.

[139] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[140] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

[141] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

[142] Hou, X., Shen, L., Sun, K., and Qiu, G. (2017). Deep feature consistent variational autoencoder. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1133–1141. IEEE.

[143] Hou, Y., Zhai, J., and Chen, J. (2021). Coupled adversarial variational autoencoder. *Signal Processing: Image Communication*, 98:116396.

[144] Huang, E. H., Socher, R., Manning, C. D., and Ng, A. Y. (2012). Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 873–882.

[145] Huang, X. and Belongie, S. (2017). Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510.

[146] Huang, X., Tan, H., Lin, G., and Tian, Y. (2018). A lstm-based bidirectional translation model for optimizing rare words and terminologies. In *2018 international conference on artificial intelligence and big data (ICAIBD)*, pages 185–189. IEEE.

[147] Islam, A. K., Corney, M., Mohay, G., Clark, A., Bracher, S., Raub, T., and Flegel, U. (2010). Fraud detection in erp systems using scenario matching. In *IFIP Int. Information Security Conf.* Springer.

[148] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134.

[149] Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

[150] Jiang, C., Nian, Z., Guo, K., Chu, S., Zhao, Y., Shen, L., and Tu, K. (2019). Learning numeral embeddings. *arXiv preprint arXiv:2001.00003*.

[151] Jin, Z., Jiang, X., Wang, X., Liu, Q., Wang, Y., Ren, X., and Qu, H. (2021). Numgpt: Improving numeracy ability of generative pre-trained models. *arXiv preprint arXiv:2109.03137*.

[152] Jin Wang, Changqing Zhao, S. H. Y. G. O. A. A. A. (2022). Loguad: Log unsupervised anomaly detection based on word2vec. *Computer Systems Science and Engineering*, 41(3):1207–1222.

[153] Johansen, A. R. and Madsen, A. (2019). Measuring arithmetic extrapolation performance. In *33rd Conference on Neural Information Processing Systems*.

[154] John, G. H. and Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, page 338–345, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[155] Johnson, J. M. and Khoshgoftaar, T. M. (2020a). Hcpcs2vec: healthcare procedure embeddings for medicare fraud prediction. In *2020 IEEE 6th international conference on collaboration and internet computing (CIC)*, pages 145–152. IEEE.

[156] Johnson, J. M. and Khoshgoftaar, T. M. (2020b). Semantic embeddings for medical providers and fraud detection. In *2020 IEEE 21st international conference on information reuse and integration for data science (IRI)*, pages 224–230. IEEE.

[157] Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*.

[158] Jordon, J., Yoon, J., and Van Der Schaar, M. (2018). Pate-gan: Generating synthetic data with differential privacy guarantees. In *International conference on learning representations*.

[159] Joulin, A. and Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets.

[160] Kalchbrenner, N., Danihelka, I., and Graves, A. (2015). Grid Long Short-Term Memory. *arXiv preprint arXiv:1507.01526*.

[161] Kanika and Singla, J. (2020). A survey of deep learning based online transactions fraud detection systems. In *2020 Int. Conf. on Intelligent Engineering and Management (ICIEM)*, pages 130–136. IEEE.

[162] Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410.

[163] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. (2020). Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119.

[164] Kingma, D. P. (2017). Variational inference & deep learning: A new synthesis.

[165] Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

[166] Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29:4743–4751.

[167] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes.

[168] Kopparti, R. and Weyde, T. (2019). Factors for the generalisation of identity relations by neural networks.

[169] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

[170] Kullback, S. and Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86.

[171] Kusner, M. J. and Hernández-Lobato, J. M. (2016). Gans for sequences of discrete elements with the gumbel-softmax distribution.

[172] Lample, G. and Charton, F. (2019). Deep learning for symbolic mathematics.

[173] Laurent, H. and Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17.

[174] Lavrač, N., Podpečan, V., and Robnik-Šikonja, M. (2021). *Introduction to Representation Learning*, pages 1–16. Springer International Publishing, Cham.

[175] Lawrence, S., Giles, C. L., and Tsoi, A. C. (1997a). Lessons in neural network training: Overfitting may be harder than expected. In *AAAI/IAAI*, pages 540–545. Citeseer.

[176] Lawrence, S., Giles, C. L., Tsoi, A. C., and Back, A. D. (1997b). Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113.

[177] Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR.

[178] LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. (2006). A tutorial on energy-based learning. *Predicting structured data*, 1(0).

[179] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690.

[180] Lee, S.-I., Lee, H., Abbeel, P., and Ng, A. Y. (2006). Efficient l˜ 1 regularized logistic regression. In *Aaai*, volume 6, pages 401–408.

[181] Leonard, J., Kramer, M. A., and Ungar, L. (1992). A neural network architecture that computes its own reliability. *Computers & chemical engineering*, 16(9):819–835.

[182] Li, C., Liu, H., Chen, C., Pu, Y., Chen, L., Henao, R., and Carin, L. (2017). Alice: Towards understanding adversarial learning for joint distribution matching. *Advances in Neural Information Processing Systems*, 30:5495–5503.

[183] Li, J., Zhang, H., and Wei, Z. (2020a). The weighted word2vec paragraph vectors for anomaly detection over http traffic. *IEEE Access*, 8:141787–141798.

[184] Li, S.-C., Tai, B.-C., and Huang, Y. (2019). Evaluating variational autoencoder as a private data release mechanism for tabular data. In *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 198–1988. IEEE.

[185] Li, W., Yu, L., Wu, Y., and Paulson, L. C. (2020b). Isarstep: a benchmark for high-level mathematical reasoning. *arXiv preprint arXiv:2006.09265*.

[186] Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE.

[187] Liu, S., Bousquet, O., and Chaudhuri, K. (2017). Approximation and convergence properties of generative adversarial learning. *arXiv preprint arXiv:1705.08991*.

[188] Liu, W., Salzmann, M., and Fua, P. (2019). Context-aware crowd counting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5099–5108.

[189] Liu, Y. and Wang, Y. (2019). A robust malware detection system using deep learning on api calls. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 1456–1460. IEEE.

[190] Lopez-Rojas, E., Elmir, A., and Axelsson, S. (2016). PaySim: A financial mobile money simulator for fraud detection. In *European Modeling and Simulation Symposium (EMSS)*, pages 249–255. Dime University of Genoa.

[191] Lopez-Rojas, E. A. and Axelsson, S. (2016). A review of computer simulation for fraud detection research in financial datasets. In *2016 Future Technologies Conference (FTC)*, pages 932–935. IEEE.

[192] Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

[193] Lu, Y. and Lu, J. (2020). A universal approximation theorem of deep neural networks for expressing probability distributions.

[194] Luo, J., Xu, Y., Tang, C., and Lv, J. (2017). Learning inverse mapping by autoencoder based generative adversarial nets. In *International Conference on Neural Information Processing*, pages 207–216. Springer.

[195] Maaløe, L., Sønderby, C. K., Sønderby, S. K., and Winther, O. (2016). Auxiliary deep generative models. In *International conference on machine learning*, pages 1445–1453. PMLR.

[196] Maas, A. L., Hannun, A. Y., Ng, A. Y., et al. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer.

[197] Madsen, A. and Johansen, A. R. (2019). Neural arithmetic units. In *International Conference on Learning Representations*.

[198] Madsen, A. and Rosenberg Johansen, A. (2019). Measuring Arithmetic Extrapolation Performance.

[199] Maes, S., Tuyls, K., Vanschoenwinkel, B., and Manderick, B. (2002). Credit Card Fraud Detection using Bayesian and Neural Networks. In *Int. Naiso congress on Neuro Fuzzy Ttechnologies*, pages 261–270.

[200] Mao, X., Su, Z., Tan, P. S., Chow, J. K., and Wang, Y.-H. (2020). Is discriminator a good feature extractor?

[201] Marsden, M., McGuinness, K., Little, S., Keogh, C. E., and O'Connor, N. E. (2018). People, penguins and petri dishes: Adapting object counting models to new visual domains and object types without forgetting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8070–8079.

[202] Marti, G. (2020). Corrgan: Sampling realistic financial correlation matrices using generative adversarial networks. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8459–8463.

[203] Masud, M. M., Al-Khateeb, T. M., Khan, L., Aggarwal, C., Gao, J., Han, J., and Thuraisingham, B. (2011). Detecting recurring and novel classes in concept-drifting data streams. In *2011 IEEE 11th International Conference on Data Mining*, pages 1176–1181. IEEE.

[204] McAfee, L. (2018). Economic impact of cybercrime report.

[205] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[206] Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (2013). *Machine learning: An artificial intelligence approach*. Springer Science & Business Media.

[207] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space.

[208] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality.

[209] Mikolov, T., Yih, W.-t., and Zweig, G. (2013c). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751.

[210] Mishra, S., Mitra, A., Varshney, N., Sachdeva, B., and Baral, C. (2020). Towards question format independent numerical reasoning: A set of prerequisite tasks. *arXiv preprint arXiv:2005.08516*.

[211] Mistry, B., Farrahi, K., and Hare, J. (2021). Learning division with neural arithmetic logic modules.

[212] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Icml*.

[213] Najadat, H., Altiti, O., Aqouleh, A. A., and Younes, M. (2020). Credit card fraud detection based on machine and deep learning. In *2020 11th International Conference on Information and Communication Systems (ICICS)*, pages 204–208. IEEE.

[214] Nalmpantis, C. and Vrakas, D. (2019). Signal2vec: Time series embedding representation. In *International conference on engineering applications of neural networks*, pages 80–90. Springer.

[215] Nazabal, A., Olmos, P. M., Ghahramani, Z., and Valera, I. (2020). Handling incomplete heterogeneous data using vaes. *Pattern Recognition*, 107:107501.

[216] Ndubuaku, M. U., Anjum, A., and Liotta, A. (2019). Unsupervised anomaly thresholding from reconstruction errors. In Montella, R., Ciaramella, A., Fortino, G., Guerrieri, A., and Liotta, A., editors, *Internet and Distributed Computing Systems*, pages 123–129, Cham. Springer International Publishing.

[217] Ng, A. and Jordan, M. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In Dietterich, T., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press.

[218] Ng, A. Y. (2004). Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78.

[219] Ngo, P. C., Winarto, A. A., Kou, C. K. L., Park, S., Akram, F., and Lee, H. K. (2019). Fence gan: Towards better anomaly detection. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 141–148. IEEE.

[220] Nilson, S. (2021). The nilson report. *HSN Consultants, Oxnard (1970 to present). A twice-monthly newsletter on the payment card industry*.

[221] Norouzzadeh, M. S., Nguyen, A., Kosmala, M., Swanson, A., Palmer, M. S., Packer, C., and Clune, J. (2018). Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *Proceedings of the National Academy of Sciences*, 115(25):E5716–E5725.

[222] Odena, A., Olah, C., and Shlens, J. (2017). Conditional image synthesis with auxiliary classifier gans. In *International conference on machine learning*, pages 2642–2651. PMLR.

[223] Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature visualization. *Distill*, 2(11):e7.

[224] Omohundro, S. M. (1989). *Five balltree construction algorithms*. International Computer Science Institute Berkeley.

[225] Onoro-Rubio, D. and López-Sastre, R. J. (2016). Towards perspective-free object counting with deep learning. In *European conference on computer vision*, pages 615–629. Springer.

[226] Orosz, T. (2011). Analysis of sap development tools and methods. In *2011 15th IEEE International Conference on Intelligent Engineering Systems*, pages 439–443. IEEE.

[227] Oussidi, A. and Elhassouny, A. (2018). Deep generative models: Survey. In *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, pages 1–8.

[228] Ozenne, B., Subtil, F., and Maucort-Boulch, D. (2015). The precision–recall curve overcame the optimism of the receiver operating characteristic curve in rare diseases. *Journal of Clinical Epidemiology*, 68(8):855–859.

[229] Pantelidis, E., Bendiab, G., Shiaeles, S., and Kolokotronis, N. (2021). Insider threat detection using deep autoencoder and variational autoencoder neural networks. In *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 129–134. IEEE.

[230] Park, N., Mohammadi, M., Gorde, K., Jajodia, S., Park, H., and Kim, Y. (2018). Data synthesis based on generative adversarial networks. *arXiv preprint arXiv:1806.03384*.

[231] Parzen, E. (1962). On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065 – 1076.

[232] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

[233] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

[234] Perera, P., Oza, P., and Patel, V. M. (2021). One-class classification: A survey.

[235] Perera, P. and Patel, V. M. (2019). Learning deep features for one-class classification. *IEEE Transactions on Image Processing*, 28(11):5450–5463.

[236] Phua, C., Lee, V., Smith, K., and Gayler, R. (2010). A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*.

[237] Piatetsky-Shapiro, G. (1990). Knowledge discovery in real databases: A report on the ijcai-89 workshop. *AI Magazine*, 11(4):68.

[238] Pinkus, A. (1999). Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195.

[239] Popov, I. (2017). Malware detection using machine learning based on word2vec embeddings of machine code instructions. In *2017 Siberian symposium on data science and engineering (SSDSE)*, pages 1–4. IEEE.

[240] Prechelt, L. (1998). Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer.

[241] Press, O., Bar, A., Bogin, B., Berant, J., and Wolf, L. (2017). Language generation with recurrent generative adversarial networks without pre-training. *arXiv preprint arXiv:1706.01399*.

[242] Qin, T., Wu, K., and Xiu, D. (2019). Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 395:620–635.

[243] Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

[244] Quinlan, J. R. (1993). C 4.5: Programs for machine learning. *The Morgan Kaufmann Series in Machine Learning*.

[245] Raileanu, L. E. and Stoffel, K. (2004). Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93.

[246] Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707.

[247] Rebekić, A., Lončarić, Z., Petrović, S., and Marić, S. (2015). Pearson's or spearman's correlation coefficient-which one to use? *Poljoprivreda*, 21(2):47–54.

[248] Reed, S. and De Freitas, N. (2015). Neural Programmer-Interpreters. *arXiv preprint arXiv:1511.06279*.

[249] Ren, Y. and Du, Y. (2020). Enhancing the numeracy of word embeddings: A linear algebraic perspective. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 170–178. Springer.

[250] Richardson, E. and Weiss, Y. (2018). On gans and gmms.

[251] Ring, M., Landes, D., Dallmann, A., and Hotho, A. (2017). Ip2vec: Learning similarities between ip addresses. *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 657–666.

[252] Ring, M., Otto, F., Becker, M., Niebler, T., Landes, D., and Hotho, A. (2015). Condist: A context-driven categorical distance measure. In *Machine Learning and Knowledge Discovery in Databases*, pages 251–266, Cham. Springer International Publishing.

[253] Ring, M., Schlör, D., Landes, D., and Hotho, A. (2019a). Flow-based network traffic generation using Generative Adversarial Networks. *Comput. Secur.*, 82:156.

[254] Ring, M., Schlör, D., Landes, D., and Hotho, A. (2019b). Flow-based Network Traffic Generation using Generative Adversarial Networks. *Computer & Security*, 82:156–172.

[255] Ring, M., Schlör, D., Wunderlich, S., Landes, D., and Hotho, A. (2021). Malware Detection on Windows Audit Logs using LSTMs. *Computers & Security*, 109:102389.

[256] Roberts, A., Engel, J., and Eck, D. (2017). Hierarchical variational autoencoders for music. In *NIPS Workshop on Machine Learning for Creativity and Design*, volume 3.

[257] Roberts, A., Engel, J., Raffel, C., Hawthorne, C., and Eck, D. (2019). A hierarchical latent vector model for learning long-term structure in music.

[258] Roberts, S. and Tarassenko, L. (1994). A probabilistic resource allocating network for novelty detection. *Neural Computation*, 6(2):270–284.

[259] Rong, X. (2016). word2vec parameter learning explained.

[260] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[261] Roy, A., Sun, J., Mahoney, R., Alonzi, L., Adams, S., and Beling, P. (2018). Deep learning detecting fraud in credit card transactions. In *2018 Systems and Information Engineering Design Symposium (SIEDS)*, pages 129–134. IEEE.

[262] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[263] Ruff, L., Kauffmann, J. R., Vandermeulen, R. A., Montavon, G., Samek, W., Kloft, M., Dietterich, T. G., and Müller, K.-R. (2021). A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*, 109(5):756–795.

[264] Ruff, L., Vandermeulen, R. A., Görnitz, N., Binder, A., Müller, E., Müller, K.-R., and Kloft, M. (2020). Deep semi-supervised anomaly detection. In *International Conference on Learning Representations*.

[265] Russac, Y., Caelen, O., and He-Guelton, L. (2018). Embeddings of categorical variables for sequential data in fraud context. In *International conference on advanced machine learning technologies and applications*, pages 542–552. Springer.

[266] Sabau, A. S. (2012). Survey of clustering based financial fraud detection research. *Informatica Economica*, 16(1):110.

[267] Sahlgren, M. (2008). The distributional hypothesis. *Italian Journal of Disability Studies*, 20:33–53.

[268] Sahlgren, M. and Lenci, A. (2016). The effects of data size and frequency range on distributional semantic models. *CoRR*, abs/1609.08293.

[269] Saito, T. and Rehmsmeier, M. (2015). The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432.

[270] Sak, H., Senior, A. W., and Beaufays, F. (2014). Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling. In *International Speech Communication Association (INTERSPEECH)*, pages 338–342.

[271] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. *Advances in neural information processing systems*, 29:2234–2242.

[272] Saxena, D. and Cao, J. (2021). Generative adversarial networks (gans) challenges, solutions, and future directions. *ACM Computing Surveys (CSUR)*, 54(3):1–42.

[273] Saxton, D., Grefenstette, E., Hill, F., and Kohli, P. (2019). Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*.

[274] Schermann, M. and Boss, S. R. (2014). The white-collar hacking contest: A novel approach to teach forensic investigations in a digital world. In *2014 Dewald Roode Workshop on Information Systems Security Research*.

[275] Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., and Langs, G. (2017). Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International conference on information processing in medical imaging*, pages 146–157. Springer.

[276] Schlör, D., Ring, M., and Hotho, A. (2020a). iNALU: Improved neural arithmetic logic unit. *Frontiers in Artificial Intelligence*, 3:71.

[277] Schlör, D., Ring, M., Krause, A., and Hotho, A. (2020b). Financial Fraud Detection with Improved Neural Arithmetic Logic Units. Fifth Workshop on MIning DAta for financial applicationS.

[278] Schölkopf, B., Herbrich, R., and Smola, A. J. (2001). A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer.

[279] Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., and Platt, J. (1999). Support vector method for novelty detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, pages 582–588.

[280] Schreyer, M., Sattarov, T., Borth, D., Dengel, A., and Reimer, B. (2018). Detection of Anomalies in Large Scale Accounting Data using Deep Autoencoder Neural Networks. In *GPU Technology Conference - Silicon Valley*.

[281] Schultz, M. and Tropmann-Frick, M. (2020). Autoencoder neural networks versus external auditors: Detecting unusual journal entries in financial statement audits. In *Proceedings of the 53rd Hawaii Int. Conf. on System Sciences*.

[282] Schütze, H., Manning, C. D., and Raghavan, P. (2008). *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge.

[283] Scott, D. W. (2009). Sturges' rule. *WIREs Comput. Stat.*, 1(3):303–306.

[284] Scott, D. W. (2012). Multivariate density estimation and visualization. In *Handbook of computational statistics*, pages 549–569. Springer.

[285] Scott, D. W. and Sain, S. R. (2005). Multidimensional density estimation. *Handbook of statistics*, 24:229–261.

[286] Seguí, S., Pujol, O., and Vitrià, J. (2015). Learning to count with deep object features.

[287] Shao, H., Yao, S., Sun, D., Zhang, A., Liu, S., Liu, D., Wang, J., and Abdelzaher, T. (2020). Controlvae: Controllable variational autoencoder. In *International Conference on Machine Learning*, pages 8655–8664. PMLR.

[288] Shen, A., Tong, R., and Deng, Y. (2007). Application of Classification Models on Credit Card Fraud Detection. In *Int. Conf. on Service Systems and Service Management*, pages 1–4. IEEE.

[289] Shental, N., Bar-Hillel, A., Hertz, T., and Weinshall, D. (2004). Computing gaussian mixture models with em using equivalence constraints.

[290] Shwartz-Ziv, R. and Armon, A. (2021). Tabular data: Deep learning is not all you need. *arXiv preprint arXiv:2106.03253*.

[291] Siblini, W., Fréry, J., He-Guelton, L., Oblé, F., and Wang, Y.-Q. (2020). Master your metrics with calibration. *Advances in Intelligent Data Analysis XVIII*, page 457–469.

[292] Simard, P. Y., Steinkraus, D., Platt, J. C., et al. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3.

[293] Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science.

[294] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

[295] Sofaer, H. R., Hoeting, J. A., and Jarnevich, C. S. (2019). The area under the precision-recall curve as a performance metric for rare binary events. *Methods in Ecology and Evolution*, 10(4):565–577.

[296] Somepalli, G., Wu, Y., Balaji, Y., Vinzamuri, B., and Feizi, S. (2021). Unsupervised anomaly detection with adversarial mirrored autoencoders. In *Uncertainty in Artificial Intelligence*, pages 365–375. PMLR.

[297] Spindler, A., Geach, J. E., and Smith, M. J. (2021). Astrovader: astronomical variational deep embedder for unsupervised morphological classification of galaxies and synthetic image generation. *Monthly Notices of the Royal Astronomical Society*, 502(1):985–1007.

[298] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[299] Stearns, D. L. (2011). *Electronic value exchange: Origins of the VISA electronic payment system*. Springer.

[300] Stolfo, S. J., Prodromidis, A. L., Tselepis, S., Lee, W., Fan, D. W., and Chan, P. K. (1997). Jam: Java agents for meta-learning over distributed databases. In *KDD*, volume 97, pages 74–81.

[301] Suh, S. and Choi, S. (2016). Gaussian copula variational autoencoders for mixed data.

[302] Sun, T. and Vasarhelyi, M. A. (2018). Predicting credit card delinquencies: An application of deep neural networks. *Intelligent Systems in Accounting, Finance and Management*, 25(4):174–189.

[303] Sundararaman, D., Si, S., Subramanian, V., Wang, G., Hazarika, D., and Carin, L. (2020). Methods for numeracy-preserving word embeddings. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4742–4753.

[304] Sung, Y., Jang, S., Jeong, Y.-S., and Park, J. H. J. J. (2020). Malware classification algorithm using advanced word2vec-based bi-lstm for ground control stations. *Computer Communications*, 153:342–348.

[305] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR.

[306] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[307] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

[308] Takahashi, S., Chen, Y., and Tanaka-Ishii, K. (2019). Modeling financial time-series with generative adversarial networks. *Physica A: Statistical Mechanics and its Applications*, 527:121261.

[309] Tamura, K. and Matsuura, K. (2019). Improvement of anomaly detection performance using packet flow regularity in industrial control networks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 102(1):65–73.

[310] Testolin, A. (2020). The challenge of modeling the acquisition of mathematical concepts. *Frontiers in Human Neuroscience*, 14:100.

[311] Thawani, A., Pujara, J., Szekely, P. A., and Ilievski, F. (2021). Representing numbers in nlp: a survey and a vision. *arXiv preprint arXiv:2103.13136*.

[312] Theil, H. (1970). On the estimation of relationships involving qualitative variables. *American Journal of Sociology*, 76(1):103–154.

[313] Theis, L., Oord, A. v. d., and Bethge, M. (2015). A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*.

[314] Tikhonov, A., Yamshchikov, I. P., et al. (2017). Music generation with variational recurrent autoencoder supported by history. *arXiv preprint arXiv:1705.05458*.

[315] Todeschini, R. (1989). k-nearest neighbour method: The influence of data transformations and metrics. *Chemometrics and intelligent laboratory systems*, 6(3):213–220.

[316] Tolstikhin, I., Gelly, S., Bousquet, O., Simon-Gabriel, C.-J., and Schölkopf, B. (2017). Adagan: Boosting generative models. *arXiv preprint arXiv:1701.02386*.

[317] Trask, A., Hill, F., Reed, S. E., Rae, J., Dyer, C., and Blunsom, P. (2018). Neural Arithmetic Logic Units. In *Advances in Neural Information Processing Systems*, pages 8035–8044.

[318] Trottier, L., Giguere, P., and Chaib-Draa, B. (2017). Parametric exponential linear unit for deep convolutional neural networks. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 207–214. IEEE.

[319] Valvano, G., Leo, A., and Tsaftaris, S. A. (2021a). Re-using adversarial mask discriminators for test-time training under distribution shifts. *arXiv preprint arXiv:2108.11926*.

[320] Valvano, G., Leo, A., and Tsaftaris, S. A. (2021b). Stop throwing away discriminators! re-using adversaries for test-time training. In *Domain Adaptation and Representation Transfer, and Affordable Healthcare and AI for Resource Diverse Global Health*, pages 68–78. Springer.

[321] van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016). Conditional image generation with pixelcnn decoders.

[322] Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

[323] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[324] Villani, C. (2009). *Optimal transport: old and new*, volume 338, page 107. Springer.

[325] Wallace, E., Wang, Y., Li, S., Singh, S., and Gardner, M. (2019). Do nlp models know numbers? probing numeracy in embeddings. *arXiv preprint arXiv:1909.07940*.

[326] Wang, J., Tang, Y., He, S., Zhao, C., Sharma, P. K., Alfarraj, O., and Tolba, A. (2020). Logevent2vec: Logevent-to-vector based anomaly detection for large-scale logs in internet of things. *Sensors*, 20(9).

[327] Wang, M., Xu, L., and Guo, L. (2018). Anomaly detection of system logs based on natural language processing and deep learning. In *2018 4th International Conference on Frontiers of Signal Processing (ICFSP)*, pages 140–144. IEEE.

[328] Wang, Q., Fang, L., Zhu, Z., and Huang, J. (2021). Detection algorithm of the mimicry attack based on variational auto-encoder. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 114–120. IEEE.

[329] Wang, R., Fu, B., Fu, G., and Wang, M. (2017). Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, pages 1–7.

[330] Wang, S., Liu, G., Li, Z., Xuan, S., Yan, C., and Jiang, C. (2018). Credit card fraud detection using capsule network. In *IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC)*, pages 3679–3684.

[331] Wei, R. and Mahmood, A. (2020). Recent advances in variational autoen-coders with representation learning for biomedical informatics: A survey. *Ieee Access*.

[332] Weng, H., Ji, S., Duan, F., Li, Z., Chen, J., He, Q., and Wang, T. (2019). Cats: cross-platform e-commerce fraud detection. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1874–1885. IEEE.

[333] White, T. (2016). Sampling generative networks.

[334] Wolters, T. (2000). 'carry your credit in your pocket': The early history of the credit card at bank of america and chase manhattan. *Enterprise & Society*, 1(2):315–354.

[335] Wu, Y., Burda, Y., Salakhutdinov, R., and Grosse, R. (2016). On the quantitative analysis of decoder-based generative models. *arXiv preprint arXiv:1611.04273*.

[336] Wunderlich, S., Ring, M., Landes, D., and Hotho, A. (2019). Comparison of system call representations for intrusion detection. In *International Joint Conference: 12th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2019) and 10th International Conference on EUropean Transnational Education (ICEUTE 2019)*, pages 14–24. Springer.

[337] Wunderlich, S., Ring, M., Landes, D., and Hotho, A. (2020). The impact of different system call representations on intrusion detection. *Logic Journal of the IGPL*.

[338] Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

[339] Xu, C., Ren, J., Zhang, D., Zhang, Y., Qin, Z., and Ren, K. (2019a). Ganobfuscator: Mitigating information leakage under gan via differential privacy. *IEEE Transactions on Information Forensics and Security*, 14(9):2358–2371.

[340] Xu, H., Chen, W., Zhao, N., Li, Z., Bu, J., Li, Z., Liu, Y., Zhao, Y., Pei, D., Feng, Y., et al. (2018a). Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, pages 187–196.

[341] Xu, K., Zhang, M., Li, J., Du, S. S., ichi Kawarabayashi, K., and Jegelka, S. (2021). How neural networks extrapolate: From feedforward to graph neural networks.

[342] Xu, L., Skoularidou, M., Cuesta-Infante, A., and Veeramachaneni, K. (2019b). Modeling tabular data using conditional gan. *arXiv preprint arXiv:1907.00503*.

[343] Xu, L. and Veeramachaneni, K. (2018). Synthesizing tabular data using generative adversarial networks. *arXiv preprint arXiv:1811.11264*.

[344] Xu, Q., Huang, G., Yuan, Y., Guo, C., Sun, Y., Wu, F., and Weinberger, K. (2018b). An empirical study on evaluation metrics of generative adversarial networks. *arXiv preprint arXiv:1806.07755*.

[345] Xue, F., Li, X., Zhang, T., and Hu, N. (2021). Stock market reactions to the covid-19 pandemic: The moderating role of corporate big data strategies based on word2vec. *Pacific-Basin Finance Journal*, 68:101608.

[346] Yan, X., Yang, J., Sohn, K., and Lee, H. (2016). Attribute2image: Conditional image generation from visual attributes. In *European Conference on Computer Vision*, pages 776–791. Springer.

[347] Yan, Y., Swersky, K., Koutra, D., Ranganathan, P., and Hashemi, M. (2020). Neural execution engines: Learning to execute subroutines.

[348] Yang, Y., Guan, X., and You, J. (2002). Clope: a fast and effective clustering algorithm for transactional data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 682–687.

[349] Yannikos, Y., Franke, F., Winter, C., and Schneider, M. (2010). 3lspg: Forensic tool evaluation by three layer stochastic process-based generation of data. In *Int. Workshop on Computational Forensics*. Springer.

[350] Yao, R., Liu, C., Zhang, L., and Peng, P. (2019). Unsupervised anomaly detection using variational auto-encoder based feature extraction. In *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 1–7. IEEE.

[351] Ye, N. and Chen, Q. (2001). An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and reliability engineering international*, 17(2):105–112.

[352] Yeh, H.-Y., Yeh, Y.-C., and Shen, D.-B. (2020). Word vector models approach to text regression of financial risk prediction. *Symmetry*, 12(1).

[353] Yeung, D.-Y. and Chow, C. (2002). Parzen-window network intrusion detectors. In *Object recognition supported by user interaction for service robots*, volume 4, pages 385–388. IEEE.

[354] Yoon, J., Jarrett, D., and Van der Schaar, M. (2019). Time-series generative adversarial networks.

[355] Zaremba, W. and Sutskever, I. (2014). Learning to Execute. *arXiv preprint arXiv:1410.4615*.

[356] Zaremba, W. and Sutskever, I. (2015). Learning to execute.

[357] Zenati, H., Foo, C. S., Lecouat, B., Manek, G., and Chandrasekhar, V. R. (2018a). Efficient gan-based anomaly detection. *arXiv preprint arXiv:1802.06222*.

[358] Zenati, H., Romain, M., Foo, C.-S., Lecouat, B., and Chandrasekhar, V. (2018b). Adversarially learned anomaly detection. In *2018 IEEE International conference on data mining (ICDM)*, pages 727–736. IEEE.

[359] Zhai, S., Cheng, Y., Lu, W., and Zhang, Z. (2016). Deep structured energy based models for anomaly detection. In *International Conference on Machine Learning*, pages 1100–1109. PMLR.

[360] Zhan, Q. and Yin, H. (2018). A loan application fraud detection method based on knowledge graph and neural network. In *Proceedings of the 2nd International Conference on Innovation in Artificial Intelligence*, pages 111–115.

[361] Zhang, C., Li, H., Wang, X., and Yang, X. (2015). Cross-scene Crowd Counting via Deep Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 833–841.

[362] Zhang, L., Zhang, S., and Balog, K. (2019). Table2vec. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in IR*.

[363] Zhang, Y., Hu, A., Wang, J., and Zhang, Y. (2021). Detection of fraud statement based on word vector: Evidence from financial companies in china. *Finance Research Letters*, page 102477.

[364] Zhao, J. and Mau, J. (2020). Discovery of governing equations with recursive deep neural networks.

[365] Zhong, P., Mo, Y., Xiao, C., Chen, P., and Zheng, C. (2019). Rethinking generative mode coverage: A pointwise guaranteed approach. *Advances in Neural Information Processing Systems*, 32:2088–2099.

[366] Zhou, Y., Li, D., Huo, S., and Kung, S.-Y. (2020). Soft-root-sign activation function. *arXiv preprint arXiv:2003.00547*.

[367] Zhu, J.-Y., Krähenbühl, P., Shechtman, E., and Efros, A. A. (2016). Generative visual manipulation on the natural image manifold. In *European conference on computer vision*, pages 597–613. Springer.

[368] Zhu, X. and Goldberg, A. B. (2009). Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130.

[369] Zixu, T., Liyanage, K. S. K., and Gurusamy, M. (2020). Generative adversarial network and auto encoder based anomaly detection in distributed iot networks. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–7. IEEE.

[370] Łukasz Kaiser and Sutskever, I. (2016). Neural gpus learn algorithms.