Erklärung

Hochschule Albstadt Sigmaringen

Fakultät für Informatik  
Studiengang Data Science

Poststraße 6, 72458 Albstadt

In Kooperation mit der   
BMW Financial Services Group  
BMW Bank GmbH

Erstprüfer: Prof. Dr. Goran Glavaš (Universität Mannheim)  
Zweitprüfer: Prof. Dr. Simone Paolo Ponzetto (Universität Mannheim)

06.01.2021

Master-Thesis

Building a Text Classification Model in a sparse and noisy Data Environment

Andreas Barth  
Matrikel-Nr. 88408

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst, keine  
anderen als die angegebenen Quellen und Hilfsmittel verwendet, sowie die aus fremden Quellen direkt oder indirekt übernommenen Stellen und Gedanken als solche kenntlich gemacht habe.

Diese Arbeit wurde noch keiner anderen Prüfungskommission in dieser oder einer ähnlichen Form vorgelegt. Sie wurde bisher auch nicht veröffentlicht.

Ich bin damit einverstanden, dass die vorliegende Arbeit von den Prüfern in elektronischer Form mit entsprechenden Softwaretools auf Plagiate geprüft werden kann.

Donauwörth, 06.01.2021

Andreas Barth

Table of Content

[1 Introduction 1](#_Toc60127229)

[1.1 Use case: BMW Financial Services 1](#_Toc60127230)

[1.2 Challenges 2](#_Toc60127231)

[1.3 Problem definition 3](#_Toc60127232)

[1.4 Outline 4](#_Toc60127233)

[2 Theoretical Background on Text Classification 6](#_Toc60127234)

[2.1 Vector representations of text data 6](#_Toc60127235)

[2.1.1 Weighted Words 6](#_Toc60127236)

[2.1.2 Word Embeddings 8](#_Toc60127237)

[2.2 Selected Machine Learning Approaches to Text Classification 13](#_Toc60127238)

[2.2.1 Lazy Learning: *k*-Nearest-Neighbor Classification (*k*-NN) 13](#_Toc60127239)

[2.2.2 Logistic Regression 14](#_Toc60127240)

[2.2.3 Support Vector Machines 15](#_Toc60127241)

[2.3 Selected Deep Learning Approaches to Text Classification 17](#_Toc60127242)

[2.3.1 Key Concepts applied in Deep Learning Models 18](#_Toc60127243)

[2.3.2 Recurrent Neural Nets 25](#_Toc60127244)

[2.3.3 Long Short Term Memory (LSTM) & Bi-LSTM 25](#_Toc60127245)

[2.3.4 CNN 25](#_Toc60127246)

[2.3.5 Transfer Learning using Attention: BERT 25](#_Toc60127247)

[3 Data & Preprocessing [6] 26](#_Toc60127248)

[3.1 Building training data from source systems 26](#_Toc60127249)

[3.2 Preprocessing & Anonymization 26](#_Toc60127250)

[3.3 EDA 26](#_Toc60127251)

[4 Methods and Experiments [8] 27](#_Toc60127252)

[4.1 Metrics 27](#_Toc60127253)

[4.2 Machine Learning approaches (Scikit Learn) 28](#_Toc60127254)

[4.2.1 KNN 28](#_Toc60127255)

[4.2.2 Linear Models 28](#_Toc60127256)

[4.2.3 Stochastic Gradient Descent 28](#_Toc60127257)

[4.3 Deep Learning approaches (TensorFlow) 28](#_Toc60127258)

[4.3.1 BiLSTM 28](#_Toc60127259)

[4.3.2 CNN 28](#_Toc60127260)

[4.3.3 BiLSTM with pretrained GLOVE 28](#_Toc60127261)

[4.3.4 CNN with pretrained GLOVE 28](#_Toc60127262)

[4.3.5 Multilingual BERT 28](#_Toc60127263)

[4.3.6 … 28](#_Toc60127264)

[5 Results & Discussion [18] 29](#_Toc60127265)

[5.1 Performance on key metrics 29](#_Toc60127266)

[5.2 Learnings 29](#_Toc60127267)

[5.3 Error Discussion 29](#_Toc60127268)

[5.4 Limitations 29](#_Toc60127269)

[5.5 Improvements 29](#_Toc60127270)

[5.6 Deployment 29](#_Toc60127271)

[6 Conclusions [4] 30](#_Toc60127272)

[List of Abbreviations 31](#_Toc60127273)

[List of Tables 32](#_Toc60127274)

[List of Figures 33](#_Toc60127275)

[Bibliography 34](#_Toc60127276)

[Appendix 35](#_Toc60127277)

# Introduction

Managing large volumes of inbound document traffic efficiently in a real-time manner is a competitive skill for companies facing millions of customers and stakeholders. People use various channels, such as email, postal services, facsimiles (fax) and in an increasing fashion posts and uploads on the company website. Customer service excellence requires a fast and highly accurate distribution of those documents to the relevant subject matter experts within the organization. To manage the volume and heterogeneity of the technical formats a document management system can be employed.

## Use case: BMW Financial Services

BMW Group Financial Services specializes in financing and leasing of automobiles and motorcycles for private, retail and commercial customers. It manages a portfolio of more than 4 million lease and credit financing contracts across 54 countries. Additional insurance and banking products complement a holistic customer experience around modern mobility solutions. BMW Bank an entity within BMW Group Financial Services serves a number of regional markets, predominantly Germany (any others?).

Throughout 2020 BMW Bank implemented a new document management system (DMS) to support daily operations in managing all inbound document traffic. This system seeks to process 10 million pages per year and strives for handling 80% of this volume in a fully automated fashion. The software chosen, a third party commercial solution, is customizable to existing needs and processes of the bank. A cross functional team fitted this system with a complex set of lookup logics and if-then rules to automatically classify inbound documents into document types (i.e. “Schadenschreiben”). Based on the assigned document type the system can extract further relevant information and terms for subsequent processing thereof (i.e. contract number, bank account). The number of topics covered by the inbound document traffic requires a set of more than 150 document types (classes) to categorize the expressed intents, before they are forwarded to the relevant inboxes.

Documents that fall short of automatic classification due to insufficient confidence in allocating a relevant document type are not automatically forwarded to the subsequent departments. They are picked up by a service team for manual inspection, classification and indexing (information extraction as described above). Figure 1‑1 illustrates the process described above.

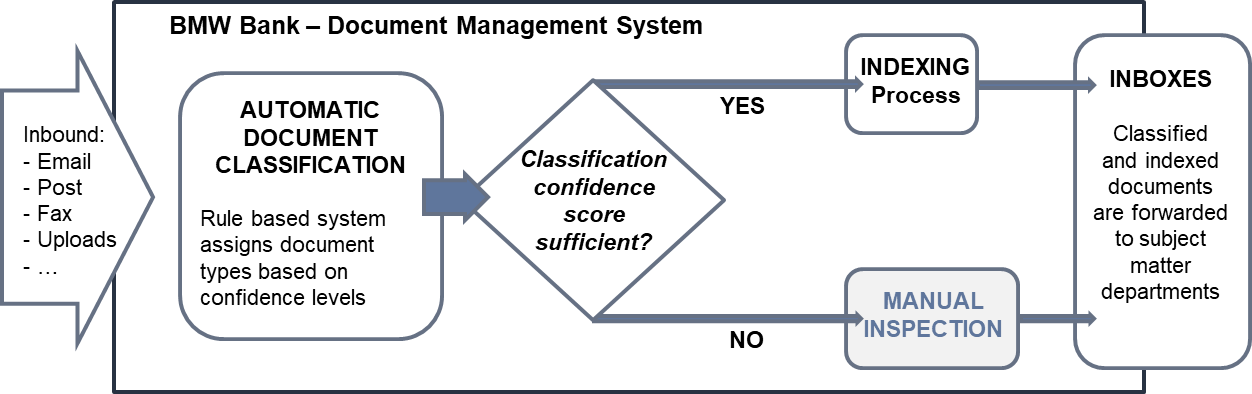


Figure 1‑1: Document flow in the BMW Bank document management system

Despite the planned regular adjustments and fine tuning of the underlying rule based logic it is expected at the time of writing that a substantial part of the documents (~25%) will continue to require manual intervention (meaning manual classification and indexing) to ensure seamless operations.

BMW Bank wants to explore in a proof of concept, if an additional machine or deep learning document classification model could mitigate the remaining need for manual classification of the inbound document traffic. Thus reducing operational expenses by a substantial factor.

## Challenges

The nature, volume, variety and heterogeneity of the received inputs gives rise to a number of challenges for an automatic document classifier:

The variance in technical formats in which information is provided spans from simple emails to OCR-scans of complex identification documents such as passports, driver licenses, vehicle registrations etc.. Thus introducing content with a big variety in length of text: From a short abstract (e.g. a short confirmation email) to several dozens of pages (e.g. legal correspondence or technical assessments).

Furthermore a substantial share of documents consist of uploaded forms, introducing textual patterns that are difficult to process in a sequential fashion.

Many documents make reference to previous (perhaps not presently included) communications in the past. Their content only makes sense in the context of previous messages, making it difficult to extract the full semantics of the document.

Letterheads, email signatures and boilerplates with legal or administrative copy introduce clutter and make it difficult to determine those relevant parts of text, which are most discriminative to determine the right intention.

The transactional nature of this data involves a big number of personal information such as names, contact details, addresses, banking data, vehicle details, contractual data, references etc.. Handling sensitive personal data requires particular efforts to comply with legislative and corporate data protection requirements. Anonymization techniques have to be applied during preprocessing phase, causing further dilution to the semantics of the content.

On top of the aforementioned the communication towards the bank is rich in specific language of the domains automotive, banking and insurance. Thus containing many terms and expressions that can lead to out-of-vocabulary situations when applying more general (German) language solutions to this domain specific classification task.

Limited volume of available training data

Given the sensitive nature of the documents data protection requirements restrict preserving and storing data over longer periods. However to employ supervised learning techniques it is imperative to gain labeled training data in sufficient volume and quality regarding the labels. A complex migration project like the one described has to be managed within tight budgets for resources. Employing additional manual labeling beyond the daily operations. Labeled training data therefore had to be extracted from the real productive system, once it went live. These factors led to a limited amount of labeled training data.

## Problem definition

Automatic document classification in this use case is the act of assigning documents their estimated intent, based on a predefined set of categories (document types). It is a special form of the general text classification problem. As the number of possible labels exceeds the binary case of 2 classes only, this task more specifically can be described as a multi-class text classification, where every document will be finally assigned to exactly one category out of a set with multiple (more than two) categories.

Provided with sufficient amounts of training data machine learning algorithms manage to learn patterns that allow for a mapping from training data to the respective labels. The classifier is deemed sufficient for the task if it generalizes well enough, thus predicting labels for new, unseen data with a low error rate.

Formally:   
In a given training set of labeled documents every document belongs to a set. Each label of a document maps to a predefined set of categories .

Document classification is the induction of algorithms that are capable of accurately classifying unseen documents, when sufficiently trained on (Bekkerman, 2004).

Building a text classification system requires a number of strategic choices:

Documents need encoding to transform unstructured text into a structured, numerical representation for a machine or deep learning algorithm to work the math. Many approaches for transforming text have been developed. They can be roughly categorized into either weighted word techniques or word embedding methods.

Choosing the right classification algorithm for the task at hand is a vital factor for success. Potential candidates range from the more traditional machine learning models to recent state-of-the art deep learning approaches with impressive results on a number of downstream tasks.

Those advances in the field of Natural Language Processing (NLP) are driven by complex neural network architectures and transfer learning approaches. However given the aforementioned challenges in our sparse and noisy data environment the question arises if those recipes provide better results when applied to the task at hand. Especially the limitation on volume of available training data requires consideration for the right approach:

*“In the small data regime, depending on how the features are hand-engineered, traditional algorithms may or may not do better. For example, if you have 20 training examples, it might not matter much whether you use logistic regression or a neural network; […] But if you have 1 million examples, I would favor the neural network”* (Ng, 2018, p. 12).

## Outline

This thesis explores several approaches to solving the document categorization problem defined herein and concludes on a deployment recommendation for the proof of concept that BMW Bank is expecting within the scope of this project.

Chapter 2 will discuss key concepts in machine and deep learning, providing the theoretical background for the experiments applied later on. Particular focus will be on the process of encoding the textual data and the discussion of different classification algorithms and deep learning models.

Chapter 3 will provide more insights on the available data environment, the data extraction necessary to build up a reasonable training corpus and the preprocessing steps to ready the data for the different classification techniques applied.

Chapter 4 details on the applied techniques in the experiments, covering some of the more traditional machine learning approaches but also applying selected deep learning methods and transfer learning ideas.

Chapter 5 follows with a discussion of the results in detail, looks at particular strengths and weaknesses of the different approaches and providse thoughts on how to improve those results.

Chapter 6 concludes this thesis with the learnings, provides a suggestion for the “best choice” given the task at hand and weighing-in the applicable criteria for a deployment decision.

# Theoretical Background on Text Classification

To solve for the described document classification task a number of different approaches and experiments are applied. The design of the experiments and their respective results will be explained in chapter 4 and 5 subsequently, before a deployment recommendation on a certain model can be expressed.

This chapter discusses briefly some of the theoretical fundament of the applied concepts. It starts with the illustration of different techniques to encode text into a numerical format and then moves on to cover different classification approaches. On an abstract level the methods applied can be grouped into the more traditional machine learning approaches and into the more recent developments in the field of neural networks, the deep learning approaches. In the context of this project several classifiers from both groups were applied and tested. Hence those selected representatives of both groups are discussed to provide a basic understanding of the theoretical foundation to these concepts.

## Vector representations of text data

Machine learning algorithms usually apply their math on a fixed sized numerical input of data. Documents however present themselves as unstructured data sets of text. For tasks in the NLP domain this requires transformation of every textual input into a vector of digits or floating point values. Mathematically these vectors correspond to points in multi-dimensional spaces (Raaijmakers, 2021est., p. 56 (v8)).

Different strategies exist for encoding text. On a higher abstraction level they can be categorized either by the simple approaches using weighted word schemes or the more sophisticated strategies learning dense representational vectors from the input data in the form of word embedding (Kowsare, et al., 2019, p. 2).

### Weighted Words

A straightforward technique is to leverage word distributions within a given document.

The Bag of Words (BoW) technique does exactly this. First, a vocabulary containing every unique word in the training corpus is generated. Building on that a term document matrix is created with rows representing documents and columns representing words in [[1]](#footnote-1). For a classification task with thousands of documents and vocabulary sizes in the tens of thousands the vector space grows very high dimensional. To observe computational cost and feasibility commonly gets reduced to the most frequent words. It’s a hyper parameter that can be tuned for better results during model training.

One hot encoding is the simplest form of BoW. Every word occurrence in the document vector has a binary representation of one or zero, indicating whether the word is contained in this document or not.

Term frequency (TF) count can be used instead of the plain Boolean representation to enrich the information value. However, very common words like propositions or articles are found in almost every document. Thus their frequency count provides limited or no value to distinguish documents from another.

“It turns out, however, that simple frequency isn’t the best measure of association between words. One problem is that raw frequency is very skewed and not very discriminative” (Jurafsky & Martin, 2019, p. 105)

To put more emphasis on words with more distinguishable power, a mathematical weighting scheme combining TF with inverse document frequency (IDF) is applied (Sparck Jones, 1972).

with  
and  
Equation 2‑1: Term Frequency-Inverse Document Frequency

Each word is represented by the product of its term frequency and its inverse document frequency (Jurafsky & Martin, 2019, p. 106). The IDF-term of this equation varies inversely with the document frequency, the number of documents a term is assigned in a corpus of total documents (Salton & Buckley, 1987, p. 7). This way commonly found words in the corpus that are not as helpful (like propositions) get their high frequency heavily discounted by multiplication with the term, pushing their total weight towards zero.

Despite a reduction of the vocabulary size BoW generally results in a high dimensional but sparse vector model. Every vector accounts only for a small fraction of non-zero values because the majority of words in the vocabulary is not found in a document.

The BoW approach carries a number of limitations for the document classification task. A classification algorithm processing a BoW input has to learn parameters for every dimension in the feature space. With tens of thousands of dimensions this can result in a highly (over) parametrized classifier with a tendency to over fit training data and poor generalization to new and unseen data (Jurafsky & Martin, 2019, p. 111).

Another more descriptive but heavy limitation of BoW models is their incapability to neither capture the sequential order of words in a document nor similarities between words, because in a BoW vector space every word of the vocabulary is individually represented atomically by its own index.

“For example, [the] words “airplane”, “aeroplane”, “plane”, and “aircraft” are often used in the same context. However, the vectors corresponding to these words are orthogonal in the bag-of-words model. This issue presents a serious problem to understanding sentences within the model” (Kowsare, et al., 2019, p. 7).

Those shortcomings led to the development of Word Embedding approaches.

### Word Embeddings

“You shall know a word by the company it keeps” (Firth, 1957).

Simple count-based methods cannot describe the semantic a sentence or a document. A fundamental principle is that specific words co-occurring with each other more often can describe the same concept in its semantic.

Word embeddings are constructed on this insight. They are a buildup of short (the length of hundreds of dimensions) and dense vectors with the majority of values being non-zero real numbers. *“It turns out that dense vectors work better in every NLP task than sparse vectors”* (Jurafsky & Martin, 2019, p. 110)*.*

For the document classification problem given in this project, two very popular concepts are selected: This chapter introduces the fundamental theory of the Word2Vec and GloVe method, their general benefits and weaknesses for the given task.

**“Word to vector”** (Word2Vec) was presented in 2013 (Mikolov, et al., 2013). It makes use of a shallow neural network with two hidden layers. The intuition of Word2Vec is that the semantic of words can be captured by the contextual words found in their neighborhood. Thus words that share the same contexts are more likely to represent a specific semantic concept than other words. Referring to the earlier example, the expressions “aircraft” and “plane” are commonly surrounded by the same context words (e.g. pilot or wing), this indicating a semantic similarity between the two.

Word2Vec implements this concept in two variations: the **continuous bag-of-words (CBOW)** or the **continuous skip-gram** architecture (Mikolov, et al., 2013, p. 4). Both leverage a shallow neural network that is trained on a huge corpus (i.e. an entire Wikipedia dump), with the objective to optimize a binary classification task. Both techniques are maximizing the likelihood that a target word and some context words in a given window size[[2]](#footnote-2) around the target word co-occur. Given the contextual input words CBOW predicts the best target word and vice versa skip-gram predicts the contextual words based on a presented input word. Both algorithms can be categorized as unsupervised learners, thus they don’t require explicitly labeled data for the training. The existing words within any given real text sequence provide the learning signal.

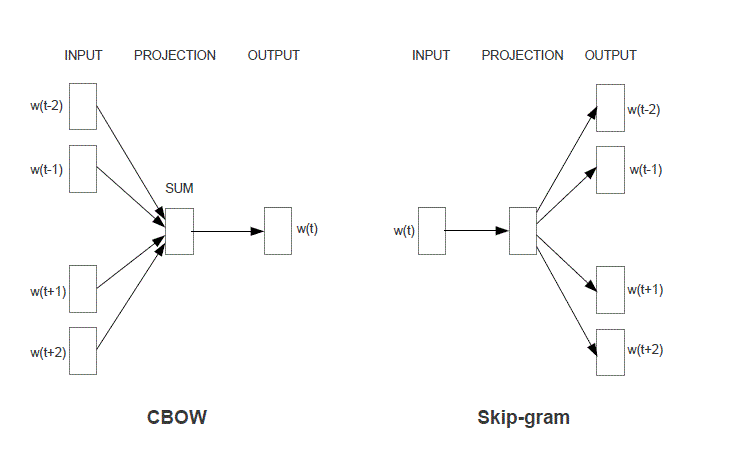
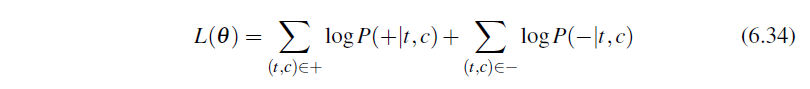


Figure 2‑1: The CBOW and Skip-gram architecture  
“The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word” (Mikolov, et al., 2013, p. 5).

To illustrate the learning process, we focus on the skip-gram architecture: Based on an initial set of embeddings for every word, the algorithm is optimizing for the maximum similarity between a target word t and its context word c (positive examples) and the minimum similarity between t and a non-context word randomly sampled from the corpus (the negative samples). The objective function of this learning task across the entire training set can be formally expressed as:



Equation 2‑2: Skip-gram objective function (Jurafsky & Martin, 2019, p. 114)

By leveraging stochastic gradient descent during training phase the weight vector theta is adjusted to optimize the log-likelihood of the positive sample word pairs (+|t,c) plus the log-likelihood of the negative examples, the randomly generated context-target word pairs (-|t,c).

While the result of the classification is not of particular use, the learned weights in theta are the interesting result. By iteratively optimizing on the given objective function the learning algorithm is pushing the weights of contextually similar words closer to each other and the weights of dissimilar words further apart from each other. The final result is a dense *d*-dimensional representation for every word in the vocabulary. Again *d* is an architectural parameter that can be experimentally chosen depending on computational considerations.

Word2Vec produces dimensions of meaning, when sufficiently trained on large enough training corpuses. Pre trained embedding matrices of Word2Vec for numerous languages exist and can be used as an input layer to encode textual training input for a deep neural network on a downstream task like the document classification described in this project. Compared to a deep neural net that would start training on a complete randomly initialized embedding matrix the usage of a pre trained embedding should give a reasonable advantage in training time and classification result.

While Word2Vec delivers on the much desired semantic learning it only takes local contexts into account: “… they poorly utilize the statistics of the corpus since they train on separate local context windows instead of on global co-occurrence counts” (Pennington, et al., 2014, p. 1532).

Because it’s streaming texts sequentially during training it does not differentiate between word sets that are commonly used in conjunction (like in “it is” or “should have”) but possess only little descriptive power and words appearing in conjunction, because they describe a specific semantic (e.g. “aircraft” and “pilot”).

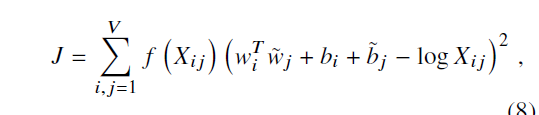
Amongst other considerations this drove the development of **Global Vectors** (GloVe) (Pennington, et al., 2014). It’s combining the strengths of the Word2Vec model with those of matrix factorization techniques encoding statistical information about an entire corpus.

The fundamental underpinning of GloVe is that co-occurrence ratios between two words in a context are strongly connected to meaning. For that GloVe establishes a co-occurrence matrix where every value denotes the number of times a word *j* is presented in the context of a word *i.* With the entire vocabulary *V* represented in the probability for every co-occurrence of *i* and *j* can be calculated by simply dividing with the number of times *i* appears in the entire corpus. With all possible values available it’s easy to determine the ratios of them with another probe word *k*.

The intuition of calculating with denoting the probability of *k* in proximity of *i* and the probability of *k* in context of *j* is that a large ratio above 1 indicates a much stronger relationship of *k* with *i* than *k* with *j*, while a ratio below 1 indicates a weaker relationship of *k* with *i* than *k* with *j*. Consequently a ratio close to 1 indicates an equally strong relationship of *k* to both words *i* and *j*, or no relationship to both terms.

“Compared to the raw probabilities, the ratio is better able to distinguish relevant words […] from irrelevant words […] and it is also better able to discriminate between the two relevant words. The above argument suggests that the appropriate starting point for word vector learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves” (Pennington, et al., 2014, p. 1534).

Introducing a number of constraints, mathematical conveniences[[3]](#footnote-3) and transformations plus two additional bias terms lead to the final model, presented as a least squares problem with an additional weighting function included to ensure some important desired behaviours of the objective function *J* described further below:



Equation 2‑3: Loss function of the GloVe model (Pennington, et al., 2014, p. 1535)

1. = 0. In the event of two terms not co-occurring (), is defined to negative infinity. In that case the weighing function must take on the value zero, to prevent an undesired explosion of values inside the objective function by setting the term inside the summation to zero.
2. Very rare and also very frequent co-occurrences should not be over weighted, so should be chosen to be non-decreasing and in the latter case be relatively small if takes on large values.

As with the Word2Vec technique GloVe optimizes the loss function *J* during training phase and by doing so produces vectors of weights that can be extracted. These vectors can then be used as a first text vectorization layer in a neural network designed to solve a downstream task like document classification.

The authors of GloVe proof that their technique produces better embeddings much faster than CBOW or Skip-gram. However since it is common practice in many real-world applications to use available pre trained versions (build with very large general corpuses and potent computational resources) training time might not be the key advantage from a practical point of view.

Arguably GloVe is seen as a more principled approach than Word2Vec as it combines a mathematically quite similar derivation with the potency of global corpus statistics. Hence there is an expectation of delivering better results when applied in real word scenarios.

In the light a practical application however both algorithms share a strong limitation. Both techniques are not equipped for dealing with out-of-vocabulary situations. That is textual input containing words not known during production time of the embedding. Those out-of-vocabulary terms will not contribute to the classification task as they add a zero-vector to the process. The ratio of the amount of zero to total vectors in the input matrix (after the training text is converted with the embedding vectors support) needs careful observation and might prevent a meaningful application of those techniques when this ratio is unfavorably high.

This chapter described several approaches to encode text data into a numerical representation using either a BoW approach or leveraging word embedding techniques. Two popular embedding models were introduced to illustrate the general idea of word embedding techniques. Within this project the TF-IDF encoding was applied in conjunction with more traditional classification algorithms. Other experiments used a German version of Word2Vec and Skip-Gram embeddings in combination with different neural network architectures. Embeddings can be trained on-the-fly by a neural network purposed to a specific classification task, this was also tested within the experiments of this project.

All experiments are fully described in chapter 4 and their results will be compared and discussed in chapter 5 of this thesis.

Chapter 2 will continue with discussing several algorithms of the machine learning and the deep learning field.

## Selected Machine Learning Approaches to Text Classification

### Lazy Learning: *k*-Nearest-Neighbor Classification (*k*-NN)

The *k*-NN algorithm makes use of the principle idea, that data points of the same class are represented in feature space within close proximity to each other. *k*-NN is a lazy learner, it simply stores instances of the training data. For a new data point to be classified it then produces a ranking with the distances of this data point to all existing training instances. The top *k* nearest neighbors based on these distances are chosen and the new label is assigned according to the known labels of those *k* neighbors. If different categories are present in this selection, then the category most often accounted for is the one to be assigned. The specific distance of each neighbor to the new data point can additionally be taken into account by applying a weighting scheme, giving a higher contribution to records with lower distance within the majority vote.

*k*-NN requires a data set of stored records, a distance measure that computes the distance between records and a preset value *k*, determining how many neighboring data points should be evaluated for the majority vote determining the class label.

The choice of the distance function used and the k number of neighbors impact the overall accuracy of the classification task. If *k* is chosen too small the classification is sensitive to noise and outliers within the training data. Increasing *k* can mitigate this but will lead to more indecisive decision bounds. It’s common practice to experiment with different distance functions and settings for *k* to arrive at the best solution.

*k*-NN requires the entire training data to be stored. A large number of training instances and a large multi-dimensional feature space, like a TF-IDF matrix, will make the computation heavy. The distance for each data point needs to be calculated across the feature space every time a new instance arrives for classification. This may lead to unfavorable response times during inference if those demands are not adequately met with computational resources.

Despite this practical limitation in a productive environment, *k*-NN was applied as one of multiple base line models within this project.

### Logistic Regression

The Logistic Regression Classifier is a binary classifier that computes the probability of a data instance belonging to a specific (positive) class or not. If the estimated probability exceeds a certain threshold (generally defaulted to 0.5) that data point is predicted to be of the positive class and vice versa if the probability is below the threshold.

As a generalized linear model Logistic Regression computes a dot product of input features and weights and transforms this weighted sum with the logistic function to present a posterior probability (aka a likelihood) as result of the calculation (Bishop, 2006, pp. 205, 197).

Equation 2‑4: Posterior probability of a class and the logistic (sigmoid) function

To train a Logistic Regression model the weight vector *w* containing the weights for each single feature needs to be adjusted to minimize an error function. This function can be calculated using the negative logarithm of the likelihood for each prediction. For every label *t* with and the error function is calculating the cross entropy *E* (*w*):

Equation 2‑5: Cross entropy error function (Bishop, 2006, p. 206)

When the predicted probabilities diverge from the actual labels the cross entropy (aka log loss) increases. The form of the log loss used in equation Equation 2‑5 presents a mathematical convenience to combine the two loss functions of both cases (tn = 0 or tn= 1) inside one formula. By multiplying each loss function with tn and (1-tn) one of the two loss terms always cancels out and the negated summation across the entire training set of length N produces the total error.

There is no closed form to calculate the optimal value for *w*. Thus the algorithm needs to adjust *w* incrementally in multiple iterations to find the optimum with the smallest error. Since the cost function is convex though it is guaranteed that an optimizing algorithm (like gradient descent) will find the best solution provided given enough training time and a sufficient learning rate (Géron, 2019, p. 144).

Per definition above Logistic Regression is a strict binary classifier. To deal with this limitation when presented a multi-class problem (*N* classes with *N* > 2) the chosen software package needs to adapt a special strategy to use Logistic Regression. Under the hood a “one versus rest” (OVR) strategy can be applied: For every class *n*  a separate Logistic Regression classifier is trained to predict whether a record belongs to *n* or not. Thus decomposing the multi-class task to an ensemble of *N* binary classifiers. The final prediction of a record is determined by the classifier giving back the highest decision score (Géron, 2019, p. 100).

### Support Vector Machines

Another supervised classification technique that is well suited for text classification problems is the Support Vector Machine (SVM). Like Logistic Regression it’s a model designed for binary classification tasks but again OVR and other strategies can be applied to overcome this binary constraint if in need for a multi-class labeling solution.

For a linearly separable data set “There may of course exist many such solutions that separate the classes exactly. […] The support vector machine approaches this problem through the concept of the margin, which is defined to be the smallest distance between the decision boundary and any of the samples, …” (Bishop, 2006, p. 326). In m-dimensional feature space a SVM is fitting a (m-1)-dimensional hyperplane to the training data, so that a perpendicular distance between the decision boundary and the nearest data points is achieved. Thus the decision function is defined only by a subset of the data points, aka the support vectors. They define the maximum margin possible for the hyperplane that is fit. This property of a maximum margin classifier is particular useful as it may reduce the generalization error when the model is applied to new and unseen data later.

A hard margin SVM constructs a margin with no allowance for any errors. Every data point classified needs to be on the correct side of the decision boundary. For a given set of labeled training data denoting the positive (+1) and negative classes (-1) the decision function needs to take on values  
> 1 for all instances positive and < 1 for all instances labeled negative. This can be formed as a one line constraint and finding the maximum margin decision boundary can then be expressed as a optimization problem to find the optimal weight vector *w* and bias term *b* (Géron, 2019, p. 166):

Equation 2‑6: Hard margin linear SVM classifier objective

A hard margin classifier is very sensitive to outliers in the data and many real word problems present data that just isn’t strictly linearly separable. Thus a hard margin classifier will not solve this problem sufficiently. Those settings require a variation of the constraints given above: There should be some misclassifications allowed as long as the remaining data points can be separated optimally. This intuition is met by the soft margin classification: Some data points are allowed to be on the “wrong side” of the decision boundary but are penalized with increasing size of distance to that boundary. To relax the constraints imposed by the hard margin classifier a set of slack variables is introduced with. Every data point is assigned to one slack variable, serving as a penalty for margin violations and scaling this effect proportionally to the distance from the margin (Bishop, 2006, p. 332):

* For an instance on the correct side of the margin the slack variable takes on the value zero.
* An instance on the wrong side of the margin gets assigned a slack value  
   if it still respects the decision boundary, meaning this instance is correctly classified but violates the maximum margin set by the classifier.
* For a misclassified the slack value is set to

With the introduction of the slack variables the optimization problem now contains two conflicting objectives: On the one hand the maximum margin still needs to be found, whereas the margin violations expressed by the sum over the slack variables should be minimized. To manage this trade off a new parameter is introduced into the equation to scale the penalizing effect of the slack variables.

Including these conceptual additions the optimization problem for the soft margin classifier can now be expressed as (Géron, 2019, p. 167):

+

Equation 2‑7: Soft margin linear SVM classifier objective

To solve efficiently for non-linear data situations another powerful feature of SVMs is the ability to apply the “kernel trick”. It describes the process of applying kernel functions to map the non-linear data input to higher dimensions with the objective to find better conditions for a linear separation in a higher-dimensional feature space. Mathematically this implies exchanging every dot product in the optimization problem with a non-linear kernel function.

“SVMs are very universal learners. In their basic form, SVMs learn [a] linear threshold function. Nevertheless, by a simple "plug-in" of an appropriate kernel function, they can be used to learn polynomial classifiers, radial basic function (RBF) networks, and three-layer sigmoid neural nets” (Joachims, 1998, p. 138).

For the document classification domain SVMs are well equipped to deal with the high dimensional input space resulting from the usage of weighted word schemes like TF-IDF. *“One remarkable property of SVMs is that their ability to learn can be independent of the dimensionality of the feature space. SVMs measure the complexity of hypotheses based on the margin with which they separate the data, not the number of features. This means that we can generalize even in the presence of very many features …”* (Joachims, 1998, p. 139).

One remarkable property of SVMs is that their ability to learn can be independent

## Selected Deep Learning Approaches to Text Classification

The classifiers presented so far - all the same developed some decades ago - have been proofing their strengths in many practical applications and domains of NLP over time. But the advent of deep learning techniques in recent years has drawn a lot of attention: *“Deep Learning has emerged in the last decade as the vehicle of the latest wave in AI. Results have consistently redefined the state-of-the-art for a plethora of data analysis tasks in a variety of domains. For an increasing amount of deep learning algorithms, better-than-human (human-parity or superhuman) performance has been reported …”* (Raaijmakers, 2021est.).

The rise of deep learning has spawned a number of interesting model architectures particularly suited for the NLP domain. Amongst other properties these models convince with advantages to process sequentially structured data like texts in documents. *“Theoretical results […] suggest that in order to learn the kind of complicated functions that can represent high-level abstractions (e.g. in vision, language, and other AI-level tasks), one may need deep architectures”* (Glorot & Bengio, 2010, p. 249).

The remainder of this chapter 2 will introduce some of those deep learning ideas that were tested for this document classification task together with their theoretical foundation.

### Key Concepts applied in Deep Learning Models

There are some universal concepts applied to the design and training of a deep neural net (DNN): An adequate loss function and an optimizer needs to be chosen, and to ensure a more robust network a couple of strategies can be applied.

#### Loss Function for Multi-Class Classification

An artificial neural net (ANN) produces an estimation, which is to be compared against the actual real value of a training instance in a supervised learning context. The result is computed by a loss function. Training a neural network is an optimization problem with regards to this function. *“Given neural network parameters, find the value of that minimizes the cost function J”* (Goodfellow, 2015). Choosing the adequate loss function is a central decision for a project and numerous candidates exist to cater for different needs.

A multi-class classification task can be expressed by the comparison of two distributions with each other. A distribution of *K* multiple given target labels and a distribution of K multiple classes estimated by the classifier. For multi-class classification tasks records are assigned mutually exclusive to labels. Thus the label distribution of one instance can be expressed by a sparse vector with targets , in a *1-of K* coding scheme.

Equation 2‑5 that was discussed for the binary case (*K*=2) of Logistic Regression can easily be extended to a multi-class task with *k = 1,…, K* and *K*>2. The neural networks final output layer needs to produce its result in the vectorized format that the labels are presented in. *“If we have K separate binary classifications to perform, then we can use a network having K outputs each of which has a logistic sigmoid activation function”* (Bishop, 2006, p. 235). The network will produce a *K*-sized vector with probability scores (the output of each sigmoid) , one for each *k*-class. To ensure the mutual exclusivity of the targets the softmax function transforms those likelihoods into a probability distribution by normalizing them so that the network’s outputs add up to 1.

Equation 2‑8: Softmax Activation Function (Bishop, 2006, pp. 235-236)

Equation 2‑9: Categorical Cross Entropy (Bishop, 2006, pp. 235-236)

#### The loss for the multi-class case can then be expressed by the categorical cross entropy computed between the distribution of actual targets (1-of-K encoding) and the predicted probability distribution of classes (K probabilities). The optimization problem is to find the optimal weight vector by minimizing the categorical cross entropy. That’s the objective of the training process.

#### Optimizers

With the loss function the backpropagation algorithm can compute the partial derivatives with regard to the loss function, compute the Gradient (pointing uphill) and incrementally update in the opposite direction of the Gradient (going downhill). The scale of this increment is determined through a learning rate . The process is formally expressed:

Equation 2‑10: Gradient Descent

The learning rate is an important hyper parameter as it drives the speed of the training process. The size of is the scaling factor of the incremental update at every training step. While it will generally accelerate training a too big might lead to overshooting, in that the global minimum of the cost function is not found because the algorithm overshoots when in the area of the best solution and perhaps even deteriorates away. A too small on the other hand would probably work better for that matter but would conversely prolong the network training substantially and perhaps even run the risk of not converging because it keeps optimizing a local minimum. Captured within a local minimum it might not be able to escape and search for the global minimum.

Batch Gradient Descent makes use of the entire training set. However there is variations like Stochastic Gradient Descent (randomly picking one instance) and Mini-Batch Gradient Descent (using smaller mini batches for every update step) that bring improvements in terms of training speed and memory management (Hinton, et al., 2012, pp. 4-6). Changing Gradient Descent for a faster optimizer can bring significant improvements. Numerous optimization algorithms have been developed to select from. For this project the very popular RMSProp algorithm has been chosen developed by Geoffrey Hinton and Tijmen Tieleman (Hinton, et al., 2012, p. 29). RMSProp computes the gradient update by considering the velocity of recent gradient upgrades. It is able to adapt the learning rate better to the current environment of the optimization problem. The algorithm takes two computational steps: For each weight RMSProp computes a moving average *s* of the squared gradient using the squared gradients multiplied with a decaying factor .[[4]](#footnote-4) The gradient of the current time step is then divided by the root mean squared average:[[5]](#footnote-5)

Equation 2‑11: RMSProp algorithm (Géron, 2019, p. 356)

#### Strategies to prevent vanishing Gradients

A key principle of every ANN is that the output of every neuron is channeled through a non-linear activation function propagating the signal forward to the next layer of neurons. The output of the final layer is matched with the given target values and a loss function calculates the estimation error. This loss is distributed backwards through the network and used to compute an incremental upgrade of all the ANNs weights contained in . This process is managed by the backpropagation algorithm. It computes a Gradient Descent with regard to the loss function, upgrades the weights accordingly with a small increment before the next prediction iteration going forward can start. This forward-backward circling process continues until the ANN converges.

Unfortunately this process is prone to a misbehavior called the “vanishing gradients” phenomena. For a deep network with many layers, neurons and even more weights in between them backpropagation translates into a complex process of calculating a vast number of chained partial derivatives. Computing derivatives implies using the chain rule of calculus[[6]](#footnote-6). Most of the activation functions used back in the day generally produce small output values around zero. With that determining the gradients of a deep neural net means multiplying lots of small numbers. Because of this the gradients in a deep net with many layers can get smaller and smaller until they vanish. The error signal traveling backwards through the net reduces exponentially. Thus preventing an effective upgrade of for the first lower layers. But with no further change induced the network cannot learn anymore. The training stops long before a satisfying result is accomplished.

Effective strategies to counter this behavior were developed in 2010 (Glorot & Bengio, 2010). It was found that a poor choice of the activation function strongly contributed to the problem. Inspired by the natural design of biological neurons, it was very common to use the logistic function at that time. Figure 2‑2 illustrates the unfavorable behavior of the sigmoid in this context: The output saturates with large valued inputs (negative or positive) to either 0 or 1. Computing the derivate in these areas results in values of or near zero.

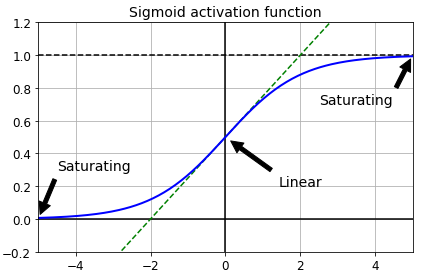


Figure 2‑2: Logistic activation function saturation (Géron, 2019, p. 333)

“Thus when backpropagation kicks in it has virtually no gradient to propagate back thought the network; and what little gradient exists keeps getting diluted as backpropagation progresses down through the top layers, so there is really nothing left for the lower layers” (Géron, 2019, p. 332).

This insight kicked of the search for a number of activation functions suited better for this purpose. The Rectified Linear Unit (ReLU) is an alternative activation function commonly used because it’s not saturating for positive values and given it’s simplicity is fast to compute:

Equation 2‑12: ReLU and Leaky ReLU (Géron, 2019, p. 335)

“This activation function is the default activation function recommended for use with most feedforward neural networks” (Goodfellow, et al., 2016).

But ReLU still is handicapped with not being differentiable at zero and in that it produces a zero valued derivative for negative inputs. In some scenarios when the majority of the neurons in the ANN produces negative valued outputs this results in gradients filled with zeros. A behavior described as “dying ReLUs”. It can be countered with a variation called the leaky ReLU. For a value of it multiplies *z* with a small scaling factor, typically sized 0.01 (Géron, 2019, p. 335).

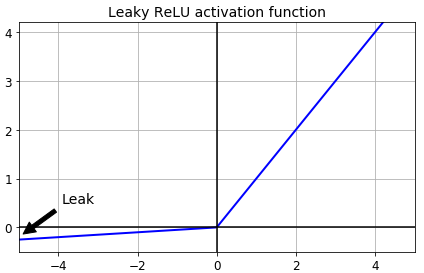


Figure 2‑3: Leaky Rectified Linear Unit function (Géron, 2019, p. 336)

In their 2010 paper Xavier Glorot and Yoshua Bengio also noted that the initialization scheme of the network played an important contribution to the saturating gradients (Glorot & Bengio, 2010). Networks then were commonly initialized with a normal distribution. They noticed that the variance of the outputs of every layer should be equal to the inputs’ variance of that layer and when backpropagation kicks in the same should hold for the reverse direction so that the input and output variance of the gradients is kept equal for every layer. To approximate a solution they developed the “Glorot initialization” scheme for the number of inputs (called: *fan-in*) and the number of outputs (*fan-out*) to be used in conjunction with the logistic function:

Equation 2‑13: Glorot Initialization (Géron, 2019, p. 334)

The insight that special emphasis should be given to the initialization is considered to be one of the break-through findings leading to the success of deep learning methods. Other initialization schemes were consequently developed like an initialization scheme optimal for usage in conjunction with ReLUs called “He-Initialization” after its author (He, et al., 2015). However the search for methods to improve the training process of DNNs didn’t stop there.

#### Batch Normalization

In a 2015 paper Ioffe and Szegedy proposed a technique called Batch Normalization. “Training Deep Neural Networks is complicated by the fact that the distribution of each layer’s inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as internal covariate shift, and address the problem by normalizing layer inputs” (Ioffe & Szegedy, 2015, p. 1). In a DNN where one layer takes its input from lower layers and again channels its output as input to higher layers applying Batch Normalization smoothens the various input distributions and prevents the unfavorable high variance between inputs and outputs of a layer.

The key idea of Batch Normalization is to integrate a normalization step into the network architecture, ideally adding one Batch Normalization layer before every activation of the network. This way every single mini-batch[[7]](#footnote-7) of inputs gets first zero centered with a unit variance and then linearly transformed by a scale and a shift parameter vector . The model learns these two vectors for every layer[[8]](#footnote-8) so that the ideal rescaling and shifting can be applied to its inputs. For a mini-batch of inputs the algorithm computes the mini-batch mean and variance and uses them to normalize every into :

is linearly transformed using the scaling vector , resulting in the rescaled and shifted output vector

Equation 2‑14: Batch Normalization Algorithm (Ioffe & Szegedy, 2015, p. 3)

The authors demonstrated that Batch Normalization can improve DNNs significantly: The vanishing gradients phenomena is kept at hand, the network is less sensitive to the choice of activation function and because of a more stable and robust behavior larger learning rates can be applied, thus accelerating the training process considerably.

#### Regularization strategies

A DNN with tens of thousands or even millions of parameters is prone for overfitting the training data. *“If the relationship between the input and the correct output is complicated and the network has enough hidden units to model it accurately, there will typically be many different settings of the weights that can model the training set almost perfectly, especially if there is only a limited amount of labeled training data”* (Hinton, et al., 2012). Given this freedom the model learns residual noise as if it was a structural property of the data. For that matter it fails to predict with sufficient accuracy when it is exposed to new and unseen data at inference time. The model fails to generalize. To prevent overfitting and reduce the test error regularization needs to be applied.

A number of strategies derived from traditional machine learning theory such as and regularization (a penalty term is added to the loss function) are likewise applicable to DNNs (Goodfellow, et al., 2016, pp. 227-231).

The batch normalization algorithm (see above) in addition to its many described positive contributions also imposes regularization to a DNN, thus reducing the need for other regularization techniques (Ioffe & Szegedy, 2015, p. 5).

A commonly applied strategy is to apply dropouts. A drop-out implemented into the architecture of the DNN will randomly switch off neurons of the connected layer during model training. Every neuron (excluding the output neurons) in the respective layer will be temporarily dropped with a probability *p* in one training step. With a sufficient dropout rate *p* assigned the network learns to generalize because it cannot rely on the entire predictive power of all the neurons. Given the huge number of possible permutations with neurons being switched on or off, adding dropout can be seen as sampling one version from an exponentially large set of different thinner neural networks at every training step. *“Random dropout makes it possible to train a huge number of different networks in a reasonable time. There is almost certainly a different network for each presentation of each training case but all of these networks share the same weights for the hidden units that are present”* (Hinton, et al., 2012, p. 2).

Of course the neurons are only dropped during training of the net. *“At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.”* (Srivastava, et al., 2014, p. 1929)

Another effective strategy to prevent overfitting is early stopping. During training the learned model is applied to a hold-out validation set of unseen data. The key idea is that a low validation set error translates to a low test set error. During training if the chosen metric (i.e. loss or accuracy) on the validation set improves a copy of the model weights is saved. With a defined tolerance and a specified number of training steps (batches or epochs) the training process can terminate when the validation metric does not improve beyond the tolerance for the specified patience period. The model will then roll-back to the last stored weight parametrization, the best version so far (Goodfellow, et al., 2016, p. 243).

### Recurrent Neural Nets

Humans understand text because they can put the words they read in context with information given to them before. Likewise it’s easy to complement missing words within a known sequence (like the *ABC* taught in elementary school). For real comprehension processing the sequential property of text is fundamental. Regular feed-forward neural networks can’t do that.

Recurrent Neural Nets (RNNs) can share their parameters and information across different parts of the model. This is a powerful property allowing for processing different texts of arbitrary length and generalize well across them. *“A traditional fully connected feedforward network would have separate parameters for each input feature, so it would need to learn all the rules of the language separately at each position in the sentence. By comparison, a recurrent neural network shares the same weights across several time steps”* (Goodfellow, et al., 2016, p. 368).

RNNs have recurrent loops that channel information from an earlier time step back into the network. They operate on a sequence of vectors , with *t* indexing the sequential position of a time step.[[9]](#footnote-10) The RNN processes the data *x* provided with a function *f* and parameters contained in transforming it into a hidden state *h* and propagates *h* forward through time, were it is again combined with further sequential information of the next time step. This can be formally expressed by:

 (Goodfellow, 2015, p. 370)

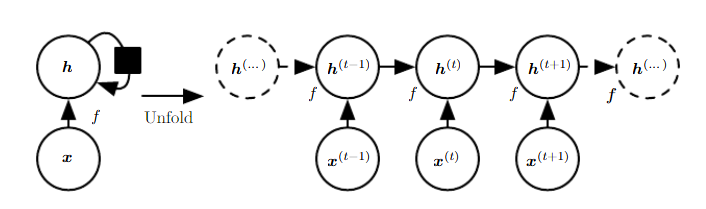


Figure 2‑4: An RNN with no outputs

Figure 2-4 describes this process on the left side with a circuit diagram with the black rectangle indicating the passage of several time steps. On the right this RNN is represented in its unfolded graph with each time step process explicitly illustrated and being connected to the time step before and thereafter.

It’s easy to see the text processing power of RNNs by mapping each time step to a word of an input sentence, abstract or a document. The RNN depicted above (figure 2-4) is missing an important feature as it shows no outputs. RNNs can be designed very flexible according to the task they are purposed to. The architectural core elements of inputs, hidden states and outputs can be combined in many ways to cater for different learning tasks:

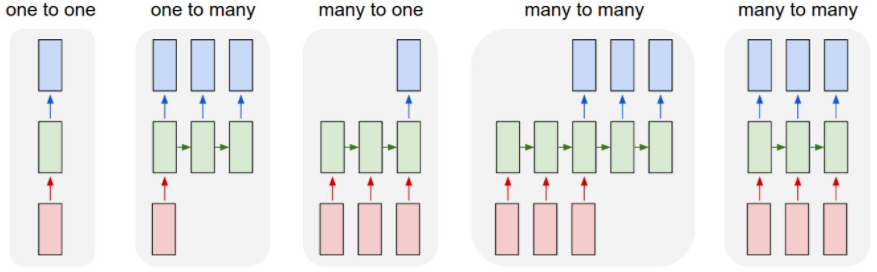
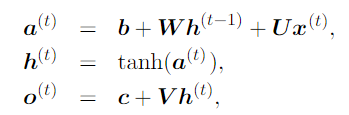


Figure 2‑5: Examples of different RNN architectures for different purposes (Karpathy, 2015)

For the document classification task at hand an RNN is supposed to have multiple inputs (the text), share connections between its hidden states and produce an output for every time step. Thus the many -to-many model shown in figure 2-5 abstracts best the architecture needed for this problem. This RNN produces an output vector *o* (illustrated with the blue boxes), which is channeled further into other layers of a final network architecture including a dense network with a softmax activation to produce the distribution of probabilities over the classes.[[10]](#footnote-11) contains the output for every time step *t* that is computed from the hidden states . Three weight matrices keep track of all the parameters used in the network:

* U for connections between inputs and hidden states
* W for connections between hidden states and neighboring hidden states
* V for connections between hidden states and outputs

With two additional bias vectors *b* and *c* the forward propagation of input through the network can then be expressed by:



Equation 2‑15: Forward propagation in a multi-output RNN (Goodfellow, et al., 2016, p. 374)

The strength of RNNs derives from two powerful properties: A distribution of hidden states, capable of preserving information through time and non-linear functions (like the hyperbolic tangent function (tanh) in equation xx) that allow to model highly complex vector spaces of input data (Hinton, et al., 2012).

Training the RNN means learning the best parametrization of the weight matrices U, W and V to minimize the loss function. Like with any other neural net this is done by backpropagation. *“Computing the gradient through a recurrent neural network is straightforward. […] No specialized algorithms are necessary”* (Goodfellow, et al., 2016, p. 379). One important difference in an RNN however is that the flow of computed derivatives needs to mirror the sequential chaining of the hidden states and outputs. For this reason that process is called Backpropagation through Time.

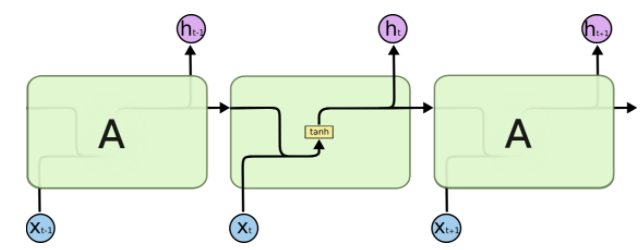
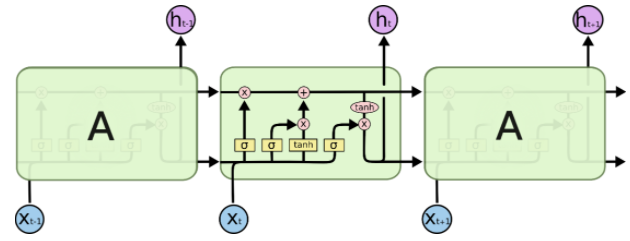
But RNNs suffer two major limitations that ask for additional strategies, particularly when trying to process long (text) sequences (i.e. 100 time steps) like in the document classification task: The complex chaining of hidden states over many time steps makes RNNs especially deep, thus being susceptive for the vanishing or exploding gradients problem, resulting in an unfavorably instable training process.

The second major limitation is the short-term memory problem. As every hidden state gets constantly rewritten when the information is propagated through the network, it’s difficult to keep information present over many time steps. *“After a while, the RNN’s state contains virtually no trace of its first inputs”* (Géron, 2019, p. 514). RNNs suffer from a short term memory limitation.

### Long Short-Term Memory (LSTM)

In a 1997 paper the idea of a Long Short-Term Memory architecture was introduced (Hochreiter & Schmidhuber, 1997). The authors showed that LSTM behave much more robust and provide a sufficient solution for the limitations of RNNs on long sequences.

The key idea of LSTM is to incorporate memory cells that learn what part of a previously stored context is not any longer needed, thus can be removed and what part of new incoming information is added to this long short-term memory, so that it is available in later time steps. Figure 2-6 illustrates (left panel) the schematic of a regular RNN with a hyperbolic tangent (tanh) activation function. In the right panel it visualizes the concept of a memory cell in a LSTM network augmenting what is a hidden state in a regular RNN.

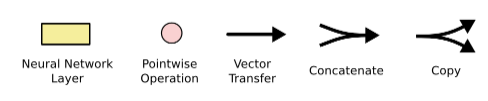


Figure 2‑6: Schematic information flow of a regular RNN (left panel)  
and (in the right panel) an LSTM memory cell. (Olah, 2015)

Memory cells store and update long-term information based on the input. Thus the new input information is not only introducing new data but also setting three corresponding gating units (sigmoid activation functions). They are the forget gate , the input gate and the output gate . Based on the combined input of and the previously computed hidden state the gates are set to be ...

* *open*: The sigmoid outputs 1, indicating to keep everything (zero filter)
* *closed*: The sigmoid outputs 0, signaling to remove everything (full filter)
* *in-between*: If the sigmoid outputs a value >0 and <1 it will take over some parts of the vector it is applied to (partial filter).

With that the inner workings in the memory cell can be formally described (Jurafsky & Martin, 2019, pp. 184-185) and illustrated (Olah, 2015) as follows:

1. **Setting the gates**: At time step *t* each gate is computed by a respective sigmoid that is applied to the sum of two matrix multiplications: The weight matrices , or are multiplied with (the short term state from the previous time step) and the respective weight matrices , or are multiplied with the current input . In the following steps the respective output vectors of the gates (all valued between 0 and 1) will be pointwise multiplied with the vectors they need to take control of. Thus masking the information of the gated vectors as described above. The three gates are indicated with in the illustrations below:

|  |  |
| --- | --- |
|  |  |

1. **Removing context from the long term context vector**: The forget gate is elementwise multiplied with the context vector to remove context that is not needed any longer, resulting in the modified context vector :

|  |  |
| --- | --- |
|  |  |

1. **Computing the new current cell context:** the context to be added currently to the memory cell is computed by a hyperbolic tangent (tanh) activation function applied to the sum of the multiplication of the input and the hidden state with their respective weight matrices and With elementwise multiplication of and the input gate the filter is applied selecting what parts of the new context remain in the current context

|  |  |
| --- | --- |
|  |  |

1. **Updating the long term context vector :** The current cell context is added to the context vector :

|  |  |
| --- | --- |
|  |  |

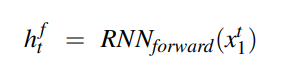
1. **Writing the current hidden state** **:** The updated context vector is transformed by another hyperbolic tangent (tanh) activation function and elementwise multiplied with the output gate to determine the current hidden state that will be passed on the next cell, as well to the output vector collecting the results of all memory cells.

|  |  |
| --- | --- |
|  |  |

With all appropriate weights set, an LSTM takes the hidden state and the long term context vector from the last time step and combines it with the current input vector. The previous hidden state and the current input determine the state of the gates in conjunction with the respective weights. Then based on the gates the long term context is modified and passed to the next layer and the current hidden state is computed and passed on to the next layer as well as to an output layer (if the architecture is designed to accept outputs from every time step).

### Bidirectional Recurrent Neural Networks

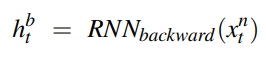
In a conventional RNN the hidden state incorporates only the information that the RNN has seen and functionally processed from the front (denoted with index *f*) of the sequence up to the current time step *t*.



Equation 2‑16: Hidden state computed from front context  
in a conventional RNN (Jurafsky & Martin, 2019, p. 182)

However determining semantic from a sequence of words (text) can be very limited if only context from previous time steps is available. Very often the true meaning of a text can only be comprehended with all context taken into account. In a document classification task the full context is available at training and usually also at inference time. For this it might be a better approach not only to use the previous context but also the context that follows in later time steps.

This can be achieved if another layer of the RNN gets to learn from the inverse sequence. With that the two vectors and can be concatenated to , representing the full context at time step *t*:





Equation 2‑17: Hidden state computed from front and back context  
in a conventional RNN (Jurafsky & Martin, 2019, p. 182)

### Convolutional Neural Nets

Convolutional Neural Networks (CNNs) contributed strongly to the domain of image classification. The key principal of CNNs is the training of filters that can identify immanent structures in the data and extract them to features for subsequent layers of a network. *“A convolutional neural network is designed to identify indicative local predictors in a large structure, and combine them to produce a fixed size vector representation of the structure, capturing these local aspects that are most informative for the prediction task at hand”* (Goldberg, 2015, p. 42).

In a document classification task it’s unlikely for all words and sentences to carry the same descriptive power. Some word co-occurrences, some collocations or some sentences might proof to be very distinguishing for a category, so there should be emphasis to find exactly those expressions. This is the driving insight for applying CNNs to a text classification task:

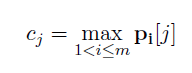
In the NLP domain convolutions are computed over a 1-dimensional sequential input, typically a sentence, an abstract or an entire document. With that the convolution can be formally described as follows (Goldberg, 2015, p. 43):

In a sequence of *n* words x = , every is mapped to a corresponding dimensional word embedding A *k*-sized window is moving *m* times[[11]](#footnote-12) across the sequence of words, so that the series represents every window with . The vector is fed to a linear transformation and element-wise applied non-linear activation function *g()*. Thus producing *m* times a resulting vector for every window with a dimension of . :



Equation 2‑18:

A CNN layer is typically fed to a subsequent forward network layer, expecting every input to be of same size and dimension. Naturally input texts can vary in length of input, thus violating this requirement. As a remedy a max pooling[[12]](#footnote-13) layer is applied producing a single vector , representing every dimension *j* of the *m* vectors with its respective max value:



Equation 2‑19: Max pooling (Goldberg, 2015, p. 43)

A convolutional layer doesn’t apply a filter just one time but many times. Thus the resulting vector *c*, representing the input sequence is featuring as many dimensions, as number of filters are applied. With each filter operating upon an independent weight matrix, bias and activation function the idea is to learn focused filters for different immanent features within an input sequence. *“Ideally, each dimension will ‘specialize’ in a particular sort of predictors, and max operation will pick on the most important predictor of each type”* (Goldberg, 2015, p. 43). During training backpropagation will upgrade the gradients through the network and tune the corresponding parameters to minimize the loss function. With sufficient training the final CNN layer consists of a number of expert filters trained on the respective classification task.

Figure xy shows exemplarily a 1d convolution and pooling performed on an example sentence, using a window size of 3, an assumed 2-dimensional embedding vector for every word and the application of 3 filters, resulting in a final 3 dimensional pooling vector:

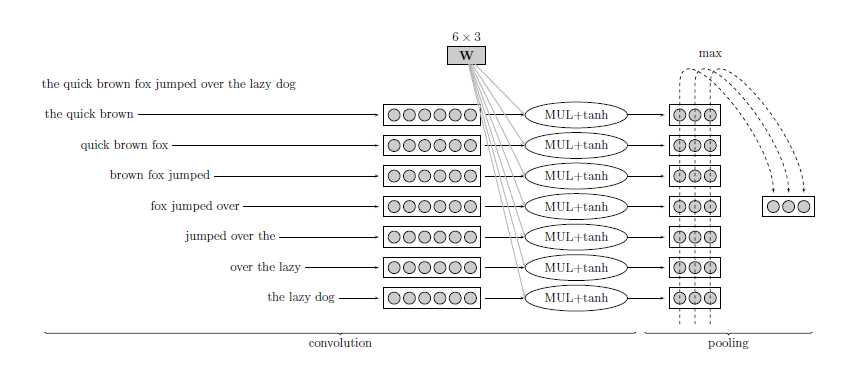


Figure 2‑7: Example of 1d-convolution and max-pooling with 3 filters (Goldberg, 2015, p. 44).

### Self-Attention & Transformer Networks

For capturing the meaning of a text it’s beneficial to account for the order of the sequential input and the context of words when processing the text. While proximity of words is often helpful to decode a semantical concept, it doesn’t work always, especially not in cases where the context is spread across an entire sentence or an abstract of a document with multiple sentences co-referring to each other. Very long sentences or abstracts of text still are a challenging task for an LSTM: The sequential processing is a fundamental constraint precluding parallel processing and the vanishing gradient problem still might occur: Training LSTMs on very big text corpora is a cumbersome task, when the gradients have to travel long distances.

A widely recognized paper on the task of neural machine translation (Bahdanau & Yoshua, 2016) introduced the concept of Attention: A global rather than a local or sequential approach to identify semantically relevant structures in a given text. The key idea of attention is to encode textual input together with contextual focus information. Each token carries additional information to determine which other tokens it might depend upon or is referring to. This gave way for a team of Google researchers to implement a variation of attention into a network architecture they called a Transformer network. Because their design of an encoder decoder model broke away from the so far predominant usage of RNNs, CNNs and LSTMs they claimed: “Attention Is All You Need” (Vaswani, et al., 2017).

The Transformer architecture is depicted in Figure. The model is designed for a sequence-to-sequence task (language translation) with an encoder and decoder unit. As this project focuses on document classification the decoder (the right part of the architecture) can be neglected for the further discussion. For a document classification it would make sense to use the encoder (the left part) only. The output of the encoder is not fed into the decoder module but it is fed to a separate network to classify the documents (not shown here).

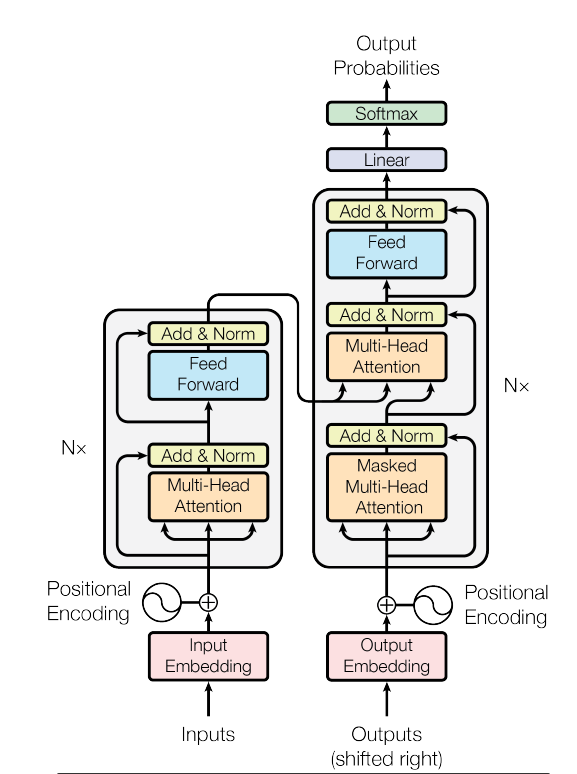


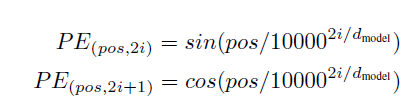
Figure 2‑8: The Transformer – model architecture (Vaswani, et al., 2017, p. 3)

The encoder is made up of *N* stacked identical layers[[13]](#footnote-14). Within each layer the data traverses two consecutive sub-layers a Multi-Head Attention block and a Feed Forward block. Two residual connections skipping the two blocks are also implemented. They support the gradient signal from thinning out during backpropagation. The Positional Encodings and the Multi-Head Attention are two particular ideas that differentiate this architecture even more from other models and will be discussed in more detail.

#### Positional Encodings

Because RNNs and LSTMs process input sequentially one embedding at a time step, the order of the tokens is maintained and there is no need for a dedicated positional encoding. Not so with the transformer network: Like other models it’s vectorizing input text into an embedding space. Every token is represented by a 512-dimensional embedding vector. But unlike other models it’s taking in the entire input sequence at once. There is a huge benefit to this: With the usage of modern Graphical Processing Units (GPU) and their ability to parallelize computing operations this reduces the training time significantly.

But this advantage comes with a price: With no sequential information encoded any longer, the order of the tokens is lost and cannot support the training.The authors thought of a smart way of bringing back the order of the inputs. Every embedding vector is combined with a positional embedding vector of same size encoding the unique position of the respective word into a unique numerical representation. To compute the positional encoding they used wave frequencies: Embedding values with an even positional index are transformed using the sine function and values with an odd positional index are likewise transformed by the cosine function. Both functions take the positional index of the token (*pos*), the embedding dimension *i* and the size of the embedding vector for arguments to calculate a positional embedding vector.



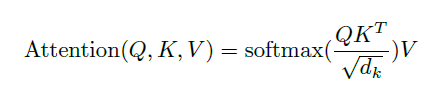
Equation 2‑20: Positional Encodings in the Transformers Model (Vaswani, et al., 2017, p. 6)

With that for every embedding vector there is a respective positional embedding vector. Being of the same size the two vectors are added. The following layers simply use one vector for each token but this vector now is carrying the initial embeddings plus the positional information.

#### Scaled Dot-Product Attention

The Positional Encodings are fed into a Multi-Head Attention layer. It can be decomposed into several single attention units called heads. Each single head performs what the authors describe as a scaled dot product attention: The incoming embedding vectors are channeled into three matrices: the query matrix *Q*, the key matrix *K* and the value matrix *V*. These three matrices are of dimensionality and .

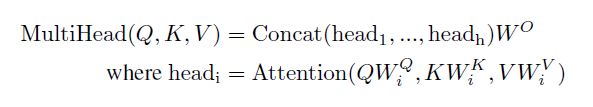
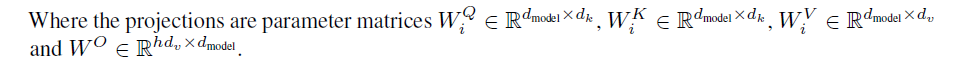
The Self-Attention block makes use of the cosine similarity concept: The cosine of the angle between two vectors is approaching 1 the more similar they are and -1 with decreasing similarity. Hence the dot product between the query matrix Q and the keys K is computed to identify the keys most similar to the queries. The resulting matrix is scaled down[[14]](#footnote-15) to keep numerical computation stable and then normalized with a softmax function to produce weights that add up to 1. This weight matrix contains for each token the attention that should be given to every other token. These weights are fed into another matrix multiplication together with the values V. Thus transforming V into a new representation with the attention scores encoded:



Equation 2‑21: Scaled Dot-Product Attention (Vaswani, et al., 2017, p. 4)

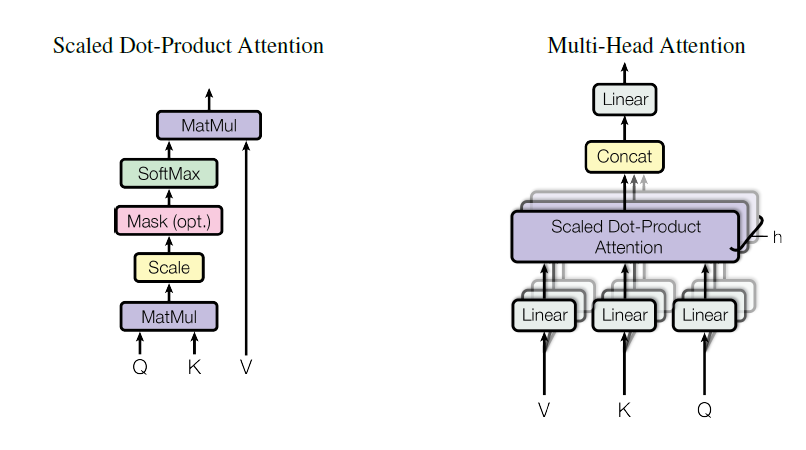
#### Multi-Head Attention

Given a specific utterance for an input sequence, attention can be given to different aspects in parallel, expressing different perspectives and usages of the attention mechanism. In the transformers architecture this is implemented with a Multi-Head Attention Layer. The input matrices *Q*, *K* and *V* are transformed to different linear projections for *h* times. Each transformation makes use of a respective weight matrix: *“On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding -dimensional output values“* (Vaswani, et al., 2017, p. 4). By using *h* different starting points for *Q*, *K* and *V* the algorithm can train jointly on different aspects of attention and isn’t limited to only one representation. The *h* output vectors of the scaled dot-product attention functions are finally concatenated and one more time linearly transformed with a trainable weight matrix :

Equation 2‑22: Multi-Head Attention (Vaswani, et al., 2017, p. 5)

The inner workings of the attention layers are depicted in figure xy:



# Data & Preprocessing [6]

This chapter will detail the procurement of training and test data, explain the necessary preprocessing, especially the anonymization part and give a short description of the training data, its structure, properties and the challenges associated with that.

## Procuring training data from production system

During 2020 a cross-functional team of BMW Bank experts implemented their new DMS. The commercial solution that was implemented allows for a complex customization to the companies processes and needs. A complex rule-based logic was developed, configured, tested and finally deployed into a final production system.

The classifier to be developed in this project is set to solve for all documents that the DMS cannot handle fully automatically. They require manual intervention through a service team. Documents falling short of automatic classification in the first place are added to a specific backlog within the DMS. Service agents are scheduled daily to log on and clear the backlog manually, thus labeling the correct document type for every document.

Following the Go-Live of the DMS a weekly set of training data was exported from the DMS productive system. The export always covered all documents (automatically and manually classified) from the DMS.[[15]](#footnote-16) Every set presented itself as a hierarchical file structure in the following form:

🗁 BatchClass => 🗁 BatchID => 🗁 DocumentID => 🗐 Data Files

Each folder representing a DocumentID accomodates two text files (both in string format): A content file containing the text of the specific document and an index file holding all available meta-data for the content file. A typical example would be content file *002D0353.txt* with its associated index file *002D0353\_index.txt*:

|  |  |
| --- | --- |
| Contentfile[[16]](#footnote-17)  /Schaden/1182502/2949971/002D0353.txt:  'Page 1\nFrom:\t"Kanzlei Meyer" <info@muster.de>\n Date:\tThu, 17 Dec 2020 11:18:36+0100\n To:\t\'"BMW Leasing GmbH\'" schaden@bmw.de>\nSubject:\tKunde: Muster Erik, Muster Weg 5, 12454 Musterstadt Verkehrsunfall vom:\n11.12.2020 Pkw, amtl.Kennz.: XX-JK1234 - Müller ./. Huber - Unser Zeichen: 1234/20-GG\nSehr geehrte Damen und Herren,\nIhr o.g. Kunde hat uns mit der zivilrechtlichen Schadensregulierung betreffend oben genannten Verkehrsunfall\n beauftragt.\nDas Fahrzeug wurde bei dem Verkehrsunfall beschädigt Der Schadensumfang wird derzeit von einem\nSachverständigen ermittelt. … | Indexfile with meta data …/002D0353\_index.txt:  "Schaden","Schaden PDF Image + Text","{Batch ID}","1182502","{DocumentID}","2949971","DocumentType","SCHADENSCHREIBEN","ClassificationResultWithConfiden","SCHADENGUTACHTEN;60;P|SCHADENANZEIGE;40;P|SCHADENSCHREIBEN;20;P","PageCount","1","{DocumentCount}","1","{$InputChannel}","EMAIL","{$sourceSystem}","EMAIL","AutoClassificationConfidence","0",""\n |

A weekly batch of training contained up to 90.000 documents accounting for twice this amount of input files (2 files per document). The file structure was uploaded to a high security data analytics platform (HSDAP) providing a Python 3.7.9 environment for data preprocessing and development. With a dedicated file parser the structure was read and the two files for each document were fed into one record of a pandas dataframe. First the content and index information were read into separated columns of the dataframe. Then the index string was subsequently extracted into further columns capturing different descriptors of the data including the training label (column DOCTYPE):

|  |  |  |
| --- | --- | --- |
| Column Name | Data type | Meaning |
| INDEXSTRING | object | Meta data regarding the document |
| RAWBODY | object | Content of the document (textual) |
| BATCHKLASSE | categorical | Source of origin (department) |
| BATCHCONTENT | categorical | Additional source information |
| BATCHID | int64 | Unique identifier for a batch |
| DOCID | int64 | Unique identifier for a document |
| DOCTYPE | categorical | The target class for the classifier |
| CONFIDENCE | object | Confidence levels for doc’types |
| AUTOCLASS | bool | Flag for manually labeled documents |
| PAGECOUNT | int64 | # of pages of the document |
| DOCCOUNT | int64 | # of documents belonging to a document |
| INPUTCHANNEL | categorical | Source of origin (Email, Fax, …) |
| SOURCESYSTEM | categorical | Source of origin (System A, B, …) |
| NBR\_DOCTYPES | int64 | # of d’types suggested by the DMS |

Table 3‑1: Structure of the raw data procured from the production system

This dataframe held all the raw data retrieved weekly. It served as the basis for further imperative preprocessing steps before the data had clearance to be used for model training.

## Anonymization of sensitive data in the documents

Given the sensitive nature of the documents retrieved a thorough data risk assessment had to be conducted and consequently an anonymization concept to be developed to get clearance with the data protection officer. The following routine of data protection and cleaning preprocessing was applied to every new dataframe retrieved:

1. Removal of critical groups of senders (i.e. employees, Rolls Royce clients, etc.):   
   A blacklist (a python set) containing 109k email addresses was matched against the textual content of each record. Every document in the dataframe was tokenized using an existing open source German language model[[17]](#footnote-18) and then scanned for positive matches. This routine removed between 6-9% of records within a weekly training batch.
2. The remaining documents in the dataframe were then cleaned and shortened with a number of string operations: Non-printable characters (i.e. “\t” or “\n”) were removed and typical conversational phrases and boiler plate copy was replaced with a short tag (i.e. “Sehr geehrte Damen und Herren,” => „<Anrede>“ or “"BMW Financial Services BMW Bank GmbH " => „<BMWB>)”.
3. With that another set of string operations was performed applying a number of regex-rules to identify critical information contained in the documents and replacing it with a descriptive tag. For example the vehicle registration number in is identified by the regex expression:  
   *r'\b[a-zA-ZöüäÖÜÄ]{1,3}[ -][a-zA-ZöüäÖÜÄ]{1,2} ?[1-9]{1}[0-9]{1,3}[^.]\b'.* Appendix xy lists all rules for identifying and replacing sensitive information.
4. Observing computational cost every document was trimmed to a maximum character limit of 10k characters. Prior data analysis indicated that this would trim roughly 10% of the documents and leave 90% of documents unchanged.
5. Finally a named entitiy recognition (NER) was performed to screen the tokenized documents for names, locations and organizations using the build-in pretrained NER-functionality methods of the spacy software package. Tokens recognized as peoples’ names were replaced with the tag “<PER>”. Locations such as city names were replaced with “<LOC>” and organizational names (i.e. “Huber GmbH”) were exchanged with <ORG>.

Applying this routine to the example above would result in this cleaned document:

|  |
| --- |
| 'Page 1 From: "<ORG>" <EMAIL> Date: Thu, 17 Dec 2020 11:18:36+0100 To: \'<ORG>\'"<EMAIL> Subject: Kunde: <PER>, Muster Weg 5, <PLZ> <LOC> Verkehrsunfall vom: 11.12.2020 Pkw, amtl. Kennz.: <KFZKZ> - <PER> ./. <PER> - Unser Zeichen: 1234/20-GG Sehr geehrte Damen und Herren, Ihr o.g. Kunde hat uns mit der zivilrechtlichen Schadensregulierung betreffend oben genannten Verkehrsunfall beauftragt. … |

Depending on the different classification approaches different additional preprocessing steps were necessary. More details regarding this are given in the description of the individual experiments.

The training of the models was performed on a dedicated Deep Learning Platform (DLP) of the BMW AG. The preprocessed and anonymized dataframes were first persisted on the HSDAP and then exported to the DLP, after removal of the raw textual information, leaving only the cleaned version of the documents.

## Explorative Data Analysis (EDA)

Over the course of several weeks this routine yielded a total consolidated sample of labeled training data with *N* = 62734 records, accounting only for the manually classified documents that the DMS could not classify automatically. The data retrieved provided for the textual data but also for some Meta data regarding the input channels, the number of pages, etc.. Most importantly it also contained the target information to train classifiers: the document types.

### Document Types (Labels)

The document type maps a document to a specific intention. Based on this a document is allocated to the relevant inboxes. Document types can describe very specific departmental intents but they can also be of a more generic nature. Thus they can be allocated across inbound traffic of multiple departments.

The data sample provides for 154 different document types. The left panel in Figure xy depicts the distribution of instances of the 20 most frequent.

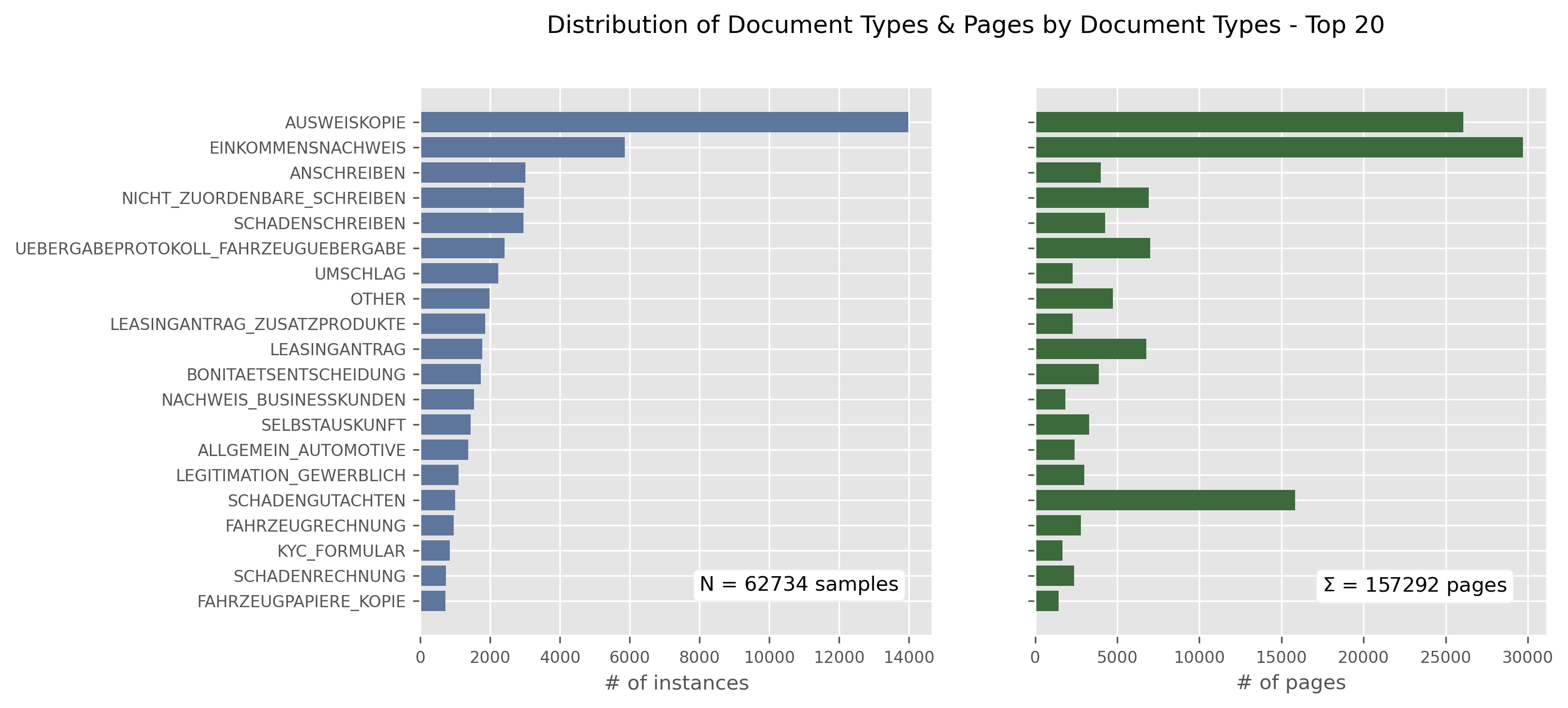


Figure 3‑1: Distribution of the Top 20 document types

The distribution of the 154 different categories[[18]](#footnote-19) skews towards a small number of classes: The top 5 labels account for 46%, the top 10 for 62% and the top 20 account for 80% of all *N* instances. The business objective however is to minimize the number of pages (and not documents) that require manual classification, as the cost for manual inspection is driven by page volume. The right panel in Figure xy illustrates the volume of pages related to each document type. This distribution corresponds well with the occurrence of the documents. Some exceptions though exist with a low document count but a respectively higher page volume (i.e. “SCHADENGUTACHTEN” accounting for 10% of the total page volume). Thus the importance of the pagecount feature as a cost for misclassifications has to be considered and is addressed adequately later with the setting of the metrics and the training parameters of the models.

### Other Meta Data

Table 3‑1 shows a full listing of all meta data retrieved in conjunction with the text and the document type of each document. This information is automatically determined by the banks’ DMS and is available to be used as additional features. There are 4 categorical features describing different aspects of the incoming channel and 3 numerical ones. Table xxx depicts the distribution of the categorical descriptors across the total sample of *N* documents:

|  |  |
| --- | --- |
| BATCHKLASSE | BATCHCONTENT |
| NeugeschäftUpload 0.572  NeugeschäftSeitenBasiert 0.132  Bestand 0.105  Schaden 0.105  Generic 0.065  Banking 0.021 | DS PDF Image + Text 0.383  DS PDF Image Only 0.189  Schaden PDF Image + Text 0.105  BestandDoc 0.104  PDF Image + Text 0.092  GenericDoc 0.065  PDF Image Only 0.040  BankingDoc 0.021  OTHER 0.002 |
| INPUTCHANNEL | SOURCESYSTEM |
| UPLOAD 0.391  POST 0.240  EMAIL 0.230  FASTLANE 0.122  FAX 0.015  OTHER 0.003 | DOC\_STORE 0.391  EMAIL 0.340  GAA 0.240  FAX 0.021  OTHER 0.008 |

Table 3‑2: Distributions of categorical features

The feature “BATCHKLASSE” describes different business units, managing certain corresponding inbound channels (i.e. specific email addresses). The department “Neugeschäft” is the predominant subject area accounting for more than 70% of all documents with an upload channel for the BMW retailers that accounts for 57% of all inbound traffic. The batch class property might serve as a strong descriptor for the document classification task as certain document types show a higher frequency in a given batch class than others or might even be exclusive to a given batch class.

Regarding the “BATCHCONTENT” it’s apparent that more than 80% of all documents are derived from an OCR scan of an image (e.g. photocopied ID cards, photographs, scans, etc.) with or without additional textual information and 23% of all documents are derived from an image-only original. This explains the poor quality in text and very noisy nature of the documents.

The “INPUTCHANNEL” and “SOURCESYSTEM” features are very similar and can be seen as a variation of the same information. The dominance of uploaded content again is very apparent.

“PAGECOUNT”, “DOCCOUNT” and “NBR\_DOCTYPES” are numerical features. The first feature describes the number of pages for each document, document counts describes how many other documents a single document is related to[[19]](#footnote-20) and the number of document types is calculated from the suggested list of document types that the DMS has already estimated for. Their distribution by the different batch classes is shown in figure xy.

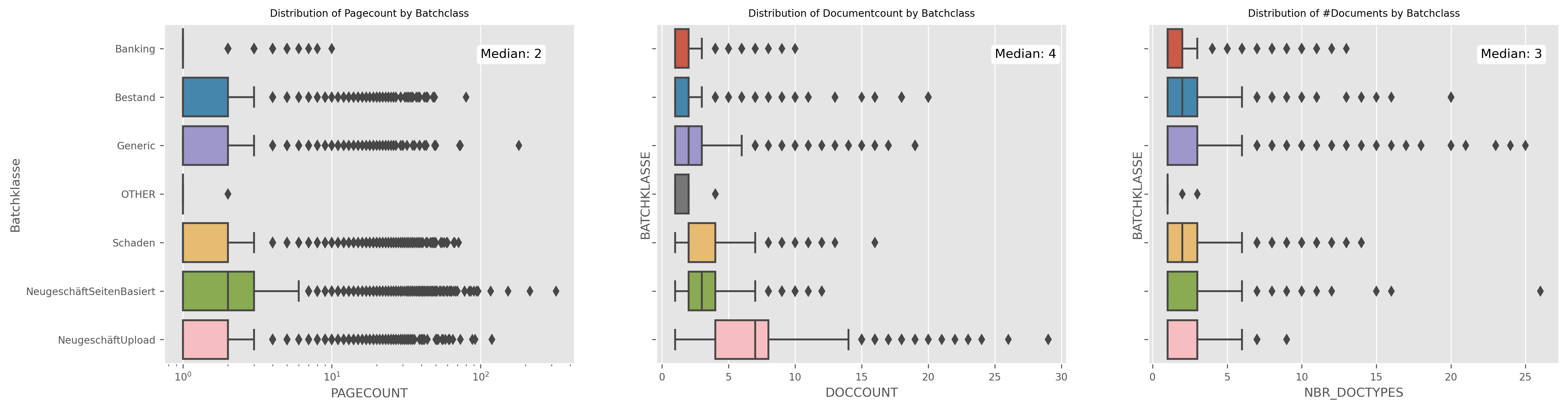


Figure 3‑2: Distributions of the numerical features (Meta Data)

Regarding the volume of pages with a median page count of 2 pages, the distribution shows that there are many outliers, some of them even exceeding 100 pages. For the document count the distribution of “NeugschäftUpload” shows a higher variance underlining that in this batch class interrelated single documents occur more frequently. As to the number of document types the DMS typically estimates around 3 different document types to be valid candidates, but outliers show that there can be up to 25 different document types suggested as candidates for a given document.

For model training the categorical features were encoded using the OneHotEncoder class of the Scikit-learn package. To avoid unfavorable effects of the outliers, the numerical features were normalized using the StandardScaler class in Scikit-learn.

### Text Data

# Methods and Experiments [10]

## Metrics

Accuracy & Weighted Accuracy

F1, Precision and Recall

## Machine Learning approaches (Scikit Learn)

### KNN

### Linear Models

#### Logistic Regression

#### Linear Support Vector Machine

#### Support Vector Machine

### Stochastic Gradient Descent

## Deep Learning approaches (TensorFlow)

### BiLSTM

### CNN

### BiLSTM with pretrained GLOVE

### CNN with pretrained GLOVE

### Multilingual BERT

### …

# Results & Discussion [14]

## Performance on key metrics

## Learnings

Efficiency

## Error Discussion

## Limitations

## Improvements

## Deployment

# Conclusions [4]

List of Abbreviations

**ABK** Abkürzung

**ACM** Association of Computing Machinery

**PDF** Portable Document Format

**IEEE** Institute of Electrical and Electronics Engineers

**ISO** International Organization for Standardization

List of Tables

[Tabelle 2.1: Ebenen der Kopplung und Beispiele für enge und lose Kopplung **Fehler! Textmarke nicht definiert.**](#_Toc437546466)

List of Figures

[Abbildung 2.1: Mars Rover der NASA **Fehler! Textmarke nicht definiert.**](#_Toc346226275)

[Abbildung 2.2: Point-to-Point **Fehler! Textmarke nicht definiert.**](#_Toc346226276)

Bibliography

Appendix

Aggarwal, C. C. & Zhai, C., 2012. A Survey of Text Classification Algorithms. In: *Mining Text Data.* s.l.:s.n., pp. 163 - 222.

Bekkerman, R., 2004. "Automatic Categorization of Email into Folders: Benchmark Experiments on Enron and SRI Corpora". *Computer Science Department Faculty Publication Series*, p. 218.

Joachims, T., 1998. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In: *ECML.* s.l.:s.n.

Joachims, T., 2001. A Statistical Learning Model of Text Classification for Support Vector Machines. In: *SIGIR '01.* s.l.:s.n.

Kowsare, K. et al., 2019. Text Classification Algorithms: A Survey. *Information (Switzerland)*, 23 April.

Ng, A., 2018. Machine Learning Yearning. s.l.:s.n.

Raaijmakers, S., 2021est.. *Deep Learning for Natural Language Processing.* s.l.:Manning Early Access Programm (MEAP).

1. Commonly term document matrices are described with rows accounting for words and columns representing documents (Jurafsky & Martin, 2019, p. 100). However the chosen transposed form with a matrix the shape of (documents *x* words) resembles the feature space that a machine learning algorithm would expect for input. [↑](#footnote-ref-1)
2. Commonly a window size of 4-5 words is applied (Kowsare, et al., 2019, p. 8) [↑](#footnote-ref-2)
3. Since the distance of two words should be interchangeably equal between the two the chosen function for the model should be invariant to exchanges of those roles. The properties of the exponential function cater for this requirement (Pennington, et al., 2014, p. 1534). [↑](#footnote-ref-3)
4. The authors suggest a value of 0.9 for the decay rate . [↑](#footnote-ref-4)
5. Again a small smoothing term is added to prevent zero division [↑](#footnote-ref-5)
6. This chaining of operations can also produce the unfavorable exploding of the gradients problem. That is when values larger than 1 are pushed through thousands of multiplications scaling the upgrades of the ANN to exponentially large portions. [↑](#footnote-ref-6)
7. In training process the entire input of a training set is portioned into equally sized batches. During one training step an entire batch is fed to the network. Networks are commonly configured to apply backpropagation for an entire batch. When all batches have been processed, the network has completed one epoch. ANNs get trained over many epochs. [↑](#footnote-ref-7)
8. contains one scaling parameter for every input, and respectively one shifting parameter for every input. So every input into the layer is scaled and shifted by its respective parameter set. [↑](#footnote-ref-8)
9. Note that time step in this context expresses foremost the sequential order of an information, not necessarily a period of real world time. [↑](#footnote-ref-10)
10. The final model architectures used within the experiments of this project are detailed in chapter 4. [↑](#footnote-ref-11)
11. This equation represents a ‘narrow convolution’, which applies a total number of *n-k+1* window-movements over the input, potentially skipping a remainder of input words that don’t fit into a full k-sized window. Should it be favorable to include those remaining words, then a padding of an adequate number of zero-valued vectors can be applied to the beginning and the end of the sequence. This operation is called a ‘wide convolution’ (Kalchbrenner, 2014). [↑](#footnote-ref-12)
12. The intuition of selecting the max value is to choose the most salient information from the convolutional representation. While max pooling is most commonly chosen, other mathematical operations like taking the average or the min value can be employed instead. [↑](#footnote-ref-13)
13. The authors use *N* = 6 [↑](#footnote-ref-14)
14. The authors used to scale down the matrix multiplication [↑](#footnote-ref-15)
15. For the proof of concept (POS) management decided to focus on the most important departments with regard to inbound traffic. [↑](#footnote-ref-16)
16. Document content presented for illustrative purpose within this thesis has been cleared of any real personal data like names, addresses, registrations, account numbers etc.. They were always replaced with random values to comply with the sensitive nature of this data. [↑](#footnote-ref-17)
17. All preprocessing steps were conducted with the support of the Spacy German Language Model “de\_core\_news\_md”. Documentation and detailed description can be found at https://spacy.io/models/de [↑](#footnote-ref-18)
18. With emphasis on the most relevant tasks in daily operations document types with an occurrence of less than 100 instances in the sample (0.15%) were consolidated to a generic label (“OTHER”). [↑](#footnote-ref-19)
19. Several single documents can be interrelated. For example a new client could send in an application for a lease together with other documentation of income, id-cards, proof of address, etc.. [↑](#footnote-ref-20)