Deep Representation Learning for Analyzing Financial and Cybersecurity Data

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

von
David Biesner
aus
Düsseldorf

Bonn, 2023

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter:	Prof. Dr. Christian Bauckhage
2. Gutachter:	Prof. Dr. Stefan Wrobel

Tag der Promotion:	25.04.2024
Erscheinungsjahr:	2024

Abstract

Recent years have shown a surge in the capability of deep learning models for various domains of interest and different types of data. Machine learning models are now capable of understanding, categorizing and generating unstructured data on par with or even surpassing human performance. While many tasks requiring human intelligence and expertise are still out of reach for machine learning models, these systems have the potential to automate tasks that require a large amount of manual work by human experts. In this thesis, we will investigate the application of machine learning models in the financial auditing process and in the field of cybersecurity, with data primarily consisting of unstructured text data and numerical tabular data.

Take for example the financial auditing process, in which expert auditors are required to manually check the compliance of financial reports with accounting standards. The process of handling the text documents and finding relevant information in the reports is time-consuming and prone to errors. Machine learning models have the potential to significantly reduce the amount of manual work required by auditors by automating repetitive tasks and providing a first pass of the data to the human expert.

When working with textual data, one of the most important aspects of any machine learning model is the representation of the data. Text data is inherently unstructured and difficult to process for machine learning models due to its high dimensionality and discrete nature. To address this issue, we transform the data into a lower-dimensional continuous vector space, a process that requires a model to learn a useful representation of the data. We call this process *representation learning*. Having learned a useful representation of the data, we can apply various machine learning models to the data for a variety of tasks in a flexible manner, meaning we can use the same learned representation for different tasks.

In this thesis, we will show how neural networks are able to learn a useful representation of data and how this representation can be used to reduce the need for manual work by human experts significantly. We will further show how representations with meaningful geometric properties allow for the generation of new data beyond the capabilities of classical generation algorithms.

We first discuss a method for extraction of interpretable word embeddings from a corpus of text data, based on a matrix factorization approach which is novel for the field of natural language processing. We demonstrate how the resulting word embeddings are able to capture semantic relationships between words and are interpretable in the sense that a word embedding is given as a combination of a number of topics extracted from the corpus.

We then discuss the application of neural networks for named entity recognition and anonymization of financial report data. We analyze the limitations of word embedding methods and propose the application of character-based recurrent neural networks for sequence tagging tasks. We show that the use of these models allows for anonymization of free-form text data without the need for a fixed vocabulary.

Following this, we introduce transformer architectures, which have shown great potential for a variety of natural language processing tasks in recent years. We apply a transformer-based architecture for textual matching in financial report data. We show that the embeddings generated by transformer-based language models are expressive enough to be used in a textual similarity matching task without any additional classification layers.

For the second field of applications, we discuss the use of deep representation learning for password recovery, in which we aim to generate a large number of novel and realistic password strings based on a given training set. We first show that variational autoencoders and generative transformer architectures can generate passwords of significantly higher quality than previous deep-learning-based approaches and consequently, demonstrate how a novel architecture combining generative transformers and variational autoencoders can increase the generation quality above the performance of the individual architectures.

Finally, we consider a further application in the field of automated auditing and describe a method for consistency checks for numerical tables in financial reports. Given a table in a text document, we apply binary optimization on Hopfield networks to check the validity of sums in the table. We show that this approach consistently solves binary optimization problems in public financial reports and discuss the potential of running the algorithm on specialized quantum hardware.

Acknowledgements

I would like to express my deep gratitude to everyone who has supported me during my time as PhD student and in the writing of this thesis.

A special thanks to my professor and advisor Christian Bauckhage for his invaluable guidance and expertise. I am also thankful to Rafet Sifa for his constant support, his advise, his supply of new ideas for research and industry projects, and his feedback on the thesis itself. Additionally, I extend my appreciation to all of my co-authors for their collaboration and their contribution to our joint research. Their collective efforts have significantly enhanced the quality and impact of this thesis.

Contents

Abstract iii

1 Introduction 1
- 1.1 Motivation 1
- 1.2 Structure of the Thesis 5

2 Applications in Finance and Cybersecurity 6
- 2.1 Auditing and Financial Data Analysis 6
- 2.2 Text Generation for Cybersecurity – Password Recovery 10

3 Representation Modeling 14
- 3.1 Embedding and Classification 14
  - 3.1.1 Topic Modeling 15
  - 3.1.2 Text Matching 15
  - 3.1.3 Sequence Classification and Named Entity Recognition 15
  - 3.1.4 Word Embeddings 16
  - 3.1.5 Recurrent Neural Networks 19
  - 3.1.6 Transformers 20
  - 3.1.7 BERT 20
- 3.2 Generative Models 21
  - 3.2.1 GANs 22
  - 3.2.2 Variational Autoencoders 23
  - 3.2.3 Latent Space Geometry for Text Generation 24
  - 3.2.4 GPT and GPT2 25
- 3.3 Tokenization 26
- 3.4 Transfer Learning 28
- 3.5 Binary Optimization and Hopfield Networks 28

4 Interpretable Topic Extraction and Word Embedding Learning Using Non-Negative Tensor DEDICOM 30

5 Anonymization of German Financial Documents using Neural Network-based Language Models with Contextual Word Representations 32

6 Zero-Shot Text Matching for Automated Auditing using Sentence Transformers 35

7 Advances in Password Recovery Using Generative Deep Learning Techniques 37

8 Combining Variational Autoencoders and Transformer Language Models for Improved Password Generation 39

9 Solving Subset Sum Problems using Quantum Inspired Optimization Algorithms with Applications in Auditing and Financial Data Analysis 41

10 Conclusion and Future Work 43
10.1 Summary of Results 43
10.2 Large Language Models 44

Bibliography 48

List of Figures 56

CHAPTER 1

Introduction

In this thesis, we investigate the evolution of neural networks for representation learning, mainly in the field of natural language processing (NLP). We demonstrate how developments in model architectures, from comparatively simple word embeddings to deep neural networks with large parameter counts, help to solve increasingly complex NLP-related tasks, such as named entity recognition, text matching and text generation. We focus the applications on the domains of financial text data, analyzing financial report texts to aid the development of automated auditing systems, and text data related to cybersecurity, applying text generation algorithms to password data. We further give an outlook on the application of neural networks on financial data beyond NLP, namely binary optimization algorithms using recurrent neural networks for numerical consistency checks in financial reports.

In this chapter, we present an overview of the motivation for this thesis and describe why representation learning is especially valuable for the tasks we consider in this thesis.

1.1 Motivation

When developing machine learning solutions to various problems, we often want to align the mechanisms of the algorithms with our human intuition. In many real-world applications, the data we are given makes this challenging. If we want to classify an image, we are unlikely to analyze the color values of every single pixel but rather want to take in the picture as a whole. When reading a sentence, we do not think about words as sequences of individual characters but know that words need the context of other words to make sense. To this end, we need methods to encode the raw data into a representation that makes these abstract concepts readable to further machine learning algorithms. If we have a reliable method of encoding data into a unified format, we can apply various algorithms for different downstream tasks regardless of the specific properties of a single data point.

All of the models we describe in this thesis share a common mechanism. Each algorithm is trained to learn a compressed representation of input data, which contains all relevant information of the data in a lower-dimensional space. We call this process embedding and denote the lower-dimensional space as embedding space or latent space. This is important for many downstream tasks, as it allows for both robust classification of input data and the generation of new data. We call the process of learning a model for embedding input data into a compressed representation *representation learning* [1].

Embedding models are especially important for text data and other sequences of discrete data. Consider an image made up of $W \times H$ pixels, in which each pixel can be represented as a float value for red, green, and blue color. The vector representation $x \in \mathbb{R}^{W \times H \times 3}$ already is machine-readable, meaning it can be directly processed by machine learning algorithms. We can add perturbations to the image by adding artificial noise to the pixel values, shifting each pixel value by a random small amount. The resulting image will still be machine-readable and will still be part of the same data space as the original image [2].

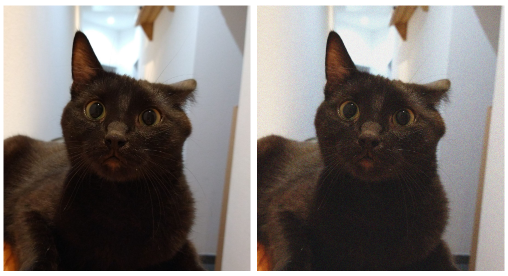
Consider, in contrast, a sentence made up of T words. The sequence of words is by itself not yet useful, to make text readable for machines and algorithms, we need to transform it into a numerical representation. This can be done on a word basis, where we have a vocabulary of all words in the corpus, and each word is represented by a unique index. Alternatively, we can use a character-based approach, where each character is represented by a unique index. Further, we can adopt the approach of byte-pair encoding, where we use a vocabulary of substrings of the corpus, and each substring is represented by a unique index. In all of these cases, we transform the sequence of characters into a sequence of indices or one-hot encoded vectors. This sparse representation of the data perfectly represents the data in a way that is readable for machines, but it has several issues.

First, it differs from the natural representation of images in that it is discrete and not continuous, therefore not robust to small perturbations. Adding artificial noise to the input data can result in a completely different sentence or lead to the output not being a valid sentence in data space at all. In Figure 1.1, we demonstrate this concept in an example sentence and an example image. While the text loses its meaning entirely by small changes, the image is robust to such perturbations.

```
The cat sat on the mat.
Uif dbu tbu po uif nbu.

The cat sat on the mat.
The cat sat on the car.
```

(a) Adding small artificial noise, here shifting each character by one index in the alphabet, results in a completely different sentence.



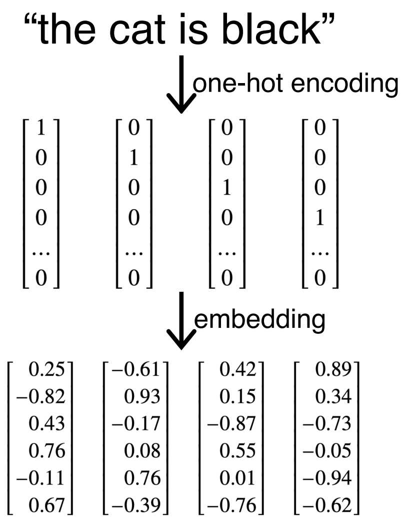
(b) Even adding larger amounts of artificial noise, here shifting each pixel by around 20%, does not change the content of the image significantly.

Figure 1.1: We demonstrate how discrete data like text and continuous data like images behave differently to the addition of artificial noise.

Further, this discrete representation is not very expressive. Consider the words `cat`, `dog`, and `car`. As one-hot encoded words, they are all orthogonal to each other and have no relation to each other in data space. A classification model learning to classify words in a sentence into classes would have to learn a separate weight vector for each of these words, and could not learn that `cat` and `dog` are more similar to each other than to `car`. Moreover, in a character based encoding, the words `cat` and `car` are closer to each other than the words `cat` and `dog`, which makes the classification task even harder.

To solve these issues, we can use embedding models. These models act as a preprocessor to the actual task (e.g., classification) and produce a compressed representation of the input data. An embedding model takes as input a single word index or a sequence of indices representing words, characters, or substring tokens and outputs one or multiple vectors of a fixed size, which we call the dimension of the embedding.

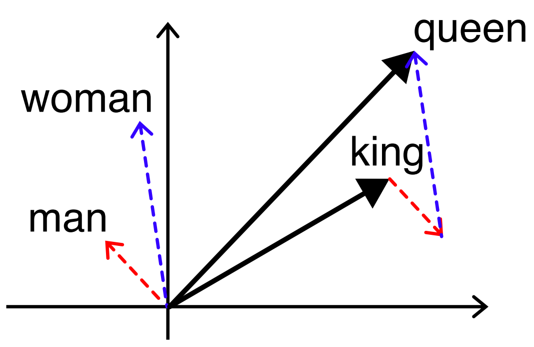
This compressed representation is usually much lower-dimensional than the original encoding, for example, a vector between 256 and 4 096 dimensions as opposed to a one-hot word encoding of over 100 000



$$\begin{bmatrix}1\\0\\0\\0\\\vdots\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\\0\\\vdots\\0\end{bmatrix}\begin{bmatrix}0\\0\\1\\0\\\vdots\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\\\vdots\\0\end{bmatrix}$$

$$\begin{bmatrix}0.25\\-0.82\\0.43\\0.76\\-0.11\\0.67\end{bmatrix}\begin{bmatrix}-0.61\\0.93\\-0.17\\0.08\\0.76\\-0.39\end{bmatrix}\begin{bmatrix}0.42\\0.15\\-0.87\\0.55\\0.01\\-0.76\end{bmatrix}\begin{bmatrix}0.89\\0.34\\-0.73\\-0.05\\-0.94\\-0.62\end{bmatrix}$$

(a) An embedding is a representation of the input data in latent space. Here we encode a series of words into high-dimensional one-hot vectors and embed them into a much lower-dimensional space.



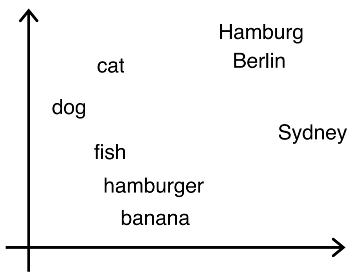
(b) Embeddings in latent space encode semantic relationships in linear algebraic relations. In this simple example, we see the latent representations for `man`, `woman`, `king`, and `queen` behave in such a way that $v_{\texttt{queen}} = v_{\texttt{king}} - v_{\texttt{man}} + v_{\texttt{woman}}$.

Figure 1.2: The process of text embedding transforms text from discrete indices or one-hot encoded vectors into float valued numerical vectors in latent space. We require a well-trained embedding model to assign semantic meaning to the embedding vectors, which results in geometric properties of the embeddings mirroring semantic properties of the respective words.

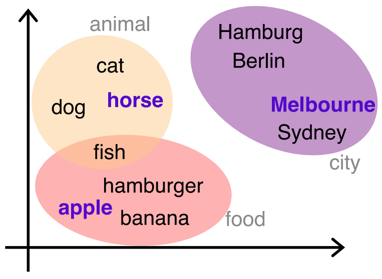
dimensions or a one-hot character encoding of $V \times T$ dimensions, where V is the number of characters in the vocabulary, and T is the length of the word or sequence.

Ideally, we require the representation to be more expressive, words that are similar in meaning should be closer to each other in the latent space. Consider the example above. A good embedding model will embed the words `cat`, `dog`, and `car` in such a way that the vectors representing `cat` and `dog` are closer to each other than to the vector representing `car`. This allows for a classification model to not learn a separate weight for each word but to learn that certain regions in latent space correspond to certain classes. This not only reduces the number of parameters that the classification model has to learn but also allows for better generalization. Consider a classification model that has learned that both `cat` and `dog` are animals. Even if the model has never seen the word `horse`, it can still classify it as an animal because its latent representation will be close to the latent representations of `cat` and `dog`. This is not possible with a one-hot encoding because the word `horse` is orthogonal to the words `cat`, `dog`, and `car`. For an example of this property of text embedding, consider Figure 1.2(b), in which we encode the words `man`, `woman`, `king`, and `queen` into two-dimensional latent space. We see that, for one, the embedding makes sure that similar words, `king` and `queen`, appear in similar regions of latent space. Further, we see that the embeddings have geometric properties which mirror the semantic properties of the input text. Just as a king is a male queen, the vectors have the algebraic property that $v_{\texttt{queen}} = v_{\texttt{king}} - v_{\texttt{man}} + v_{\texttt{woman}}$ [3].

For an example of how this embedding can aid the classification process, see Figure 1.3. We again show



(a) A word embedding model maps semantically similar words into similar regions in latent space. These word embeddings can be used to train a downstream model, e.g. for classification.



(b) A classification model can learn the semantic properties of the regions in latent space for robust classification.

Figure 1.3: An example for a classification task on word embeddings. We see that the model can easily classify the individual words into their respective categories. Words that fit multiple categories, here `fish`, are located on the borders of the decision spaces. New words that have not been seen during training, here `apple`, `horse`, and `Melbourne`, can be correctly classified because of the location of their embeddings in latent space.

an embedding in two dimensions, in which various words are embedded into latent space. A model classifying these words in categories (e.g. animals, food, and cities) can learn that certain regions in latent space correspond to their respective categories and can also classify terms it has not seen during training correctly, like the words `horse`, `apple`, and `Melbourne` in Figure 1.3(b). Considering Figure 1.3, additionally note how the words `hamburger` and `Hamburg` are very similar in terms of characters but are embedded into entirely different regions in latent space due to their semantic dissimilarity.

The capability for expressive representations depends on the embedding model and its training. Embedding models can be trained in an unsupervised manner, where the model is trained on raw data and learns the embedding in such a way that similar data points are close to each other in latent space by their appearance in similar contexts. Or the models can be trained in a supervised manner, where the model is trained on a task such as classification or generation and learns to embed data in such a way that the task can be solved more easily. During this thesis, we will investigate the effect of different types of training on the performance downstream models for their respective tasks.

A further advantage of embedding models is the general process of moving the representation of the data from a discrete space to a continuous space. This allows for training techniques that are also found in machine learning models for image data and other continuous data, such as adding noise to the input data to prevent overfitting. Compare the examples in Figure 1.1, where we add noise to both an image and a sentence. Embedding text in a continuous space additionally allows for generating artificial latent vectors by sampling from a known continuous distribution, like a normal distribution. In later chapters, we will demonstrate how adding noise to an embedding and sampling from a distribution on the latent space allows for the generation of new artificial text data, and how geometric properties of latent space embeddings can be utilized for the generation of text with beneficial semantic properties.

In the course of this thesis, we will track the evolution of data representation and embeddings from comparatively simple word embedding models to complex deep-learning-based models for embedding long sequences, tables, and entire documents. We will not only look at representation learning from a theoretical perspective but always apply the ideas, algorithms, and model architectures to problems in two distinct domains. These domains are the analysis of financial report documents towards systems for automated auditing and the analysis of password data for applications in cybersecurity. We will show how the methods presented in the coming chapters are crucial building blocks for automated systems that greatly reduce the need for manual labor in the processing of financial documents and allow for the generation of data for cybersecurity applications beyond the capabilities of classical algorithms.

1.2 Structure of the Thesis

We continue in Chapter 2 by introducing the application context for the works presented in this thesis, where we provide an overview of the specific problems that arise in the fields of financial data analysis (Section 2.1) and cybersecurity (Section 2.2). We detail how they relate to representation learning and natural language processing and provide a brief overview of the related work relevant to the use cases.

In Chapter 3, we will discuss the technical background needed for the individual works presented in later chapters. We introduce the basic machine-learning tasks we will be dealing with, namely text classification (Section 3.1), text generation (Section 3.2) and binary optimization (Section 3.5). We will describe how each of them relates to the aforementioned applications and consequently introduce the machine learning model architectures we will employ. We will introduce text embedding models (Sections 3.1 to 3.1.7), generative models (Sections 3.2.1 to 3.2.4), and neural networks for binary optimization (Section 3.5). We provide additional details on training concepts in Sections 3.3 and 3.4.

In Chapters 4 to 9, we will present the individual works which make up the main content of this thesis, starting with a general method for interpretable word embeddings in Chapter 4, continuing with the analysis of financial report documents in Chapters 5 and 6 and consequently presenting the research on text generation for password data in Chapters 7 and 8. We conclude the main part of the thesis by presenting an application of binary optimization in financial report analysis in Chapter 9.

Finally, in Chapter 10 we will summarize the results of this thesis and give an outlook on future work, mainly investigating the effect of large language models on the use cases presented in the previous chapters.

CHAPTER 2

Applications in Finance and Cybersecurity

In this thesis, we consider solutions based on representation learning to various problems that arise in the analysis of financial documents during the auditing process and in the field of cybersecurity with relation to password data. This section gives an overview of both fields, the relevant workflows, the available data, and the areas that benefit from integrating machine learning components.

2.1 Auditing and Financial Data Analysis

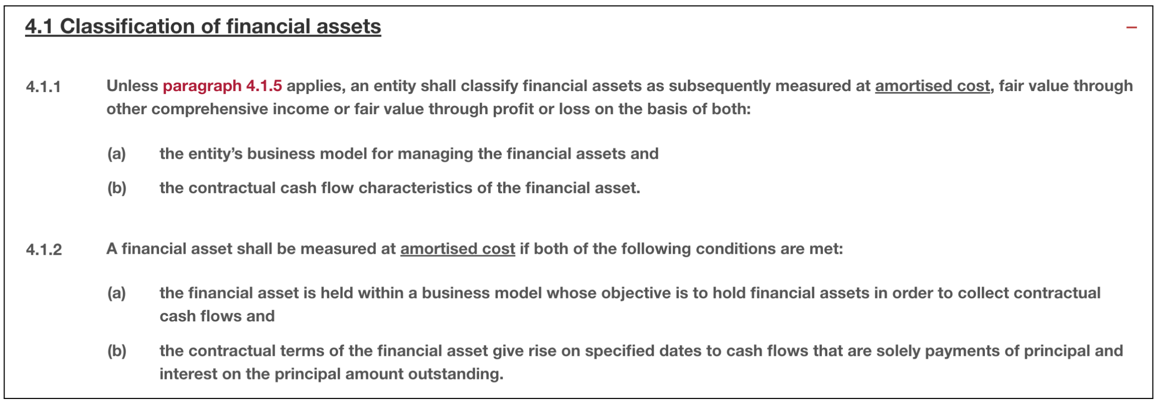
Auditing is the process of reviewing and examining an organization's financial records, operations, and activities to ensure that they are accurate, reliable, and comply with relevant laws and regulations [4]. The objective of auditing is to provide an independent and objective assessment of the organization's financial and operational performance and to identify any areas where improvements or corrective actions may be required.

The auditing process may include reviewing financial statements, conducting interviews with key personnel, performing tests of internal controls, and analyzing data and other information [4].

The process analyzed more closely in this work is the review of annual financial reports, i.e., text documents detailing a company's financial situation in a given year. These reports contain financial statements, providing an overview of the company's financial performance in terms of income statement, balance sheet, and cash flow statement. This is followed by notes to the financial statement, which provide context and additional information relevant to the stakeholders. Further, the reports contain a section on the management's discussion and analysis, in which an analysis of the company's financial results, operating performance, future prospects, and possible risks is given by the management,

Writing and publishing these reports in Germany is mandatory for public corporations above certain thresholds in terms of total assets, net income, or number of employees [5]. The reports must be audited by independent auditors [6].

Such an audit of a financial report involves legal frameworks known as accounting standards. These provide requirements that a financial report must follow regarding content and presentation. Examples of these types of accounting standards are the internationally recognized *International Financial Reporting Standards* (IFRS [7]) or *Generally Accepted Accounting Principles* (GAAP [10]), and *Handelsgesetzbuch* (HBG [11]) in Germany. Which accounting standard a company has to follow depends on the location of the company and the areas of operation. The applicable accounting standards can be interpreted as a checklist, in which each checklist item corresponds to a specific legal requirement.



4.1 Classification of financial assets

4.1.1 Unless paragraph 4.1.5 applies, an entity shall classify financial assets as subsequently measured at amortised cost, fair value through other comprehensive income or fair value through profit or loss on the basis of both:

(a) the entity's business model for managing the financial assets and

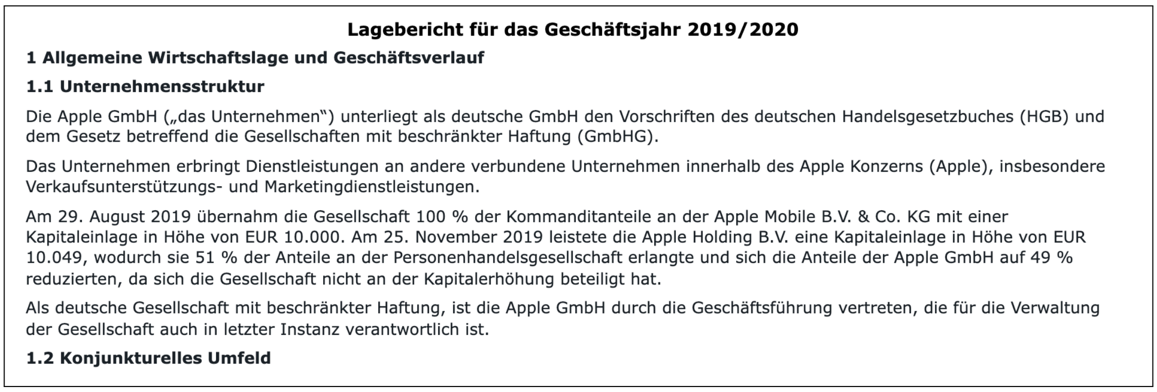
(b) the contractual cash flow characteristics of the financial asset.

4.1.2 A financial asset shall be measured at amortised cost if both of the following conditions are met:

(a) the financial asset is held within a business model whose objective is to hold financial assets in order to collect contractual cash flows and

(b) the contractual terms of the financial asset give rise on specified dates to cash flows that are solely payments of principal and interest on the principal amount outstanding.

(a) Excerpt from the auditing requirements under the IFRS standard. Screenshot from [7].



Lagebericht für das Geschäftsjahr 2019/2020

1 Allgemeine Wirtschaftslage und Geschäftsverlauf

1.1 Unternehmensstruktur

Die Apple GmbH („das Unternehmen") unterliegt als deutsche GmbH den Vorschriften des deutschen Handelsgesetzbuches (HGB) und dem Gesetz betreffend die Gesellschaften mit beschränkter Haftung (GmbHG).

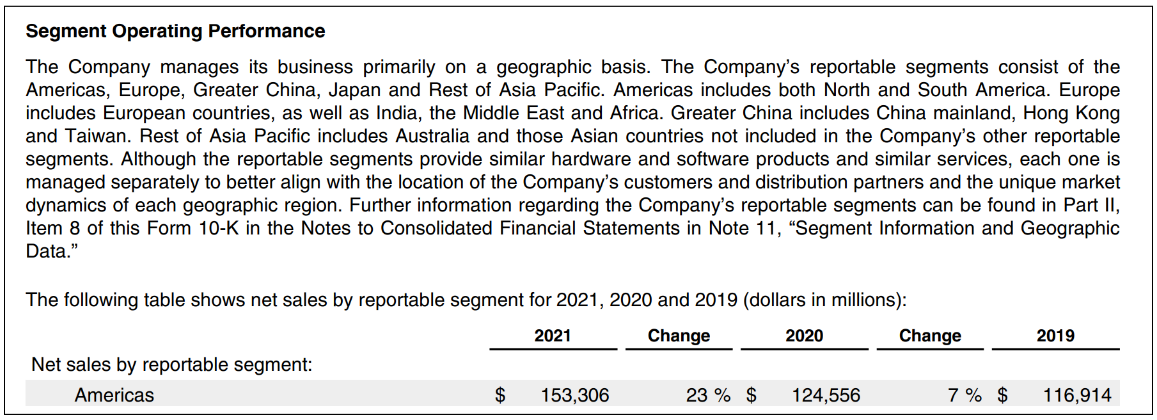
Das Unternehmen erbringt Dienstleistungen an andere verbundene Unternehmen innerhalb des Apple Konzerns (Apple), insbesondere Verkaufsunterstützungs- und Marketingdienstleistungen.

Am 29. August 2019 übernahm die Gesellschaft 100 % der Kommanditanteile an der Apple Mobile B.V. & Co. KG mit einer Kapitaleinlage in Höhe von EUR 10.000. Am 25. November 2019 leistete die Apple Holding B.V. eine Kapitaleinlage in Höhe von EUR 10.049, wodurch sie 51 % der Anteile an der Personenhandelsgesellschaft erlangte und sich die Anteile der Apple GmbH auf 49 % reduzierten, da sich die Gesellschaft nicht an der Kapitalerhöhung beteiligt hat.

Als deutsche Gesellschaft mit beschränkter Haftung, ist die Apple GmbH durch die Geschäftsführung vertreten, die für die Verwaltung der Gesellschaft auch in letzter Instanz verantwortlich ist.

1.2 Konjunkturelles Umfeld

(b) Excerpt from a German annual *Jahresabschluss* financial report audited by HGB standards. Screenshot from [8].



Segment Operating Performance

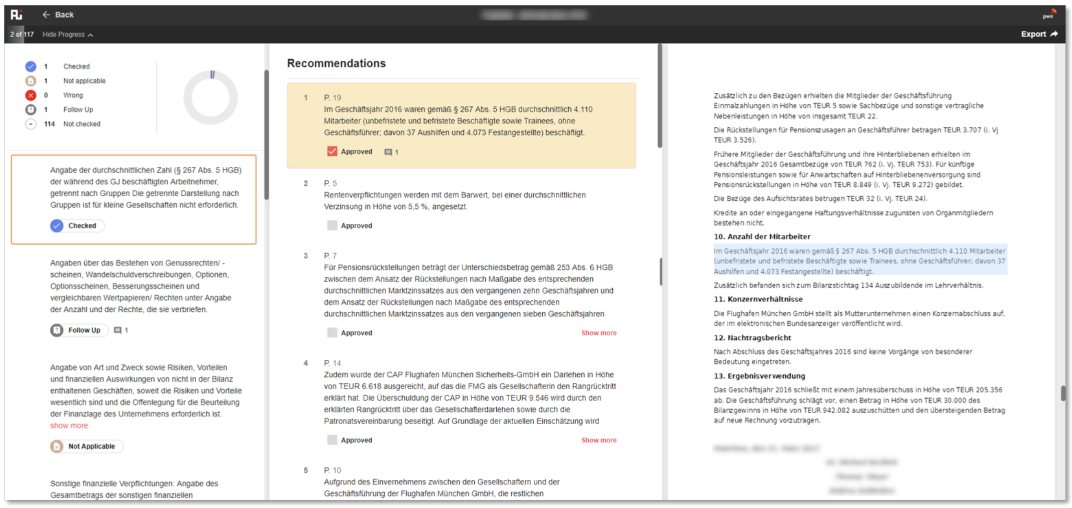
The Company manages its business primarily on a geographic basis. The Company's reportable segments consist of the Americas, Europe, Greater China, Japan and Rest of Asia Pacific. Americas includes both North and South America. Europe includes European countries, as well as India, the Middle East and Africa. Greater China includes China mainland, Hong Kong and Taiwan. Rest of Asia Pacific includes Australia and those Asian countries not included in the Company's other reportable segments. Although the reportable segments provide similar hardware and software products and similar services, each one is managed separately to better align with the location of the Company's customers and distribution partners and the unique market dynamics of each geographic region. Further information regarding the Company's reportable segments can be found in Part II, Item 8 of this Form 10-K in the Notes to Consolidated Financial Statements in Note 11, "Segment Information and Geographic Data."

The following table shows net sales by reportable segment for 2021, 2020 and 2019 (dollars in millions):

	2021	Change	2020	Change	2019
Net sales by reportable segment:					
Americas	$ 153,306	23 %	$ 124,556	7 %	$ 116,914

(c) Excerpt from an English annual *Form 10-K* financial report audited by IFRS standards. Screenshot from [9].

Figure 2.1: Examples for auditing requirements (IFRS standard) and the audited financial reports in English and German.


Back [redacted]

2 of 117 Hide Progress

Export

✓	1	Checked
☒	1	Not applicable
✗	0	Wrong
?	1	Follow Up
–	114	Not checked

Angabe der durchschnittlichen Zahl (§ 267 Abs. 5 HGB) der während des GJ beschäftigten Arbeitnehmer, getrennt nach Gruppen Die getrennte Darstellung nach Gruppen ist für kleine Gesellschaften nicht erforderlich.

Checked

Angaben über das Bestehen von Genussrechten/ -scheinen, Wandelschuldverschreibungen, Optionen, Optionsscheinen, Besserungsscheinen und vergleichbaren Wertpapieren/ Rechten unter Angabe der Anzahl und der Rechte, die sie verbriefen.

Follow Up 1

Angabe von Art und Zweck sowie Risiken, Vorteilen und finanziellen Auswirkungen von nicht in der Bilanz enthaltenen Geschäften, soweit die Risiken und Vorteile wesentlich sind und die Offenlegung für die Beurteilung der Finanzlage des Unternehmens erforderlich ist.
show more

Not Applicable

Sonstige finanzielle Verpflichtungen: Angabe des Gesamtbetrags der sonstigen finanziellen

Recommendations

1 P. 19
Im Geschäftsjahr 2016 waren gemäß § 267 Abs. 5 HGB durchschnittlich 4.110 Mitarbeiter (unbefristete und befristete Beschäftigte sowie Trainees, ohne Geschäftsführer; davon 37 Aushilfen und 4.073 Festangestellte) beschäftigt.
☑ Approved 1

2 P. 5
Rentenverpflichtungen werden mit dem Barwert, bei einer durchschnittlichen Verzinsung in Höhe von 5,5 %, angesetzt.
☐ Approved

3 P. 7
Für Pensionsrückstellungen beträgt der Unterschiedsbetrag gemäß 253 Abs. 6 HGB zwischen dem Ansatz der Rückstellungen nach Maßgabe des entsprechenden durchschnittlichen Marktzinssatzes aus den vergangenen zehn Geschäftsjahren und dem Ansatz der Rückstellungen nach Maßgabe des entsprechenden durchschnittlichen Marktzinssatzes aus den vergangenen sieben Geschäftsjahren
☐ Approved Show more

4 P. 14
Zudem wurde der CAP Flughafen München Sicherheits-GmbH ein Darlehen in Höhe von TEUR 6.618 ausgereicht, auf das die FMG als Gesellschafterin den Rangrücktritt erklärt hat. Die Überschuldung der CAP in Höhe von TEUR 9.546 wird durch den erklärten Rangrücktritt über das Gesellschafterdarlehen sowie durch die Patronatsvereinbarung beseitigt. Auf Grundlage der aktuellen Einschätzung wird
☐ Approved Show more

5 P. 10
Aufgrund des Einvernehmens zwischen den Gesellschaftern und der Geschäftsführung der Flughafen München GmbH, die restlichen

Zusätzlich zu den Bezügen erhielten die Mitglieder der Geschäftsführung Einmalzahlungen in Höhe von TEUR 5 sowie Sachbezüge und sonstige vertragliche Nebenleistungen in Höhe von insgesamt TEUR 22.

Die Rückstellungen für Pensionszusagen an Geschäftsführer betragen TEUR 3.707 (i. Vj TEUR 3.526).

Frühere Mitglieder der Geschäftsführung und ihre Hinterbliebenen erhielten im Geschäftsjahr 2016 Gesamtbezüge von TEUR 762 (i. Vj. TEUR 753). Für künftige Pensionsleistungen sowie für Anwartschaften auf Hinterbliebenenversorgung sind Pensionsrückstellungen in Höhe von TEUR 8.849 (i. Vj. TEUR 9.272) gebildet.

Die Bezüge des Aufsichtsrates betrugen TEUR 32 (i. Vj. TEUR 24).

Kredite an oder eingegangene Haftungsverhältnisse zugunsten von Organmitgliedern bestehen nicht.

10. Anzahl der Mitarbeiter

Im Geschäftsjahr 2016 waren gemäß § 267 Abs. 5 HGB durchschnittlich 4.110 Mitarbeiter (unbefristete und befristete Beschäftigte sowie Trainees, ohne Geschäftsführer; davon 37 Aushilfen und 4.073 Festangestellte) beschäftigt.

Zusätzlich befanden sich zum Bilanzstichtag 134 Auszubildende im Lehrverhältnis.

11. Konzernverhältnisse

Die Flughafen München GmbH stellt als Mutterunternehmen einen Konzernabschluss auf, der im elektronischen Bundesanzeiger veröffentlicht wird.

12. Nachtragsbericht

Nach Abschluss des Geschäftsjahres 2016 sind keine Vorgänge von besonderer Bedeutung eingetreten.

13. Ergebnisverwendung

Das Geschäftsjahr 2016 schließt mit einem Jahresüberschuss in Höhe von TEUR 205.356 ab. Die Geschäftsführung schlägt vor, einen Betrag in Höhe von TEUR 30.000 des Bilanzgewinns in Höhe von TEUR 942.082 auszuschütten und den übersteigenden Betrag auf neue Rechnung vorzutragen.

[redacted]

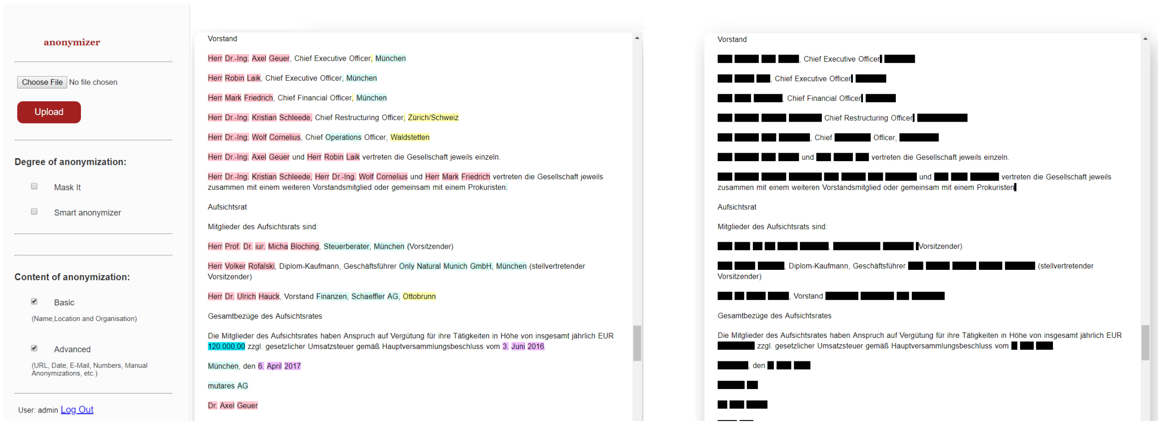
Figure 2.2: Screenshot of automated auditing software that illustrates the text matching workflow. The left column shows the auditing standards checklist, the middle column shows relevant passages from the financial report for the selected checklist item, and the right column shows the corresponding text passage from the financial report in the context of the document.

We present examples for requirements and financial reports in Figure 2.1. The top screenshot, Figure 2.1(a), presents an excerpt from the IFRS accounting standard, describing the requirements under which a company's financial assets must be measured at amortized cost instead of fair value. Figures 2.1(b) and 2.1(c) show excerpts from annual financial reports in German, audited by HGB standards, and in English, audited by IFRS standards. The German excerpt is the beginning of the report and describes the basic structure of the company, the English excerpt is taken from the middle of the report and details the operating performance by net sales in different geographic regions.

The main task in the audit of a financial report is to check whether the report complies with every relevant item in the accounting standard checklist. To this end, an auditor must review each checklist item, find the corresponding section in the financial report, and determine whether the legal requirements described by the checklist item are followed correctly and completed entirely. Note that a financial report is usually not structured in an order corresponding to the order of requirements in an account standards checklist, and report structure varies from company to company. Considering the usual length of annual reports of large companies, finding the appropriate section in these documents is a manual task that is both time-consuming and prone to human error [12, 13].

Large parts of this thesis will relate to automating this process in a way that improves an auditor's workflow and frees up time for tasks which require human understanding and expert knowledge. The main task we consider is matching requirements to financial report text paragraphs in a recommender setting. This approach has previously been explored using basic natural language processing methods [12]. The result of this has been auditing software with a machine-learning-based recommender system. In Figure 2.2, we show a screenshot of the developed software currently used by our industry partner. In Chapter 6 we will present an extension of the algorithms presented in [12], which allows for matching requirements and financial reports beyond fixed individual auditing standards.

A second area of interest related to the auditing process we consider in this thesis is data privacy concerns regarding the distribution of financial reports and the use of financial reports as training data for machine



Degree of anonymization:

- [ ] Mask It
- [ ] Smart anonymizer

Content of anonymization:

- [✓ Basic]
  (Name,Location and Organisation)
- [✓ Advanced]
  (URL, Date, E-Mail, Numbers, Manual Anonymizations, etc.)

User: admin

Vorstand

Herr Dr.-Ing. Axel Geuer, Chief Executive Officer, München

Herr Robin Laik, Chief Executive Officer, München

Herr Mark Friedrich, Chief Financial Officer, München

Herr Dr.-Ing. Kristian Schleede, Chief Restructuring Officer, Zürich/Schweiz

Herr Dr.-Ing. Wolf Cornelius, Chief Operations Officer, Waldstetten

Herr Dr.-Ing. Axel Geuer und Herr Robin Laik vertreten die Gesellschaft jeweils einzeln.

Herr Dr.-Ing. Kristian Schleede, Herr Dr.-Ing. Wolf Cornelius und Herr Mark Friedrich vertreten die Gesellschaft jeweils zusammen mit einem weiteren Vorstandsmitglied oder gemeinsam mit einem Prokuristen.

Aufsichtsrat

Mitglieder des Aufsichtsrats sind:

Herr Prof. Dr. iur. Micha Bloching, Steuerberater, München (Vorsitzender)

Herr Volker Rofalski, Diplom-Kaufmann, Geschäftsführer Only Natural Munich GmbH, München (stellvertretender Vorsitzender)

Herr Dr. Ulrich Hauck, Vorstand Finanzen, Schaeffler AG, Ottobrunn

Gesamtbezüge des Aufsichtsrates

Die Mitglieder des Aufsichtsrates haben Anspruch auf Vergütung für ihre Tätigkeiten in Höhe von insgesamt jährlich EUR 120.000,00 zzgl. gesetzlicher Umsatzsteuer gemäß Hauptversammlungsbeschluss vom 3. Juni 2016.

München, den 6. April 2017

mutares AG

Dr. Axel Geuer

Vorstand

[redacted], Chief Executive Officer [redacted]

[redacted], Chief Executive Officer [redacted]

[redacted], Chief Financial Officer [redacted]

[redacted] Chief Restructuring Officer [redacted]

[redacted], Chief [redacted] Officer, [redacted]

[redacted] und [redacted] vertreten die Gesellschaft jeweils einzeln.

[redacted] und [redacted] vertreten die Gesellschaft jeweils zusammen mit einem weiteren Vorstandsmitglied oder gemeinsam mit einem Prokuristen[redacted]

Aufsichtsrat

Mitglieder des Aufsichtsrats sind:

[redacted], [redacted]Vorsitzender)

[redacted], Diplom-Kaufmann, Geschäftsführer [redacted] (stellvertretender Vorsitzender)

[redacted], Vorstand [redacted]

Gesamtbezüge des Aufsichtsrates

Die Mitglieder des Aufsichtsrates haben Anspruch auf Vergütung für ihre Tätigkeiten in Höhe von insgesamt jährlich EUR [redacted] zzgl. gesetzlicher Umsatzsteuer gemäß Hauptversammlungsbeschluss vom [redacted].

[redacted], den [redacted]

[redacted]

Figure 2.3: Screenshots of the tool for automatic anonymization of financial report documents. Left: control panel and document view with found named entities marked in colors for their respective classes. Right: Result of anonymization step in which all found sensitive entities are removed, visualized by blacking out the text. Image originally published in [14].

learning systems. Due to the sensitive information in financial reports, it is impossible to distribute reports to third parties before the report's full release to the public. This includes, for example, using these reports as training data for machine learning systems and access to reports by developers of such systems. One way to alleviate this issue is anonymization of the reports before distribution. After removing sensitive entities such as names and locations from the document, the inference of personal information is no longer possible, and one remains with a document that is safe to distribute but still features the same language and structure as the original. This anonymized document is then suitable for both the training of models and analysis by developers [14].

The anonymization of text documents is largely a manual process, in which human annotators select which text to remove or obscure from the document and which text is safe to keep. This manual process is very time-consuming and prone to error. Missing a single instance of a name in a document can jeopardize the entire anonymization process, even if this particular name is correctly removed from the remaining text. To this end, automated systems can assist human annotators in their work, by automatically reading and analyzing a given text document and either presenting the user with a de-identified version of the document or providing a recommendation system, in which the likelihood of a term being sensitive is presented to the user. Rule-based post-processing methods can ensure that names and terms that are established as sensitive are removed or obscured in the entire document [14].

In Chapter 5, we will present a method of automated de-identification of financial reports in the German language, which presents a significant step towards legally valid anonymization of such financial reports. We show how a combination of deep-learning-based sequence classification and rule-based post-processing results in a near-perfect de-identification performance, which provides a major improvement in the workflow of users. See Figure 2.3 for a screenshot of the software developed for automatic anonymization of financial reports in use.

Lastly, we will consider an application in automated auditing beyond analyzing the text of financial report documents. Financial reports do not only contain text but also numerical data in the form of tables. These tables present the financial information in a structured way, presenting the key performance indicators, such as revenue, profit and loss, and various other monetary values related to the company's financial performance.

During the auditing process, auditors need to ensure the correctness of the presented values, which applies to their representation of the correct financial situation of the company, their consistency with the actual text of the document, and their internal consistency [15].

Numerical values presented in tables are not unrelated to each other, as tables often contain aggregated values that are derived from other values in the table. For example, a company's total revenue is often the sum of the revenue of all its subsidiaries. This means that certain numerical values in the table present a sum of other values in the table. Checking the correctness of the sum is trivial. However, deciding which values are supposed to be sums of other values is more complex. Financial documents are most likely provided in .pdf or .docx format, which does not offer machine-readable information on the underlying structure of tables. Therefore, checking the internal consistency of tables in these documents remains a largely manual task with a large room for human error [15].

In Chapter 9, we will present a novel approach to this type of consistency check, encoding the internal consistency of the table as a binary optimization problem and applying a neural-network-based solving algorithm to the problem. For an analysis of machine learning methods for table-to-text consistency checks, see [16].

2.2 Text Generation for Cybersecurity – Password Recovery

The second main domain of application discussed in this thesis is the use of natural language processing algorithms for text generation in cybersecurity. In particular, we focus on password generation or password recovery, which is the process of attempting to reconstruct a user's password by guided brute-force attacks.

Consider a locked device or account protected by a password. In password recovery, we aim to generate a password string that unlocks the respective account or device without knowing the actual password beforehand. This password is stored internally as a hash value, and checking whether an input password is correct is done by comparing the hashed input to the stored password hash [17, 18].

Hash functions are mathematical functions that take an input of arbitrary length and return a fixed-size string of characters or bytes. These functions have several defining properties. The most important to note here is the non-reversibility, meaning it is computationally infeasible to determine the original input from the hash output [19]. Therefore, password recovery is equivalent to reconstructing the input for a given hash value and known hash function. Since hash functions are non-reversible, the only possibility for reconstruction is a brute-force approach, in which we hash a large number of possible password candidates and compare the hash values to the target hash. See Figure 2.4 for an illustration of this approach, in which we generate multiple password candidates, but only the exact matching string generates the target hash. Even small changes in the input string result in entirely different hash values.

Considering the exponentially large number of possible passwords (there are $1.4e14$ possible combinations for passwords of length 10 of only lower-case characters), a pure brute-force attack is infeasible on any computational hardware. Therefore password recovery algorithms attempt to produce a large number of possible password candidates which are both novel and realistic. By novel, we mean that the outputs are not already part of a known set of passwords since these can be hashed and compared without needing a text-generation algorithm. By realistic, we mean that they resemble passwords that a human user might use for authentication.

Note that in practice, the feasibility of this approach also depends on the specific hash function since hash functions that are slow to compute limit the number of password guesses one can attempt in a certain time [20]. Additionally, note that randomly generated passwords are safe from this type of attack. A ran-

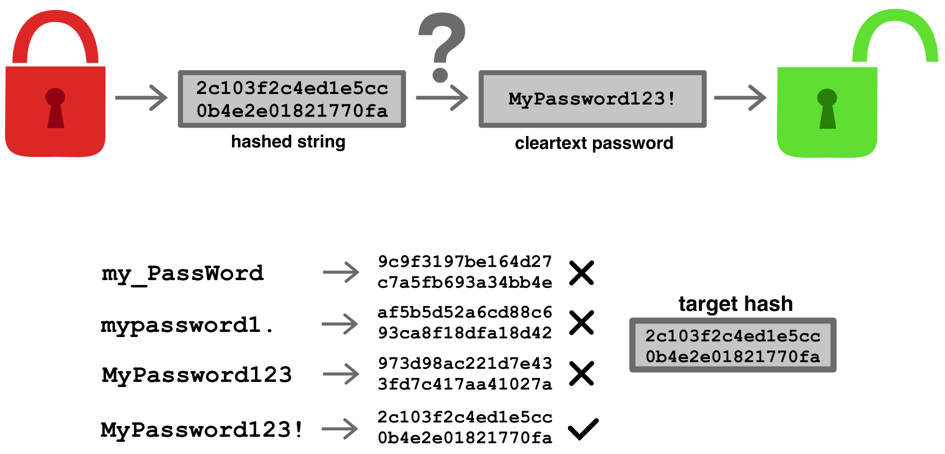

2c103f2c4ed1e5cc 0b4e2e01821770fa
hashed string
?
MyPassword123!
cleartext password
my_PassWord
9c9f3197be164d27 c7a5fb693a34bb4e
mypassword1.
af5b5d52a6cd88c6 93ca8f18dfa18d42
MyPassword123
973d98ac221d7e43 3fd7c417aa41027a
MyPassword123!
2c103f2c4ed1e5cc 0b4e2e01821770fa
target hash
2c103f2c4ed1e5cc 0b4e2e01821770fa

Figure 2.4: Illustration of the password recovery process. Given a locked device or account with a corresponding password hash, one aims to recover the original string that produces the hash. Since hash functions are non-reversible, the only way to restore the original password is by generating and hashing multiple password candidates.

domly generated password contains no patterns that a password generation algorithm might exploit, so any algorithm producing passwords is at most as efficient as pure brute-force over the entire password space, less efficient if the algorithm produces any duplicated outputs.

Password recovery is an active field of research in the domain of cybersecurity. For example, password generation algorithms are applied during penetration testing, in which the security of computer systems and networks is tested using simulated cyberattacks by authorized security auditors [21].

There are various password-generation algorithms and tools available. Before providing details on our contributions in Chapters 7 and 8, we introduce a selection of notable algorithms and tools in more detail and describe how they differ from a purely data-driven deep-learning text generation approach.

The first tool we consider is *hashcat* [22], which offers GPU-accelerated hashing of password strings and can expand an input list of known passwords, called a dictionary, by predefined rules. A rule defines a specific string manipulation, such as lower-casing of the input, reversing the string, or appending characters and numbers. Rules can be chained for more extensive string manipulations. There are several lists of rules publicly available, from small hand-written lists [23] to lists of several thousand rules [24]. These large lists of rules usually are the result of explicit statistical analysis of large password sets or randomly generated and then selected by evaluated password recovery performance [25].

Rule-based extensions of known password lists prove powerful and expressive in practice and require minimal computational power. However, the rules are always predefined and limited in scope, meaning the algorithm might not be able to produce every possible output and miss parts of the output space, or the algorithm might be able to generate each possible password string but produce a large number of unrealistic passwords in the process.

The second algorithm we consider is *PCFG* [18], an application of the *probabilistic context-free grammar* [26] probabilistic modeling theory for password recovery. A probabilistic context-free grammar is, informally, a type of grammar in natural language processing that assigns probabilities to a set of rules that defines how sentences can be formed. The probability of using a certain rule to construct a sentence might be inferred from how often sentences in a training dataset adhere to the rule. In the context of passwords, PCFG generates a password structure (e.g., `ULLLLD` for an upper-case letter followed by four lower-case

123456	12345
123456789	password
iloveyou	princess
1234567	rockyou
12345678	abc123
nicole	daniel
lovely	jessica
654321	michael
ashley	qwerty
111111	iloveu
000000	michelle
tigger	sunshine
chocolate	password1
soccer	anthony

(a) Most common passwords in the *rockyou* dataset.

gasukidesu	tp6e6npzsa
2183june	cindybryant
princesa8294	Ricky6
rbnc1125	sweden21
usheraymond	tommy21+
25146313	rachel87
551AME	obrazeczek
GALLO	valc22
eemmiinneeiizz	ng_honay
chirita	toothpaste^*
1poohstyle	ilovemadiha
jessie.25	81806180
matthew012992	hot819
nostalgiex3	joe90

(b) Random passwords in the *rockyou* dataset.

Figure 2.5: Example passwords from the *rockyou* password dataset. The dataset stems from a leak of cleartext passwords from a data breach on RockYou, a social application site. The list of passwords is available from various sources [27, 28] and has become a default benchmark for password recovery algorithms [17, 29].

letters and a digit) and then fills this structure by sampling substrings based on the frequency of their occurrence in a training dataset. In practice, this algorithm provides a good trade-off between expressiveness and computational complexity, resulting in more diverse output passwords than the purely rule-based approach of hashcat, while still being able to produce a large number of passwords in a reasonable amount of time.

Note that all password-generation algorithms can be used in combination with hashcat, which takes the output passwords of the algorithm as its input dictionary and applies its rule-based string manipulation to produce an even larger number of password candidates [17].

For the development and evaluation of password recovery algorithms, there is a need for datasets of passwords, i.e., lists of strings that were used by real users for password authentication. There are various datasets of such passwords publicly available, the most commonly used of which is the *rockyou* list [27], which is a list of cleartext passwords acquired from a data breach on the social application website RockYou [30]. In Figure 2.5, we show some sample passwords from the dataset. Further well known password dataset include *linkedin* [31, 32], *yahoo* [33], and *youku* [34].

All of these datasets can be used to evaluate password data in the following manner. We split the dataset into a training, a validation, and a test split, making sure that there is no data overlap between the splits. We use the training set for the initialization of the password generation algorithm. For example, the training set can provide a word list for the application of hashcat rules, can be statistically analyzed by the PCFG algorithm, or, as we will see in later chapters, can be used as training data for deep learning text generation algorithms. We use a trained algorithm to generate a fixed number of password candidates, e.g., all evaluations in [35] and [36] consider 1.0e9 generated passwords, and count the number of passwords generated that are also found in the validation and test sets. A generated password found in the validation or test set is both novel, since it was not part of the training set and the algorithm has therefore not seen the password before, and realistic, since it is part of another password dataset and therefore has been used by human users as an actual password. We can then measure the quality of generated passwords as the total number of found

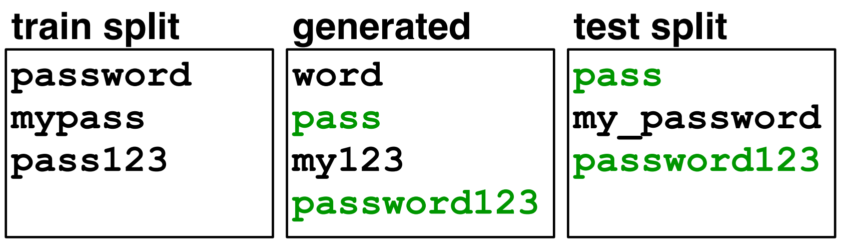


Figure 2.6: Illustration of the training and evaluation process of a password recovery algorithm. We use the train split to train the algorithm or model. The model generates new data based on the information present in the train split. The test split contains only data not found in the train split. We count the number of generated passwords that are also present in the test split. In this case, we recover two of three passwords in the test split for a recovery rate of 66%.

passwords for a given number of generated strings or as a percentage of the total validation and test set. In Figure 2.6, we illustrate this process with a toy example. In this case, we generate four new passwords based on a training split, two of which are found in the test split for a recovery rate of 66% of the test split.

There are several further password-generation algorithms, such as Markov chains [37, 38], recurrent neural networks [39] (see also Section 3.1.5), advanced dictionary attacks [40], and many more [41–43]. In recent years, with the advent of deep learning text generation models, the investigation of such models for password generation has commenced.

One of the first attempts to use deep learning generative models for password generation was *PassGAN* [29], which uses a generative adversarial network (GAN [44], see also Section 3.2.1) to generate passwords. This work received attention from both the research community and the general public [45, 46]. While being an effective approach, the PassGAN algorithm proved less expressive than competing algorithms like PCFG. This is partly due to the inherent nature of GANs, which provide impressive performance when generating images but struggle with discrete data like text [47]. In Chapters 7 and 8, we present a significant enhancement of deep generative models for password recovery beyond PassGAN by applying various architectures that lend themselves better to the generation of text data. We will describe the implementation of a variational autoencoder (VAE [48], Section 3.2.2), a generative transformer language model (GPT2 [49], Section 3.2.4) and a combined architecture, meaning a variational autoencoder with transformer-based encoder and decoder blocks. All of these models present major improvements over PassGAN, with the number of recovered passwords for the same number of generated strings approximately tripled across various test datasets.

CHAPTER 3

Representation Modeling

In this chapter, we provide a high-level overview of the machine learning techniques we apply in the development of solutions for the application domains described in the last chapter. We will describe the tasks, the optimization objectives, the model architectures, and the training techniques in a general manner. Most of this chapter will focus on natural language processing, as most applications presented in the later chapters will deal with unstructured textual data. Section 3.1 will describe text classification models and introduce text embedding architectures. Section 3.2 will introduce the task of text generation and generative models specifically for text. In Sections 3.3 and 3.4, we provide further information on the training of the models introduced previously, namely the tokenization of text and the idea of transfer learning. In addition to this, we present a special case of binary optimization for numerical data and a solving algorithm using recurrent neural networks in Section 3.5. In the later chapters, we provide details of the individual implementations for the specific use cases.

3.1 Embedding and Classification

One of the most basic tasks in natural language processing is text classification, in which a text, i.e., any sequence of characters like a document, a sentence, or a single word, is read by a classification model and classified as belonging to one or more predefined classes. We call a classification task on two classes binary, a task on more than two classes multi-classes and a task in which a text can belong to more than one class multi-label.

Any classification task will follow a similar architectural pattern. The text, provided as a sequence of characters, will be processed by an embedding model, which provides a single N-dimensional vector representation for the entire text, independent of the number of characters. As we will show in this thesis, most progress in natural language processing has come from developments in the embedding of text data [50].

The vector representation is mapped onto the set of possible labels by a classification layer, often composed of linear layers or similar architectures, less complex than the embedding model. The output vector must be transformed into a vector of class probabilities by an activation function, where the choice of activation function depends on the task at hand.

For binary classification, in which the proposed text can either belong or not belong to a single class, we apply a sigmoid activation to the single numerical output value of the classification layer to map the output to the interval $[0, 1]$ and interpret the result as the probability of the input being a positive example of class. Given a single-label task in which only one of the C classes can be correct, we need to map the C-

dimensional numerical vector into a probability distribution over all C classes. We do this using a softmax function which forces all values of the vector to be in the interval $[0, 1]$ and the vector entries to sum up to 1. In a multi-label task, where more than one of the C labels can be correct, the task breaks down into C individual binary classification tasks. Therefore, we apply a sigmoid activation function to each element of the C-dimensional output vector of the classification layer [51].

3.1.1 Topic Modeling

Topic modeling is a special case of text classification in which the classes are not predefined but learned from the data. The goal of topic modeling is to find a set of classes, i.e., topics, which describe and cluster the data appropriately. Often the number of topics K to be learned from the data is predefined, and documents are assumed to be a mixture of these topics [52].

For example, take a corpus of news articles. We can assume that each article is a mixture of topics such as politics, sports, finance, and others. The goal of topic modeling is to find these topics and assign each article a probability distribution over these topics, such that we can interpret an article about the current state of the stock market as a mixture of the topics finance and politics. Since the topics are not predefined but learned from the data, we can not directly assign labels such as “finance” or “politics” to the topics, but have to interpret the topics by looking at the documents clustered into the respective topics and infer a topic name from the distinct texts found in these documents [53].

Standard algorithms for topic modeling include Latent Dirichlet Allocation (LDA [52]), Latent Semantic Analysis (LSA [54]), and Non-Negative Matrix Factorization (NMF [55]). In Chapter 4, we will introduce a novel topic modeling algorithm based on matrix factorization for interpretable word embedding and topic modeling.

3.1.2 Text Matching

Moving from text classification with a predefined set of classes or a set of learned topics, we now consider text matching, in which we want to find a matching text for a given input document or text. More generally, we want to rate the similarity of two texts and their relevance to each other [56].

Technically, for two texts $x \in \mathcal{X}, y \in \mathcal{Y}$ we want to find a distance function $d : \mathcal{X} \times \mathcal{Y} \to [0, 1]$, such that semantically meaningful texts are mapped to values close to 1, and semantically unrelated texts are mapped to values close to 0. Once this distance function is found, we can find a matching text $y' \in \mathcal{Y}$ for a given input text $x \in \mathcal{X}$ by

$$y' = \arg\max_{y \in \mathcal{Y}} d(x, y). \tag{3.1}$$

The distance function can be defined independently of the data or learned from the data in an unsupervised or supervised manner. We will consider unsupervised and supervised learning approaches to text matching in Chapter 6.

3.1.3 Sequence Classification and Named Entity Recognition

So far, we have considered text classification tasks in which we want to classify entire documents or text sequences. A more granular task is given by sequence classification. Given a text comprised of multiple words, we want to classify each word into one or multiple predefined classes [50].

A special case of sequence classification is named entity recognition, in which we want to determine which words or sequences are named entities. These named entities describe key information present in the text.

Named entities can be names of persons, organizations, and locations, as well as any other category of information to be extracted from the input document, such as financial key performance indicators, pathologies in medical text, and many others [57].

To this end, we classify a substring of the input text as belonging to one of the predefined categories of named entities or no named entity. What compromises a substring depends on the problem definition and the model architecture. One can, for example, classify each character in a sequence, each word or span compromised of multiple words.

A type of named entity recognition is de-identification. The process of de-identification contains finding named entities that allow for the identification of persons, locations, organizations, and others and removing or replacing them in the text such that the resulting text does not enable the identification of these key figures. The task of anonymization is similar but different since it does not only contain the de-identification of the document but also the removal of any sensitive information that might make conclusions about the entities in the document possible. Such information might be dates, numbers, document metadata, and many types of contextual information present in the document [14].

De-identification can be tackled as a named entity recognition problem if we define the named entity categories as the types of identifying information present in the document (such as names, locations, and organizations), classify each word as belonging to one of these categories or not and redact each word classified as sensitive information from the text. This redaction can be done by replacing the word with generic tokens (a general `REDACTED` token or category-specific tokens like `PERSON`, `ORGANIZATION`, ...) or pseudonymization, in which the redaction algorithm keeps track of the entities to be redacted and ensures that the same entity is replaced with the same identifier in the whole document (e.g. `PERSON_1`, `PERSON_2`) [57].

In Chapter 5, we will present an algorithmic approach to de-identification and anonymization, in which we classify words in financial reports as belonging to a set of classes of named entities using recurrent neural networks and apply rule-based post-processing methods to ensure the reliability of the prediction and the removal of various additional types of sensitive information.

3.1.4 Word Embeddings

In order to apply any classification algorithm to text data, we first need to embed the text into a latent space. Word embeddings are a basic but powerful method of embedding text. To demonstrate the mechanism of training word embeddings, we introduce the popular word embedding model GloVe [58].

GloVe is a matrix factorization-based model, which means that the training process is equivalent to factorizing a matrix of word cooccurrences in the text. Word embeddings are trained on large collections of text documents. We split the text into a sequence of words and generate a cooccurrence matrix by counting how often each word appears in a context window of fixed length (the original GloVe implementation uses a context window of 10 words in each direction) of each other word. We then factorize this matrix of cooccurrences $\boldsymbol{X} \in \mathbb{R}^N$ into matrices $\boldsymbol{W}, \tilde{\boldsymbol{W}} \in \mathbb{R}^{d \times N}$, such that

$$\boldsymbol{W}_i^T \tilde{\boldsymbol{W}}_k + +\boldsymbol{b}_i \tilde{\boldsymbol{b}}_k = \log(\boldsymbol{X}_{ik} + 1), \tag{3.2}$$

where N is the vocabulary size, d is the embedding dimension, and $\boldsymbol{b}, \tilde{\boldsymbol{b}} \in \mathbb{R}^d$ are bias vectors. The rows of $\boldsymbol{W} + \tilde{\boldsymbol{W}}$ then are the word embeddings for the respective vocabulary entry. For more details on the motivation and the derivation of this training procedure, see [58].

Having extracted the cooccurrence information and calculated the embedding vectors, using word embed-

Text	`The (black) cat's name is Mittens.`
Lower-Casing	`the (black) cat's name is mittens.`
Remove Punctuation	`the black cats name is mittens`
Remove Stop-Words	`black cats name mittens`
Stemming	`black cat name mitten`

Figure 3.1: Preprocessing steps for an example text before whitespace tokenization. We apply lower-casing, remove punctuation, remove stop words, and apply stemming. The resulting text is suitable for application in word embedding models.

dings is easy and computationally inexpensive. We only have to split the target text into individual words and find the pre-calculated vector corresponding to the respective vocabulary entry.

One issue of word embeddings is their lack of interpretability. The original GloVe implementation generates word embeddings of size $d = 300$. The individual entries in the word embeddings do not correspond to any specific semantic meaning. Suppose a system trained on a downstream task corresponds more to certain dimensions in the input embedding than to others. In that case, we can not extract any useful information about the system from this observation. To this end, in Chapter 4, we introduce a method for calculating interpretable word embeddings, where we factorize the logarithmic cooccurrence matrix into three matrices and interpret the individual dimensions of the word embeddings as topics that occur in the text corpus.

For training and retrieval of word embedding models, we need to tokenize the text data first, so that we split the text into individual words which we can map to the vocabulary. The simplest way of tokenization into words is splitting on whitespace characters, however, the direct application of this approach runs into some issues. Consider, for example, the following sentences:

```
I love my cat.
Cats are my favorite animal.
```

A simple splitting on whitespace will create the tokens `Cats` and `cat.`. While both refer to the same animal, they will have to be treated differently by a word-based model.

To avoid this, we apply preprocessing steps [50]. These steps include lower-casing, removal of punctuation and numbers, removal of stop words (i.e., common words such as `and` or `the`), and stemming (i.e., removing inflectional forms from words). See Figure 3.1 for an example of preprocessing on a simple text.

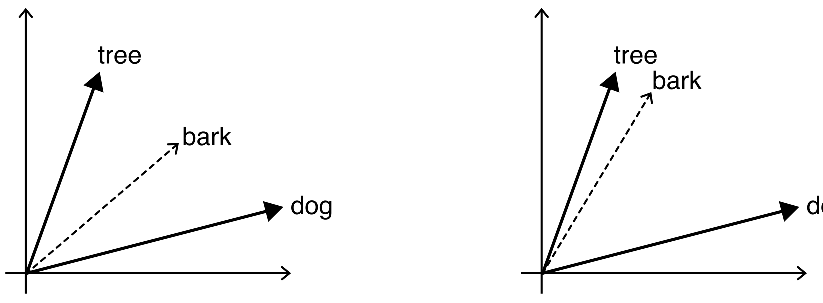
This tokenization requires a large vocabulary of words in the respective language. For example, word2vec contains a vocabulary of 662 109 individual words [59]. However, even with this large vocabulary, not all words in the target language can be encoded. Words that, even after preprocessing, are not part of the vocabulary, like names or compound words, will be tokenized with a special token for unknown words (such as `<UNK>`).

All preprocessing steps reduce noise in the text data but make the text more prone to ambiguity by homonyms, i.e., words that are spelled the same but have different and unrelated meanings. For example:

```
The dog would always bark at people.
Tree bark is the protective outer layer of a tree trunk.
```

The word `bark` has two entirely unrelated meanings, one as a verb and one as a noun.

Homonyms, in general, pose a problem for word embedding-based machine learning models. If we pre-computed and embed for the word `bark`, we can not know which meaning of the words is represented by the



(a) Word embedding of `bark` has the same distance to `dog` as to `tree`. The embedding does not capture either meaning of the word.

(b) Word embedding of `bark` is closer to `tree` than to `dog`. The embedding captures the meaning of tree bark but does not relate to dogs.

Figure 3.2: Word embeddings exhibit limitations when dealing with homonyms, i.e., words that are spelled the same but have multiple meanings. In this example, there is no optimal way to encode the word `bark` since we can not feasibly encode both meanings at the same time.

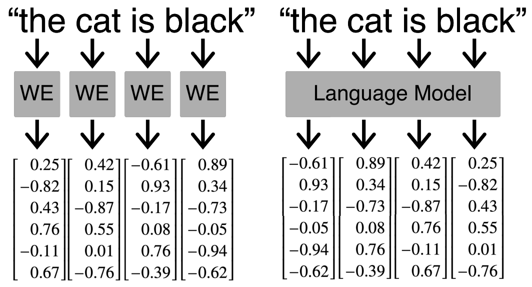


Figure 3.3: Left: Word embeddings produce a fixed output for each word regardless of the context in the sentence. The computation of each vector is independent. Right: Contextual embeddings calculate embeddings based on the entire sequence. A word will have different embeddings depending on the context, i.e., the other words in the sequence.

embedding. We demonstrate this issue in Figure 3.2, in which we show how the word embedding for `bark` either favors one of the word embeddings for `tree` or `dog` or resides in the middle of both. The embedding will either only contain information about one meaning of the word and not make sense in the context of the other meaning, or the embedding will contain information about multiple meanings and therefore make less sense in the context of each meaning.

One possible solution to this problem is to use different embeddings for each meaning of a word, e.g. `bark_verb` and `bark_noun`, which requires a larger vocabulary and a method to distinguish between different meanings of a word in the text. Another solution to this problem is to use embeddings that contain information about the context of a word in addition to the word itself. This contextual embedding approach will be discussed later, starting with Section 3.1.5 and we demonstrate its effectiveness in applications starting at Chapter 5.

3.1.5 Recurrent Neural Networks

In a text document, words or sentences are usually not independent of each other. The order of characters, words, and sentences is important for the meaning of the text. Taking into account the context of a word when embedding solves the issue of homonyms with static word embeddings described in Section 3.1.4. If we have a way of adding information about the context of the remaining sentence when embedding a word, we can safely embed the word `bark` in both sentences displayed above. Both embeddings for the word will be different and capture the respective meaning of the term in the sentence. We call this approach to contextual embedding by analyzing the relationships between words in sequences *language modeling* [60]. Consider Figure 3.3, in which we demonstrate the difference between both approaches. We encode the same sentence two times, once with a static word embedding in which each word is embedded separately into a pre-computed embedding vector and once using a language model which calculates the embeddings at inference time from the entire sequence.

Recurrent neural networks (RNNs) are a class of neural networks that are able to model such dependencies. In its most basic form, the network has an internal hidden state and takes as input an input token (for example, a character in a sequence as a one-hot encoded vector) and calculates an output and a new hidden state from the input and the previous hidden state. The input adds new information to the calculation, while the hidden state retains previous information from the sequence. We initialize the RNN with a random hidden state.

Regular RNNs often suffer from a problem called vanishing gradients [61, 62], in which the model fails to learn long dependencies between input tokens in the sequence since the gradients given by new information become increasingly small at each time step. A version of the RNN which addresses this issue is the Long Short-Term Memory (LSTM) network [61], which enables learning of long dependencies by implementing so-called forget gates, which allows information to flow freely between time steps.

For embedding of text, one can apply an RNN as a language model on text tokenized by characters or sub-word tokens (see Section 3.3) and use a one-hot encoded vector of a token as input. The RNN calculates an output from the internal hidden state, and the input and transforms the output to a probability distribution over the token vocabulary to predict the next token in the sequence. We calculate a loss from the predicted token and the actual next token and repeat the process for the next time step with the next token in the sequence and the updated hidden state [63].

After training such a model on a corpus of text data, for which we do not need annotations since the model is trained only on the prediction of the next token in the sequence, we can apply the model for token embedding by reading a new text with the model and interpreting the hidden state of the RNN as an embedding of the current token. This embedding contains contextual information on the sequence so far. To incorporate left and right context for one token, i.e., information from the entire sequence, we can pass the text through the RNN twice, once in each direction. This way, we receive two hidden state vectors, one of which contains information on the sequence left of the token, one which contains information on the sequence right of the token. We concatenate both vectors to obtain a single token embedding. We call this type of language model *bidirectional* [63].

In Chapter 5, we will apply a bidirectional LSTM language model, trained on both general language data and domain-specific data, as an embedding model for a sequence classification task, namely named entity recognition for de-identification of financial report data.

3.1.6 Transformers

In recent years, transformer architectures have emerged as state-of-the-art in many NLP tasks. They are based on the attention mechanism, which allows the model to focus on certain parts of the input. This allows the model to learn long-range dependencies in the input, which is important for tasks such as machine translation or text generation [64].

Transformers do not have a recurrent structure but instead, use a self-attention mechanism to model dependencies between words in the input. The main functionality of the self-attention mechanism depends on three weight matrices. originally named queries $\boldsymbol{Q}$, keys $\boldsymbol{K}$ and values $\boldsymbol{V}$. We denote by N the vocabulary size, d_E the input embedding size, d_k the dimension of the keys and queries, and d_v the dimension of the value vectors. The learnable weight matrices for the queries, keys, and values are denoted by $\boldsymbol{W}^q, \boldsymbol{W}^k \in \mathbb{R}^{d_E \times d_k}$ and $\boldsymbol{W}^v \in \mathbb{R}^{d_E \times d_v}$, respectively.

For an input string of T tokens, we embed each token using the input embedding for a matrix $X \in \mathbb{R}^{T \times d_E}$. We multiply the embedding matrix with the query, key, and value weights to obtain the queries, keys, and values for the respective input sentence:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^q \in \mathbb{R}^{T \times d_k}, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^k \in \mathbb{R}^{T \times d_k}, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^v \in \mathbb{R}^{T \times d_v}. \tag{3.3}$$

Given these matrices, we calculate the attention vector for each token as a weighted sum of the values vectors of all tokens in the sequence:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \operatorname{soft\,max}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \in \mathbb{R}^{T \times d_v}. \tag{3.4}$$

In a transformer block, we apply self-attention to the input embeddings and map the obtained values back onto the input or output dimension d_E by a linear layer in a feed-forward network. This way, we can use the output of one transformer block as input for another transformer block and concatenate multiple blocks. For more details on the transformer, the self-attention mechanism, attention between encoder and decoder, masked attention in the decoder and positional embeddings, we refer to the original transformer paper [64].

We will apply different transformer-based models in the course of this thesis, namely a BERT [65] architecture for embedding and text matching in Chapters 5 and 6 and a GPT2 [49] architecture for text generation in Chapters 7 and 8. Before describing these applications in the coming chapters, we will first introduce both architectures and provide details on their use of the self-attention mechanism in the following subsections.

3.1.7 BERT

The BERT (*Bidirectional Encoder Representations from Transformers* [65]) model is a transformer-based language representation model. Unlike other transformer-based models like the original transformer [64] or GPT [49] (see also Section 3.2.4), BERT is a pure encoding architecture, which reads an input sequence and outputs embeddings for each token in the sequence. By adding a single additional output layer, these embeddings can be used for various tasks, such as sequence tagging or text classification.

Internally, BERT is a multilayer bidirectional transformer similar to the original transformer implementation [64]. The model is trained in an unsupervised manner using masked language modeling, in which some individual tokens in a sequence are masked (either by a specific `MASK` token or a random token) before input and the embeddings of the masked tokens are used to predict the original token. The model is called bidirectional since it applies attention over the entire sequence, meaning the left and right context for the prediction of each token. Additionally, the model is trained unsupervised using next sentence prediction, in which two sequences from the dataset are passed to the model as input, and the model has to decide whether these two sequences are random pairs or consecutive sentences appearing in the dataset.

For supervised training, any dataset with labels for either individual tokens or entire sequences can be used. Combining both supervised and unsupervised training, BERT achieves state-of-the-art results for many NLP tasks, such as question answering on various benchmark datasets [65].

In Chapter 5, we apply a BERT model for embedding tokens in financial report texts and classify the tokens using linear layers or sequential models for a named entity recognition task. We compare the BERT and LSTM architectures for this task and find that both approaches achieve near-perfect anonymization (99% recall) when combined with rule-based post-processing. In Chapter 6, we apply the BERT model for embedding sequences and text matching in financial report data, where we calculate the distance in latent space of the embedding of different texts to rank the most relevant texts in financial reports. We investigate the effect of multiple unsupervised and supervised training approaches and present a novel training paradigm based on multiple training steps on a series of datasets for a both powerful and flexible text matching model.

3.2 Generative Models

The previous section dealt with embedding and classification, in which we are given input data that we aim to classify or compare. In contrast to this, in this chapter, we consider generative models, which aim to generate new data either randomly or based on input data. In the context of natural language processing, this is relevant for applications like question-answering, summarization, and translation. In Chapters 7 and 8, we will describe the application of generative models for the generation of password strings, an application for which we require the generation of new data both with and without input data.

Formally, given an input sequence $y_0, y_1, \dots y_N$ of tokens, we want to model the probability of the next tokens in the sequence $y_{N+1}, \dots, y_{N+M}$. This probability can be factorized into the conditional probabilities of each output token given the input sequence and the previous output tokens. We denote by p_θ the probability distribution calculated by a model with parameters θ and see that

$$p_\theta(y_{N+1}, \dots y_{N+M} | y_0, y_1, \dots y_N) = \prod_{i=1}^{M} p_\theta(y_{N+i} | y_0, y_1, \dots y_{N+i-1}). \tag{3.5}$$

This way, at every step of the generation, we can model the probability of the next token given the input sequence and the tokens generated so far [66]. We choose a new generated token $\hat{y}_i$ by sampling from this probability distribution, we discuss strategies for effective sampling of $\hat{y}_i$ below. During training, at generation step i we infer a loss function by maximum likelihood estimation [66], calculating the negative log-likelihood of the true token y_i under the predicted probability distribution p_θ,

$$\mathcal{L}(p_\theta, y_i) = -\log p_\theta(y_i | y_0, y_1, \dots y_{i-1}). \tag{3.6}$$

During inference, we do not have access to the true tokens and have to use the tokens generated so far by the model itself as input. At generation step $N+i$ we already generated the tokens $\hat{y}_{N+1}, \hat{y}_{N+2}, \dots \hat{y}_{N+i-1}$ and model the probability of the next token $\hat{y}_{N+i}$ as

$$p_\theta(y_{N+i} | y_0, \dots y_N, \hat{y}_{N+1}, \hat{y}_{N+2}, \dots \hat{y}_{N+i-1}). \tag{3.7}$$

To improve the efficiency of the training, we likely apply a method called teacher forcing, in which we use the true tokens $y_0, y_1, \dots y_{N+i-1}$ as input to the model instead of the tokens generated by the model itself [67].

Given a probability distribution $p_\theta(y_i)$, there are several ways of choosing a token $\hat{y}_i$. The simplest way is to choose the token with the highest probability at each time step, the so-called greedy approach, which often leads to repetitive and uninteresting texts. To alleviate this problem, we can consider more than one candidate sequence and choose the sequence of multiple tokens with the highest joint probability. This approach is called beam search and results in more diverse texts, which are, however, still prone to repetition [68].

To avoid repetition, we can use a sampling approach, in which we sample the next token from the probability distribution $p(y_i)$. To increase the coherence of the generated text, we can use a top-k sampling approach, in which we only consider the k most likely tokens for sampling [69]. This way, we can avoid sampling from the tail of the probability distribution, which often contains unlikely tokens. To further increase coherence, we can use a top-p sampling approach, in which we only consider the tokens with the highest cumulative probability mass of $p_\theta(y_i)$ for sampling [69].

Which decoding strategy to use depends on the application and the model architecture [70]. We present algorithms for text generation in Chapters 7 and 8. We will consider both models that use a greedy decoding strategy, namely the architectures based on generative adversarial networks (see Section 3.2.1) and variational autoencoders (Section 3.2.2), as well as models that use a top-k sampling approach, namely the transformer-based language models (Section 3.2.4).

3.2.1 GANs

Generative adversarial networks (GAN [44]) are a class of generative models that are able to generate new data. They are composed of two networks, a generator and a discriminator. After training, the discriminator is discarded, and the generator is used to generate new data.

The loss function for the GAN is given by

$$\mathcal{L}(x, z) = \mathbb{E}_{x\sim\mathcal{X}}\left[\log D(x)\right] + \mathbb{E}_{z\sim\mathcal{Z}}\left[\log(1 - D(G(z)))\right] \tag{3.8}$$

where $x \sim \mathcal{X}$ are samples from the real data distribution, $z \sim \mathcal{Z}$ are samples from the noise distribution, D is the discriminator network, and G is the generator network. The term $D(x)$ describes the probability of the discriminator correctly classifying real data as real, while $D(G(z))$ describes the probability of the discriminator correctly classifying fake data as fake. The first term in the loss function is the loss of the discriminator, which is trained to minimize the probability of the discriminator correctly classifying real data as fake. The second term is the loss of the generator, which is trained to maximize the probability of the discriminator classifying fake data as real. The training follows a min-max game in which the generator tries to generate samples that are indistinguishable from real data, while the discriminator tries to distinguish between real and fake data. In the ideal case, the generator is able to generate data that is indistinguishable from real data, and the discriminator is not able to distinguish between real and fake data.

The original GAN architecture [44] was applied to image data, however, further advances in the architecture and training process [47, 71] have made the GAN applicable to discrete data such as text as well. In Chapter 7, we apply a GAN architecture with a convolutional neural net with residual connections [72] as the generator and discriminator for the task of text generation for password data.

3.2.2 Variational Autoencoders

A variational autoencoder (VAE [48]) is a type of generative model that learns a probabilistic mapping between a high-dimensional data space (like images or text) and a lower-dimensional latent space. Like any autoencoder, it is composed of an encoder and a decoder, both of which are usually parametrized as deep neural networks [2].

The encoder is a parametrized mapping $q_\theta : \mathcal{X} \to \mathcal{Z}$, where $\mathcal{X}$ is the high-dimensional data space and $\mathcal{Z}$ is the low-dimensional latent space. The decoder is a parametrized mapping $p_\phi : \mathcal{Z} \to \mathcal{X}$, that takes a point in the latent space and maps it to a point in the data space. Both encoder and decoder are usually implemented as a type of deep neural network, for example, a convolutional neural network for images or a recurrent neural network for text. Additionally, the mapping is often not static but rather a stochastic process, in which the mappings q_θ and p_ϕ define parametrized distributions from which we sample points in $\mathcal{Z}$ and $\mathcal{X}$ [2].

A regular autoencoder is trained to minimize the reconstruction error. The reconstruction error for an encoder q_θ, a decoder p_ϕ and a data point $x \in \mathcal{X}$ is usually defined as the negative log-likelihood of the input data under the output distribution:

$$\mathcal{L}_{\text{rec}}(x) = \mathop{\mathbb{E}}_{z \sim q_\theta(z|x)} \left[-\log p_\phi(x|z) \right]. \tag{3.9}$$

This method of training exhibits some drawbacks: During training, an optimal behavior for the encoder-decoder pair is mapping each data point onto its respective vector in data space and increasing the space between individual learned points in latent space. This way, the decoder can learn which latent space points correspond to which data point and provide a good reconstruction. This behavior leads to issues with a generalization: If the latent space is learned in such a way that single meaningful latent vectors exist with large ‘holes’ in the latent space that the decoder has never seen, reconstruction of a data point from the test set will be challenging: the data will be encoded into an area of latent space the decoder has never handled during training, and the reconstruction will suffer as a result [48].

To alleviate these issues, one can add noise to the encoding process or implement various other regularization techniques. The variational autoencoder enforces a regularization on the distribution of encoded data in latent space. Unlike a regular autoencoder, we additionally require the latent space of the variational autoencoder to resemble a prior distribution $q(z)$. This allows us to sample from the latent space and generate new data. In order to achieve this, a variational autoencoder is additionally trained to minimize the Kullback-Leibler divergence between the encoder's output distribution and the prior distribution. This prior distribution is usually chosen to be a normal distribution [48]. The Kullback-Leibler divergence between the encoders output distribution and the prior distribution is defined as

$$\mathcal{D}_{KL}(q_\theta(z|x)||p(z)) = \mathop{\mathbb{E}}_{z \sim q_\theta(z|x)} \left[\log \frac{q_\theta(z|x)}{q(z)} \right]. \tag{3.10}$$

The total loss function for the variational autoencoder for a single data point x is then given as

$$\begin{aligned} \mathcal{L}(x) &= \mathcal{L}_{\text{rec}}(x) + \mathcal{D}_{KL}(q_\theta(z|x)||q(z)) \\ &= \mathop{\mathbb{E}}_{z \sim q_\theta(z|x)} \left[-\log p_\phi(x|z) \right] + \mathop{\mathbb{E}}_{z \sim q_\theta(z|x)} \left[\log \frac{q_\theta(z|x)}{q(z)} \right]. \end{aligned}$$

We minimize this loss via gradient descent on the parameters ϕ and θ of the encoder and decoder in batches over the training data.

Note that the formulation above requires the distribution of the latent encoding of each individual datapoint $q(z|x)$ to resemble the prior distribution $q(z)$. In practice, this may lead to issues where the optimization reaches a local minimum and each latent distribution perfectly replicates the prior $q(z)$. More recent work addresses this issue and proposes an alternative regularization method in which we consider the distribution of the latent variables over an entire batch of data [73].

To generate new data using a trained variational autoencoder, we can discard the encoder of the model and sample new latent vectors directly from the prior distribution. Since we enforced the encoded data to resemble data from the prior distribution, the decoder will be able to decode the latent vector into a point resembling the data distribution. In Chapters 7 and 8, we apply a variational autoencoder model to generate new text data by sampling random vectors from the prior distribution and decoding into data space using different decoding architectures.

3.2.3 Latent Space Geometry for Text Generation

So far, we have only explored the geometry of latent spaces implicitly. We discussed the theoretical advantages of latent space encoding over direct encoding like one-hot vectors, for one, the dimensionality reduction and on the other hand, the encoding of similar data into similar regions of the latent space. As shown illustrated in Figure 1.3, embedding models encode data in such a way that geometric regions in latent space correspond to similar semantic meanings, which has implicit advantages for text classification tasks: if similar text (where the semantic similarity is dependent on the specific training objective) is encoded in similar regions in latent space, a classification model will have an easier time generalizing from training data to unseen test data.

For text generation, we can see an explicit application of the geometry of latent space. Consider the example of the variational autoencoder above. During training, the VAE encodes data into latent space and tries to reconstruct the original data from its latent representation. During generation, we sample data from a distribution in latent space and decode it into a data point in data space. If we sample from the standard prior distribution, we will generate entirely random data points resembling the original data distribution [48].

However, we can adjust the sampling distribution to target specific regions in the data space. By encoding a certain data point and only sampling new latent vectors from a small region around this encoded data point, we can decode data that will resemble the target data point [48]. See Figure 3.4 (middle) for an illustration of this approach, where we encode the blue pentagon into latent space, sample latent vectors from the same region and decode into pentagons of similar color.

We can further apply this method to generate interpolations between data points. Given two data points, we want to generate new data that is semantically in between the two data points. Direct linear interpolation in data space is not feasible since the resulting data is unlikely to resemble real data [48, 74]. We demonstrate this issue in Figure 3.5, where we show the difference between interpolation in data space (left) and the kind of semantic interpolation we expect (right).

To achieve this, we encode both data points into latent space, interpolate linearly between both points and decode from a point along the interpolation. The resulting decoded data point will lie in the data distribution (i.e., resemble real data) and be semantically similar to both input data points (i.e., have properties of both input data points). For an early application of this method to text data, see [75].

We will apply this method for text generation in password data in Chapter 7. As an example of this, consider Figure 3.6. We use the geometric properties of the latent space to generate password strings that are similar to an input password string (here they contain the substring `love`) and use two real passwords

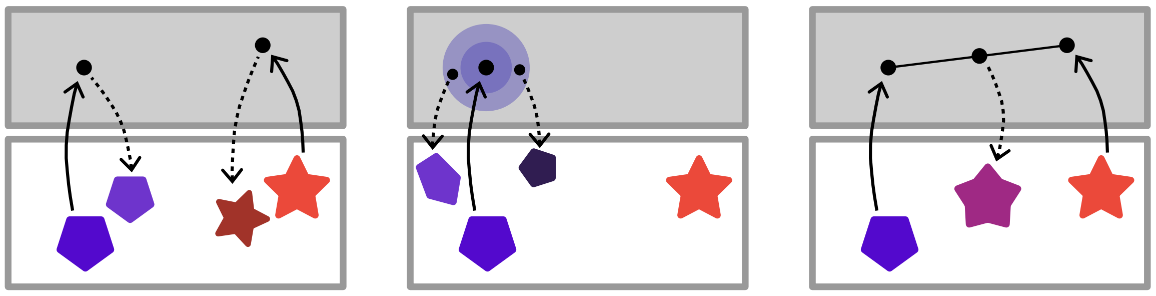
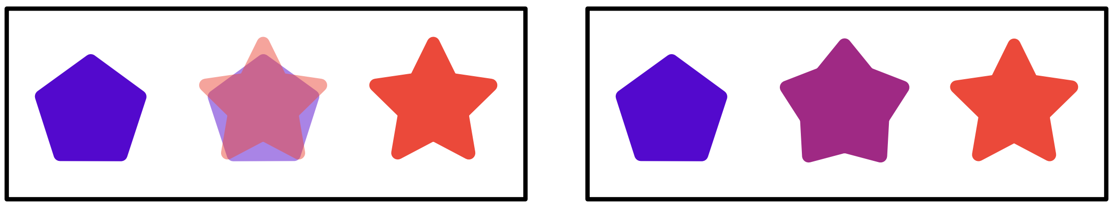


Figure 3.4: Illustration of the concept of latent space geometry for targeted generation. Left: During training, points in data space (lower area) are encoded in latent space (upper area) and reconstructed. Middle: To generate data similar to some target data, we encode the data into latent space, sample latent points around the encoded data, and decode it into data space. Right: We can generate mixtures of multiple data points by encoding both into latent space and interpolating between their latent representations.



(a) Linear interpolation in data space.

(b) Linear interpolation in latent space.

Figure 3.5: Illustration of interpolations between data points. Left: Linear interpolation in data space results in generated data that does not lie in the data distribution and does not make semantic sense. Right: Linear interpolation in latent space and decoding into data space results in data from the data domain that is a mixture of both data points.

as the start and end points of a linear interpolation in latent space, which results in a smooth interpolation in data space. This is important for generation tasks in which we have prior information. Suppose in a password recovery task we already have information about other passwords used in similar situations. We can encode the known password strings and use targeted sampling and interpolation to improve the quality of password guesses for this specific application.

This feature is only possible if the generative model architecture enforces such a structure on the latent space. For example, a standard GAN architecture as described in Section 3.2.1 is only trained to generate data from latent points sampled from a predefined prior distribution (usually a standard Gaussian). Since we do not have the ability to encode data into latent space in a meaningful way, we can not apply the targeted sampling approach described above. Note, however, that there are approaches to adjusting the GAN architecture to enable targeted sampling, for example, by combining variational autoencoders and generative adversarial networks [76, 77] or enhancing the latent space with additional information [78].

3.2.4 GPT and GPT2

The *generative pre-trained transformer* (GPT [79]) model is an auto-regressive decoder-only architecture for language modeling. This means that it reads an input sequence and outputs a probability distribution over the vocabulary for the next token in the sequence. Unlike BERT, GPT is not bidirectional but autoregressive,

```
                                          remington223   pepegrillo16

nublove85/9   miblovenv11                 rtninton2131   popegrillo16
siclove00me   riglover2k                  ntn123to2131   hoperillel62
failoveye4    n2ulovemswo                 rmnm2et21213   agterillo652
gemloveso1    irolovesor                  nnd231523465   allininal505
vatlover10    melovey4u                   ontit1321642   allinllang08
cetlovesder   9alolove71u                 andiew123456   alliannan680

                                          andrew123456   alliatnna068
```

Figure 3.6: Using the geometry of the latent space allows for targeted generation of text data. Left: We encode the string `love` and decode random passwords from a neighbourhood of the encoding. Right: We encode two real password strings (top and bottom of each column) and interpolate linearly between the encoded vectors. We decode multiple password strings along the interpolation.

meaning when predicting a new token, it only considers the context before the new token. The original transformer uses an encoder-decoder architecture, where the encoded context is fed into the decoder using cross-attention during the decoding process. In contrast to this, GPT only features a decoder that reads a prompt, for example, the beginning of a sentence and outputs the next token without additional context from an encoder.

This architecture can be trained unsupervised on raw text data by having the model read the beginning of a sequence and learning to output the next token, or in a supervised manner, for example, by prompting the model with a question and making it learn to output the corresponding answer. Unlike BERT, which is generally applied to text classification and general embedding tasks, GPT is particularly used for text generation. Later GPT models [49, 80, 81] allow for state-of-the-art performance in text generation.

A follow-up model to GPT, GPT2 [49], features minor architectural improvements, such as added layer normalization and adjusted weight scaling, as well as a larger parameter count. In Chapter 7, we will apply GPT2 as a language model for the generation of password data and see that the language modeling approach successfully generates usable password data, showing a performance comparable to the variational autoencoder approach and improving on the previous state-of-the-art for deep learning password generation significantly. In Chapter 8, we implement GPT2 as both the encoder and decoder of a variational autoencoder in order to leverage both the expressive power of GPT2 and the natural data generation capabilities of the autoencoder. We find that the combination of both models further improves on each of the individual architectures.

3.3 Tokenization

A key component of any NLP pipeline is tokenization. In order to make a text machine-readable, we must split it into smaller substrings and assign each substring an index from a given vocabulary.

In Section 3.1.4, we have seen how word-based tokenization requires preprocessing of the text in order to match the tokens to the limited vocabulary. Preprocessing and vocabulary limits remove information from the input text irreversibly.

In order to avoid this, we can apply lossless tokenization, such as character-based tokenization. The vocabulary is given by a set of characters, for example, lowercase letters `a-z`, uppercase letters `A-Z`, numbers `0-9`, and some special characters and punctuation. Any text consisting of these characters can be described

Text	The cat is black.
Tokens	T h e _ c a t _ i s _ b l a c k .
Indices	45 7 4 62 2 0 19 62 8 18 62 1 11 0 2 10 63

(a) Character tokenization of a sentence. The _ token denotes a whitespace character.

Text	The cat is black.
Tokens	The cat is bl #ack .
Indices	141 2543 19 3772 7865 12

(b) Subword tokenization of a sentence. Each token is implied to be prefixed by whitespace unless a token starts with a # character.

Text	The cat is black.
Preprocessed	the cat is black
Tokens	the cat is <UNK>
Indices	786 15420 11956 0

(c) Word tokenization of a sentence. The sentence is preprocessed (lower-casing and removal of punctuation) and the text is split on whitespace and matched to the large vocabulary. If a word (here the word `black`) is unknown, i.e., not found in the vocabulary, it is replaced by a special `<UNK>` token.

Figure 3.7: Comparison of character, subword, and word tokenization. The same original sentence is tokenized using the three methods, we show both the split text tokens and the corresponding indices from the vocabulary. Note that the difference in vocabulary size between character tokenization and the other two is apparent by the magnitudes of the respective indices.

perfectly by mapping each character to the corresponding index in the vocabulary [63].

The character set described above results in a rather small vocabulary of less than 100 individual characters, which has advantages, for example, during a text generation task. At each step, the model only has to decide between a small number of individual characters for the next character to generate. However, this tokenization also results in long sequences. Consider common words found in many texts such as `and`. A character tokenized model must generate three individual characters each time it wants to generate this word, which is computationally expensive and can provide issues with long context windows.

A way to improve on this issue is the use of *subword tokenization* [82]. Subword tokenization is a technique in which a training corpus of raw text data is used to extract a vocabulary of single characters and strings of multiple characters in order to find efficient tokenization of the data. The tokenization is lossless, meaning, like character-based tokenization, we can perfectly encode any input text and recover the original text from the calculated tokens. This type of tokenization increases the size of the vocabulary significantly, by default BERT has a vocabulary size of around 30 000 and GPT a vocabulary size of around 50 000, but decreases the number of tokens in each sequence [49, 65]. Most transformer-based architectures for language modeling apply a version of subword tokenization [49, 64, 65, 83–85].

In Chapters 5, 7 and 8 we apply character tokenization for language modeling and text generation, in Chapters 5, 6 and 7 we train transformer based language models using subword tokenization.

3.4 Transfer Learning

Various studies [86–89] have shown that training machine learning models on data not entirely related to the proposed task can improve the performance of downstream tasks. In practice, this means that given a limited amount of training data, one can first train a model on a large corpus of publicly available data and later continue training on the actual dataset.

We call this process *transfer learning* and the individual steps *pre-training* and *finetuning*. Transfer learning builds a base for the application of many text analysis models. Given the large number of trainable parameters in state-of-the-art models, a large amount of training data is needed to reach their full potential. This training data is seldom available for a specific task, which is why pre-training on a large corpus of publicly available data, such as CommonCrawl [90] or Wikipedia [91], is beneficial or even required for the application of these models.

The large number of parameters and the size of the public datasets make pre-training very costly in terms of computing time and resources. In order to facilitate this process, many models are publicly available with pre-trained weights. We apply transfer learning and compare the performance of pre-trained models to models trained from scratch on the relevant datasets in Chapters 5, 6, and 7.

3.5 Binary Optimization and Hopfield Networks

Moving on from classic natural language processing applications, we will focus on a further application in the field of automated auditing in Chapter 9. As described in Section 2.1, we will expand the automated analysis of financial reports to consistency checks within numerical tables in these documents.

In [15], we present a way of encoding the task of finding sums of values in numerical tables as the subset sum problem, which is an NP-hard binary optimization problem [92]. We show that this problem has an equivalent formulation as a *quadratic unconstraint binary optimization* problem (QUBO [93]), in which we optimize a quadratic polynomial function $f : \{0,1\}^n \rightarrow \mathbb{R}$ of the form

$$f(\boldsymbol{z}) = \sum_{ij} \boldsymbol{P}_{ij} \boldsymbol{z}_i \boldsymbol{z}_j \tag{3.11}$$

to find the optimal binary vector

$$\boldsymbol{z}^* = \underset{\boldsymbol{z} \in \{0,1\}^n}{\arg\min}\, f(\boldsymbol{z}) = \underset{\boldsymbol{z} \in \{0,1\}^n}{\arg\min}\, \boldsymbol{z}^T \boldsymbol{P} \boldsymbol{z} \tag{3.12}$$

with a symmetric matrix $\boldsymbol{P} \in \mathbb{R}^{n \times n}$. We encode the subset sum problem into a QUBO in a way that the binary vector $\boldsymbol{z}$ indicates which values of the table belong to the sum and the function $f(\boldsymbol{z})$ has a global minimum if the sum of all values indicated by $\boldsymbol{z}$ is equal to a target sum.

While finding the solution to a QUBO problem is NP-hard, there are several algorithmic approaches to finding optimal or approximate solutions like heuristic algorithms [94] and genetic algorithms [95].

In order to solve the corresponding QUBO to the underlying subset sum problem will present a GPU-accelerated algorithm for which we apply a special type of recurrent neural network, called a Hopfield network [96]. The network consists of n interconnected neurons, each with a value in $\{-1, 1\}$. The state of the network can be described by a binary vector $\boldsymbol{s} \in \{-1, 1\}^n$. Each neuron is connected with every other neuron by connection weights $\boldsymbol{W} \in \mathbb{R}^{n \times n}$, such that $\boldsymbol{W}_{ii} = 0$ and $\boldsymbol{W}_{ij} = \boldsymbol{W}_{ji}$ for all i, j. Each neuron additionally contains a threshold value $\boldsymbol{\theta}_i \in \mathbb{R}$. We define the energy of the network for a certain state as

$$E(\boldsymbol{s}) = -\frac{1}{2}\boldsymbol{s}^T \boldsymbol{W} \boldsymbol{s} + \boldsymbol{\theta}^T \boldsymbol{s}. \tag{3.13}$$

Given the constraints on $\boldsymbol{W}$ stated above, updating the state of a single neuron by

$$s_i = \operatorname{sign}\left(\boldsymbol{W}_i^T \boldsymbol{s} - \boldsymbol{\theta}_i\right) \tag{3.14}$$

can never increase the energy value of the network. By updating the network as in (3.14) multiple times for different neurons, we perform gradient descent on $E(\boldsymbol{s})$. Since there are only 2^n possible states of the network, this procedure is guaranteed to find a local or global minimum after a finite amount of update steps [97].

This type of optimization procedure is useful for many problems in binary optimization. We initialize the weights $\boldsymbol{W}$ and thresholds $\boldsymbol{\theta}$ such that an energy minimum of the network as in (3.13) corresponds to a solution to the optimization problem and performs the gradient descent update steps until a convergence state of minimal energy is reached, at which point the state of the network is the solution to the binary optimization problem [97].

In Chapter 9, we show how GPU-accelerated gradient descent can find solutions to these complex binary optimization problems. We show that the algorithm is capable of solving subset sum problems that arise in the consistency check of real financial report documents and explore its limitation by testing on artificial data. Additionally, we show how this architecture is related to the field of quantum computing and apply a quantum computing architecture to this type of problem.

CHAPTER 4

Interpretable Topic Extraction and Word Embedding Learning Using Non-Negative Tensor DEDICOM

This chapter is based on the publication "Interpretable Topic Extraction and Word Embedding Learning Using Non-Negative Tensor DEDICOM" by Lars Hillebrand, David Biesner, Christian Bauckhage and Rafet Sifa, published in *Machine Learning and Knowledge Extraction 3* (2021) [53].

Word embeddings are a vital tool for natural language processing. As previously established in Section 3.1.4, in order to be processed by machine learning models, a text must be converted into numerical vectors first. One way of embedding text is splitting the text into words and assigning each word a predefined vector. This converts a sentence into a sequence of numerical vectors, one vector for each word in the text. These word embeddings must contain information about the word itself and its usage in order to make it usable for downstream tasks.

Many word embedding models are based on a matrix factorization approach [58, 98]. Take GloVe (Section 3.1.4), which extracts an adjusted concurrency matrix of words from a text corpus and factorizes this matrix into a matrix of lower dimensionality. This method of embedding text has been shown to yield good performance on downstream tasks while being computationally inexpensive: Once the word embeddings are pre-computed, they can be easily stored and shared in lookup tables, and for embedding a text, one must only split the text into single words and retrieve the corresponding embedding for each word.

While this method of embedding text is more intuitive and requires fewer parameters than deep-learning language models, the resulting word embeddings are not any more interpretable in the sense that individual entries of a word embedding do not have any extractable semantic meaning. The calculated embeddings are n-dimensional numerical vectors, where n is usually relatively large, e.g. 300 for the default GloVe implementation. This poses a limitation for the development of machine learning solutions for downstream tasks, as we can not infer any information about the models from knowing which dimensions of the embedding it focuses on.

In [53], we present a novel approach for learning interpretable word embeddings in combination with topic modeling. The method is based on the *DEDICOM* algorithm [99], which factorizes a square matrix $\boldsymbol{S} \in \mathbb{R}^{n \times n}$ into three matrices by

$$\boldsymbol{S} \approx \boldsymbol{A}\boldsymbol{R}\boldsymbol{A}^T, \quad \boldsymbol{A} \in \mathbb{R}^{n \times k}, \boldsymbol{R} \in \mathbb{R}^{k \times k}, \tag{4.1}$$

where we call $\boldsymbol{A}$ the loading matrix and $\boldsymbol{R}$ the affinity matrix.

We apply this factorization approach to word embeddings by factorizing a logarithmic cooccurrence matrix $\boldsymbol{S} \in \mathbb{R}^{n \times n}$, where n is the vocabulary size, into a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times k}$ and an affinity matrix $\boldsymbol{R} \in \mathbb{R}^{k \times k}$, where $k \ll n$ is the dimension of the word embeddings. We enforce row-stochasticity on the matrix $\boldsymbol{A}$, such that each word embedding describes a probability distribution over the k dimensions. By enforcing this constraint, we can interpret each word embedding as a mixture of k individual components, which we call topics. A word is relevant to a topic if its entry in the word vector in the respective dimension is large and we identify the content of the topics by taking into account the most relevant words for each dimension.

For example, if the words with the largest word embedding entries for a topic dimension are `cup`, `football` and `fifa`, we can interpret the topic to be `soccer`. Since

$$\boldsymbol{S}_{ij} \approx \sum_{b=1}^{k} \sum_{c=1}^{k} \boldsymbol{A}_{ib} \boldsymbol{R}_{bc} \boldsymbol{A}_{jc}, \tag{4.2}$$

we can estimate the probability of the cooccurrence of any two words w_i and w_j in the vocabulary by calculating a weighted sum of their word embedding entries, where the weights are given by the relationship matrix $\boldsymbol{R}$. This way, we can interpret matrix $\boldsymbol{R}$ as a matrix describing relationships between topics in the corpus: A large value in $\boldsymbol{R}_{bc}$ describes a likely cooccurrence of words that belong to topics t_b and t_c and a semantic relationship between the respective topics. Further, we extend this method to a tensor factorization approach, where the cooccurrence matrix is not only calculated on one corpus but on multiple text sources. The resulting factorization yields word embeddings for the entire data, meaning one set of topics for the entire data, and one relationship matrix for each text source. This way, one can identify the different relationships between topics depending on the individual text source.

In [53], we show that this approach generates interpretable word embeddings for a variety of text corpora. We demonstrate how the algorithm models logical relationships between the topics in the text and how the topics are identifiable by the words that are assigned the greatest weights in their respective dimensions. Additionally, we establish how the tensor factorization method successfully models the relationships of topics in a series of text corpora, for example extracting the changing focus of news outlets on various political events over time.

Contributions

My contributions to these papers as first author (shared first authorship) are as follows: I was in equal parts with my first co-author responsible for research, planning, and conceptualization of the project, as well as the deduction of the update rules for the matrix formulation of the DEDICOM algorithm. I was in equal parts with my first co-author responsible for the conduction of the experiments presented in the paper and the main contributor to the evaluation of the interpretability of the algorithms. I was in equal parts with my first co-author responsible for the text, tables, and figures of both papers and the main contributor to the respective section on interpretability.

CHAPTER 5

Anonymization of German Financial Documents using Neural Network-based Language Models with Contextual Word Representations

This chapter is based on the publication “Anonymization of German financial documents using neural network-based language models with contextual word representations” by David Biesner, Rajkumar Ramamurthy, Robin Stenzel, Max Lübbering, Lars Hillebrand, Anna Ladi, Maren Pielka, Rüdiger Loitz, Christian Bauckhage, and Rafet Sifa, published in the *International Journal of Data Science and Analytics 13* (2022) [14].

The previous chapter introduced a method of word embedding, which is a powerful tool for converting text into machine-readable information with various properties that enable their use on downstream tasks (see also Section 3.1.4). They are easy to implement and computationally inexpensive to use. However, they present multiple downsides.

First, they deal with a fixed vocabulary. Any word-embedding model has a vocabulary of N predefined words that it can embed meaningfully. While this vocabulary may be very large (the pre-trained GloVe vocabulary contains between 400k and 2.2M entries, depending on the training corpus), all finite vocabularies will eventually run into out-of-vocabulary issues, where we can not find a suitable word embedding for a string in the text and have to ignore the word or use the embedding of a generic `<UNK>` token. Preprocessing text, like lower-casing, removing punctuation, stemming, and lemmatization, reduces the risk of out-of-vocabulary words. Nevertheless, many texts will contain words, often names of persons and places, for which no embedding is available.

Second, pre-trained word embeddings contain no context information. Consider the following sentences.

```
The early bird catches the worm.
Mr. Bird is a data scientist.
```

After lowercasing, both contain the token `bird`, which will receive the same word-embedding using any word-embedding model. However, it is evident that both tokens refer to completely different concepts, one to an animal, the other to the name of a person. Any model for downstream tasks will not be able to differentiate between the two concepts.

A way to deal with these issues are language models, which we introduced in Section 3.1.5. Language models are machine learning models that read tokenized text and output embeddings for each token. Instead of assigning each token a fixed pre-calculated embedding, the embeddings are calculated during inference time with regard to context, i.e., the previous and next tokens in the sequence. Such models are trained by reading a large corpus of non-annotated text data and trying to predict the next token in a sequence.

In the present work, we tackle the issue of de-identification and anonymization of financial text documents. Text in financial reports may contain sensitive information, such as names of people, organizations, and locations, that needs to be anonymized (replaced by generic tokens) or pseudonymized (replaced by tokens that still allow for tracking of entities in a document) before publication or distribution to another party. Manual anonymization of text is possible but very time-consuming and prone to errors. A machine learning model trying to anonymize text using word embeddings would soon run into problems, as evident considering the example given above. A model processing both sentences would have to decide whether or not to anonymize the token `bird` based on its word embedding. Either decision would result in errors, either anonymization of non-sensitive tokens or not anonymizing sensitive information.

To this end, we employ a language model based on a character-based bidirectional LSTM, trained on a large corpus of raw German financial document text data (without annotations for sequence tagging). The model embeds each word by processing a text paragraph by characters, reading the text bidirectionally and concatenating the embeddings of each word's first and last character. During language modeling training, the model is trained to predict the next character in the sequence.

In the paper, we compare the performance of various pre-training data corpora (general language and financial domain data) and the effect of the size of the embedding, i.e., the dimension of the hidden state of the LSTM, on the classification performance. We infer that training on domain-specific language is beneficial to the model in any configuration and the size of the embedding is a hyperparameter that provides a trade-off between classification performance and inference time. While the best model is based on embeddings of dimension 4096, a model with embedding size 2048 provides very similar performance while reducing the inference time by approximately 60%.

We additionally consider the transformer-based BERT model (see also Sections 3.1.7 and 6) as the embedding model and find that it provides similar performance to the 2048 dimensional RNN model and does not reach the best RNN models both in terms of performance or inference speed.

The actual named entity recognition is based on a sequence classification architecture with the contextual word embeddings calculated by the language model as input. Each architecture outputs a probability distribution over the classes of names entities (`O`, `ORG`, `PER`, `LOC`, `PROD`, `SEG` and `OTH`). We compare three different architectures: a regular feed-forward neural network with one layer mapping the word embedding to a probability distribution over the number of classes, an RNN, which is a bidirectional LSTM reading the sequence of word embeddings, and the RNN with an additional conditional random field (CRF [100]), which models the conditional dependencies between labels in the sequence. We find that the RNN with CRF provides the best classification results, while the choice of classifier does not impact inference time significantly.

We evaluate the model on recall, precision, and the resulting F1-score. We can interpret the recall as the performance on anonymization. A high recall means that little sensitive tokens were missed during the sequence tagging. We interpret precision as a readability score, a high precision means that little tokens which are not sensitive were anonymized unnecessarily. To improve the performance on the classification task, we employ a post-processing step of anonymization of the same token in the entire document if it is tagged as a sensitive entity once. Before post-processing, the best model shows very good performance for anonymization and readability (97.3% precision, 97.1% recall, 97.2% F1-score). After post-processing, the best model shows nearly perfect anonymization performance while keeping the text readable (90.3% precision, 99.0% recall, 94.7% F1-score).

Contributions

My contributions to this paper as first author (shared first authorship) are as follows: I was in equal parts with my first co-author responsible for the research, planning, and conceptualization of the project. I was the main contributor to overseeing the data acquisition and annotation and the implementation of the data infrastructure. I was the main contributor to the implementation and tuning of the post-processing methods and the evaluation of their efficiency in the anonymization process. I was responsible for the generation of the figures and tables in the final version of the paper and was together with my first co-author in equal parts responsible for the writing of the text.

CHAPTER 6

Zero-Shot Text Matching for Automated Auditing using Sentence Transformers

This chapter is based on the publication "Zero-Shot Text Matching for Automated Auditing using Sentence Transformers" by David Biesner, Maren Pielka, Rajkumar Ramamurthy, Tim Dilmaghani, Bernd Kliem, Rüdiger Loitz, and Rafet Sifa, published in the proceedings of the *21st IEEE International Conference on Machine Learning and Applications* (2022) [101].

The last chapter considered the application of character-level language models for the task of anonymization of financial reports. Anonymization of these reports may be necessary for a number of reasons, one of which is the use of reports in the development of further machine learning systems for analysis of financial report texts. One of these systems is the recommender for automated auditing, which we introduced in Section 2.1 and shown in Figure 2.2. The goal of this tool is to find the paragraphs in a financial report which are relevant to a given regulatory requirement or vice versa.

Previous work [12, 13] interpreted this task as multilabel text classification: Given a set of N regulatory requirements and one trains a predictor to predict which of the N possible requirements are relevant to a given text passage. Since more than one requirement can be relevant to a given text passage, this is a multilabel classification task on N labels. To recommend the most relevant text paragraphs for a given requirement, one encodes and predicts each paragraph in a document separately and either recommends each paragraph with a positive prediction for the requirement or ranks the paragraphs by their prediction score for the corresponding label. In [12], the authors use the word-based *tf-idf* algorithm [102] to encode the text passages and train a set of logistic regression models to predict the labels. In [13], the authors use a BERT model to encode the text passages and investigate the effect of RNN and MLP classification layers. The application of transformer architectures for text embedding provides significant performance gains over the more simple tf-idf-based embedding.

While these approaches are capable of classifying texts to the correct requirements for the given datasets of financial reports and auditing standards, the multilabel classification interpretation has a number of disadvantages: While both works show that the chosen approach works very well for the given set of requirements, it is very inflexible and cannot be applied to new requirements. Should the regulatory checklist change, which happens fairly regularly [103], the set of prediction labels changes and the model (at least the prediction layers) must be retrained on new annotated data. There is no mechanism with which to apply the model to new requirements without retraining. Furthermore, a model trained on one checklist can not be applied to other checklists or checklists in other languages at all.

In contrast to this approach, in [101] we apply the SentenceBERT [56] architecture to the task, converting the multilabel classification task into a text similarity matching task. For this, we use a BERT model to encode both the paragraph text from the financial report and the description text from the regulatory requirement and calculate the cosine similarity between the two embeddings. We then rank the paragraphs by their similarity score to the requirement and recommend the top k paragraphs. This interpretation of the assignment of pairs of text paragraphs and requirements more closely models the way a human auditor would approach the task. An auditor does not view a requirement as a numbered label and learns which types of text fit this label, but rather reads the description text of the requirement and finds text in the document that corresponds to this description.

The base architecture of this model is a regular BERT model in a siamese architecture, meaning that the same model is used to produce embeddings for both sequences. During training, the model encodes two sequences which may or may not match and produces one embedding for each sequence. The training loss is given by the cosine distance between the two embeddings, where we want the distance to be small for matching sequences and large for non-matching sequences.

We show that this approach leads to prediction performance that is comparable to the previous approaches, while providing usable results on new requirements and entirely new checklists, which is not possible at all with the multilabel classification methods. We compare two methods of unsupervised pre-training, an unsupervised similarity matching approach based on [104] and a denoising autoencoding approach based on [105], and find that the autoencoding approach improves performance significantly. We compare supervised training on different combinations of datasets, German data, English data, and German and English data combined, and find that training on both languages is preferable even when the inference language is only German or English, and that training on one language and using it for inference on the other is not recommendable in any configuration. Additionally, we analyze the language modeling step of the base BERT model, which is conducted before the unsupervised and supervised training begins. We compare training on general language data and domain-specific data, i.e., not-annotated financial reports, and find that the effect of this training is negligible once the model is trained in the unsupervised and supervised manner.

This work provides the basis for recommender systems for automated auditing to be applied beyond the confines of a single auditing standard and language, such that trained models can be used in multiple regions of interest.

Contributions

My contributions to this paper as first author are as follows: I implemented parts of the models and the training procedures, mainly the TSDAE algorithm and an overhaul of the training and evaluation routine. I conducted all experiments presented in the paper, compiled and processed the evaluation results and generated their presentations in the text. Finally, I was the main contributor to the text, figures, and tables in both the first draft and the final version of the paper.

CHAPTER 7

Advances in Password Recovery Using Generative Deep Learning Techniques

This chapter is based on the publication "Advances in Password Recovery Using Generative Deep Learning Techniques" by David Biesner, Kostadin Cvejoski, Bogdan Georgiev, Erik Krupicka, and Rafet Sifa, published in the proceedings of the *30th International Conference on Artificial Neural Networks and Machine Learning* (2021) [35].

The previous chapters focused on text embedding models for text classification and text similarity matching, in which the embeddings are either used directly or provide the bases for classification models. We have seen that transformer models improve on more traditional NLP models like tf-idf and RNNs in terms of performance significantly.

The following chapters will consider the effect of transformer architectures on the task of text generation. We will consider a text generation task in the domain of cybersecurity data, namely the generation of password strings, and investigate how latent variable models and generative transformers allow for the generation of realistic text.

As discussed in Chapter 2.2, the password generation task requires a model to learn the general structure of password data from a dataset of cleartext passwords and generate a large number of password candidate strings. The generation can be either unsupervised, without context information for the generated string, or supervised, with additional context information (like an input prompt), which should guide the generation process.

One of the first works to apply modern deep-learning text generation models to the password generation task was *PassGAN* [29], in which the authors used a generative adversarial network for the generation of password strings.

Generative adversarial networks are inherently not ideal for text generation due to the discrete nature of text data. Tokenized and vectorized text exists as discrete indices or one-hot encoded vectors, and non-integer values are not part of the data domain. Unlike images, which can exist as real-valued vectors, text generation must include a step in which one chooses a specific token from a given real-valued probability distribution. This step is non-differentiable in general and provides issues with GAN architectures that require an entire output sample to be passed to the discriminator [106].

While there are workarounds to this issue [106, 107], in [35], we investigate the capability of other architectures for the password generation task and find that both variational autoencoders and generative transformers are able to generate passwords of much higher quality than the GAN approach presented in [29].

We construct the variational autoencoder with encoder and decoder based on ResNet [72], which is a convolutional neural net with residual connections between the layers. We train the model on password text tokenized by characters and generate new data by sampling new latent vectors from a standard normal distribution and passing it to the decoder.

For the generative transformer, we train the GPT2 [49] (see also Section 3.2.4) architecture on password data. For this, we generate a dataset of raw text documents consisting of randomly arranged passwords concatenated into continuous text. We compare both a GPT2 model trained from scratch and a GPT2 model pre-trained on general language data and finetuned on password data. Both models read the password text as subword tokenized input.

We additionally show how models that encode data into latent space, in our case the variational autoencoder model, are able to use the geometric properties of the latent space for targeted generation. We show that the trained model is able to encode a data point, in our case a specific password string, and output various passwords that are similar, for example, because they share a certain substring. In the same manner, we can apply the idea of linear interpolation in latent space introduced in Section 3.2.3 to encode two passwords and generate an interpolation of password strings between them.

Quantitatively, we show that both variational autoencoder and generative transformer produce passwords of higher quality than the GAN approach, resulting in a significant increase in the number of passwords reconstructed across all test sets, e.g., 44.9% reconstruction of *rockyou* for the VAE, 45.1% for GPT2, in contrast to 15.9% for GAN. We investigate the ability to generate high-quality password strings for all models qualitatively and demonstrate the ability of the VAE for targeted generation of password strings. We show that both generation of passwords similar to an input string as well as smooth interpolation between two input passwords is possible and results in output data usable for the password recovery task. As an example of this, see also Figure 3.6.

Contributions

My contributions to this paper as first author (shared first authorship) are as follows: I was in equal parts with my first and second co-authors responsible for the research, planning of the project, and conceptualization of the machine learning components. I was in equal parts with my first and second co-authors responsible for the general code base for training and evaluation of the models. I was the main contributor to the acquisition of training and evaluation data and the implementation of data preprocessing. I was responsible for the development, training, and evaluation of the generative transformer models and conducted various experiments and evaluations regarding the targeted generation of password data. Additionally, I was responsible for the implementation and evaluation of the classic (i.e., not deep-learning) algorithms for comparison. During the paper writing process, I was in equal parts with my first and second co-authors responsible for the majority of the text, and was the main contributor to the section on the generative transformer model.

CHAPTER 8

Combining Variational Autoencoders and Transformer Language Models for Improved Password Generation

This chapter is based on the publication “Combining Variational Autoencoders and Transformer Language Models for Improved Password Generation” by David Biesner, Kostadin Cvejoski, and Rafet Sifa, published in the proceedings of the *17th International Conference on Availability, Reliability and Security* (2022) [36].

The previous chapter has shown that both encoder-decoder architectures, like the variational autoencoder, and decoder-only language models with transformer-based components are capable of generating a large number of passwords with high diversity and good quality. However, both models have their limitations.

On the one hand, the variational autoencoder trained in the previous work is limited in terms of its architecture, which was based on convolutional neural nets. While CNNs are a powerful architecture, recent advances in NLP have shown that architectures based on self-attention are often superior to sequential architectures like CNNs and RNNs [64].

On the other hand, the decoder-only language model GPT2 employs a transformer-based architecture that is capable of generating high-quality passwords, but the generation process itself is less a controlled random process and more of an artifact of the training procedure. During the training, we let the model learn from a concatenated string of random passwords. There are no patterns in the sequence of passwords, and the model can only learn to generate a random password after each whitespace character. Ideally, we aim to combine the features of both architectures, having a generative latent variable model with meaningful latent space that has the same expressive power as the transformer-based language model.

In this work, we proposed a novel architecture, the *variational GPT2* (VGPT2), which combines the variational autoencoder with the transformer-based language model GPT2, and show that it surpasses both previously established methods of generating passwords using deep-learning generative models in terms of the quality of generated passwords.

The model is an encoder-decoder architecture trained to encode a single password string into latent space and decode the latent vector back into the original string. The loss is given, just like the regular variational autoencoder discussed in Section 3.2.2 and Chapter 7, by a reconstruction loss and a loss on the distribution of the latent vector.

Both the encoder and the decoder are individual GPT2 models. While GPT2 is a decoder-only architecture in the sense that there is only a mechanism to read an input prompt and predict the next token in the sequence, internally, the input prompt must be encoded by the transformer blocks. We take the token embeddings of the input string, apply a pooling operation and map the input embedding onto the mean and variance vector for the latent distribution. We sample a latent vector from the distribution and provide the needed context for the reconstruction by concatenating the latent vector to the embedded input token to the decoder at every time step. We provide further details and illustrations of the architecture in [36].

We conduct experiments on the same datasets for training and testing as in [35] and see that the model is capable of generating passwords of higher quality, reconstructing more passwords from the test set than the other models for each tested number of generated passwords, for example 48.3% reconstruction of the *rockyou* test set as opposed to 44.9% reconstructed by the variational autoencoder from [35] and 23.6% reconstruction of the *linkedin* test set as opposed to 21.8% reconstructed by the variational autoencoder from [35].

The combination of variational autoencoders and GPT2 paves the way for the application of any type of transformer language model as the encoder and decoder module of the autoencoder, enabling the use of powerful generative models for the generation of password data.

Contributions

My contributions to this paper as first author are as follows: I was the sole contributor to all additional code to the data, models, training, and evaluation in [35]. Specifically, I implemented the combined VAE and transformer model and conducted all experiments and evaluations regarding this model. I was the sole contributor to all new experiments and evaluations presented in the paper. I was responsible for the first draft of the paper and was the main contributor to the final version of the text, tables, and figures presented in the paper.

CHAPTER 9

Solving Subset Sum Problems using Quantum Inspired Optimization Algorithms with Applications in Auditing and Financial Data Analysis

This chapter is based on the publication “Solving Subset Sum Problems using Quantum Inspired Optimization Algorithms with Applications in Auditing and Financial Data Analysis” by David Biesner, Thore Gerlach, Christian Bauckhage, Bernd Kliem, and Rafet Sifa, published in the proceedings of the *21st IEEE International Conference on Machine Learning and Applications* (2022) [15].

The main part of this thesis has described the various ways in which representation learning models with a focus on natural language processing can be applied to problems in the realms of automated auditing and cybersecurity. For the final chapter of the thesis, we will take another look at the field of automated auditing and present a novel approach to numerical consistency checks in financial documents. We show how encoding an NP-hard optimization problem using a recurrent neural network allows for the usage of GPU-accelerated binary optimization algorithms.

Previously, Chapters 5 and 6 introduced methods of preparing financial reports for the development of machine learning algorithms and using recommender systems to improve the auditing workflow. The recommender systems decrease the need for human auditors to search manually through long text documents and increase the time an auditor can spend applying their expert knowledge to analyze the content of the document.

However, even with a reliable recommender system in place, there are aspects of the auditing workflow that remain manual work done by auditors and show potential for improvement by automation. As mentioned in Section 2.1, one such task is the internal consistency check of numerical tables in financial documents. Financial reports feature numerical data in structured tables, that highlight key indicators like revenue and profit. These numbers are connected, often aggregating values from within the table, such as total revenue deriving from subsidiary revenues. Auditors must ensure their accuracy and internal consistency. For this application, we focus on the correctness of the aggregated values in the table. If these sums are incorrect, that indicates a potential error in the financial report since either the sum value is incorrect or one or multiple of the values that are summed up are incorrect. This type of correctness check must be performed by auditors for each numerical table in the document.

During the auditing process, financial documents are prepared in .pdf or .docx format, which does not include machine-readable information on the underlying structure of tables. Therefore, one must find an algorithmic way of identifying which values are supposed to be sums of other values, which becomes very complex for tables with many rows and columns, complex structure, and little regulation on the formatting.

In [15], we present a novel approach to this type of numerical consistency check, in which we do not try to identify the exact sum structure of the table but rather treat finding sums in tables as a combinatorial optimization problem. The problem of finding the correct values in a table that sum up to a certain target element is equivalent to the subset sum problem, which is a well-known NP-complete problem. Due to its complexity, we can not solve the subset sum problem exactly for large tables. Therefore, we consider a reformulation of the problem as a quadratic unconstrained binary optimization problem (QUBO, Section 3.5). We encode the subset sum problem as a QUBO and apply Hopfield networks, introduced in Section 3.5, such that the state of the network represents the current solution to the subset sum problem and that the energy minimum of the network corresponds to a state which is an optimal solution.

We present a novel algorithm for efficient minimization of the energy of Hopfield networks using parallelized computing on GPUs. In a quantitative evaluation, we show that this reformulation of the problem allows us to find correct sums in real-world financial documents and various types of artificial tables. This method of solving QUBOs is inspired by the idea of solving QUBOs on specialized quantum hardware, so-called quantum annealers. This type of computer can solve complex binary optimization problems very efficiently, given that the problem can be encoded in the number of available quantum bits [97]. We demonstrate the ability of quantum hardware to solve real-world application problems by calculating the correct sum structure for multiple financial report tables. By encoding consistency checks in this manner, we can apply both Hopfield networks and gradient descent on regular GPU-accelerated hardware and use quantum hardware as soon as the required number of quantum bits is available, without needing to adjust the underlying representation of the problem.

Contributions

My contributions to this paper as first author are as follows: I was the main contributor to the implementation of the models and training routines and the sole contributor to the acquisition and preprocessing of suitable data. I was the sole contributor to the conduction of experiments using the Hopfield networks and the evaluation of the results, as well as the evaluation of the results of the experiments on quantum hardware. I was responsible for the first draft of the paper and the main contributor to the final version of the text, tables, and figures presented in the paper.

CHAPTER 10

Conclusion and Future Work

In this thesis, we investigated the potential of representation learning for various tasks in the fields of automated auditing and cybersecurity. To conclude, we will summarize the results of the individual chapters and works that build the basis of the thesis. Additionally, we will give an overview of more recent advancements in natural language processing and discuss their potential for the tasks presented in this thesis and how future work can build on the results presented here.

10.1 Summary of Results

The individual works referenced in this thesis have shown how architectures for representation learning have evolved over the last years, from simple embedding models to complex generative deep learning models with semantically interesting latent space properties.

The methods for word embedding introduced in Chapter 4 provide powerful methods of embedding text with very little computational cost at inference time and the embeddings provide a layer of interpretability to the machine-learning system that apply them to downstream tasks. We showed how the novel DEDICOM word embedding algorithm is able to generate interpretable word embeddings and extract topics from text corpora. However, all word embedding models suffer from the limited word vocabulary that results in issues with homonyms and out-of-vocabulary words.

The introduction of character-based recurrent neural nets in Chapter 5 provided a clear advantage over the use of word embeddings for many tasks. For one, since the entire text can be tokenized lossless by characters, we do not have to deal with out-of-vocabulary words anymore, which is especially important for the task of de-identification, which we tackled for financial report documents. In these reports, many of the words to be tagged will be names, places, and organizations that are unlikely to have a pre-trained embedding. Second, the context-aware nature of the embeddings lets the system differentiate between common words and names with the same spelling. Our experiments show that this type of context-aware embedding can provide the basis for near-perfect anonymization while keeping the output text readable.

In light of the advancements in language modeling provided by the transformer architecture, we applied a language model based on a bidirectional transformer to the task of text similarity matching in Chapter 6. We showed that this architecture is able to encode text into latent space in such a way that the model can match text passages from new financial reports to known auditing requirements with competitive performance and still perform well when matching new financial reports to previously unknown auditing requirements. This matching of auditing requirements not seen during training is a clear advantage over the more limited and less flexible multilabel classification approach of previous work.

The emergence of transformer architectures as the dominating building block for NLP models has also affected the works presented in Chapters 7 and 8. In both chapters, we discussed the capability of deep learning text generation models of various architectures for the task of generating password strings. In Chapter 7, we compared two more classical approaches, namely a generative adversarial network and a variational autoencoder, both with ResNet-based encoding and decoding blocks, to a transformer-based language model. We found that both the variational autoencoder and the transformer language model provided similar performance in the password generation task across multiple test datasets, improving significantly on the previous state-of-the-art for deep learning password generation and most classic password reconstruction methods. We showed that the variational autoencoder is able to enforce a geometry on the latent space, in which encoded vectors have geometric properties that correspond to the semantic properties of the respective data. We demonstrate this by qualitative evaluations of targeted sampling and interpolation of text data. In Chapter 8, we approach the combination of both architectures, leveraging both the expressiveness of the transformer-based language model and the capability of generating new data from latent space sampling of the variational autoencoder to build a model that improves password generation performance on both individual architectures.

Finally, in Chapter 9, we investigated a further application in the field of automated auditing by considering the use of neural networks for automated consistency checks in financial reports. We encoded the problem of verifying the correctness of sums in tables using a Hopfield network such that the state of minimal energy of the network corresponds to a solution to the problem. We presented a novel GPU-accelerated gradient descent algorithm and demonstrated it's ability to solve consistency checks for real tables and artificial data. Additionally, we detailed how this approach relates to the application of quantum hardware to the problem and described how sufficiently advanced quantum hardware is directly applicable to solve this type of consistency check reliably and exceedingly faster than traditional hardware.

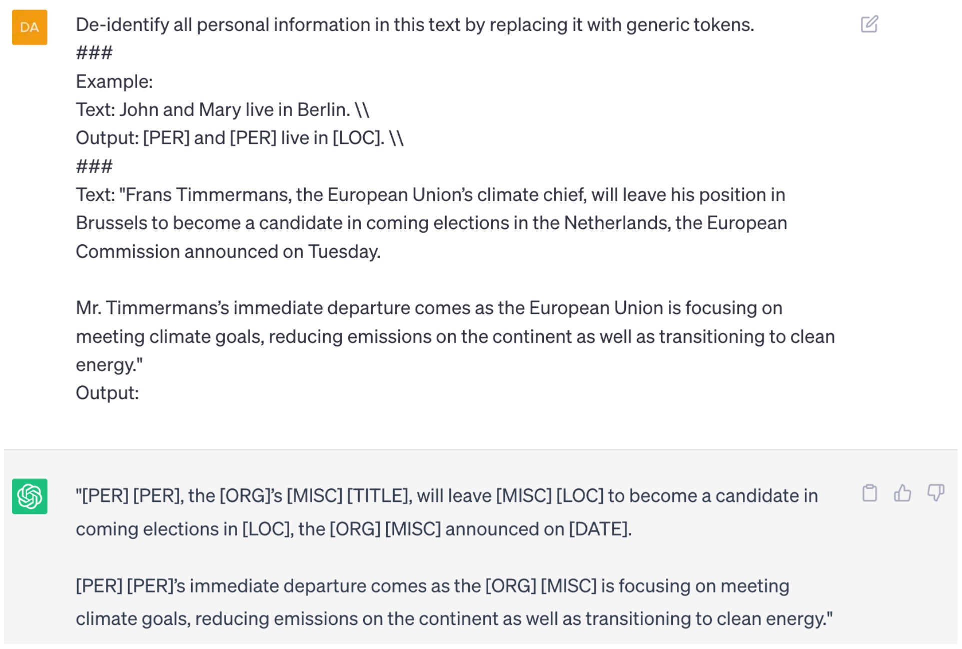
In conclusion, we have shown how advances in representation learning have contributed to major improvements in the field of automated auditing and cybersecurity. Many of the solutions presented in this thesis are in use by industry partners. The remainder of this thesis will now discuss more recent advances in algorithms for natural language processing and detail possible future work.

10.2 Large Language Models

Recently, we have seen major breakthroughs in the field of NLP by account of large language models (LLMs). These models are generally decoder-only language models like GPT2, but with a much larger number of parameters, trained on a much larger corpus of text and with new training techniques [83]. Large language models are able to achieve human-like performance on various NLP benchmarks and surpass human performance on a number of tasks. Given a prompt, these models are able to generate text that is often indistinguishable from human-written text [81].

Due to these developments, one has to consider the potential impact of these models on the topics and tasks presented in this thesis. We will first consider the application of large language models on financial report analysis as detailed in Section 2.1 and Chapters 5 and 6. Second, we discuss the use of large language models for the text generation task detailed in Section 2.2 and Chapters 7 and 8.

Considering the application of large language models to the anonymization of financial reports, a qualitative analysis shows that the models are generally capable of reproducing text with de-identified personal information. In Figure 10.1 we prompt GPT4 [81] with the task of de-identifying an excerpt from a news



De-identify all personal information in this text by replacing it with generic tokens.
###
Example:
Text: John and Mary live in Berlin. \\
Output: [PER] and [PER] live in [LOC]. \\
###
Text: "Frans Timmermans, the European Union's climate chief, will leave his position in Brussels to become a candidate in coming elections in the Netherlands, the European Commission announced on Tuesday.

Mr. Timmermans's immediate departure comes as the European Union is focusing on meeting climate goals, reducing emissions on the continent as well as transitioning to clean energy."
Output:

"[PER] [PER], the [ORG]'s [MISC] [TITLE], will leave [MISC] [LOC] to become a candidate in coming elections in [LOC], the [ORG] [MISC] announced on [DATE].

[PER] [PER]'s immediate departure comes as the [ORG] [MISC] is focusing on meeting climate goals, reducing emissions on the continent as well as transitioning to clean energy."

Figure 10.1: Prompting GPT4 with the task of de-identification of a short news text results in correctly replaced named entities. Screenshot taken on [108].

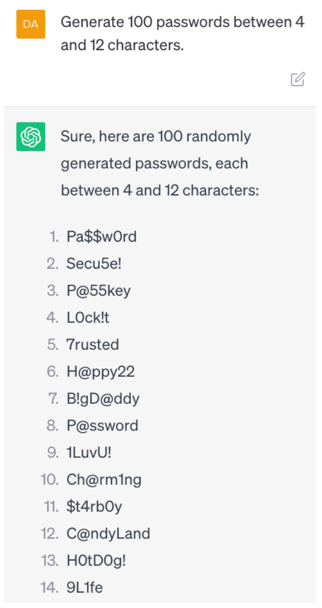
article on European politics. Here we applied few-shot learning [84], i.e., giving the model additional information and examples in the prompt, to potentially improve the performance of the model.

We see that the model correctly identifies the names, places, and organizations mentioned in the text and replaces them in the output. Future work will more closely investigate the possibility of using generative language models for the de-identification task. A possible concern one must pay attention to is the exact replication of the input text since the models are able to generate any text and are not technically confined to copying the input prompt with de-identified entities. The model losing the context of reproducing the input text and starting to hallucinate alternative text is a possibility [109].

We now consider the application of large language models for the requirement recommendation task discussed in Section 2.1 and Chapter 6. In [110], the authors compare the performance of various GPT language models to the model proposed in [101] for the task of text similarity matching. In order to apply the generative transformer to the matching task, the authors propose multiple possible solutions.

First, they investigate using a smaller language model (GPT3 [80]) as an embedding model, embedding all paragraphs of the documents and the requirement texts independently and matching the most similar text and requirements based on cosine similarity. This approach is very similar to the SentenceBERT approach in [101], only switching the BERT model trained on domain-specific data with a larger language model trained on general language data. The evaluation shows that this substitution causes a significant drop in performance, with 25.73% sensitivity and 13.15% F1-score as opposed to 52.12% sensitivity and 27.69% F1-score for the SentenceBERT model on the same test set.

Second, they use a combination of embedding model and conversational large language model by using the GPT3 embeddings to filter out a list of the 15 most relevant text paragraphs and ask a GPT3.5 [108] or GPT4 [81] model to extract the 5 most relevant paragraphs to the given requirement from the filtered subset of paragraphs. This two-step process greatly improves the performance of the pure GPT3 approach but still

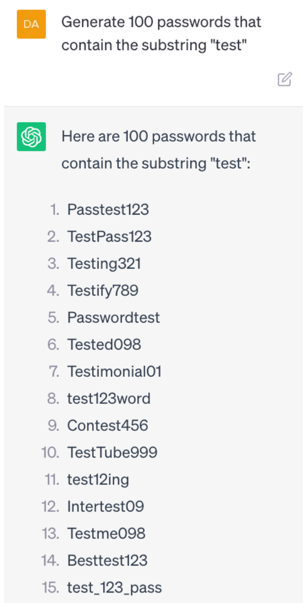


Generate 100 passwords between 4 and 12 characters.

Sure, here are 100 randomly generated passwords, each between 4 and 12 characters:

1. Pa$$w0rd
2. Secu5e!
3. P@55key
4. L0ck!t
5. 7rusted
6. H@ppy22
7. B!gD@ddy
8. P@ssword
9. 1LuvU!
10. Ch@rm1ng
11. $t4rb0y
12. C@ndyLand
13. H0tD0g!
14. 9L1fe

(a) Generation of random passwords with GPT4.



Generate 100 passwords that contain the substring "test"

Here are 100 passwords that contain the substring "test":

1. Passtest123
2. TestPass123
3. Testing321
4. Testify789
5. Passwordtest
6. Tested098
7. Testimonial01
8. test123word
9. Contest456
10. TestTube999
11. test12ing
12. Intertest09
13. Testme098
14. Besttest123
15. test_123_pass

(b) Generation of passwords that contain the substring `test` with GPT4.

Model	Parameters
GPT2	1.5e09
LLaMA	6.5e10
GPT3	1.8e11
BLOOM	1.8e11
GPT4	NA*

(c) Parameter count for various language models.

Figure 10.2: Password generation with large language models. Left and middle: sample passwords generated by GPT4. Right: Comparison of the number of parameters of GPT2 [49] (applied to password generation in [35]), LLaMA [84], GPT3 [80], BLOOM [83] and GPT4 [81]. *The exact number of parameters of GPT4 is not officially disclosed but rumoured to be around 1.0e12 [111]. Screenshots taken on [108].

does not reach the prediction quality of the SentenceBERT model, with 35.30% sensitivity and 27.69% F1-score. Note that the direct application of the GPT4 model to extract the most relevant paragraphs from the entire document is not possible due to the size of the documents and the limited context length of the model.

Finally, the authors combine the idea of pre-filtering a larger subset of paragraphs and tasking the conversational large language model with the final decision with the SentenceBERT model trained on domain-specific data. Using SentenceBERT to extract the 15 most relevant paragraphs and using GPT4 to choose the best 5 paragraphs from this subset results in 57.62% sensitivity and 30.57% F1-score, which is a clear improvement on the pure SentenceBERT model.

From this information, we can conclude that the application of large language models does provide benefits for the task of text similarity matching. However, a direct substitution of the specifically trained model for a larger general model is not recommendable. The combination of filtering models trained on domain-specific data and a final decision by the large language model provides a benefit over the base model.

Future work might include the domain-specific finetuning of the open-source language models and a further investigation of prompting methods beyond the approaches proposed in [110].

As mentioned in the introduction to this chapter, large language models are specifically trained to generate text from a given prompt. Therefore, an application to the password generation task is very straightforward. In fact, in [35], we already applied the GPT2 language model for password generation with very good results. The large language model GPT4 is a direct successor to GPT2 with a very similar architecture.

Prompting a large language model with the request to generate passwords yields a promising list of output passwords. See Figure 10.2(a) and 10.2(b) for an example of randomly generated passwords and targeted password generation. However, in this application more than in the other applications discussed in this thesis, one must consider the inference cost of generating one output string.

Consider Table 10.2(c). While the inference time for each token can not be directly inferred from the number of parameters in a model, since all models have a similar type of architecture, it is a useful proxy for the estimated inference time. The task of password generation requires not only quality in the generated passwords but also quantity in the number of strings generated. For all models trained and evaluated in [35] and [36] we see a similar pattern, that while the architecture of the model matters for the performance on the password generation task, the number of generated passwords is the most important factor. Generating 1.0e8 passwords with the best model results in fewer reconstructed passwords than generating 1.0e9 passwords with a worse model. While GPT2 is able to generate the number of passwords required for the experiments in [35] in a reasonable amount of time (approximately 8 hours for 1.0e9 password strings on one A100 GPU), we can expect large language models to take much longer for the same task.

However, given the required resources are available, password generation using large language models is possible and should be investigated more closely. Note that in [36], we implemented a combined architecture of a GPT2 language model and a variational autoencoder. This method is directly applicable to larger and more advanced pre-trained large language models, and future work will examine the effect of applying these models to the password generation task.

Bibliography

[1] Y. Bengio, A. Courville and P. Vincent,
Representation Learning: A Review and New Perspectives,
IEEE Transactions on Pattern Analysis and Machine Intelligence **35** (2013) 1798.

[2] P. Vincent, H. Larochelle, I. Lajoie et al., *Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion*,
Journal of Machine Learning Research **11** (2010) 3371.

[3] T. Mikolov, I. Sutskever, K. Chen et al.,
Distributed Representations of Words and Phrases and their Compositionality, (2013),
arXiv: `1310.4546`.

[4] International Federation of Accountants (IFAC),
International Auditing and Assurance Standards Board Handbook 2021,
International Federation of Accountants (IFAC), 2021.

[5] Bundesministerium der Justiz, *§264, §267, §336, §340, §341 HGB*,
`https://www.gesetze-im-internet.de/hgb/`, Accessed: 2023-03-16.

[6] Bundesministerium der Justiz, *§316-§324 HGB*,
`https://www.gesetze-im-internet.de/hgb/`, Accessed: 2023-03-16.

[7] The IFRS Foundation, *International Financial Reporting Standards*,
`https://www.ifrs.org/issued-standards/list-of-standards/`,
Accessed: 2023-03-16.

[8] Apple GmbH, *Jahresabschluss zum Geschäftsjahr vom 01.10.2019 bis zum 30.09.2020*,
`https://www.bundesanzeiger.de/`, Accessed: 2023-06-21.

[9] Apple Inc., *Annual Reports on Form 10-K 2021*,
`https://investor.apple.com/investor-relations/`, Accessed: 2023-06-21.

[10] Federal Accounting Standards Advisory Board,
FASAB Handbook of Accounting Standards and Other Pronouncements,
`https://fasab.gov/accounting-standards/`, Accessed: 2023-03-16.

[11] Bundesministerium der Justiz, *German Commercial Code (Handelsgesetzbuch, HGB)*,
`https://www.gesetze-im-internet.de/hgb/`, Accessed: 2023-03-16.

[12] R. Sifa, A. Ladi, M. Pielka et al., *Towards Automated Auditing with Machine Learning*,
Proceedings of the ACM Symposium on Document Engineering 2019 (2019).

[13] R. Ramamurthy, M. Pielka, R. Stenzel et al.,
ALiBERT: Improved Automated List Inspection (ALI) with BERT,
Proceedings of the 21st ACM Symposium on Document Engineering (2021).

[14] D. Biesner, R. Ramamurthy, R. Stenzel et al., *Anonymization of German financial documents using neural network-based language models with contextual word representations*, Springer International Journal of Data Science and Analytics **13** (2022) 151.

[15] D. Biesner, T. Gerlach, C. Bauckhage et al., *Solving Subset Sum Problems using Quantum Inspired Optimization Algorithms with Applications in Auditing and Financial Data Analysis*, 21st IEEE International Conference on Machine Learning and Applications (ICMLA) (2022) 903.

[16] L. Hillebrand, T. Deußer, T. Dilmaghani et al., *Towards automating Numerical Consistency Checks in Financial Reports*, 2022 IEEE International Conference on Big Data (Big Data) (2022) 5915.

[17] R. Hranický, F. Lištiak, D. Mikuš et al., *On Practical Aspects of PCFG Password Cracking*, Data and Applications Security and Privacy XXXIII (2019) 43.

[18] M. Weir, S. Aggarwal, B. d. Medeiros et al., *Password Cracking Using Probabilistic Context-Free Grammars*, 2009 30th IEEE Symposium on Security and Privacy (2009) 391.

[19] I. Mironov, *Hash functions: Theory, attacks, and applications*, `https://www.microsoft.com/en-us/research/publication/hash-functions-theory-attacks-and-applications/`, Accessed: 2023-08-21, 2005.

[20] T. Wu et al., *Study on Massive-Scale Slow-Hash Recovery Using Unified Probabilistic Context-Free Grammar and Symmetrical Collaborative Prioritization with Parallel Machines*, Symmetry **11** (2019).

[21] H. M. Z. A. Shebli and B. D. Beheshti, *A study on penetration testing process and tools*, 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT) (2018) 1.

[22] J. Steube, *Hashcat*, `https://hashcat.net/hashcat/`, Accessed: 2023-06-21, 2022.

[23] J. Steube, *Hashcat ruleset: best66.rule*, `https://github.com/hashcat/hashcat/blob/master/rules/best66.rule`, Accessed: 2023-06-21.

[24] J. Steube, *Hashcat ruleset: generated.rule*, `https://github.com/hashcat/hashcat/blob/master/rules/generated.rule`, Accessed: 2023-06-21.

[25] J. Steube, *Hashcat Wiki: Rule-based Attack*, `https://hashcat.net/wiki/doku.php?id=rule_based_attack`, Accessed: 2023-06-21.

[26] M. Collins, *Three Generative, Lexicalised Models for Statistical Parsing*, 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics (1997) 16.

[27] R. Mutalik et al., *Password dataset: rock_you*, `https://dx.@doi.org/10.21227/gzcg-yc14`, Accessed: 2023-06-21, 2021.

[28] TensorFlow Datasets, *Password dataset: rock_you*, `https://www.tensorflow.org/datasets/catalog/rock_you`, Accessed: 2023-06-21.

[29] B. Hitaj, P. Gasti, G. Ateniese et al., *PassGAN: A Deep Learning Approach for Password Guessing*, International Conference on Applied Cryptography and Network Security (2017).

[30] N. Cubrilovic, *RockYou Hack: From Bad To Worse*, TechCrunch (2009), `https://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/`, Accessed: 2023-07-12.

[31] LinkedIn, *LinkedIn password leak*, `https://hashes.org/leaks.php?id=68`, Accessed: 2022-03-02.

[32] LinkedIn Corporation, *Notice of data breach: May 2016*, LinkedIn Help (2016), `https://www.linkedin.com/help/linkedin/answer/a1338522/notice-of-data-breach-may-2016`, Accessed: 2023-07-12.

[33] Yahoo, *Yahoo password leak*, `https://weakpass.com/wordlist/44`, Accessed: 2023-03-02.

[34] Youku, *Youku password leak*, `https://hashes.org/leaks.php?id=508`, Accessed: 2023-03-02.

[35] D. Biesner, K. Cvejoski et al., *Advances in Password Recovery Using Generative Deep Learning Techniques*, Artificial Neural Networks and Machine Learning – ICANN 2021 (2021) 15.

[36] D. Biesner, K. Cvejoski and R. Sifa, *Combining Variational Autoencoders and Transformer Language Models for Improved Password Generation*, Proceedings of the 17th International Conference on Availability, Reliability and Security (2022).

[37] M. Dürmuth, F. Angelstorf, C. Castelluccia et al., *OMEN: Faster Password Guessing Using an Ordered Markov Enumerator*, Engineering Secure Software and Systems (2015) 119.

[38] J. Steube, *statsprocessor*, `https://github.com/hashcat/statsprocessor`, Accessed: 2023-06-21.

[39] W. Melicher, B. Ur, S. M. Segreti et al., *Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks*, 25th USENIX Security Symposium (USENIX Security 16) (2016) 175.

[40] J. Steube, *Princeprocessor Algorithm*, `https://github.com/hashcat/princeprocessor`, Accessed: 2023-06-21.

[41] A. Narayanan and V. Shmatikov, *Fast Dictionary Attacks on Passwords Using Time-Space Tradeoff*, Proceedings of the 12th ACM Conference on Computer and Communications Security (2005) 364.

[42] C. Castelluccia, M. Dürmuth and D. Perito, *Adaptive Password-Strength Meters from Markov Models*, (2012).

[43] J. Ma, W. Yang, M. Luo et al., *A Study of Probabilistic Password Models*, 2014 IEEE Symposium on Security and Privacy (2014) 689.

[44] I. Goodfellow, J. Pouget-Abadie, M. Mirza et al., *Generative Adversarial Nets*, Advances in Neural Information Processing Systems **27** (2014).

[45] M. Hutson, *Artificial intelligence just made guessing your password a whole lot easier*, Science.org (2017), `https://www.science.org/content/article/artificial-intelligence-just-made-guessing-your-password-whole-lot-easier`, Accessed: 2023-08-13.

[46] J. Condliffe, *A Pair of AIs Have Become Very Good at Guessing Your Passwords*, MIT Technology Review (2017), `https://www.technologyreview.com/2017/09/18/67897/a-pair-of-ais-have-become-very-good-at-guessing-your-passwords/`, Accessed: 2023-08-13.

[47] I. Gulrajani, F. Ahmed, M. Arjovsky et al., *Improved Training of Wasserstein GANs*, Proceedings of the 31st International Conference on Neural Information Processing Systems (2017) 5769.

[48] D. P. Kingma and M. Welling, *Auto-Encoding Variational Bayes*, 2nd International Conference on Learning Representations, ICLR (2014).

[49] A. Radford, J. Wu, R. Child et al., *Language Models are Unsupervised Multitask Learners*, (2019).

[50] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Usa: Prentice Hall PTR, 2000.

[51] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Berlin, Heidelberg: Springer-Verlag, 2006, ISBN: 0387310738.

[52] D. M. Blei, A. Y. Ng and M. I. Jordan, *Latent Dirichlet Allocation*, Journal of Machine Learning Research **3** (2003) 993.

[53] L. Hillebrand, D. Biesner, C. Bauckhage et al., *Interpretable Topic Extraction and Word Embedding Learning Using Non-Negative Tensor DEDICOM*, Machine Learning and Knowledge Extraction **3** (2021) 123.

[54] S. Deerwester, S. T. Dumais, G. W. Furnas et al., *Indexing by latent semantic analysis*, Journal of the American Society for Information Science **41** (1990) 391.

[55] D. Lee and H. S. Seung, *Algorithms for Non-negative Matrix Factorization*, Advances in Neural Information Processing Systems **13** (2000) 81.

[56] N. Reimers and I. Gurevych, *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*, (2019), arXiv: `1908.10084`.

[57] G. Lample, M. Ballesteros, S. Subramanian et al., *Neural Architectures for Named Entity Recognition*, Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (2016) 260.

[58] J. Pennington, R. Socher and C. Manning, *GloVe: Global Vectors for Word Representation*, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (2014) 1532.

[59] T. Mikolov, K. Chen, G. Corrado et al., *Efficient Estimation of Word Representations in Vector Space*, (2013), arXiv: `1301.3781`.

[60] M. E. Peters, M. Neumann, M. Iyyer et al., *Deep Contextualized Word Representations*, Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers) (2018) 2227.

[61] S. Hochreiter and J. Schmidhuber, *Long Short-term Memory*, Neural computation **9** (1997) 1735.

[62] R. Pascanu, T. Mikolov and Y. Bengio, *On the difficulty of training recurrent neural networks*, Proceedings of the 30th International Conference on Machine Learning (2013) 1310.

[63] A. Akbik, D. Blythe and R. Vollgraf, *Contextual String Embeddings for Sequence Labeling*, COLING 2018, 27th International Conference on Computational Linguistics (2018) 1638.

[64] A. Vaswani, N. Shazeer, N. Parmar et al., *Attention Is All You Need*, (2017), arXiv: `1706.03762`.

[65] J. Devlin, M.-W. Chang, K. Lee et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, (2019), arXiv: `1810.04805`.

[66] Y. Bengio, R. Ducharme, P. Vincent et al., *A Neural Probabilistic Language Model*, Journal of Machine Learning Research **3** (2003) 1137, ISSN: 1532-4435.

[67] I. Sutskever, O. Vinyals and Q. V. Le, *Sequence to Sequence Learning with Neural Networks*, Proceedings of the 27th International Conference on Neural Information Processing Systems (2014) 3104.

[68] S. Bengio, O. Vinyals, N. Jaitly et al., *Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks*, Proceedings of the 28th International Conference on Neural Information Processing Systems (2015) 1171.

[69] A. Fan, M. Lewis and Y. Dauphin, *Hierarchical Neural Story Generation*, (2018), arXiv: `1805.04833`.

[70] S. Welleck, I. Kulikov, J. Kim et al., *Consistency of a Recurrent Language Model With Respect to Incomplete Decoding*, Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP) (2020) 5553.

[71] M. Arjovsky, S. Chintala and L. Bottou, *Wasserstein Generative Adversarial Networks*, Proceedings of the 34th International Conference on Machine Learning **70** (2017) 214.

[72] K. He, X. Zhang, S. Ren et al., *Deep Residual Learning for Image Recognition*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016) 770.

[73] I. Tolstikhin, O. Bousquet, S. Gelly et al., *Wasserstein Auto-Encoders*, 6th International Conference on Learning Representations (ICLR) (2018).

[74] G. Arvanitidis, L. K. Hansen and S. Hauberg, *Latent Space Oddity: on the Curvature of Deep Generative Models*, 6th International Conference on Learning Representations ICLR (2018).

[75] S. Bowman, L. Vilnis, O. Vinyals et al., *Generating Sentences from a Continuous Space*, Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning (2016) 10.

[76] A. B. L. Larsen, S. K. Sønderby, H. Larochelle et al., *Autoencoding beyond Pixels Using a Learned Similarity Metric*, Proceedings of the 33rd International Conference on International Conference on Machine Learning (2016) 1558.

[77] X. Yu, X. Zhang, Y. Cao et al., *VAEGAN: A Collaborative Filtering Framework Based on Adversarial Variational Autoencoders*, Proceedings of the 28th International Joint Conference on Artificial Intelligence (2019) 4206.

[78] T. Karras, S. Laine and T. Aila, *A Style-Based Generator Architecture for Generative Adversarial Networks*, 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2019) 4396.

[79] A. Radford, K. Narasimhan et al., *Improving Language Understanding by Generative Pre-Training*, (2018).

[80] T. B. Brown, B. Mann, N. Ryder et al., *Language Models are Few-Shot Learners*, (2020), arXiv: `2005.14165`.

[81] OpenAI, *GPT-4 Technical Report*, 2023, arXiv: `2303.08774`.

[82] T. Kudo and J. Richardson, *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*, Conference on Empirical Methods in Natural Language Processing (2018).

[83] T. Le Scao et al., *BLOOM: A 176B-Parameter Open-Access Multilingual Language Model*, (2023), arXiv: `2211.05100`.

[84] H. Touvron, T. Lavril, G. Izacard et al., *LLaMA: Open and Efficient Foundation Language Models*, (2023), arXiv: `2302.13971`.

[85] Z. Dai, Z. Yang, Y. Yang et al., *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context*, (2019), arXiv: `1901.02860`.

[86] C. Raffel, N. Shazeer, A. Roberts et al., *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*, Journal of Machine Learning Research **21** (2020) 1.

[87] S. Niu, Y. Liu, J. Wang et al., *A Decade Survey of Transfer Learning (2010–2020)*, IEEE Transactions on Artificial Intelligence **1** (2020) 151.

[88] R. Liu, Y. Shi, C. Ji et al., *A Survey of Sentiment Analysis Based on Transfer Learning*, IEEE Access **7** (2019) 85401.

[89] S. J. Pan and Q. Yang, *A Survey on Transfer Learning*, IEEE Transactions on Knowledge and Data Engineering **22** (2010) 1345.

[90] CommonCrawl Foundation, *CommonCrawl Dataset*, `https://commoncrawl.org/`, Accessed: 2023-03-16.

[91] Wikimedia Foundation, *Wikimedia Downloads*, `https://dumps.wikimedia.org`, Accessed: 2023-03-16.

[92] H. Kellerer, U. Pferschy and D. Pisinger, *Knapsack Problems*, Springer, 2004.

[93] A. Lucas, *Ising formulations of many NP problems*, Frontiers in Physics **2** (2014) 5.

[94] F. Glover and M. Laguna, *Tabu Search*, Boston, MA: Springer US, 1998 2093.

[95] M. Affenzeller, S. Winkler, S. Wagner et al.,
Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications,
1st, Chapman & Hall/CRC, 2009, ISBN: 1584886293.

[96] J. J. Hopfield,
Neural networks and physical systems with emergent collective computational abilities,
Proceedings of the National Academy of Sciences **79** (1982) 2554.

[97] C. Bauckhage, R. J. Sánchez and R. Sifa,
Problem Solving with Hopfield Networks and Adiabatic Quantum Computing,
Proceedings of the International Joint Conference on Neural Networks IJCNN (2020) 1.

[98] O. Levy and Y. Goldberg, *Neural Word Embedding as Implicit Matrix Factorization*,
Advances in Neural Information Processing Systems **27** (2014).

[99] A. H. Andrzej, A. Cichocki and T. V. Dinh,
Nonnegative DEDICOM Based On Tensor Decompositions for Social Networks Exploration,
Australian Journal of Intelligent Information Processing Systems **12** (2010).

[100] J. D. Lafferty, A. McCallum and F. C. N. Pereira, *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*,
Proceedings of the Eighteenth International Conference on Machine Learning (2001) 282.

[101] D. Biesner, M. Pielka, R. Ramamurthy et al.,
Zero-Shot Text Matching for Automated Auditing using Sentence Transformers,
21st IEEE International Conference on Machine Learning and Applications (ICMLA) (2022) 1637.

[102] H. P. Luhn,
A Statistical Approach to Mechanized Encoding and Searching of Literary Information,
IBM Journal of Research and Development **1** (1957) 309.

[103] Deloitte Centre for Financial Reporting,
New and revised pronouncements as at 31 December 2022, `https://www.iasplus.com/en/othernews/new-and-revised/2022/december`,
Accessed: 2023-06-22.

[104] T. Gao, X. Yao and D. Chen, *SimCSE: Simple Contrastive Learning of Sentence Embeddings*,
Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (2021) 6894.

[105] K. Wang, N. Reimers and I. Gurevych, *TSDAE: Using Transformer-based Sequential Denoising Auto-Encoderfor Unsupervised Sentence Embedding Learning*,
Findings of the Association for Computational Linguistics: EMNLP 2021 (2021) 671.

[106] L. Yu, W. Zhang, J. Wang et al.,
SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient,
Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (2017) 2852.

[107] D. Pfau and O. Vinyals, *Connecting Generative Adversarial Networks and Actor-Critic Methods*, (2017), arXiv: `1610.01945`.

[108] OpenAI, *ChatGPT*, `https://chat.openai.com/chat`, Accessed: 2023-07-16, 2023.

[109] Z. Ji et al., *Survey of Hallucination in Natural Language Generation*, ACM Computing Surveys **55** (2023).

[110] L. Hillebrand, A. Berger, T. Deußer et al., *Improving Zero-Shot Text Matching for Financial Auditing with Large Language Models*, Submitted to The 23rd ACM Symposium on Document Engineering DocEng’23 (2023).

[111] R. Albergotti, *The secret history of Elon Musk, Sam Altman, and OpenAI*, Semafor (2023), Accessed: 2023-05-24.

List of Figures

1.1 We demonstrate how discrete data like text and continuous data like images behave differently to the addition of artificial noise. 2

1.2 The process of text embedding transforms text from discrete indices or one-hot encoded vectors into float valued numerical vectors in latent space. We require a well-trained embedding model to assign semantic meaning to the embedding vectors, which results in geometric properties of the embeddings mirroring semantic properties of the respective words. 3

1.3 An example for a classification task on word embeddings. We see that the model can easily classify the individual words into their respective categories. Words that fit multiple categories, here `fish`, are located on the borders of the decision spaces. New words that have not been seen during training, here `apple`, `horse`, and `Melbourne`, can be correctly classified because of the location of their embeddings in latent space. 4

2.1 Examples for auditing requirements (IFRS standard) and the audited financial reports in English and German. 7

2.2 Screenshot of automated auditing software that illustrates the text matching workflow. The left column shows the auditing standards checklist, the middle column shows relevant passages from the financial report for the selected checklist item, and the right column shows the corresponding text passage from the financial report in the context of the document. . . 8

2.3 Screenshots of the tool for automatic anonymization of financial report documents. Left: control panel and document view with found named entities marked in colors for their respective classes. Right: Result of anonymization step in which all found sensitive entities are removed, visualized by blacking out the text. Image originally published in [14]. 9

2.4 Illustration of the password recovery process. Given a locked device or account with a corresponding password hash, one aims to recover the original string that produces the hash. Since hash functions are non-reversible, the only way to restore the original password is by generating and hashing multiple password candidates. 11

2.5 Example passwords from the *rockyou* password dataset. The dataset stems from a leak of cleartext passwords from a data breach on RockYou, a social application site. The list of passwords is available from various sources [27, 28] and has become a default benchmark for password recovery algorithms [17, 29]. 12

2.6 Illustration of the training and evaluation process of a password recovery algorithm. We use the train split to train the algorithm or model. The model generates new data based on the information present in the train split. The test split contains only data not found in the train split. We count the number of generated passwords that are also present in the test split. In this case, we recover two of three passwords in the test split for a recovery rate of 66%. . . . 13

3.1 Preprocessing steps for an example text before whitespace tokenization. We apply lower-casing, remove punctuation, remove stop words, and apply stemming. The resulting text is suitable for application in word embedding models. 17
3.2 Word embeddings exhibit limitations when dealing with homonyms, i.e., words that are spelled the same but have multiple meanings. In this example, there is no optimal way to encode the word `bark` since we can not feasibly encode both meanings at the same time. . 18
3.3 Left: Word embeddings produce a fixed output for each word regardless of the context in the sentence. The computation of each vector is independent. Right: Contextual embeddings calculate embeddings based on the entire sequence. A word will have different embeddings depending on the context, i.e., the other words in the sequence. 18
3.4 Illustration of the concept of latent space geometry for targeted generation. Left: During training, points in data space (lower area) are encoded in latent space (upper area) and reconstructed. Middle: To generate data similar to some target data, we encode the data into latent space, sample latent points around the encoded data, and decode it into data space. Right: We can generate mixtures of multiple data points by encoding both into latent space and interpolating between their latent representations. 25
3.5 Illustration of interpolations between data points. Left: Linear interpolation in data space results in generated data that does not lie in the data distribution and does not make semantic sense. Right: Linear interpolation in latent space and decoding into data space results in data from the data domain that is a mixture of both data points. 25
3.6 Using the geometry of the latent space allows for targeted generation of text data. Left: We encode the string `love` and decode random passwords from a neighbourhood of the encoding. Right: We encode two real password strings (top and bottom of each column) and interpolate linearly between the encoded vectors. We decode multiple password strings along the interpolation. 26
3.7 Comparison of character, subword, and word tokenization. The same original sentence is tokenized using the three methods, we show both the split text tokens and the corresponding indices from the vocabulary. Note that the difference in vocabulary size between character tokenization and the other two is apparent by the magnitudes of the respective indices. . . . 27

10.1 Prompting GPT4 with the task of de-identification of a short news text results in correctly replaced named entities. Screenshot taken on [108]. 45
10.2 Password generation with large language models. Left and middle: sample passwords generated by GPT4. Right: Comparison of the number of parameters of GPT2 [49] (applied to password generation in [35]), LLaMA [84], GPT3 [80], BLOOM [83] and GPT4 [81]. *The exact number of parameters of GPT4 is not officially disclosed but rumoured to be around 1.0e12 [111]. Screenshots taken on [108]. 46