university of groningen / faculty of science and engineering

MASTER'S THESIS

University of Groningen
Department of Artificial Intelligence

Text-based Patent-Quality Prediction Using Multi-Section Attention

Author:
Xabi Krant (s2955156)

Supervisor AI:
Prof. dr. L.R.B. Schomaker

Supervisor SIM:
dr. P.J. Steinberg

March 15, 2023

Abstract

The number of patents has increased tremendously in recent years and statistics derived from patents have become the standard measurement for innovation. Patents statistics are widely available and correlate well with patent valuation and quality. They do however suffer from inherent biases and flaws, for instance caused by the characteristics of patent offices and their employees. The major drawback of using patent statistics is substantial time needs to have passed before they can be used and there are no ex-ante indicators available.

This thesis aims to solve this problem, using current advances in machine learning and language models. A new text model is introduced that can predict the innovation and market value of patents based solely on the patent text. Current state of the art machine learning text models like BERT are however not a perfect fit for patents, as patents can contain very long text and they have a multi-section structure. This thesis proposes a new model, called MSABERT, that is able to handle longer texts and the multi-section structure. Each section is handled separately, after which they are combined using attentitive pooling. This attentitive pooling also adds a layer of explainability to the model, showing the relative importance of each section.

This new model is compared to models that lack some of these capabilities in several experiments. The results show that this newly introduced model achieves similar performance as existing models when the models are trained in an end-to-end fashion. Predicting whether patents will be accepted solely on the text is a generally hard task and the performance of the models seems to hit a ceiling. When the models are used in a transfer learning scenario, the MSABERT model clearly outperforms the other models. The MSABERT model pre-trained on the acceptance task is able to accurately predict the patent value measured as the OECD quality indicator. The performance of the other models lacks far behind with an error about 5 times as high. This is a very promising result, as it shows that the MSABERT is capable of extracting the patent value from the text alone.

The MSABERT model improves performance in a transfer learning scenario and adds explainability without a decrease in performance. This allows users to use this model in applications to predict patent value early in the process. It can be a useful tool for companies to evaluate internal patents, evaluate patents of other companies for a merger or acquisition or as a tool for research.

Acknowledgements

Most of the making of this thesis has been done throughout COVID-19 lockdowns and restrictions. The writing and research for this thesis has been done from home, as university buildings had been closed for a majority of the time. Meetings with the supervisors were done online. This made it very hard to remain motivated. Throughout the process I have had several periods were it was hard to motivate myself to continue. The meetings with my supervisor Lambert Schomaker helped me continue. I am grateful for his insights, the discussions we had and the support in writing this thesis. I would also like to thank Philip Steinberg, who inspired me to work on this topic and supported me throughout. One of the experiments in this thesis was a replication of an experiment in Arts, Hou, and Gomez (2021), I want to thank Sam Arts for his help in getting started. Last, I would also like to thank the CIT for access to the peregrine HPC cluster and even providing access to a personal machine for the research. Throughout writing this thesis I have learned a lot and developed myself further. I hope this thesis can spark interest in patent-quality prediction, as it comes with some interesting challenges.

Contents

1 Introduction

Innovation and technical progress is of vital importance for the growth of companies. Innovation is a broadly used term, which is often defined as: "The implementation of a new or significantly improved product (good or service), or process, a new marketing method, or a new organisational method in business practices, workplace organization or external relations." (Eurostat, 2005). Successful innovation is essential for creating and maintaining competitive advantages in firms (Martín-de Castro, Delgado-Verde, Navas-López, & Cruz-González, 2013). Despite its importance, both in practice for companies and in research, measuring innovation is complex and challenging (Gault, 2018). Measuring the quality of a single innovation is even more of a challenge, especially for scholars (Arts et al., 2021).

Due to their public availability, patents have become the predominant indicator for innovation (Griliches, 1990; Hall, Jaffe, & Trajtenberg, 2001). A patent is an exclusive right granted for an invention that provides a new way of doing something (Organization, n.d.). As shown in Figure 1.1, the number of patents in the US approximately follows Moore's law for transistor size. Measuring the quality and impact of newly created technology in the initial phase, i.e., on the basis of patent documentation, remains challenging however (Arts et al., 2021). Innovation can be measured at different levels, such as the technological sector, country or company level. This research focuses on the level of a single technology based on a single patent document. The advantage of developing a methodology on this level is, although interactions between patents should be taken into account, that it can be aggregated to higher levels. The quality of the patent is usually measured as the technical novelty of the innovation, the technological and economic impact it has and the impact is has on both subsequent technological developments and society in general (Arts et al., 2021; Squicciarini, Dernis, & Criscuolo, 2013).

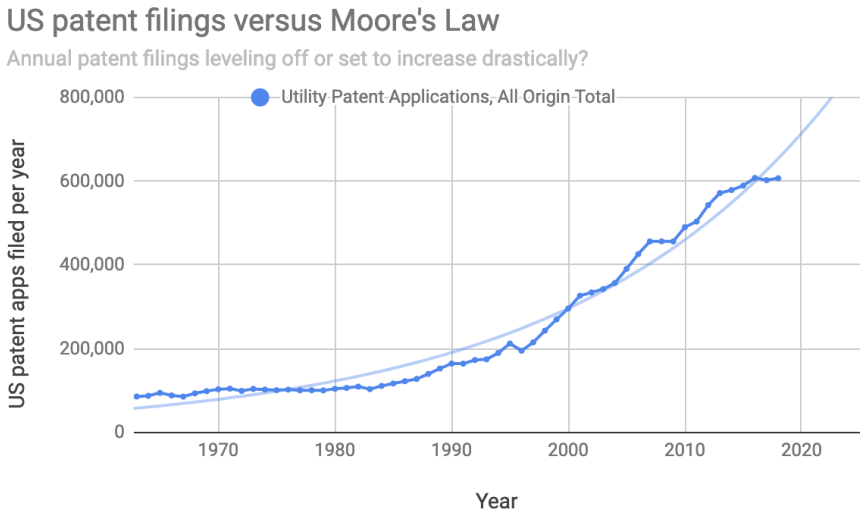


Figure 1.1: Number of patents filed in the United States over time. Next to this a graph for Moore's law is shown. Image from Schick (2019)

Many of the proposed methods to determine the quality of an newly invented technology based on patents are only possible after a substantial amount of time has passed. The most common method in literature is using forward citations and patent classifications. These statistics derived from patent metrics should however be treated carefully because not all patents represent innovations, and not all innovations are patented. The impact of patents varies hugely and is very skewed (Griliches, 1990; Lemley & Shapiro, 2005; Scherer & Harhoff, 2000) and "about one-third of the patents are not used for specific economic or commercial activities" (Giuri et al., 2007). Furthermore patent citations are an indirect measure of the technological innovation, the citations do not necessarily reflect the content of the patent. This thesis introduces a model that is not affected by these inaccuracies, as it measures innovation based on the patent text directly.

Patent citations are moreover hugely influenced by the human examiners who can add missing citations to proposed patents (Alcacer & Gittelman, 2006). Around 40% of all citations are added

by examiners. Although this does not have to be a problem, the effect of examiners varies across patents in different technological fields and origins (Alcácer, Gittelman, & Sampat, 2009). Next to this, examiners also have a strong influence on whether patents are rejected or accepted in the first place. This examination is shown to be influenced by human personal experiences. Factors like examiner characteristics (Lemley & Sampat, 2012), incentives (Retnasaba, 2008) and even the weather (Kovács, 2017) can influence whether patents are accepted or not. Although Lemley and Sampat (2012) show that the percentage of patents accepted in violation with patent office standards is lower than previously assumed, it is still around 10%.

Despite patent citations clearly being a sub-optimal measure of innovation quality, Nagaoka, Motohashi, and Goto (2010) show that patent documentation does contain relevant information about the quality of a technology. They shows that using citation information of patents can significantly predict market value and can therefore be used as a proxy for innovation levels. Mainly because of a lack of alternatives, patent citation statistics is the most widely used indicator of the quality of an innovation and technology.

More recently some researchers have tried to determine the quality of the technology directly from the patent content. These studies have applied natural language processing (NLP) methods to innovation research. A significant body of research has been developed focusing on detecting novelty in text documents (Allan, Wade, & Bolivar, 2003; C. Lee, Kang, & Shin, 2015; X. Li & Croft, 2005; Smola, Song, & Teo, 2009; Wang & Chen, 2019), patent valuations (Arts et al., 2021; Falk & Train, 2017; Hasan, Spangler, Griffin, & Alba, 2009; Reitzig, 2004) and technology forecasting (Chen, Zhang, Zhu, & Lu, 2017; Liu et al., 2017). Abbas, Zhang, and Khan (2014) provide an overview of patent analysis techniques until 2014. Table 1 provides an overview of the studies and the applied NLP techniques. Most of the traditional methods have focused on hand-crafted features like key phrase-based models. Only in the sub-field of patent-similarity research more advanced NLP methods have been applied. Younge and Kuhn (2016) have used a TF-IDF (Jones, 1972; Salton & McGill, 1983) model to compare documents, upon which Kelly, Papanikolaou, Seru, and Taddy (2018) have build a time-dependent adaptation. Hain, Jurowetzki, Buchmann, and Wolf (2020) used a Word2Vec (Mikolov, Chen, Corrado, & Dean, 2013) based approach. Even though these methods are still commonly used in natural language processing, more recent models like RNNs (Rumelhart, Hinton, & Williams, 1986) and the Transformer (Vaswani et al., 2017) have shown to achieve better performance on a variety of tasks. The goal of the current research project is to apply the state-of-the-art transformer models to the field of patent analysis.

In recent years the field of natural language processing has advanced significantly. The TF-IDF model is able to weigh important words against common words, but it is not able to represent any meaning of a word. Word2Vec is able to represent the meaning of words by looking at their context, but is not able to differentiate between homonyms. More recent models are better able to handle this polysemous disambiguation and generally perform superior. The advances in natural language processing relies on three important developments: the introduction of the Transformer, the use of pre-trained models and the advances in computer processing. The Transformer model (Vaswani et al., 2017) allowed for more parallelization, allowing to build deeper models with more parameters. GPT (Radford, Narasimhan, Salimans, & Sutskever, 2018) and BERT (Devlin, Chang, Lee, & Toutanova, 2018) were the first large-scale Transformer models. They showed that increasing the parameters indeed improves the performance of the language models. These models do however need a lot of data to train, which is scarce in many supervised tasks. Usually, these models are first trained on an unsupervised corpus and then fine-tuned to the supervised downstream task.

Table 1: Non-exhaustive overview of methods used in patent novelty and valuation research

Methodology	**Papers**
Key words and phrases as features	Bergmann et al. (2008); deGrazia et al. (2020); Gerken and Moehrle (2012); Hasan et al. (2009); Y.-R. Li et al. (2009); Noh et al. (2015); Park et al. (2012)
Newly introduced (key) words	Arts et al. (2021); Balsmeier et al. (2018)
Topic model approaches	Ashtor (2019); Kaplan and Vakili (2015); Teodoridis et al. (2020)
Text-similarity measure	Hain et al. (2020); Kelly et al. (2018); Moehrle (2010); Younge and Kuhn (2016)

This allowed for far bigger models with rich linguistic knowledge in tasks by only training with a few samples. BERT and the more recent versions of GPT (GPT-2 and GPT-3) have shown to outperform older techniques, like Word2Vec, in a variety of tasks.

BERT and other pre-trained models have mostly been developed for tasks like question answering and language inference. The texts for question answering are usually short and the language can likely be inferred from a subsection of the text. Patents are however not like these other text documents and the state-of-the-art natural language processing models used for other documents can not be applied directly. The main difference is that patents are very long and have a multi-section structure.

Patent documents are structured with a title, abstract, description and claims section. The claims section define the scope of the patented innovation and are the basis of the patent. Formally "the claims of a patent define the invention to which the patentee is entitled the right to exclude" (*Phillips v. AWH Corp.*, 2005). The description section explains the invention in detail and should enable others to replicate the invention (Office, n.d.-b). As opposed to the claims section, the description section can not be changed by examiners on later in the process. Figures A.1 and A.2 in the Appendix show an example patent. The claims and description sections have a significantly different text style and structure.

The widely used approach to deal with text in different sections is to simply concatenate them into a single piece of text. This is problematic because all sections will be treated in the same fashion, regardless of the difference in content, structure and style. Since the sections in patents are so different, this approach is possibly not valid.

Patent texts are also extremely long, especially the descriptions section. Figure 1.2 shows the text length of patents in the data set. The main issue with long-sized text chunks is that they can not be used as input for many Transformer models. For instance, the well-known BERT algorithm accepts a maximum of 512 tokens, GPT accepts a maximum of 1024 tokens. Tokens are explained in Section 2.3.3, very broadly defined a token is a piece of a word. On average a token corresponds to around 4 characters (OpenAI, n.d.), meaning that on average a patent consists of around 19 000 tokens. Even the shortest patent in the data set consists of around 650 tokens and can not be used by BERT directly, with most patents consisting of thousands of tokens. Most approaches use the first x tokens that can be accepted and discard the rest of the text. With patents this would be problematic, as in some cases only a few percent of the full text will be used.

The problem of the multi-section structure of a patent document is even bigger because of the differences in the section lengths. The average description section is around 100x as long as the abstract section and 10x the size of the claims section. Treating the entire patent as a single text chunk would mean that the description section will dominate the characterization of the patent text and important information from other sections might be underutilized.

This would be an even bigger issue when only the first 512 tokens are used. This could mean that for some patents the title, abstract and part of the claims are used, whereas for others only the title and part of the abstract would be included. These two would contain significantly different information and would be structured differently. Then treating these patents in the exact same fashion could hurt performance.

1.1 Research Questions

In this thesis, the goal is to develop a model that is able to combat the difficulties in patents and utilize the strengths of state-of-the-art pre-trained Transformer models. This leads to the following research question: "How can state-of-the-art pre-trained Transformer models be applied and adapted to the task of patent quality prediction?". To answer this question, the following sub-questions will be answered:

- Does using the full text improve the performance compared to models using a limited part of the text?
- How does leveraging the multi-section structure of patents affect the performance in patent quality prediction?
- What sections of the patent contribute most to the prediction quality?

The thesis is structured as follows. First the theoretical background of the methods relevant for this thesis will be described. This includes sections on machine learning, neural networks and

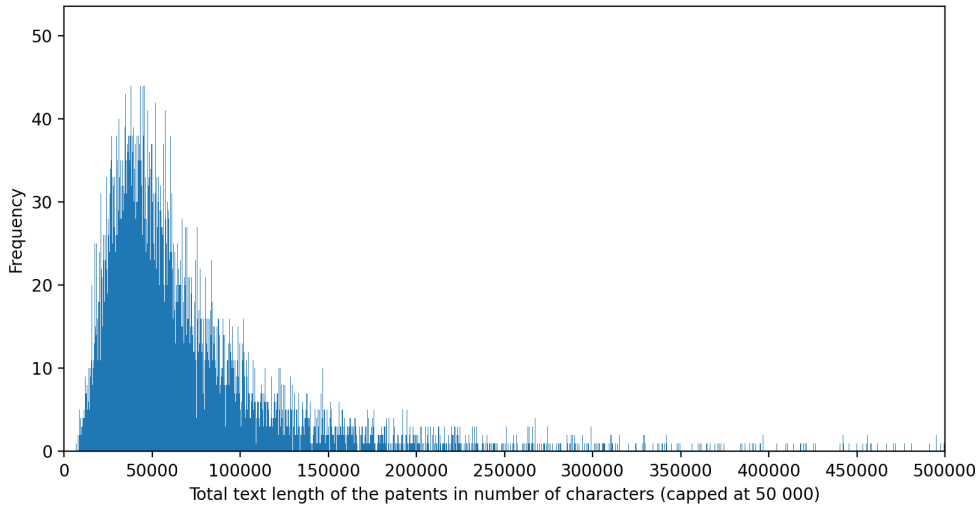


Figure 1.2: Histogram of the length of the full text of patents. The reported length is the number of characters including spaces and punctuation. For readability, patents with a length of over 500 000 were are not shown. This discards 298 out of the 48 630 patents.

natural language processing, both statistical and neural. Chapter 3 will describe the methods used and the setup of the experiments. This will include a description of the data set and the proposed model. Chapter 4 will describe the results of the experiments. Finally, Chapter 5 will relate the results to the research questions and a conclusion is given, as well as the contribution to the research field and the possibilities for further research.

2 Theoretical Background

This Chapter describes the theory behind the text-processing techniques and algorithms used in this thesis. Section 2.1 provides an overview of the core concepts of machine learning. In the section afterwards the basics of artificial neural networks (ANN) are explained. Artificial neural networks are at the basis of deep learning and most models presented in this thesis. Section 2.3 expands on this and explains the most relevant techniques used in natural language processing. Section 2.4 then explains the related techniques used in previous research. Finally, Section 2.5 shortly introduces the proposed MSABERT model.

2.1 Machine Learning

Machine learning studies algorithms that automatically learn from experiences and is an important research area within Artificial Intelligence. One widely used definition of machine learning algorithms is: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997). Machine learning tasks can generally be divided in the three categories supervised, unsupervised and reinforcement learning. Reinforcement learning is not relevant for this thesis and therefore the focus will be on supervised and unsupervised learning. Section 2.1.3 will also explain transfer learning, which is at the basis for many recent natural language processing models.

2.1.1 Supervised learning

The goal of supervised learning is finding a function that maps input x to a target output y. Both input x and output y can be anything. They can for example be images, text or sound signal and they do not have to be the same type. The outputs y are usually assigned by a human supervisor. These outputs y are shown as targets together with the input to the model, hence the name supervised learning. Probably the most well-known type of supervised machine learning are classification tasks. The goal of a classification task is to classify a certain input into a class. The algorithm learns this by being presented the input together with the target class and learns a mapping that fits best for the training data. Some classification tasks include spam or non-spam classification, movie genre classification, handwritten character recognition, object recognition and many more. Besides classification, regression tasks are also very frequent in supervised machine learning. In a regression task the output is continuous. Examples include predicting views for a Youtube video or the number of citations of a paper.

2.1.2 Unsupervised learning

The other major type of machine learning is unsupervised machine learning. As opposed to supervised learning, in unsupervised learning the target output is not shown to the model. In unsupervised learning the goal is to extract relevant patterns from the input data. These patterns can for instance be similarity between the input samples, as used in clustering where clusters of similar samples are found. Another example is to find lower-dimension patterns that still represent the most important elements as in Principal-Component Analysis or dimensionality reduction. Unsupervised machine learning does not require labeled data, the collection of which is very labour-intensive, and is therefore especially relevant to utilize bigger data sets.

2.1.3 Transfer learning

Transfer learning is a form of machine learning where the algorithm is trained to learn a certain task and afterwards the learned mapping is applied on a different problem. An example is an algorithm that is first trained to cluster the data and is then transferred to a classification task. The idea is that the learned mapping on the initial task is a good starting point for training on the new task and can speed it up. The advantage is that the initial unsupervised training does not need labeled data and can utilize more data. Transfer learning has become one of the predominant methods in natural language processing, as there is an abundance of unlabeled text data but labeled data is usually much more sparse.

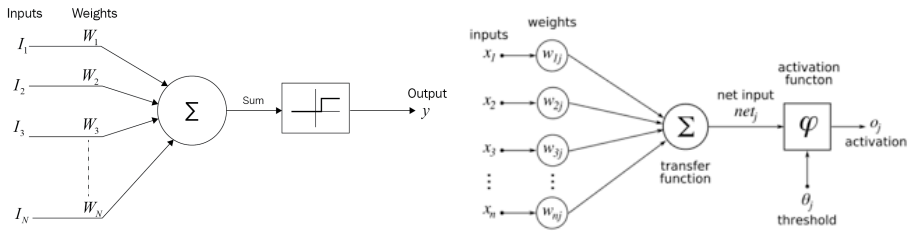


Figure 2.1: **Left:** A McCulloch-Pitts neuron with x weighted inputs and a single output y. **Right:** A perceptron with weighted inputs, a bias term and a single activation output. Images respectively from *The McCulloch-Pitts neuron* (n.d.) and *Rosenblatt perceptron* (n.d.).

2.2 Artificial Neural Networks

Artificial neural networks (ANNs) are machine learning models that are loosely based on neurons in the human brain. ANNs consist of processing units resembling neurons that connect to other neighbouring processing units resembling synapses. The output of a neuron is computed by summing its inputs and applying a non-linear function. The connections between the neurons have a certain weight that can be changed during the learning phase. Generally ANNs are trained with supervised learning and learn a function that maps given inputs to an output by changing and learning the weights of the connections between neurons.

2.2.1 McCulloch-Pitts neuron & perceptrons

McCulloch and Pitts (1943) were the first to propose a mathematical model loosely based on neurons. In the McCulloch-Pitts neuron multiple binary inputs are aggregated and based on this value a decision is made. If the summed input surpasses a certain threshold the neuron will output 1, representing a firing neuron, otherwise the output will be zero. Figure 2.1 shows a graphical representation of the McCulloch-Pitts neuron, Equations 2.1 and 2.2 show the mathematics behind it.

$$\sum_{x=1}^{n} I_x w_x = a \tag{2.1}$$

$$f(x) = \begin{cases} 0\, x < threshold \\ 1\, x \geq threshold \end{cases} \tag{2.2}$$

McCulloch-Pitts neurons take as input binary values and output a binary value. These binary values represent booleans, and McCulloch-Pitts neurons are capable of representing the boolean functions AND and OR. For the AND function the neuron should output 1 if and only if both inputs are 1, a threshold of 2 achieves this. For the OR function the neuron should output 1 if one of the two inputs is 1, a threshold of 1 achieves this. Unfortunately, the McCulloch-Pitts neuron only allows a linear decision, it can for example not represent a XOR. Although the McCulloch-Pitts neuron is very limited the underlying concepts are still at the basis of current ANNs.

Rosenblatt (1958) introduced an extension upon the McCulloch-Pitts neurons to include learning, called the perceptron. The perceptron is a binary classifier. As opposed to the McCulloch-Pitts neurons, where inputs are binary, the perceptron takes in weighted real-valued inputs. To these inputs a learnable constant, the bias, is added. The activation function of the Rosenblatt perceptron is similar to the activation of the McCulloch-Pitts neurons with a threshold value of 0. Again Figure 2.1 shows a graphical representation, Equations 2.3 and 2.4 show the mathematical functions.

$$\sum_{i=1}^{n} x_i w_i + b = a \tag{2.3}$$

$$f(x) = \begin{cases} 0\, x \leq 0 \\ 1\, x > 0 \end{cases} \tag{2.4}$$

The perceptron learns the weights and bias in a supervised way. It is initialized with random values and moves the decision boundary to a location which is best able to separate the two classes.

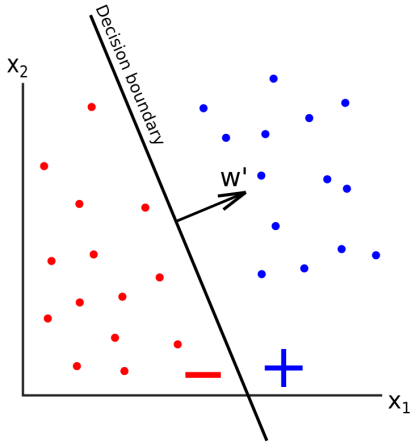


Figure 2.2: The perceptron and the boundary decision. The weights are updated such that the two classes (shown in red and blue) are separated. Image from Lazar (n.d.)

.

The simplest version of the perceptron is trained by forward propagation. In forward propagation the inputs x and targets $\hat{y_i}$ are send through the network with weights w. The network then predicts an output y and updates the weights according to the prediction. If the predicted value y is correct the decision boundary will be moved towards the input vector, if it is wrong it will be moved away. The learning rate η determines how much the weights are updated for each update and consequently how far the decision boundary is moved. Equation 2.5 shows the complete weight update function, Figure 2.2 shows a graphical representation.

$$w_i(n+1) = w_i(n) + \eta(y_j - \hat{y_j}) * x_{ij} \tag{2.5}$$

Despite the major improvement of introducing learning, the capability of the perceptron is still very limited. As with the McCulloch-Pitts neuron the perceptron is only capable of learning linearly separable functions. The perceptron is not capable of representing a boolean XOR function (Minsky & Papert, 1988), as represented in Figure 2.3. To solve this,the Multi-layer Perceptron (MLP) was introduced which will be explained in the next section.

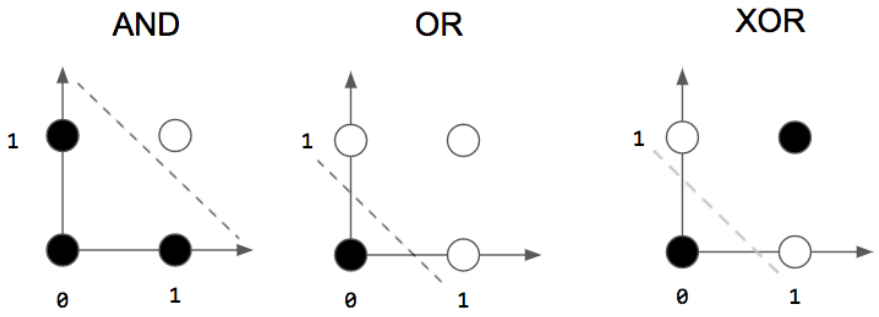


Figure 2.3: The logical boolean functions AND, OR and XOR. The black dots are false outputs, the white true. The AND and OR functions are linearly separable, as shown by the line. The XOR function is not linearly separable, the line exemplifies that it is not possible. Adapted image from Deshpande (2020).

2.2.2 Multilayer perceptrons

The single layer perceptron consists of an input layer and an output layer with an activation function, the output layer is the only layer containing trainable neurons. In order to solve the non-linearity issue a multilayer perceptron (MLP) was introduced which contains additional layers between the input and output layers, called hidden layers. A MLP can contain one or multiple hidden layers. Figure 2.4 shows a simple MLP with a single hidden layer. In a MLP all inputs or

neurons in each layer are connected to all neurons in the next layer, this is generally referred to as fully connected.

The added hidden layer is however not enough for the perceptron to learn a non-linear function. With a linear activation, the forward pass still consists exclusively of linear operations. The forward pass is namely a series of dot products between the input matrix and the weights. Therefore a MLP will never be able to learn a non-linear function without a non-linear activation. Two historically famous non-linear activation functions are the sigmoid and tanh, recently ReLU activation functions and other activation function have become more popular. Activation functions will be explained further in Section 2.2.3.

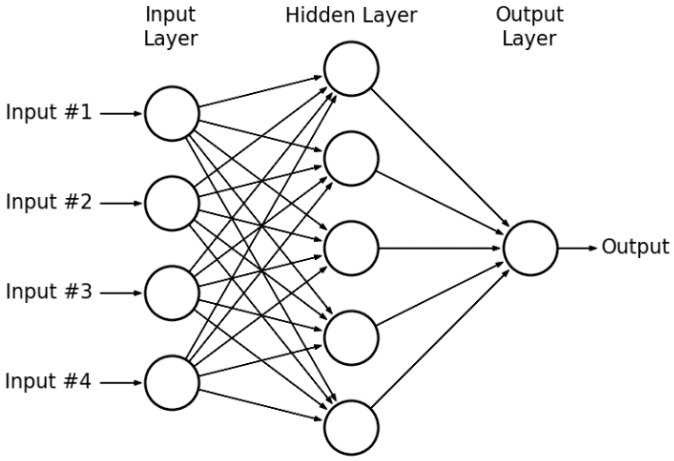


Figure 2.4: An example Multilayer perceptron with a single hidden layer. Image from Zahran (n.d.)

With the combination of multiple layers and non-linear activation functions the Multilayer Perceptron is capable of learning non-linear functions. As shown by Cybenko's theorem (Cybenko, 1989), the MLP is a universal function approximator. This means that the MLP can approximate any function, even with just one hidden layer. How well the function will be approximated depends on the data, training time and number of hidden layers and neurons. As opposed to the perceptron which uses forward propagation, the MLP uses backward propagation or backpropagation to optimitze the weights. This will be explained in more detail in Section 2.2.5

2.2.3 Activation functions

Activation functions control the output of the neurons in a artificial neural network. They determine whether a neuron "fires" or not based on the weighted sum of the inputs and the bias. To calculate the outputs, the weights in a neural network are multiplied with the input, which is a linear operation. If the activation function is linear as well, only linear functions can be learned. To learn non-linear functions, a non-linear activation function is required. Next to this, non-linear functions have the major advantage that their range is limited. Linear functions can have any value from minus infinity to infinity. The learning process of neural networks performs better with a bounded activation functions than with non-linear functions. The last advantage of non-linear functions is that they are differentiable. The derivative of linear functions are always a constant.

There are many activation functions, the most relevant ones will be discussed here. Underneath each explanation, an example equation with the derivative will be presented. A graphical representation of the sigmoid, tanh and ReLU is presented in Figure 2.5.

- **Linear activation:** The simplest form of activation is linear activation, as shown in Equation 2.6. As explained earlier, this form has some disadvantages. Solely using linear activation can not approximate non-linear functions and the output range is not bounded. Linear activations are however useful for an output layer with continuous outputs, such as prediction of house prices or citation prediction.

$$f(x) = ax \quad f'(x) = a \tag{2.6}$$

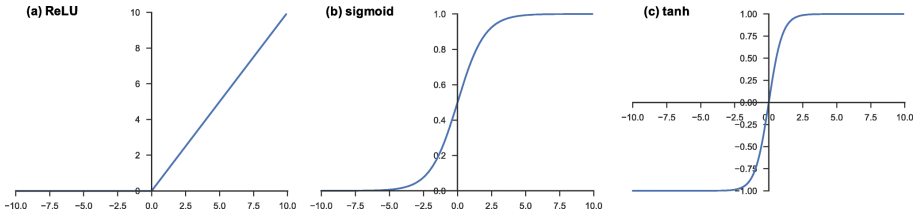


Figure 2.5: The activation functions ReLU, tanh and sigmoid shown in a graph. Image from Yamashita et al. (2018).

- **Sigmoid activation:** The sigmoid functions ranges from 0 to 1, approaching 0 for negative values and approaching 1 for positive values. This makes the sigmoid function well suitable for the last layer in binary classification. The sigmoid does however suffer from two problems. Firstly the function is centered at 0.5 and not zero. This causes the gradients to become either all-positive or all-negative. Secondly the sigmoid activation suffers from the vanishing gradient problem, this become apparent in the derivative shown in Equation 2.7. The vanishing gradient problem appears when the gradient becomes very small for large value inputs.

$$S(x) = \frac{1}{1+e^{-x}} \qquad S'(x) = S(x)(1-S(x)) \tag{2.7}$$

- **Hyperbolic tangent:** The hyperbolic tangent (tanh) function is similar to the Sigmoid but ranges from -1 to 1. As it is centered around 0, it does not suffer from the zigzag problem and has more stable learning. The tanh still suffers from the vanishing gradient problem however.

$$tanh(x) = \frac{1-e^{-x}}{1+e^{-x}} \qquad tanh'(x) = (1-tanh(x)^2) \tag{2.8}$$

- **Rectified Linear Unit:** As opposed to the tanh and sigmoid, the Rectified Linear Unit (ReLU) does not have an S curve shape. Instead the ReLU resembles the linear activation function, except that it will be zero for negative x values. The ReLU is computationally very efficient, making it extremely popular and widely used. Although the ReLU still suffers from the vanishing gradient problem it is less prominent than for tanh and the sigmoid as the ReLU has a constant gradient for high values as well. The ReLU does however suffer from another problem. The neurons can become inactive and have a gradient of zero regardless of the input.

$$ReLU(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \qquad ReLU'(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \\ undefined & x = 0 \end{cases} \tag{2.9}$$

- **Softmax:** The last activation function discussed here is the softmax activation. It is quite different from the others discussed above. The softmax normalizes the input to a probability distribution vector, where the sum of the output is equal to 1. The softmax is often used in multi-class classification tasks, where the softmax output represents the likelihood of the input being from a certain class.

$$Softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.10}$$

2.2.4 Loss functions

In supervised learning the model is presented with an input and a target output. The goal is to learn optimize the parameters in the model to maximize its performance on a task. The loss function is used to measure the error between the model output and the target output. The loss functions calculates an error that serves as a proxy for the model performance. The lower the error, the better the model performance.

For the purpose of this research, two categories of loss functions are important, namely classification losses and regression losses. A classification loss deals with a finite number of output

classes, while a regression loss deals with a continuous output. The optimization of the model is based on the loss function, so it is important to select an appropriate function. In this research two loss functions are used: cross entropy (CE) loss and mean-squared error (MSE) loss.

The mean-squared error is used for continuous numerical outputs. Example task are regression tasks like citation prediction. It measures the difference to the target output. Equation 2.8 show the MSE function, where y is the model output and $\hat{y}$ the target output.

$$Loss_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) \tag{2.11}$$

For classification tasks, the network does not output continuous values but instead outputs probabilities per class. For this the cross-entropy (CE) loss is used. Entropy is a statistical concept that quantifies the difference between two probability distributions. Equation 2.9 formalizes the cross entropy loss.

$$Loss_{CE} = -\sum_{i=1}^{C}(y_i log(\hat{y}_i)) \tag{2.12}$$

2.2.5 Backpropagation & Optimization

Based on the loss function the weights in a neural network need to be updated, to improve the results and decrease the loss. Most optimization algorithms to do this are based on a technique called gradient descent. In gradient descent algorithms, the gradient of the loss function with respect to each weight within a network has to be calculated. This is eventually used to update these weights. Backpropagation is one of the methods that uses gradient descent and does so by using the chain rule. In backpropagation the first order partial derivatives of the loss with respect to each weight are computed. The weights in the network are then updated by propagating the gradients backwards through the network. The mathematical foundation is presented in Equation 2.10, with learning rate η and loss E.

$$w_{ij} = w_{ij} + \triangle w_{ij} \quad \triangle w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}} \tag{2.13}$$

This formula is a generalization of the perceptron update function (see Equation 2.5) that is suitable for a multilayer network. These gradient-based optimization functions update the weights away from the direction in which E increases the most. Neural networks are usually trained until the loss function converges at a local minimum, at that moment the weight will also be minimal updated and they will not change significantly.

It is very expensive to calculate the gradients for the entire data set to update a single weight. A more efficient alternative of gradient descent, called stochastic gradient descent (SGD), was proposed to solve this. Instead of using all datapoints, SGD uses a single input sample to update the weights. Although SGD is a lot more efficient, it heavily suffers from noise and single outliers causing high variance. This is why in practice most neural networks use batch gradient decent, where a subset (bigger than one sample) or mini-batch of the full data set is used to compute the gradients.

Many alternative optimizers based on gradient descent have been proposed. The most widely used one is Adaptive Moment Optimization (Adam) (Kingma & Ba, 2014), likely because of its relatively simple configuration and state-of-the art results. Adam uses an exponentially decaying moving average over past gradients, which limits the effect of a single outlier. Adam also uses an adaptive learning rate based on the first and second moments of the gradients. All models in this thesis use Adam as the optimizer.

2.2.6 Dropout

Dropout is the most important and successful regularization technique in Neural Networks (Zaremba, Sutskever, & Vinyals, 2014). The idea of dropout is to omit certain randomly selected units during the training (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). As the learning should correct for the missing units it becomes more robust to noise and focuses on the broader patterns, reducing overfitting. Dropout can be applied for a specific layer within the network, it may be applied on any or all hidden layers as well as the input layer. Dropout is not used on

the output layers. Dropout is used to make the learning more robust to noise, it is therefore not applied during testing. During testing all units are therefore fully utilized.

2.3 Natural Language Processing

Natural Language Processing (NLP) is a research field focusing on representing language as computer algorithm and computer-based language processing. While computational linguistics, a subfield of NLP, traditionally focused on the formal analysis and syntax. Current research in this domain has shifted to statistics, and more recently to (latent) semantics. The latter is possible due to the developments in deep learning. Language is built upon characters as the atomic building blocks which can be used to form continuously higher-level representations of words, sentences and full texts. Although languages adhere to certain rules and conventions, there are many ambiguities and irregularities that make language hard to understand for computers.

The most relevant aspects and methods in NLP will be discussed in this section. Firstly, an overview of the traditional approaches will be presented. Next, the statistical approaches such as Bag-of-Words and Latent Semantic analysis will be explained. Lastly, the more recent neural approaches will be discussed. This includes Recurrent Neural Networks, Convolutional Neural Networks and Transformer models.

2.3.1 Traditional approaches in NLP

Hidden-Markov Models

Traditionally it has been challenging to find a model for representing language. The earliest representation made the assumption that language can be represented by modelling the probabilities of word occurrences. In such models, the assumption is that next word depends solely on the previous words. These n-gram models predict the probability of the next words based on the previous words, or in probability terms: $P(x_i \mid x_{i-(n-1)}, \ldots, x_{i-1})$.

A traditional approach for modelling these statistical properties is using Hidden Markov Models (HMM). Hidden Markov Models try to predict the probability of hidden events based on the observed events. An example in Natural Language Processing is predicting the part-of-speech tag based on the words alone, the part-of-speech tag is not in the text itself but can be inferred from the word sequences. The hidden Markov model can be characterized by three fundamental problems (Rabiner & Juang, 1986):

- **Problem 1 (Likelihood):** Given an HMM $\lambda = (A, B)$ and an observation sequence O, determine the likelihood $P(O|\lambda)$
- **Problem 2 (Decoding):** Given an observation sequence O and an HMM $\lambda = (A, B)$, discover the best hidden state sequence Q.
- **Problem 3 (Learning):** Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B.

A more intuitive example will be explained here first, afterwards the application of Hidden Markov Models for Natural Language Processing will be specified. Let's assume we want to predict the weather based on the actions of a specific person. We don't have any information on the weather itself, but the actions of the person are a likely indicator for the weather. The activities the person can perform are: *walking* and *cleaning* and *shopping*. The options for the weather are: *sunny* or *rainy*. More formally the HMM could be defined as:

States	(*sunny*, *rainy*)
Observations	(*walk*,*shop*, *clean*)
Start probability	{*sunny*=0.6, *rainy*=0.4}
Transition probability	{*rainy* : {*rainy*: 0.7, *sunny*: 0.3}, *sunny* : {*rainy*: 0.4, *sunny*: 0.6}}
Emission probability	{*rainy* : {*walk*: 0.1, *shop*: 0.4, *clean*: 0.5}, *sunny* : {*walk*: 0.6, *shop*: 0.3, *clean*: 0.1}}

The start probability represents the belief prior to getting any information of the actions performed by the person, this could for instance be based on the knowledge that it generally rains more often. The transition probability represent how likely it is that there is a change in the state,

given a certain state. So if it rains today, there is a 70% chance of it raining tomorrow as well and a 30% chance of it being sunny. The emission probability represents how likely it is that the person performs a certain action given the state. If it is rainy there is a 10% chance the person will walk, a 40% chance the person will shop and a 50% chance of the person cleaning. A visualization of the HMM is given in Figure 2.6. The Hidden Markov model now allows to predict the hidden states, based on the actions that are performed. One might predict that it is sunny if the person is walking, or that it is rainy if the person is cleaning.

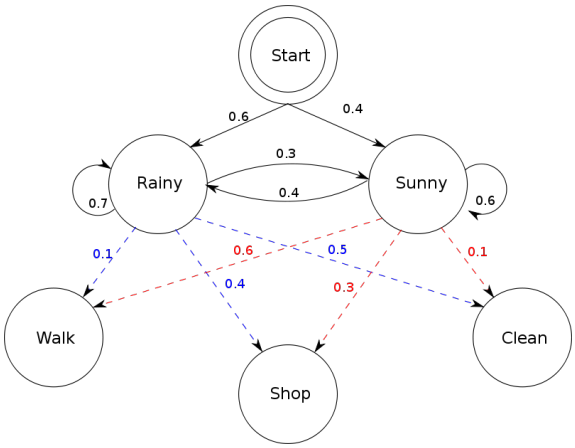


Figure 2.6: A visualization of the Hidden Markov Model example. Image from *Hidden Markov Model Graph* (n.d.)

Hidden Markov Models can be used in many Natural Language Processing problems. Examples are the prediction of the next word, part-of-speech tagging (Merialdo, 1994) and even document classification. One of the main uses of Hidden Markov Models is for n-gram representation. Instead of using the direct n-grams probabilities learned from a text, a Hidden Markov Model can be used to model the probabilities of n-grams. Until the introduction of neural models, these Markov Models were the best-performing models for representing language. The disadvantage of these models is however that they do not extract a lower-dimensional representation of language. They model the surface level of the probabilities but not the underlying mechanism that creates language.

Maximum-entropy models

Many Natural Language Problems can be reformulated as tasks where the probability of a class a occurring for a context c should be predicted. Predicting the part-of-speech tag for a word in a sentence, could be reformulated as predicting the probability for each part-of-speech tag for the context. The context depends on the specific problem, but could for example be a single word, a n-gram or a full document. The context could even consist of non-word labels, like the part-of-speech tag or the relations between words.

The task is to construct a stochastic model that accurately represents the random process at hand, so estimating $p(a|c)$. To estimate this probability many examples pairs (a_n,c_n) of a context with the class are used. The goal is then to construct a model that was most likely to generate the training samples. In maximum entropy the most uniform distribution is selected among all possible models. Equation 2.14 shows the formula to calculate the entropy for a model. The model with the highest entropy among all possible models in the set P is then selected, as in Equation 2.15.

$$H(p) = -\sum_{a,c} \tilde{p}(c)p(a|c)logp(a|c) \tag{2.14}$$

$$p = \underset{p \in P}{argmax}\, H(p) \tag{2.15}$$

Maximum entropy models have a close fit to the observed data and will not make any assumptions beyond the evidence. The main advantage of not making any assumptions, is that arbitrarily complicated features can be used to model the data, including inter-dependent features.

Table 2: Bag-of-Words representation of the example sentences: 1. "John likes to watch movies. Mary likes movies too." and 2. "Mary also likes to watch football games."

	John	likes	to	watch	movies	Mary	too	also	football	games
Sentence 1	1	2	1	1	2	1	1	0	0	0
Sentence 2	0	1	1	1	0	1	0	1	1	1

2.3.2 Statistical NLP

Basic Word & Document representations

To model language, a computer needs the language to be in a format that it can use. Neural networks, for instance, require a numerical input. Before a text can be used as input for a neural network, it needs to be transformed to a numerical input. This transformation is a difficult task, as it requires to represent the words and their connections in the best possible way despite the many ambiguities in language.

The simplest approach is one-hot encoding, where all words in a corpus are represented by a one-dimensional vector of the size of the vocabulary. For example, the sentence "Words need to be numeric" can be represented as [Words=10000, Need=01000, To=00100, Be=00010, Numeric=00001]. The main drawback of this approach is that the vector representing a word scales with the size of the vocabulary. This results in an extremely sparse matrix, where almost all values are zero. These encodings disregard any contextual information and will represent different meanings of the same word in the same way. For example the encodings of "play" in "I saw her play the piano" and "I watched the play" will be the same, even though their meaning is different.

Bag-of-Words (BOW) is a similar approach that can be applied to sentences or documents, where the document is encoded as the counts of each word. The sentences "John likes to watch movies. Mary likes movies too." (sentence 1) and "Mary also likes to watch football games." (sentence 2) would be encoded as shown in Figure 2. In BOW the order and structure of the words within the document is irrelevant, only the frequency is used (Salton, 1971). The idea behind BOW is that documents that contain similar words, are likely similar in content and meaning. Although, this idea seems simplistic, it was quite revolutionary. Until the introduction by Salton, all models focused on representing language in the sequence in which it was presented. Salton showed that disregarding the sequence information can still produce valuable representations.

In the Bag-of-Words representation all words are equally important. This is however not representative of language, where certain key words contribute more to the meaning of the text than words like "a" and "and". Luhn (1957) was the first to introduce some form of term weighting. Jones (1972) continued upon this and introduced Term frequency-inverse document frequency (TF-IDF). TF-IDF takes into account the number of occurrences in the corpus. Some words occur very often and are not very informative on their own, like "the", "of" and "and". In TF-IDF common words get a lower weight, words that are only common in the current document get a higher score. Words like "football", "game" and "ball" could for example get a higher weight in a document about sports.

The formula to calculate the term weights is given in Equation 2.16, where tf_{ij} is the number of occurrences of word i in document j, df_i is the number of documents containing the word i and N is the total number of documents.

$$w_{ij} = tf_{ij} * log(\frac{N}{df_i}) \tag{2.16}$$

These representations have all been focused around single words. However these models can also be extended to use n-grams. An n-gram is a sequence of n words. For instance the sentence "She plays the piano" would exist out of the n-grams shown in Table 3. n-grams can for instance be used together with Bag-of-Words, where instead of words the counts for each n-gram is used. The advantage of n-grams over words is that they provide more context. Using n-grams for the sentences "I saw her play the piano" and "I watched the play" will give the bigrams "her play" and "the play". Although the word "play" is ambiguous, the bigrams are not.

Table 3: n-gram representations for the sentence "She plays the piano"

4-grams:	"She plays the piano"
3-grams:	"She plays the", "plays the piano"
2-grams:	"She plays", "plays the", "the piano"
1-grams:	"She", "plays", "the", "piano"

Latent semantic analysis

A disadvantage of the Bag-of-Words approach is that due to the size of a lexicon, the dimensionality of the used vectors can become huge. For a language such as Dutch, 250k dimensions would not even be exhaustive. Additionally, although synonyms represent the same semantic information, they end up in completely different indices in the BOW list. The question then was, how to find lower-dimensional vectorial spaces which can be based on an exploitation of the correlation and redundancy in the surface terms. The idea emerged that methods such as principal-components analysis or factor analysis could be used to find conceptual spaces as opposed to the raw surface-keyword spaces.

Latent semantic analysis (LSA) is such a technique to extract a lower dimensional representation. The LSA algorithm assumes that words that are similar in meaning will occur in similar texts. The LSA algorithm was among the first widely used distributional semantic models. The LSA algorithm first calculates a term occurrence matrix, such as the TF-IDF matrix. LSA then finds a low-rank approximation of this term occurrence matrix, in other words LSA finds a low-level representation of the term occurrence matrix. This is necessary as the term occurrence matrix can become very large and therefore sparse, most elements in the matrix will be zero. There are many methods to calculate this low-rank representation, the most prominent being single value decomposition and the related method principal component analysis. Single value decomposition (SVD) is a method that decomposes the original matrix into three separate matrices that combined make up the original matrix. These matrices could be interpreted as a matrix for the documents, a weight of the relative importances of each topic and all the words. The words will however be discarded, and only the themes in the text will be kept, which lowers the number of dimensions. Principal Component Analysis uses SVD within its algorithm. However PCA subtracts the mean of each sample before applying SVD or the eigenvectors of covariance. SVD and PCA preserve the similarity structure among different documents, but lower the number of dimension needed. In this way they are able to represent the topics within a text in a lower dimension than the TF-IDF matrix, without losing much of the information.

Dumais, Letsche, Littman, and Landauer (1997) showed how powerful using LSA can be. They applied LSI, which is very similar to LSA, to a data set of French and English document pairs. Firstly, they showed that documents that have similar topics are close together in the vector space. More importantly however, they were able to create vectors for which the English and French document were aligned in the vector space as well. This means they were able to retrieve French documents with similar topics based on the vector embedding of an English document.

2.3.3 Neural NLP

Neural word representations

Although methods like Bag-Of-Words and LSA are well capable of representing documents, they can not handle very big lexicons. The matrix size for a Bag-Of-Words representations grows quadratically with the vocabulary size. Representing a document or language with n different words requires a matrix of size n^2, which becomes impossible for language like Dutch with over 250k words. LSA is able to extract lower-dimensional representation from these matrices, but to do so uses Singular value decomposition (SVD). SVD calculates the inverse of the original matrix, which is computationally very heavy for big matrices.

To combat this estimation problem, other model estimation methods were needed. Neural networks, i.e., autoencoders (Hinton & Salakhutdinov, 2006) can be used to perform dimensionality reduction in an incremental manner, over minibatches. In this manner, data sets of a virtually unlimited number of documents can be used and the number of words in the lexicon (the dimensionality) can also be huge. Computing time becomes the essential limiting factor since contemporary

disk storage is massive. If the learning rate is small enough, a subspace or embedding can be estimated which is reliable and similar to what the linear-algebra methods achieve.

Word2Vec

Word2Vec Mikolov, Chen, et al. (2013) is among the most famous prediction-based neural models to represent words. Word2Vec learns the embeddings, i.e., linear 1D vectors representing a 'semantic' latent space, based on a large text corpus.

Although Word2Vec has become famous for being able to calculate the semantic similarity of two text objects by using the two embeddings and simple cosine similarity function, earlier models like LSA were also capable of doing that. A famous example from Mikolov, Yih, and Zweig (2013) are the vectors for "man", "king", "woman" and "queen". In this example the vector for "king" minus the vector for "man" and plus the vector "woman" results in a vector close to the vector "queen". This is visualized in Figure 2.8. This example is very similar to the vector representation shown by Dumais et al. (1997), visible in Figure 2.7. In LSI contextually similar documents in different language end up in the same subspace, for Word2Vec similar semantic concepts end up in the same subspace. For both, words or documents that are similar in meaning and context are mapped to a similar subspace. Word2Vec does outperform the previous methods if enough data is present to fit the high number of parameters (Altszyler, Sigman, Ribeiro, & Slezak, 2016). Word2Vec offers a better-performing alternative for count-based models when using large data sets.

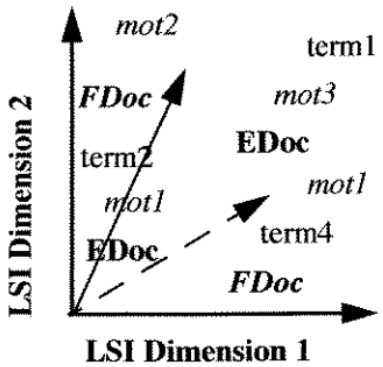


Figure 2.7: Visualization of the query vectors and the vectors created by LSI. Monolingual documents are located at the vector sum of their constituent terms. mot is the Monolingual Term, FDOC and EDOC the French and English Document respectively. Image from Dumais et al. (1997)

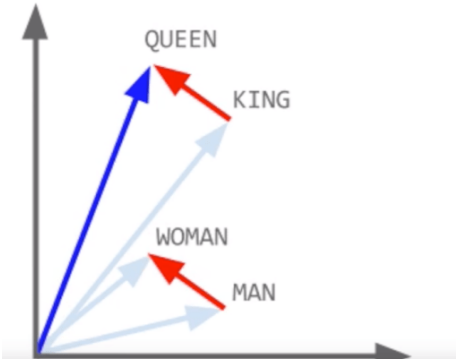


Figure 2.8: Visualization of the vectors "man", "king", "woman" and "queen". Image from Medatwal (2018)

Table 4: Actual probabilities from a 6 billion word corpus. Image from Pennington et al. (n.d.)

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

GloVe

One improvement upon Word2Vec is a method called Global Vectors for Word Representation (GloVe), proposed in Pennington, Socher, and Manning (2014). GloVe takes into account the probability of words occurring in certain contexts. Instead of using raw probabilities it uses the ratio of probabilities. To clarify this, an example for the words *ice* and *steam* is taken. As could be expected, the word *ice* co-occurs frequently with the word *solid* and the word *steam* co-occurs with the word *gas*. Both words co-occur frequently with the word *water* and rarely with the word *fashion*. The raw probabilities used in for instance Word2Vec would be P(k|ice) and P(k|steam), with k being *solid, gas, water* or *fashion*. Although these probabilities do show the expected results, the differences are very small. In GloVe the effect of *water* and *fashion* is cancelled out, resulting in high correlations for the specific terms *solid* and *gas*, as would be expected. These representations are better able to discriminate between relevant words.

Recurrent neural networks

Recurrent neural networks (RNN) are a special type of neural network that is able to learn from sequential data, first introduced by Rumelhart et al. (1986). The earlier discussed neural networks only process one fixed-size sample at a time. There is no dependence across multiple inputs. For many situations, the interdependence between different inputs is crucial. For example for natural language such as words in a sentence or for time series data such as stock market prices. When there is a dependence between different inputs and the full input is of a variable length, a RNN structure is more appropriate. As opposed to MLPs or Convolutional Neural Networks (discussed later in 2.3.3), Recurrent neural networks do take into account temporal dependencies. They do this by using connections between units, that allow for information to flow back. These recurring connections allow RNNs to keep track of previous states.

The input to a RNN is given as a sequence of inputs $[x_1, x_2, x_3 ... x_t]$, where t is the time step in the sequence. For example a sentence would be represented as $[recurrent, neural, networks, are, great]$, where input t_1 is $recurrent$, input t_2 is $neural$ and so on. The network receives this entire sequence with all elements, instead of a single input.

There are five general setups for neural networks, also visualized respectively in Figure 2.9:

- **One-To-One (1:1)**: This network structure deals with a fixed size input and maps it to another fix sized input. It also assumes interdependence between the inputs. This network structure is used in MLPs and CNNs. A RNN can also be used for this but their ability for sequential processing and temporal dependence are not used. An example tasks is image classification.

- **One-To-Many (1:N)**: This network structure applies to a RNN that receives a single input and generates a sequence of outputs. An example task is generating a sentence (sequence of words) from an image.

- **Many-To-One (N:1)**: This network structure applies to a RNN that receives a sequence of inputs and outputs a fixed-size output. Examples of this are generating an image from a sentence (sequence of words), classifying a sentence into a certain sentiment or representing/mapping a variable-length sequence as a fixed-size vector.

- **Many-To-Many (N:M)**: This network structure applies to a RNN that receives a sequence of inputs and outputs a sequence, that is not synced with the input. An example of this is machine translation, where for example English sentences are translated to French sentences. Some English words might become multiple words in French or the other way around.

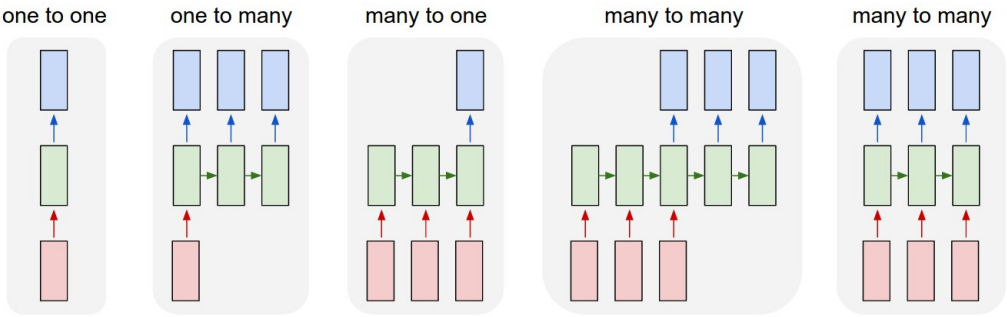


Figure 2.9: Visualization of network structures. Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state. Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like. Image and description from Karpathy (2015)

- **Synced Many-To-Many (N:N)**: Similar to the other many-to-many structure, this RNN structure also receives a sequence of inputs and outputs a sequence. However in this case the input and output sequence are synced. An example of this is predicting whether the price will be higher or lower after an hour, for each time point in a sequence.

Using regular backpropogation is not feasible for RNNs. Doing regular backpropogation would require to expand the computational graph of an RNN at a single input step in the sequence to obtain the dependencies among model variables and parameters. For a sequence of length n, this would require n matrix products as the first input in the sequence can depend on the last one. For long sequence this becomes too computationally expensive.

Jordan (1989) was the first to introduce recurrent connections in a network. The goal was to introduce memory to the network, such that inputs later in the sequence depend on earlier inputs. Jordan achieved this by creating a one-to-one connection between each output $\hat{y}$ and a state unit with a fixed weight of 1.0 that are again connected to the hidden states h. The state units keeps track of the previous states, similar to a running average. The Jordan network does not learn because of the fixed weight. Elman (1990) introduced a similar idea but used a recurrent connection from each hidden state in the sequence to context units (similar to the state units), which are again connected to the hidden units. Where in Jordan networks the output is connected to the hidden units, in Elman networks the hidden units are connected to themselves. Next to this the recurrent connections in the Elman network are actually trainable parameters. The memory units try to remember the past hidden states and learn useful weights to encode the temporal properties of the input. Figures 2.10 and 2.11 show visualizations of the two networks.

Inspired by this earlier work, Williams and Zipser (1995) introduced an adaption of backpropagation called backpropagation through time (BPTT). BPTT first unfolds the RNN in time into an unfolded network with k inputs and outputs. Every network copy of this network shares the same parameters, allowing to use the regular backpropagation algorithm. Backpropagation is used here to find the gradient of the cost with respect to all the network parameters. Regardless of how the training cost is defined, the aggregated cost is the average over all separately calculated time steps. Robinson, Hochberg, and Renals (1996) showed how successful BPTT can be for temporal data, such as speech as used in his research.

Although basic RNNs are able to handle sequences of arbitrary length, they have difficulties with long-term dependencies. The number of hidden layers in a RNN scales with the input length, meaning that long inputs will have many layers. Gradient are calculated based on the gradient of the layer before. If the update with respect to the last layer is below 1, the gradient becomes smaller. If the update is above 1, the gradient becomes larger. With many layers, there will be many of these multiplications. This can cause two problems: vanishing gradient or exploding gradient. In case of a vanishing gradient, the gradient approaches zero resulting in the weights not being updated. In case of an exploding gradient, the gradients become extremely large resulting in previously learned information being lost. These problems cause the difficulty to learn long-term dependencies in basic RNNs.

The long short-term memory network (LSTM) (Hochreiter & Schmidhuber, 1997) was intro-

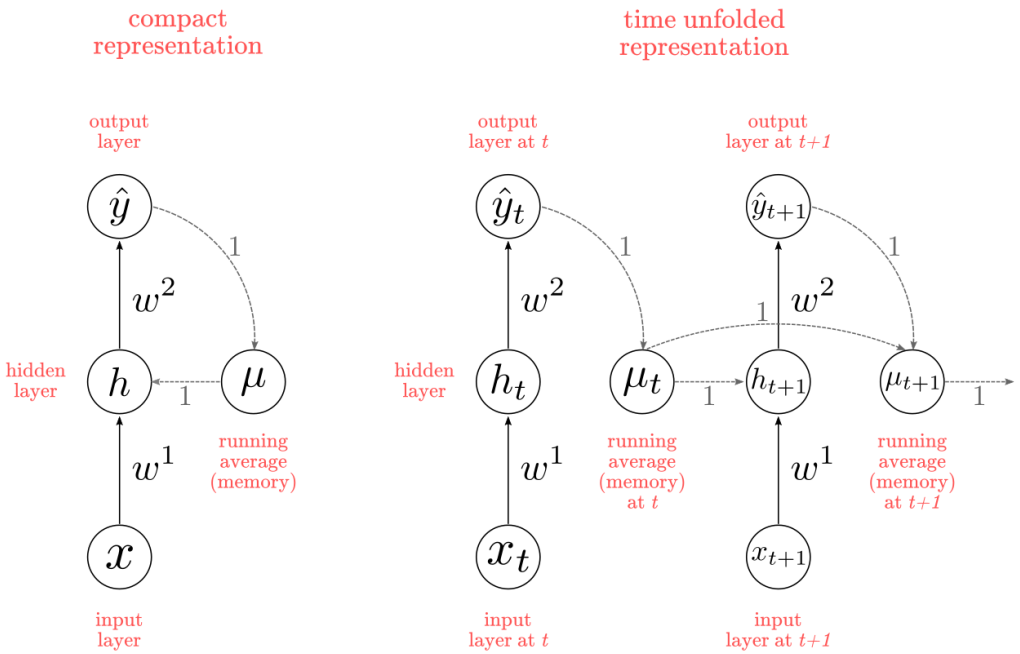


Figure 2.10: Visualization of the Jordan network, with on the left side the compact version and on the right side the unfolded representation. Image from Caceres (2020).

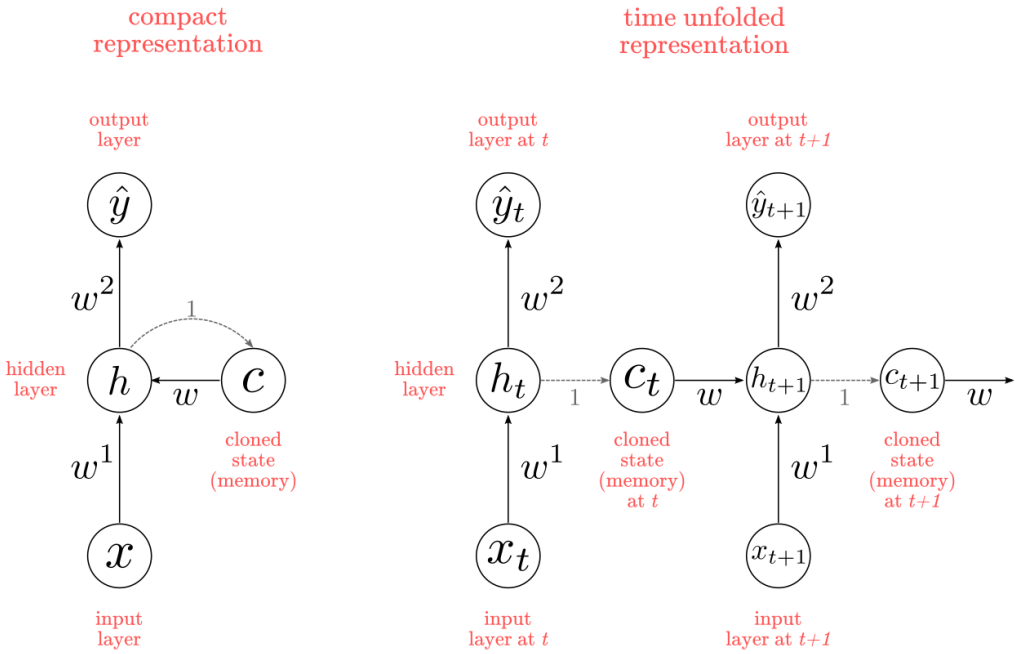


Figure 2.11: Visualization of the Elman network, with on the left side the compact version and on the right side the unfolded representation. Image from Caceres (2020).

duced to tackle the vanishing gradient problem. The LSTM introduces a memory cell which controls the information that gets through. The LSTM has three of these gates: a forget gate, an input gate and an output gate. The forget gate determines how much information from the previous cell should be kept. The input gate determines which information will be updated and what new candidate values can be added to the state. The output gate filters the output such that it only contains the outputs that should be kept. With these methods the LSTM is able to combat the vanshing gradient problem. Although it still suffers from the exploding gradient problem, this makes the LSTM better suited to learn long-term dependencies.

A variation on the LSTM is the Gated Recurrent Unit (GRU) introduced by Cho et al. (2014). The GRU combines the forget and input gates into a single gate called the update gate. It also merges the cell state and the hidden state. This results in a simpler model with fewer trainable parameters, making it faster to train. If the sequences are long and the data sets are small, the GRU is shown to perform equally well as the LSTM. In other scenarios the LSTM however may outperform the GRU (S. Yang, Yu, & Zhou, 2020).

Alternative methods: Convolutional Neural Networks

For completeness and comparison purposes it is necessary to mention convolutional neural networks (CNNs). These algorithms are very effective in image processing, so it seemed reasonable to explore their virtues in the area for document processing (Kalchbrenner, Grefenstette, & Blunsom, 2014). The advantage from CNNs over RNNs is that they only use local context as opposed to using the full context. The disadvantage is of course that long-term dependencies can not be included in this.

Traditional multi-layer perceptrons (MLP) are useful as a non-linear transform operator, but they are not well suited for high-dimensional data like images and text. This is because it expects a vector input and all layers are fully connected to the next. Next to this using a vector loses the spatial information. The location of features relative to other features is not represented in a vector. The context and order in which words are presented are not taken into account. The main issue is however the enormous number of parameters that would be needed for high-dimensional data. The input vector of for instance an image of size $h \times w$ would be sized $h \times w$. This vector would be multiplied with the weights in each layer. Lets assume we have an image of 32×32 and a single fully connected output layer of size 16. Even with this relatively small image and network there are already 16 384 parameters that need to be learned by the network. For an image of size 100×100 and the same network there would already be 160 000 learnable parameters, for a medium sized image of 500×500 there would be 4 million learnable parameters. Clearly, the number of parameters explodes for higher image sizes making the training of the MLP very unstable and inefficient.

To combat these difficulties Convolutional Neural Networks (CNN) were introduced. A CNN combines convolutional layers with pooling layers to extract features that are progressively higher level. The principal of convolutional and pooling layers was first introduced in the Neocognitron (Fukushima, Miyake, & Ito, 1983), which formed the basis for the CNN introduced by LeCun, Bottou, Bengio, and Haffner (1998).

Convolutional layers are at the core of CNNs. A convolutional layer consists of a set of filters or kernels. A kernel is a learnable weight matrix that is optimized during training. The size of the kernel corresponds to its receptive field. For each convolution layer the input into the kernel is convolved, which is an element-wise product, with the weight matrix. This computed matrix is then summed together to produce a scalar entry into the feature map. Figure 2.12 shows a visualization of this process. In the CNN only the kernel weights have to be learned, in this example 9, as opposed to 25 weights per neuron in the MLP.

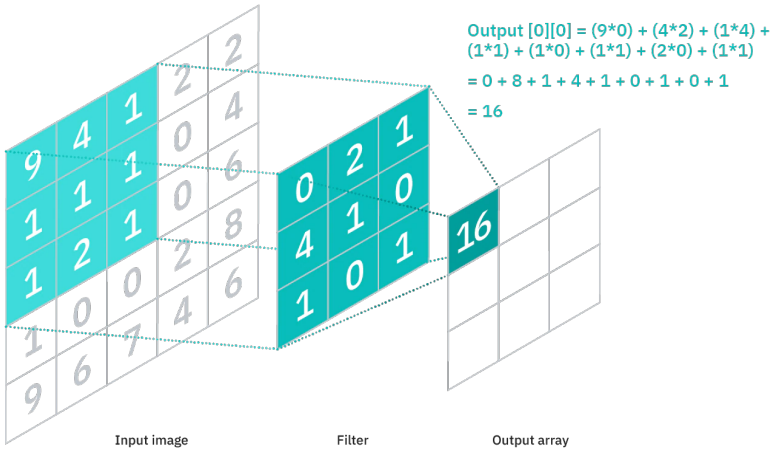


Figure 2.12: An example showing the process how a feature map is calculated within a CNN with a single kernel. Image from Education (2020)

The kernel is moved across the entire input to form the full feature map. The step size of this sliding window is called the stride. A stride of one means the filter will move by one pixel in the input, a stride of two will be two steps etc. Figure 2.13 shows an example. The kernel size with the specific stride might not match the input size perfectly. This would mean that some of the inputs will not be taken into account. To counter this padding can be applied. Padding adds additional pixels to the image to make sure the kernels fit in the input and all pixels in the input are taken into account. The most used option is zero-padding but different values, such as replicating the border, can be used. If no padding is applied the values will be ignored and dropped from the image.

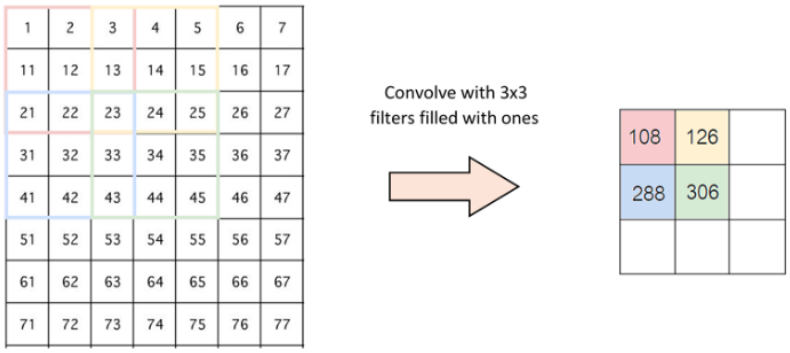


Figure 2.13: An example of a kernel being moved over the image with a stride of 2. Image from Prabhu (2018)

CNNs usually combine the convolutional layers with pooling layers. Like convolutional layers, pooling layers also move a sliding window over the input. Instead of convolving the input with a kernel, the pooling layer applies a pooling method to the input window. The two most-used pooling methods are *max* and *average* pooling, where respectively the maximum input is used or the mean over the values in the input window is taken. Figure 2.14 shows an example of both pooling methods.

To use the features extracted by the convolutional and pooling layers, most CNNs use a fully-connected layer. The fully-connected layer flattens the input and allows it to be used in tasks such as classification or regression.

Sequence-to-Sequence Models

Sequence-to-Sequence models were first introduced in 2014 (Sutskever, Vinyals, & Le, 2014). The aim of the model is to map a fixed-length input to a fixed-length output, however the sequences do not have to be the same length. An example is mapping the English sentence "I like to eat fruit"

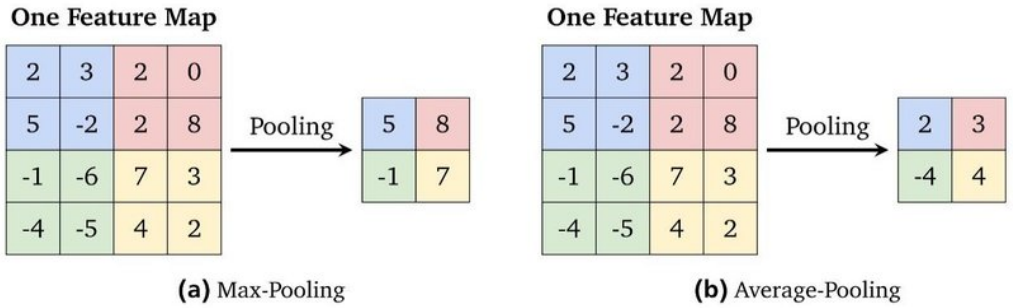


Figure 2.14: Example of *max* and *average* pooling with a 2×2 window size. Image from Guissous (n.d.)

to the Dutch sentence "Ik eet graag fruit". The English sequence consists of five words, the Dutch one contains four words.

The Encoder-Decoder model is one of the most famous and widely used Sequence-to-Sequence models. It exists of three elements: the encoder, an intermediate vector and the decoder. Conceptually the encoder gets an input that it tries to map to the intermediate vector. The decoder then tries to map the intermediate vector to another vector. It is important for the encoder to create an intermediate vector that encapsulates the information for all input elements, otherwise the decoder will not be able to predict an accurate output. The output depends on the task, in sequence-to-sequence tasks the output is a fixed-length vector. The encoder and decoder are neural networks on their own. Usually a stack of several recurrent units are used, as they are best fitting for sequential data. It is possible to use other architectures within the encoder-decoder that might fit the data better. For example when using an encoder-decoder architecture for images a CNN architecture will likely be better.

The major drawback of the encoder-decoder structure is that all information has to be encoded in the intermediate vector. The predictive power of the decoder depends on this. The longer the sequence, the harder it is to encode everything in the intermediate vector.

Attention

For long sequences the performance of the encoder-decoder becomes problematic. This is because it is very unlikely that it is needed to use the whole sequence for all parts of the output. If for example a whole paragraph should be translated to a different language, the contribution of the first sentence to translate the last sentence is likely small. The contribution of the last sentence to translate to the last sentence in the translation is however likely much bigger. In a regular encoder-decoder architecture both the first and last sentences would be handled exactly the same and they both need to be encoded in the same fixed-size intermediate vector. Bahdanau, Cho, and Bengio (2014) proposed a simple but elegant solution for this problem called attention. The idea is that a relative importance is given to each input word, next to the context vector for all the input words.

The attention mechanism can be divided into three steps: computation of the alignment scores, the weights and the context vector. Let's assume an encoder-decoder model. The encoder representations of the entire input sequence will be represented as the sequence $(h_1, ..., h_{T_x})$, in simple models only the last encoder state is used. The output of the encoder for the input at position i is given by s_i. The context vector will be determined using the following steps:

1. **Computation of the alignment scores:** The alignment model calculates a score e_{ij}, which represent how well the input at position i and the output at position j align. To determine this alignment function $a(.)$ a feedforward neural network is used, which is jointly trained with the other parts of the network.

$$e_{ij} = a(s_{i-1}, h_j) \tag{2.17}$$

2. **Computation of the attention weights:** Once the alignment score are calculated a tanh and a softmax is applied to produce the attention weights. The advantage of using a softmax

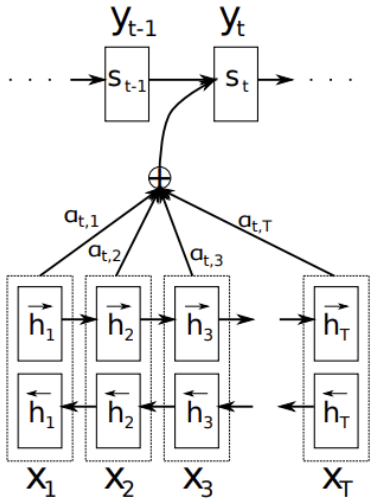


Figure 2.15: Diagram representing the structure of the attention model. Image from Bahdanau et al. (2014).

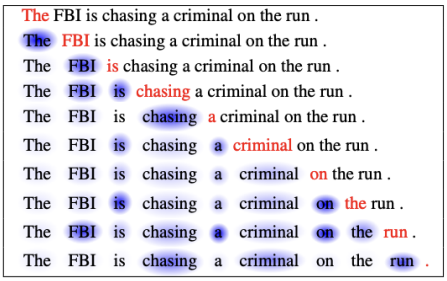


Figure 2.16: Visualization of what words self-attention pays attention to. Red represent the current input, the shade of blue represents the degree of attention. Image from Cheng et al. (2016).

is to create a probabilistic interpretation of the attention weights. The probabilistic interpretation has the advantage that one could also visualize to what the network pays attention (Vig, 2019).

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{2.18}$$

3. **Computation of the context vector:** The context vector for the input at position i is calculated by the sum of all hidden values h weighted by the attention weights.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.19}$$

With this attention mechanism, the decoder decides what part of the input sequence it should pay attention to. This approach relieves the encoder of the burden of encoding all information in the fixed-length vector. The encoder has the option to spread the information throughout the annotation sequence h. The decoder will then learn what parts of the attention sequence to use.

A number of different alignment functions and attention mechanisms have been proposed. The most notable being self-attention or intra-attention, proposed in Cheng, Dong, and Lapata (2016). The difference with attention as proposed in Bahdanau et al. (2014), is that here the relation between the inputs within the sequence are related with the other inputs in the same sequence. Figure 2.16 shows how the attention between words is distributed, for example showing that the word "FBI" is highly associated with the word "run". The self-attention for a sequence is calculated by determining the regular attention for this sequence, where the sequence is used both as the source and the target sequence.

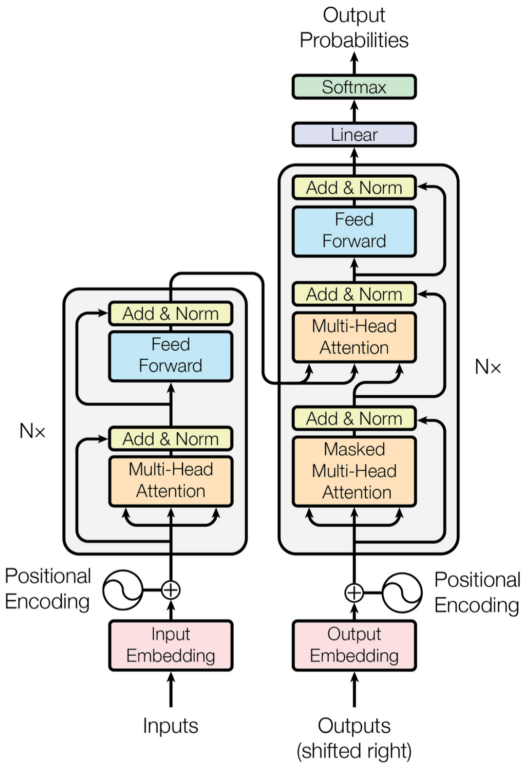


Figure 1: The Transformer - model architecture.

Figure 2.17: The architecture of the transformer model. Image from Vaswani et al. (2017).

Transformers

The transformer introduced by Vaswani et al. (2017) is an encoder-decoder model, but instead of using a recurrent neural network it only uses attention mechanisms. It was originally designed for machine translation (the language-to-language conversion task). The encoder consists of a stack of layers containing a multi-head self-attention mechanism and a fully connected feed-forward network. The decoder uses a similar stack, but adds a third sub-layer which performs multi-head attention over the encoder output. As the transformer does not use any recurrent networks, but still needs to keep the relative positions in the sequence, position encodings are added to the embedded representation of each word.

The original transformer uses a modified attention mechanism called multi-head attention. Multi-head attention as used in the original transformer model is expressed in Equation 2.20. It essentially maps a query and a set of key-value pairs to an output. The query (Q) is a matrix with the vector representation of one single word in the sequence, the keys (K) and values (V) are a matrix with the vector representations of all words generated in the sequence till thus far. K and V can be the same but they do not have to be. One could think of the procedure as calculating a weight for how each word in the sequence (Q) is influenced by all previous words (K), and then applying those weights to all the words in the sequence (V).

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.20}$$

Encoder

The encoder first applies the multi-head attention and then applies a fully connected feed-forward network (FFN). The FFN consists of two linear transformations with ReLU activation in between. The same linear transformation is applied to all inputs within the sequence.

Each layer consisting of the multi-head attention and the FFN has a residual connection around it. Residual connections are a type of skip-connection, that instead of learning unreferenced functions actually learn residual functions with reference to the layer inputs. The core idea behind this is that it is easier to optimize the residual mapping than to optimize the unreferenced mapping.

After each sub-layer a normalization layer is applied. This layer normalizes the sum computed over the input x and the generated output: $normoutput = Layernorm(x + sublayer(x))$

Decoder The architecture of the decoder is quite similar to that of the encoder, with some modifications. The first sublayer in the decoder implements a multi-head self-attention over the previous output of the decoder stack. The decoder only attends to preceding words and, as opposed to the encoder stack, can not use any future words. This is done by applying a mask.

The second and third layer are similar to the encoder layers. The second layer implements a multi-head self-attention mechanism. The attention mechanism is applied on the queries of the previous decoder layer and the keys and values from the encoder output. Since the decoder receives the output of the encoder, it can attend to all of the words in the input sequence. As in the encoder, the third layer is a fully connected feed-forward network. The decoder also uses residual connections and normalization layers.

Comparison of neural networks architectures

The authors of Vaswani et al. (2017) show that the transformer can outperform other models in the task of machine translation. Transformer model are much more parallelizable and require less time to train, a huge advantage compared to the other models. Where RNN models need to be trained in a sequential fashion, for transformers it is not necessary to process the beginning of the sequence before the end. This allows for much more parallelization and therefore a reduction in training time. Some have proposed to use CNNs instead of using RNNs, as CNNs are also parallelizable. In CNNs however the kernel width directly affects the ability to learn long-term dependencies. Learning long-term dependencies would require the use of large kernels, which are computationally expensive. The computational complexity per self-attention layer is a lot lower. The self-attention layer connects all inputs within a sequence with a constant number of operations. The lower the number of operations needed to connect these, the better capable a model will be to learn long-term dependencies. The connections are also referred to as the forward and backward path lengths. Transformers do suffer from a high computational complexity when the sequences are very long. To combat this problem the self-attention could be restricted to a specific neighborhood of size r instead of the full input sequence. This would increase the maximum path-length but lowers the computational complexity. Table 5 shows a comparison of the complexity of layer-wise computations in different neural architectures, showing the advantage of restricted self attention in transformers.

Table 5: Complexity per layer, minimum number of sequential operations and maximum path lengths for layer types in commonly-used neural architectures. With n being the sequence length, d being the representation dimension, k being the kernel size and r the neighborhood size. Table from Vaswani et al. (2017)

Layer type	**Complexity**	**Sequential Operations**	**Maximum Path Length**
Convolutional layer	$\mathcal{O}(k \cdot n \cdot d^2)$	$\mathcal{O}(1)$	$\mathcal{O}(log_k(n))$
Recurrent layer	$\mathcal{O}(n \cdot d^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Self-Attention layer (full)	$\mathcal{O}(n^2 \cdot d)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Self-Attention layer (restricted)	$\mathcal{O}(r \cdot n \cdot d)$	$\mathcal{O}(1)$	$\mathcal{O}(n/r)$

Pre-trained models

The introduction of the transformer allows for very deep neural models to be applied to natural language processing tasks. Where the training of deep recurrent neural networks quickly became unfeasible, the transformer introduced the possibility to train NLP models with millions of parameters. These models however do require massive training data sets to fully train the parameters and prevent overfitting. For most tasks, a big labeled data set is not available however. The idea of the pre-trained models is to first learn a good representation for a large unlabeled text data set and to then use this representation as the starting point for other tasks. The learned general representation of language, what words are semantically similar etc. helps in learning a specific task. Fine-tuning pre-trained models has become the consesus (Qiu et al., 2020) and achieved competive model performance on almost all NLP tasks (Han et al., 2021). The most famous models ELMo,

GPT and BERT will be discussed here. There are many variations upon the basic structure of these models.

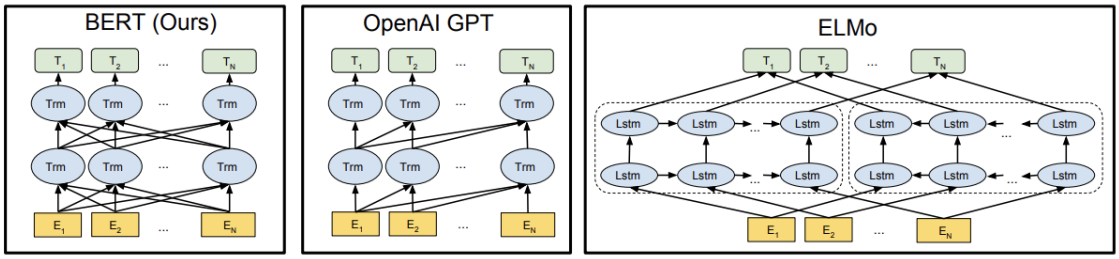


Figure 2.18: The pre-training model architectures of BERT, GPT and ELMo. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo does not use a Transfomer but uses a combination of a left-to-right LSTM and a right-to-left trained LSTM. Image from Devlin et al. (2018)

ELMo

Unlike the word embeddings discussed in Section 2.3.3, Embeddings from Language Models (ELMo) introduce embeddings that vary across linguistic contexts and still contain the complex characteristics of word use. ELMo word representations are functions of the entire input sentence. ELMo computes the word vectors using a two-layer bidirectional language model (biLM). The input to this biLM is computed from characters, not from words. A CNN is used to represent the raw word vectors from the characters. These vectors are then passed through the first layer of the biLM. A biLM combines a forward pass and a backward pass for each token. The forward pass of a language model uses the words in the sequence before the target token, the backward pass uses the words later in the sequence. The word vectors that are calculated in the first biLM are then passed through the second biLM layer. The final ELMo word representation is a weighted sum of the two biLM representations and the raw word vector.

Compared to the embeddings discussed in Section 2.3.3 ELMo is able to represent different meanings of words based on the context. Where GloVe and Word2Vec are not able to separate between the different meaning of "play" in "I saw her play the piano" and "I watched a play", ELMo would be able to differentiate between these two. This is because ELMo takes into account the context around the word, instead of representing the single word.

OpenAI GPT

The Generative Pre-Trained Transformer or GPT model (Radford et al., 2018) is the first model that combines the transformer architecture with the goal of making a model that learns a universal language representation that can be transferred to other NLP tasks. GPT optimizes a standard language model objective. The objective is shown in Equation 2.21, where $X = \{x_0, x_1, \ldots, x_n, x_n+1\}$ is a corpus of tokens. Here k is the window size, P the probability modeled by the Transformer with parameters Θ. The first token in the input sequence (x_0) and the last token (x_{n+1}) are special token that are added to the sequence, x_0 is the token [CLS], x_{n+1} is the token [SEP].

$$L_1(X) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta) \tag{2.21}$$

GPT computes the probability distributions by applying a masked multi-head self-attention over the input context tokens followed by a feed-forward neural network. The GPT model uses a similar architecture as the original transformer decoder, except that it removes the cross-attention.

GPT used hundreds of millions of parameters and took a month to train. This shows that it is unfeasible to train a similar model from scratch for each task. Using the pre-trained parameters of GPT it was possibly to achieve success in almost all supervised downstream NLP tasks.

After the introduction two extensions were created called GPT-2 (Radford et al., 2019) and GPT-3 (Brown et al., 2020). GPT-2 was aimed to learn multiple tasks with the same model. To achieve this an additional task objective was added. More importantly GPT-2 used a far bigger model and data set. The research showed that the capability of language models improves with the number of parameters and the data set size. GPT-3 again increased the number of parameters and the data set size, GPT-3 used 100 times as many parameters as GPT-2. Just like GPT-2

outperformed GPT-1, GPT-3 was able to show that increase the parameters and data set size improves the capability of language models. GPT-3 performance was specifically tested for few-shot, one-shot and zero-shot settings. In these settings the pre-trained model only gets a few or zero examples to perform the new specialised task. GPT-3 was shown to outperform state-of-the-art models in zero-shot settings.

Figure 2.19: Comparison of data sets and parameters used in the different GPT models.

Model Type	Data set Size	Parameters
GPT-1	Book Corpus (5GB)	117 million
GPT-2	WebText (40GB)	1.5 billion
GPT-3	Common Crawl, WebText2, Books1, Books2 and Wikipedia (45TB)	175 billion

BERT

Arguably the biggest breakthrough in Natural Language Processing has been the emergence of the Bidirectional Encoder Representations from Transformers model, better known as BERT Devlin et al. (2018). As opposed to GPT, BERT uses a bidirectional Transformer as its main structure. Where GPT uses autoregressive language modelling in the pre-training phase, BERT uses autoencoding language modelling. An autoregressive model uses past inputs to predict the future value. Autoencoding models like BERT however try to construct a representation that is robust to noise. The encoding should be such that if certain inputs are masked, the model is still able to reconstruct the original input. This is called masked language modelling (MLM).

Specifically BERT randomly replaces tokens in the input sequence by a special [MASK] token. The model objective is to predict the token that was in the input. Formally BERT maximizes the log-likelihood given in Equation 2.22. Here $X = \{x_0, x_1, \ldots, x_n, x_n + 1\}$ is a corpus of tokens. $\tilde{X}$ is the masked corpus of which m randomly selected tokens from X are masked. Again, P the probability modeled by the Transformer with parameters Θ. $[\text{MASK}]_i$ is the i-th masked token with y_i being the original input token.

$$L_1(X) = \sum_{i=1}^{m} \log P([\text{MASK}]_i = y_i | \tilde{X}; \Theta) \tag{2.22}$$

Besides the MLM objective, BERT also incorporates a next sentence prediction (NSP) objective. The goal of NSP is to predict whether two sentences follow each other. This should capture the relationship between sentences, which is important for some downstream tasks such as question answering. This relationship is not captured directly by the MLM objective alone. For the NSP task a binary classifier is used, which predicts whether two sentences are coherent. This is done by randomly selecting sentences A and B, where in 50% of the cases B indeed follows A and in the other half it is a random sentence from the corpus.

Tokenization

Both BERT and GPT do not use words as their input but are based upon tokens, tokens are pieces of words. The main benefit for using tokens over full words is the reduction in vocabulary size and at the same time it improves performance for very rare words (Wu et al., 2016). The smallest word pieces are characters, but as they are not very explanatory, a mix between wordpieces ranging from characters to full words is used.

GPT uses the spaCy tokenizer (*spaCy tokenization*, n.d.), which is a rule-based tokenizer. SpaCy tokenization splits sentences into words. It is able to separate words from punctuation but still combine some special characters into a single word. For example the sentence: "Don't you love tokenization? We do!" would be split into ["Do", "n't", "you", "love", "tokenization", "?", "We", "do", "!"]. Splitting into words is intuitive but results in massive vocabularies for big corpora. This would result in a huge embedding matrix and memory issues when using it. GPT-2 and GPT-3 use a simpler approach for the tokenization and only uses space tokenization.

All GPT models apply an encoding upon the tokens. GPT uses a byte pair encoding (BPE) for these determined tokens (Sennrich, Haddow, & Birch, 2015). After the pre-tokenization, BPE creates a set consisting of all symbols that occur in the vocabulary. It then learns rules to merge symbols in this set, until the vocabulary has attained the desired vocabulary size. For GPT this

size is set to 40 000. GPT-2 and GPT-3 apply a trick upon BPE, called Byte-level BPE. Here they use bytes as the base vocabulary, instead of all unicode characters. GPT-2 can tokenize all text without using an unknown symbol and with a vocabulary size of approximately 50 000.

BERT uses a tokenization method proposed in Schuster and Nakajima (2012) called WordPiece, which is very similar to BPE. WordPiece is initialized with a set of all characters in the training data. Similar to BPE it then learns merge rules. Instead of choosing the most frequent symbol pair, it merges the symbol pair that maximizes the likelihood of the training data.

The main advantage of using WordPiece and BPE becomes apparent when thinking of words like "hugger". The word "hugger" might be very rare and might not be in the vocabulary. The word could for example be split into the more meaningful tokens ["hug", "g", "e", "r"]. Although "g", "e" and "r" do not add much explainability, however the word the token "hug" might be more common and be present in the vocabulary. In this way the unknown words can be represented as a combination of known tokens, from which possibly the meaning can be derived.

Remarks on Language models
GPT-3 clearly outperforms others model in language modelling, but one has to be critical if this means that the model has an improved understanding of language. Brown et al. (2020) show that GPT-3 is capable of doing arithmetic without any task-specific training. GPT-3 is however only capable of doing this for two and three digit substractions and additions, see Equation 2.20. GPT-3 does not understand how to do these arithmetic tasks, but it is able to replicate them because it has seen these basic tasks enough times. Essentially one could argue that GPT-3 and any other current language model, is simply a very powerful lookup table (Qassemi, 2020). This however does not mean that it is any less useful in downstream tasks. For many tasks it is irrelevant how the representation is determined, as long as it is a good representation of language.

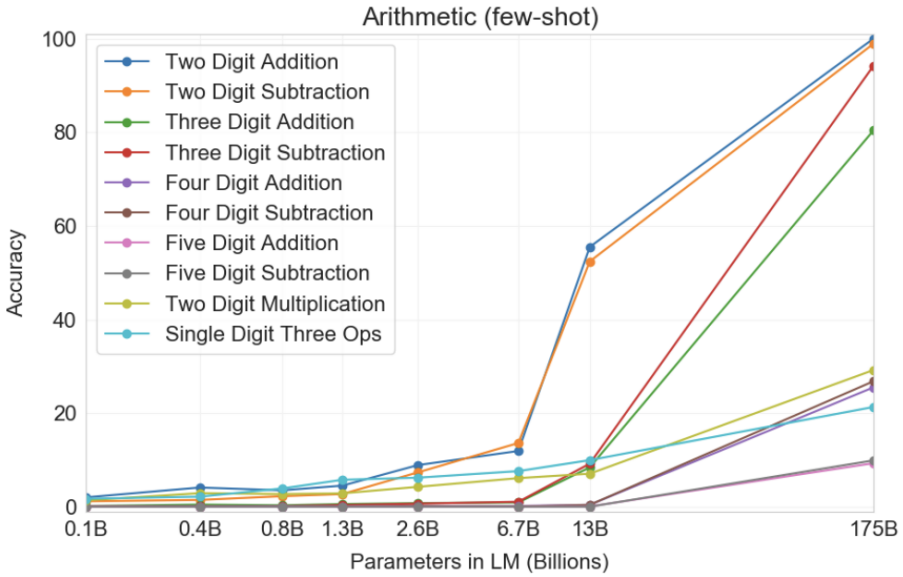


Figure 2.20: Results of GPT-3 on different arithmetic tasks in the few-shot settings. Image from Brown et al. (2020)

2.4 Related Work in document-analysis applications

2.4.1 Related Work in patent analysis

Despite the success of Neural NLP models in many tasks, most of the patent-analysis models still rely on traditional statistical models. Most of the techniques are based around the detection of key phrases. Hasan et al. (2009) and Balsmeier et al. (2018) developed models to measure patent novelty based on key phrases. Patents that contain key phrases that are only recently introduced in the vocabulary are valued as novel. A similar approach combined with a Bag-Of-Words model was used by Arts et al. (2021) to predict patent quality. deGrazia et al. (2020) use a TF-IDF based

model to determine the similarity between technologies described in the patent claims. Younge and Kuhn (2016) have developed a method to transform patent text into a vector and then compare these vectors. Their method relies on bag-of-words and TF-IDF. Kelly et al. (2018) have expanded this by applying a time-dependent adaptation of TF-IDF to identify important terms and compare patents in their use of these words.

There are some models that have focused specifically on topic modelling. Topic modeling is a technique to discover abstract topics in a collection of text documents. Kaplan and Vakili (2015) were among the first to use topic modeling in the field of patent analysis. In their approach they identified novel patents by identifying those patents that originate new topics. Ashtor (2019) created a LSA vector for each patent document and then clustered these used the similarity compared to patents within the cluster to predict citations. Teodoridis et al. (2020) have advanced this even further by tracing topics over time using a Hierarchical Dirichlet Process model. Although these models use somewhat more advanced NLP methods, they still rely on basic statistical word and document representations.

In the related task of patent classification more advanced neural methods have been proposed. The task in patent classification is to predict the technological category or the Cooperative Patent Classification (CPC) of the patent. Trappey, Hsu, Trappey, and Lin (2006) and Guyot, Benzineb, Falquet, and Shift (2010) were the first to apply simple Neural Networks to the patent classification task, with limited success. The first more advanced neural NLP model was the DeepPatent model proposed by S. Li, Hu, Cui, and Hu (2018). They proposed to use a CNN model with Word2Vec inputs. The advantage of their method is that they were able to utilize the entire text. As stated earlier CNNs are however not able to learn long-term dependencies and Word2Vec is not able to handle homonyms. J.-S. Lee and Hsiang (2019) proposed a classifier based on a pre-trained BERT model fine-tuned for patents. Their model only changes the softmax in the original vanilla BERT to a sigmoid cross entropy with logits function to handle their multilabel classification problem. Although their model is able to utilize the language capabilities of BERT, they do not use the full text.

2.4.2 Scholarly-document quality prediction

A related task is concerned with the automated evaluation of scientific articles or scholarly documents. Such a task would help publishers tremendously with pre-sifting of manuscripts that are submitted for peer review.

Scholarly documents suffer from similar difficulties when applying state-of-the-art language models as patents. Similar to patents, scholarly documents can become very long and are usually too long to be handled by models such as BERT. Although they are less formally organized than patents, scholarly documents also have a multi-section structure where the sections are heterogeneous (e.g., the Introduction/Methods/Results and Discussion format, 'IMRaD'). We will not go into depth into the field of scholarly-document quality prediction but will highlight two models/ideas that are relevant for our proposed approach.

The first model is proposed by P. Yang, Sun, Li, and Ma (2018) and deals with the multi-section problem. Their model, called Modularized Hierarchical Convolutional Neural Network (MHCNN), utilizes the differences in structure of each section of the academic paper. Figure 2.21 shows the visualization of their model. To combine the different section embeddings into a single document embedding they propose an attentive pooling layer. The attentive pooling layer uses the attention principal and weighs the contribution of each section to the document embedding. The attentive pooling layer is formally presented in Equations 2.23, 2.24 and 2.25 for a sequence $[c_1, c_2, \ldots, c_q]$ with q vectors. Where W_c is the weight matrix and b_c the bias vector, u_w is a vector that can be learned during training.

$$z_i = \tanh(W_c c^{(i)} + b_c) \tag{2.23}$$

$$a_i = \frac{z_i^T u_w}{\sum_k \exp(z_k^T u_w)} \tag{2.24}$$

$$s = \sum_i a_i z_i \tag{2.25}$$

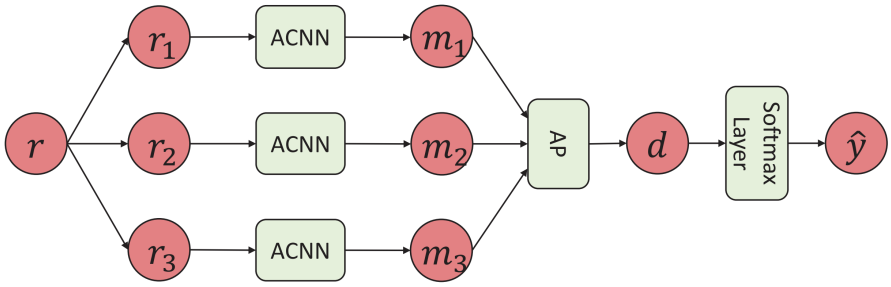


Figure 2.21: A visualization of the Modularized Hierarchical Convolutional Neural Network proposed by P. Yang et al. (2018), image taken from their paper.

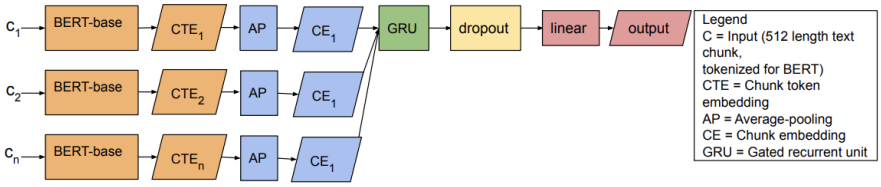


Figure 2.22: A visualization of the SChuBERT model proposed by van Dongen et al. (2020), image taken from their paper.

The second idea deals with the length of the documents. Transformer models like BERT limit the maximum length of the input sequence because they have a time complexity of $\mathcal{O}(N^2)$ with respect to the input length. For BERT the maximum input length is 512 tokens. Some adaptations to the Transformer model have been proposed to extend the input length, most famously the Reformer (Kitaev, Kaiser, & Levskaya, 2020) and Longformer models (Beltagy, Peters, & Cohan, 2020). The disadvantage of these models is that they are not as wide-spread yet and have not been pre-trained as extensively as BERT and GPT models. Relying on pre-trained Reformer or Longformer models could therefore mean that one is unable to utilize developments in other models. There is, for example, no way to utilize the recent developments of GPT-3 in the Reformer and Longformer models. A more general solution, which is able to utilize new models, is therefore to be preferred.

One such solution was proposed by van Dongen, Wenniger, and Schomaker (2020) upon the idea of Joshi, Levy, Weld, and Zettlemoyer (2019). Their SChuBERT model uses pre-trained BERT embeddings for sequences of 512 tokens and combines them into a document embedding using a RNN. The full sequence is first split up in separate sequences of each 512 tokens with an overlap of 50 tokens. An input sequence $[x_1, x_2, \ldots, x_{1200}]$ would be split in three separate sequence chunks $s_1 = [x_1, \ldots, x_{512}]$, $s_2 = [x_{462}, \ldots, x_{974}]$, $s_3 = [x_{924}, \ldots, x_{1200}]$. Each sequence s_i is then passed through a BERT model, creating a BERT embedding for each chunk. The chunks are then concatenated into a single variable-sized vector. This vector is then passed through a Gated Recurrent Unit (GRU), since it can handle variable-sized inputs and the inputs in the sequence have a sequential relationship. The GRU outputs a fixed-size document embedding, which is then passed through a fully-connected layer and a softmax, to create the final task-specific output.

The main advantage of their approach is that the BERT model to create the chunk embeddings can easily be replaced by any other model. To take advantage of the success of GPT-3, the BERT model could simply be replaced with GPT-3 without adapting the other elements. The other advantage is that this model is easily able to handle long input sequence. The sequences in their experiments had an average length of 23 787 with a maximum length of 1 261 656 characters (van Dongen et al., 2020).

2.5 Our proposed approach for patent analysis - MSABERT

Our proposed model, called Multi-Section Attention BERT or MSABERT, combines the strengths of MHCNN and SChuBERT. In essence the model consists of an adapted SChuBERT model for each section, combined with an attentive pooling layer as used by the MHCNN. This approach

counters both the multi-section issue and the problem that the documents are too long for state-of-the-art pre-trained language models.

A patent contains four sections: the Title, Abstract, Description and Claims. MSABERT expects the text of each section to be separated. The text of each section is then tokenized and split into chunks of 512 tokens with an overlap of 50 tokens. These token chunks are then passed through a BERT model, of which the last hidden state is used. To create the intermediate section embedding these chunks are then combined, creating a single variable-sized embedding for each section. To create a fixed-sized section embedding, the intermediate section embedding is passed through a GRU. Although the use of a recurrent neural network looks like a step back, relative to the transformer approach, it solves a concrete problem. In BERT, the limitation of the context size is 512 tokens. The use of an additional recurrent network (GRU) with its leaky-integrator, soft memory, allows for taking a larger textual context into account, in a bidirectional manner.

The GRU outputs an embedding of a fixed hidden size per token. This is done for each section separately and all the GRUs are trained in a modular fashion. For example, the GRU for the title and the GRU for the description are trained separately and can therefore focus on different elements.

These section embeddings are then combined by using an attentive pooling layer, as proposed in P. Yang et al. (2018). The attentive pooling layer learns to weigh the contribution importance of each section. It outputs a single document embedding. The attentive pooling layer also adds additional explainability to the model. The attention weights for each section can be visualized, to get a feeling of what sections contribute most to the patent embedding. Finally the pooled output is passed through a task-specific fully-connected linear layer. The model will be discussed in more detail in Section 3.3.1.

3 Methods

3.1 Data set

A custom data set was collected for this research, containing the patent text, a quality indicator the patent class and a rejected/accepted label. The database is collected out of information from three publicly available data sets: the USPTO patent database, the OECD Triadic Patent Family data set and the OECD Patent Quality data set. After combining all data sets, the newly collected data set contains a total of 48 630 patents. Due to restrictions in processing time, 6000 patents will be used for training and 3000 for validation. If needed in future research, the data set can easily be expanded by scraping more patents from the USPTO patent database.

3.1.1 USPTO patent database

The USPTO patent database forms the basis for the research. The USPTO offers data sets available for download and an API (Patent & Office, n.d.). For this research the API was scraped. The API returns the patent in an easily structured format and is therefore easy to use. Specifically the title, abstract, description and claims are extracted for the patent text. Each section is kept separated and is not concatenated. The USPTO patent databse is also used to extract the CPC sections and can be used for alternative classifications.

3.1.2 OECD Triadic Patent Family

The OECD Triadic Patent Family is a data set of patents that have been filed in various countries but relate to the same invention (for Economic Co-operation & Development, 2022). The patent family is a grouping of these related single patent records. The OECD "Triadic" Patent Families contain patent families that were filed for at the European Patent Office (EPO), the Japan Patent Office (JPO) and the United States Patent and Trademark Office (USPTO). This data set is particularly interesting as it is one of the few data sets that contains information on filed patents instead of only focusing on granted patents. This feature is used to determine the accepted/rejected label. Patents that are granted by all three patent offices are labeled as accepted, the patents that are accepted by the USPTO but not by one of the other offices are labeled as rejected. Patents that are not accepted by the USPTO are discarded. In general the USPTO is the most lenient patent office, the Japanese and European patent offices are more strict (Arts et al., 2021). Patents that are rejected by the Japanese or European patent offices are likely of lower quality than the ones that are accepted by all offices. Next to the determined accepted/rejected label, the CPC section provided in the data set is also extracted.

3.1.3 OECD Patent Quality data set

The last data set is the OECD Patent Quality data set, which contains an indicator for technological and economic value per patent (Squicciarini et al., 2013). The data set contains a variety of possible quality indicators includes basic statistics like the grant lag, the number of claims and the number of forward citations and index statistics like a breakthrough indicator, a generality and an originality index. The OECD Patent Quality data set offers two quality indicators, called the "Patent quality index 4" and the "Patent quality index 6". These indicators are a compounded unweighted index,

Table 6: Statistics on the text length of patents. All reported numbers are the number of characters, including spaces and punctuation.

Patent section	**Minimum length**	**Maximum length**	**Average length**
Title	3	358	61
Abstract	7	3 295	655
Description	1914	3 100 613	70 188
Claims	3	263 569	6 112
Total	3218	3 116 453	77 016

of respectively four and six components. The components are "normalised according to patent cohorts stratified by year and technological field" (Squicciarini et al., 2013). The components are:

- The number of forward citations (up to 5 years after publication)
- Patent family size
- The number of claims
- The patent generality index
- The number of backward citations (only included in "Patent quality index 6")
- The grant lag index (only included in "Patent quality index 6")

In this research we will use the "Patent quality index 4", as this only contains information known prior to the application.

3.2 Tasks

There are three tasks on which model performance will be evaluated. The first is the Patent Acceptance prediction task and related to predicting the quality of patents. In this task the goal is to predict whether the patent will be granted or rejected by the EPO and JPO. The second task is the prediction of the CPC section, which is a classification used for patents. This will focus more on classifying the correct patent topic. After these two tasks, the models will be evaluated on a transfer learning task. The models trained in the first two tasks will then be used to perform the other task.

3.2.1 Patent Acceptance prediction

The goal of the Patent Acceptance prediction task is to predict a binary label accepted or rejected. For this experiment the final fully connected linear layer maps to an output vector of 2, after which a softmax is applied to get the class probabilities. A second experiment is conducted where a one-hot encoded vector for the CPC section is added to the patent embedding just before the final task layer. The CPC section is assigned during the review of the patent and not known at the moment of submission. This experiment was conducted to test if the rejection process is dependent on the type of patent, or whether there is an underlying inherent reason in all patent, which determines which are accepted and which are rejected.

Accuracy is used as the evaluation metric for this task. The accuracy is the proportion of correct predictions among the total. For the correct predictions both the true positives and true negatives are counted. Accuracy is a widely used metric and therefore suitable. For the binary case it is formally defined in Equation 3.1.

$$\text{Binary Accuracy} = \frac{\text{True Positive count} + \text{True Negative count}}{\text{Total number of samples}} \tag{3.1}$$

For the loss function the unweighted binary cross-entropy with logits loss function is used. Formally given in Equation 3.2, where x is the input, y is the target, and N the size of the minibatch.

$$l(x, y) = L = \{l_1, \ldots, l_N\}^T \quad \text{where} \quad l_n = -w_n[y_n \cdot \log\sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n))], \tag{3.2}$$

3.2.2 CPC Section Classification

Similar to Patent Acceptance prediction task, the CPC Section classification is also a classification problem. The difference between the two is that the CPC Section classification is a multi-class problem, where the Patent Acceptance prediction task is a binary classification. There are 8 classes in the CPC Section classification. This task is not focused on the patent quality but used as a baseline to compare the proposed model to other models. This task is similar to the work of J.-S. Lee and Hsiang (2019) and S. Li et al. (2018). Instead of the main CPC section they have used the CPC subclass, which has 656 labels. They also used a far bigger dataset of 2 and 3 million

patents compared to the 6000 training patents used in this study.
As for the previous task, the unweighted Cross Entropy loss is used as a loss. Formally given in Equation 3.2, where x is the input, y is the target, c is the number of classes and N the size of the minibatch. As for the Patent Acceptance prediction task, accuracy (as shown in Equation 3.3) is used as the evaluation metrics.

$$\text{Multi-class Accuracy} = \frac{\text{Correct classification count}}{\text{Total number of samples}} \tag{3.3}$$

3.2.3 Transfer Learning Tasks

After each epoch the performance of the model is evaluated on a separated dataset, which was also used to perform the results above. The model with the highest evaluation accuracy during the training phase will be saved. This allows to use that particular model with the best performance to be re-used again.

Not the entire model is saved, the last fully connected layer (task-specific part of the model) is not included. This way, the model can be used to output the patent embedding is has learned. In the transfer learning task, the saved models are used to train a new model. The models trained on the Acceptance task are first used in the CPC classification task, and the CPC classification models are used for the Acceptance task. After this, all pre-trained models are used for a new task, where the goal is to predict the OECD quality indicator of the patent.

The transfer learning models are very simple: they only consist out of a single fully connected layer. The size of the layer depends on the size of the output of the saved model. The single fully connected layer is the same as the task-specific output layer in the earlier models. In the training phase, the transfer learning model gets the patent embedding output of the saved model as its input, passes this through the task-specific fully connected layer and outputs its classification.

3.3 Models

3.3.1 MSABERT

The model proposed here is the Multi-Section Attention BERT (MSABERT) model. This model has a modular structure for each section. For each section a pre-trained BERT model is used in combination with a GRU. The pre-trained BERT model creates embeddings for chunks of text, which serve as the input to the GRU. The GRU then learns to create a different embedding, specific to that section. An attentive pooling layer is used to combine the different section embeddings into a single document embedding. A fully-connected linear layer is then used to predict an output from the document embedding. A visualization is shown in Figure 3.1 and Figure 3.2

For this research a pre-trained uncased BERT-base model is used. The BERT-base model is trained on English Wikipedia and BookCorpus. BookCorpus is a data set consisting of 11 038 unpublished books (Zhu et al., 2015).

BERT-base uses 12 layers of transformer blocks with a hidden size of 768 and 12 self-attention heads. It has a total of around 110M trainable parameters.

The input to the BERT-base model is a sequence of tokens, with a maximum sequence length of 512 tokens. Since the sections within a patent often exceed this limit, the text is split into chunks. Firstly the entire text sequence is tokenized. This sequence is then split into chunks of 512 tokens with an overlap of 50 between the chunks. An input sequence $[x_1, x_2, \ldots, x_{1200}]$ would be split in three separate sequence chunks $s_1 = [x_1, \ldots, x_{512}]$, $s_2 = [x_{462}, \ldots, x_{974}]$, $s_3 = [x_{924}, \ldots, x_{1200}]$. Figure 7 shows how an example sentence would be chunked.

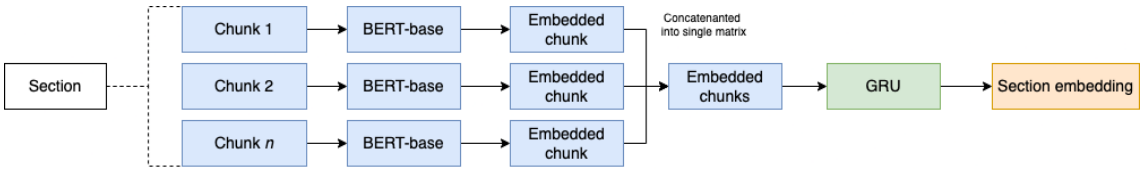


Figure 3.1: Visualization of the MSABERT model proposed in this research.

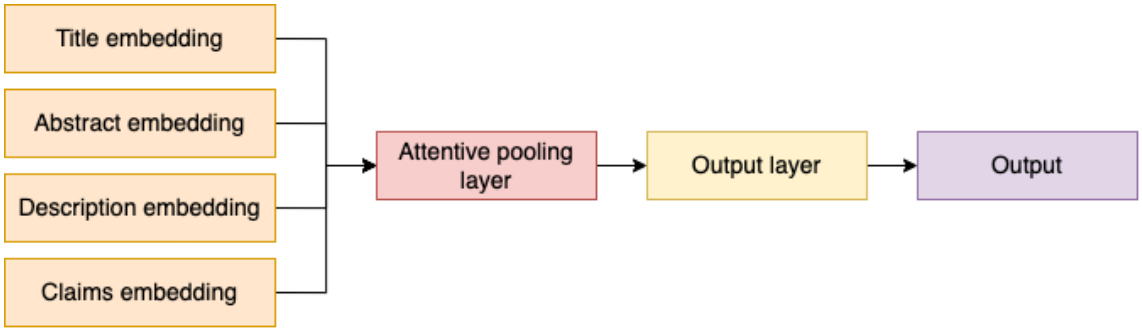
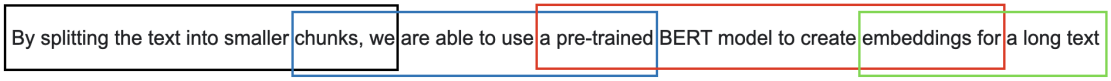


Figure 3.2: Visualization of the full MSABERT model proposed in this research.

Table 7: Example sentence split into chunks of 8 tokens with an overlap of 2. Here a token matches a word, in reality a word would likely consist of multiple tokens.



c_1 =["by", "splitting", "the", "text", "into", "smaller", "chunks", "we"]
c_2 =["chunks", "we", "are", "able", "to", "use", "a", "pre-trained"]
c_3 =["a", "pre-trained", "bert", "model", "to", "create", "embeddings", "for"]
c_4 =["embeddings", "for", "a", "long", "text"]

These chunks are then used as the input for the pre-trained BERT-base model. Each chunk is passed through the BERT model sequentially, creating an embedding for each token. There are three embeddings that can be used: all hidden states, the last hidden state and the pooler output. Similar to van Dongen et al. (2020), we use the last hidden state of the BERT model. The last hidden state is a vector of size 768 for each token in the input. The resulting chunk embedding is a matrix of size $t_c \times 768$, where t_c is the number of tokens in a chunk. To create an intermediate section embedding the matrices for each chunk are concatenated, resulting in a matrix of size $t \times 768$ where t is the total number of tokens in all chunks. As the chunks have some overlap, it is not required that t matches the number of tokens in the input sequence. The example in Figure 7 contains 23 input tokens but t is 29.

The intermediate section embedding is variable-sized, as the number of chunks can vary per document. To create a fixed-sized embedding for the section a GRU is used. A GRU is used because the elements in the sequence have a sequential relation and a GRU has fewer trainable weights than a LSTM. The GRU expects all inputs in the batch to have the same size so all inputs are first padded to the size of the longest sequence in the batch. Each section embedding is then passed through the matching GRU. The model consists of four separately trained GRUs. This ensures that the GRU learns an embedding specifically for that section. The GRU outputs a vector for each token in the input sequence. To extract an embedding three options can be used:

- **Last element of the GRU output sequence:** A method commonly used in NLP is to use the output for the last element in the sequence. This does expect that the output for the last element contains information for the entire sequence.
- **Average over all elements in the GRU output sequence:** As not all information might be encoded in the last element, an average of all elements in the sequence can also be used.
- **The final hidden state of the GRU:** Another option is to use the final hidden state of the GRU instead of the output. This is similar to what is for instance done with pre-trained BERT models.

We will treat the selection of the embedding method as an hyperparameter, all three methods will be tested in the experiments. A bi-directional GRU with zero, one or three hidden layers will be used, where in the case of three hidden layers there will be three bi-directional GRUs stacked together. The hidden size of each GRU will be treated as a hyperparameter. Dropout will be applied to the outputs of each GRU layer except the last layer, the dropout probability is also

treated as a hyperparameter.
After creating a fixed-sized embedding for each section, these need to be combined. One option would be to apply a simple average over the four section embeddings. In this case, each section would contribute equally to the final embedding. One can however assume that a title might contain less information than the abstract or description. This is why an attentive pooling layer is used, as used in P. Yang et al. (2018). As formally shown in Equations 2.23, 2.24 and 2.25, the attentive pooling layer learns a weight for each section. The final document embedding is created by taking the weighted average of the sections.
To map the final document embedding to a task-specific output a final fully-connected linear layer is used. For the Acceptance/Rejection prediction task an the linear layer outputs a vector of 2, for the quality index the output is a single value, for the CPC classification task an output of 8 is used.

3.3.2 Baseline Models

As discussed earlier in 2.4, there are very few models that can serve as a baseline. The baseline models will constitute of restricted adaptations of the MSABERT model. Specific features of the MSABERT model will be removed to create a baseline model. Next to these baseline models, the results for the Patent Acceptance prediction task will be qualitatively compared to the results in Arts et al. (2021). They did not use an end-to-end trained neural network, but collected hand-crafted features and applied a logistic regression model on these features.

Baseline A - BERT

This model will be a simple BERT model with a task-specific output layer. This is similar to the fine-tuning model proposed in J.-S. Lee and Hsiang (2019). This model does not utilize the multi-section structure and is limited in the sequence length. Specifically, the input will be a concatenation of all sections, in the order title, abstract, claims, description. The entire sequence will be tokenized, after which only the first 512 tokens will be used. As in MSABERT the last hidden state of the BERT model will serve as the embedding. This embedding is then used as the input for a fully-connected linear layer that maps the input to the required output.

Baseline B - Chunked BERT

Similar to baseline A, this model first concatenates all the sections together. After this, time-ordered chunks of 512 tokens with an overlap of 50 tokens are created. These chunks are then concatenated and passed through a single GRU. This model does utilize the entire text but not the multi-section structure. It treats all of the sections equally. This model is similar to the SChuBERT model (van Dongen et al., 2020). This baseline mainly serves to evaluate the impact of using the full text as opposed to using the first 512 tokens. Compared to MSABERT this model lacks the ability to utilize the multi-section structure. Each section will be treated as one piece of text and therefore similarly.

Baseline C - Multi-Section BERT without Attention

This baseline model is the most similar to the MSABERT model, but it replaces the attentive pooling with an average pooling layer. This model allows for a comparison between the performance of average pooling and attentive pooling. This model is however capable of using the full text and has a multi-section structure. The difference to MSABERT is however that each section contributes equally to the final embedding, whereas in MSABERT this is weighted.

3.3.3 Hyperparameters

Table 8: Hyperparameters

Hyperparamater	**Values**
Batch size	48
Maximum epochs	70
Optimizer	Adam
Learning rate	0.00005, 0.0001 & 0.0005
Weight decay	0.0001
β_1	0.9
β_2	0.999
GRU Size	512, 768 & 1024
GRU Layers	0, 1 & 3
GRU Output Type	Average output over the document, last output of the document & last hidden-GRU activation
Dropout	0, 0.3, 0.5 & 0.7

4 Results

In this section, the results of the various experiments are described. The first part will discuss the results of the experiments performed to find the optimal hyperparameter configuration. Only the structured search results are shown here. Before these experiments, unstructured experiments were performed. This was used to find an optimal subspace to perform the structured search in, to test the model performance and to find valid configurations for the hyperparameters set to remain constant. For each hyperparameter an experiment was setup, where one hyperparameter was selected and differentiated, while all other hyperparameters remained constant. Compared to a full grid search, this significantly limits the required number of experiments.

4.1 Hyperparameter testing results

For the hyper-parameter scan only the validation sets were used for decision making concerning the best parameter setting. The best performing models were selected based on test accuracy on the validation set. Generally, one uses the loss to select the hyperparameters. The initial experiments however showed that the lowest losses were already achieved in the first few epochs, at which accuracy was still suboptimal. The optimal accuracy was often achieved much later in the training process with a much higher loss. This behavior could suggest that the selected loss function does not correctly represent the task at hand. In the initial experiments, not reported here, different loss functions were also tested. The other loss functions had significantly worse performance and the difference between loss and accuracy was even higher. The effect of this choice has a very minimal effect on the hyperparameter selection. Models that achieve a higher accuracy also achieve a lower minimal loss compared to models with a lower accuracy. Although the selection is based on different evaluation moments in the training procedure, the overall model quality is similar for both evaluation metrics. Selecting the hyperparameters based on the loss would have resulted in the same hyperparameter configuration. As there is a big difference between when the optimal loss is achieved and the moment where the optimal accuracy is achieved, one could select a different learning rate based on in which epoch the optimal loss was achieved.

GRU Output type
Remember, the GRU takes as input a variable-sized intermediate section embedding. To create a fixed-sized embedding for each document, a GRU is used. There are three options to create a fixed-sized embedding based on the GRU. These three options are the *average output over the document*, the *last output of the document* and the *last hidden-GRU activation*. From the results shown in Table 9 it becomes clear that using the *average output over the document* type performs better than the other types (i.e., *last output* and *last hidden-GRU activation*). Using the hidden layers has a somewhat worse performance but is still performing quite decently. Only using the last output element of the GRU is performing significantly worse than the other two options.

Table 9: Results for comparing the different manners of creating a fixed size embedding from the variable-sized section embeddings. The learning rate was set at 0.0001, dropout at 0.5 and 3 GRU layers were used.

Accuracy CPC Section ($N_{class} = 8$) classification			
Method	**GRU 512**	**GRU 768**	**GRU 1024**
Last element GRU Output	0.654	0.6383	0.657
Average over all GRU Output elements	0.708	0.6977	0.704
Last hidden GRU Layer	0.688	0.6957	0.699

Number of GRU layers
A GRU can consist of multiple stacked layers. For all cases bidirectional GRUs were used, meaning the actual number of GRU layers is double the amounted listed here. In the case with 0 GRU layers, a simple average over all embeddings was used instead of a GRU. As shown in Table 10, using a single GRU layer performs almost as well as using three layers when doing class prediction. There is only a small difference, but using fewer GRU layers has some benefits. Using only one GRU layer significantly speeds up the training and allows for using a higher GRU size. The case of using an

average instead of a GRU is interesting. For the CPC classification task the effect seems to be quite limited, but for the accepted/rejected task the performance drops to below chance. The model performance on the accepted/rejected task heavily depends on the temporal dependencies within the patent text. The performance on the CPC classification task barely suffers from removing the temporal dependencies. This is likely because the topic of the text is usually not described by using temporal dependencies, but is already clear from the choice of specific words.

Table 10: Results for comparing the number of hidden GRU layers. The learning rate was set at 0.0001, GRU size at 1024 and the hidden layers were used. The dropout was set to 0, as dropout was performed between the GRU layers, which is not applicable with a single layer.

Model	**Accuracy CPC classification** ($N_{class} = 8$)	**Accuracy Acceptance** ($N_{class} = 2$)
0 GRU Layer	0.675	0.466
1 GRU Layer	0.719	0.606
3 GRU Layers	0.721	0.627

Learning rate
Table 11 shows a comparison of the results for different learning rates. There is no clear relation between the learning rate and the performance. The performance for the highest learning rate (0.0005) appears to be somewhat worse than for the other tested learning rates. It is however relevant to note that is possible to achieve a fairly good performance within a few epochs with this learning rate. In a scenario where training time is restricted, one could use a high learning rate and still achieve a reasonably high performance. Even with a lower learning rate, the model already achieves a near optimal performance early on but then takes longer to achieve an optimal performance.

Table 11: Results for comparing the learning rate. A single GRU layer with size 1024 was used, the dropout was set to 0 as this is not applicable. For the class prediction the hidden layers of the GRU were used, for the rejected task the average of the output was used.

Learning rate	**Accuracy CPC classification** ($N_{class} = 8$)	**Accuracy Acceptance** ($N_{class} = 8$)
0.00005	0.714 (epoch 44)	0.590 (epoch 60)
0.0001	0.738 (epoch 26)	0.559 (epoch 18)
0.0005	0.704 (epoch 6)	0.570 (epoch 3)

Selected hyperparameters
Overall the best hyperparameter configuration, looking at both performance and training time is: using the average GRU output over the document, three GRU layers, a GRU size of 1024 and a learning rate of 0.0001. The training time when using three GRU layers is around four times as long as compared to using a single GRU layer. In these experiments we have chosen to use three, as it has a slightly better performance. One could however argue that this increase does not weigh up against using the additional training time.

The selection of the learning rate was mostly based on the number of training epochs required to achieve optimal performance and not necessarily on the best performance. The learning rate 0.00005 and 0.0001 achieved similar performances but the latter required around half of the epochs to achieve its optimal performance and was therefore chosen.

4.2 Experiments - CPC section classification task

The first experiment is the CPC section classification. The CPC section identifies the technical field of a patent. This is a multi-class problem, with eight different sections to be identified. The results for the CPC section classification experiments are shown in Table 15. The Chunked BERT baseline model outperforms the other models on this task. The MSABERT w/o attention has the worst performance. The MSABERT model is outperforming the BERT model with an improvement of almost 4%. The performance of the BERT and MSABERT w/o attention models are also outside the standard deviation range for the other two models.

The t-test results show that there is no significant difference between the Chunked BERT model and MSABERT-4096, and there is no significant difference between BERT and MSABERT-4096 w/o attention. There is a significant difference between Chunked BERT model and MSABERT-4096 on the one hand and BERT and MSABERT-4096 w/o attention on the other hand. The MSABERT-4096 and Chunked BERT models are significantly outperforming the other two models.

Table 12: Results on the CPC classification task with 5 repetitions of the experiment.

Model	**Accuracy CPC classification** ($N_{class} = 8$)
BERT	0.698 ±0.014
Chunked BERT	**0.731 ±0.003**
MSABERT-4096 w/o attention	0.692 ±0.012
MSABERT-4096	0.725 ±0.010

Table 13: t-test results on the results of the CPC classification task with 5 repetitions of the experiment (8 degrees of freedoms). The astriks marks significant differences.

Model	BERT	Chunked BERT	MSABERT-4096 w/o attention	MSABERT-4096
BERT		t=5.1537, p=0.0009*	t=0.7276, p=0.4876	t=3.5092, p=0.0080*
Chunked BERT	t=5.1537, p=0.0009*		t=7.0502, p=0.0001*	t=1.2851, p=0.2347
MSABERT-4096 w/o attention	t=0.7276, p=0.4876	t=7.0502, p=0.0001*		t=4.7239, p=0.0015*
MSABERT-4096	t=3.5092, p=0.0080*	t=1.2851, p=0.2347	t=4.7239, p=0.0015*	

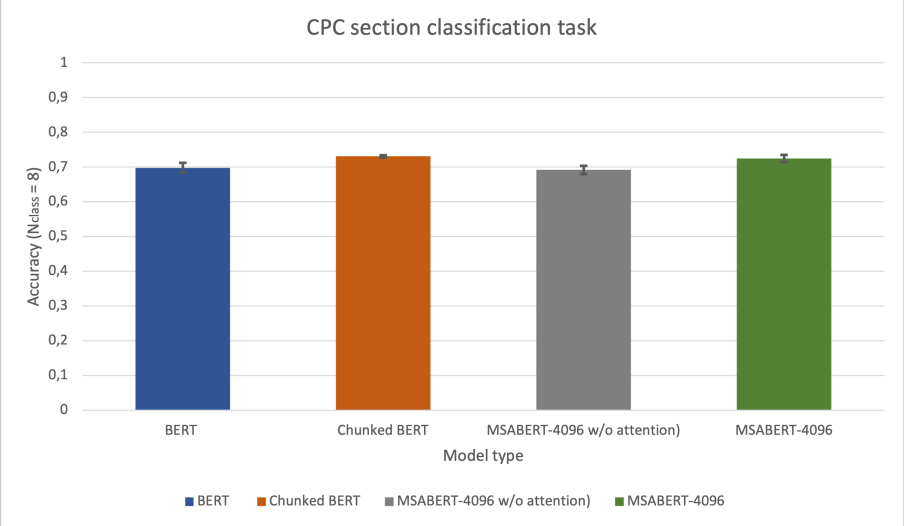


Figure 4.1: Bar chart visualization of the results on the CPC classification task

4.3 Experiments - Acceptance prediction task

The second experiment is based on predicting whether a patent will be accepted or not. Patents that are accepted by all three of the major patent offices are categorized as accepted, if one of the patent offices rejects a patent it is categorized as rejected. This is a binary classification problem.

Within this experiment an additional experiment was conducted, where the CPC section was included as additional input to the final output layer next to the text vector. A one-hot encoded vector for the CPC section was concatenated to the text embedding before the output layer. Figures 4.2 and 4.3 show the different setups.

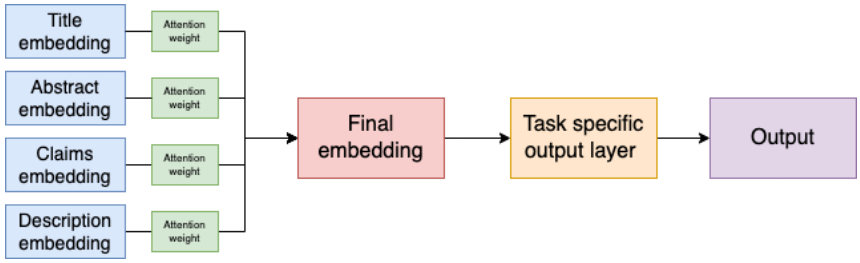


Figure 4.2: Visualization of the regular task based on only the text.

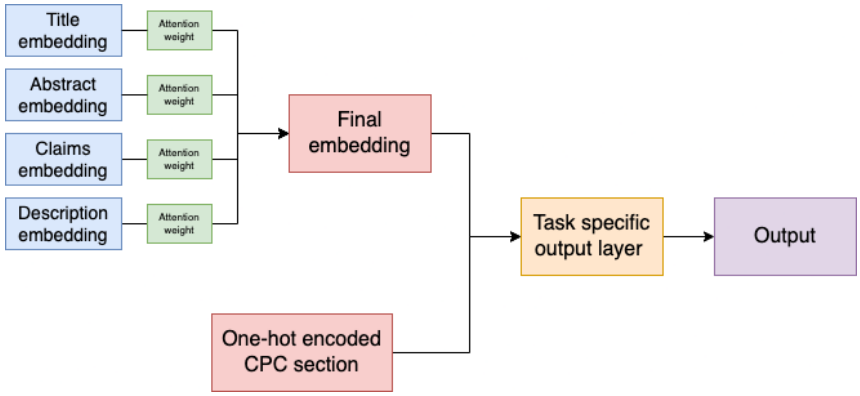


Figure 4.3: Visualization of the multi-input task where next to the text also the CPC section is included as input to the final task output layer.

The results for the Acceptance prediction task experiments are shown in Table 14. The newly introduced MSABERT model is compared to the other baseline models for both the Acceptance prediction task. As could be expected, all models are performing much better on the CPC section classification task than the Acceptance prediction task.

The Chunked BERT baseline model outperforms the other models on this tasks, by around 1.6% compared to BERT, 1.3% compared to MSABERT and around 2.2% compared to the MSABERT without attention. Where for the CPC classification task there was a difference in performance between the BERT and MSABERT model, the difference here is negligible. The t-tests however show that these difference are not significant. Only the difference between Chunked BERT and MSABERT-4096 w/o attention is approaching significance. All of the models perform equally well.

Table 14: Results on the Acceptance prediction task, with 5 repetitions of the experiment.

Model	**Accuracy** ($N_{class} = 2$)
BERT	0.604 ±0.006
Chunked BERT	**0.614** ±0.012
MSABERT-4096 w/o attention	0.601 ±0.008
MSABERT-4096	0.606 ±0.009

The results of the experiment with the CPC section as additional input are presented in Table 16. There is a clear difference between the results of the first experiment and this one. In the first experiment the different models were close together and Chunked BERT was the best performing. Based on the t-tests shown in table 17 and table 18 the Chunked BERT and BERT models perform significantly worse, whereas the MSABERT models perform better. The MSABERT model outperforms the BERT and Chunked BERT model by around 8.5%. The performance for the MSABERT model and the version without attention increases with 1.9% and 1.7% respectively, whereas the performance for the BERT model decreases with 5.8%. The Chunked BERT model has the biggest difference, with a decrease of 6.9%. The MSABERT models benefit from including the CPC section in the embedding, where the other two models actually suffer significantly.

Table 15: t-test results on the results of the Acceptance prediction task with 5 repetitions of the experiment (8 degrees of freedoms). The astriks marks significant differences.

Model	BERT	Chunked BERT	MSABERT-4096 w/o attention	MSABERT-4096
BERT		t=1.6667, p=0.1341	t=0.6708, p=0.5212	t=0.4134, p=0.6901
Chunked BERT	t=1.6667, p=0.1341		t=2.0156, p=0.0786	t=1.1926, p=0.2672
MSABERT-4096 w/o attention	t=0.6708, p=0.5212	t=2.0156, p=0.0786		t=0.9285, p=0.3803
MSABERT-4096	t=0.4134, p=0.6901	t=1.1926, p=0.2672	t=1.2851, p=0.2347	

Table 16: Results on the Acceptance prediction with one-hot encoded CPC with 5 repetitions of the experiment.

Model	**Accuracy** ($N_{class} = 2$)
BERT	0.569 ±0.025
Chunked BERT	0.571 ±0.005
MSABERT-4096 w/o attention	0.611 ±0.010
MSABERT-4096	**0.617** ±0.019

Table 17: t-test results on the results of the Acceptance prediction task with the CPC section one-hot encoded with 5 repetitions of the experiment (8 degrees of freedoms). The astriks marks significant differences.

Model	BERT	Chunked BERT	MSABERT-4096 w/o attention	MSABERT-4096
BERT		t=0.1754, p=0.8651	t=3.4879, p=0.0082*	t=3.4181, p=0.0091*
Chunked BERT	t=0.1754, p=0.8651		t=8.0000, p=0.0001*	t=5.2354, p=0.0008*
MSABERT-4096 w/o attention	t=3.4879, p=0.0082*	t=8.0000, p=0.0001*		t=0.6249, p=0.5495
MSABERT-4096	t=3.4181, p=0.0091*	t=5.2354, p=0.0008*	t=0.6249, p=0.5495	

Table 18: t-test results on the results of the Acceptance prediction task compared with the results for the task with the CPC section one-hot encoded. 5 repetitions of the experiment (8 degrees of freedoms). The astriks marks significant differences.

Model	result
BERT	t=3.0441, p=0.0160*
Chunked BERT	t=7.3962, p=0.0001*
MSABERT-4096 w/o attention	t=1.7461, p=0.1189
MSABERT-4096	t=1.1699, p=0.2757

4.4 Experiments - Transfer Learning

Several experiments have been performed to test the transfer learning capabilities of the models. The best performing models trained in the previous tasks are stored and applied to different tasks. Figure 4.5 shows a visualization of this process.

For the first task, described in Sections 4.4.1, the models that were trained on the Acceptance task are used as input for the CPC classification task. In the Section 4.4.2 this is reversed and

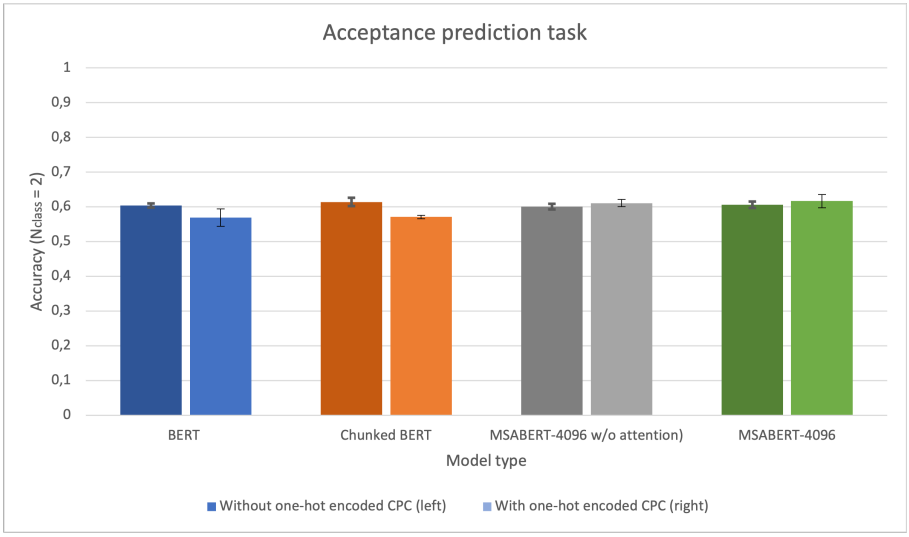


Figure 4.4: Bar chart visualization of the results on the acceptance prediction tasks

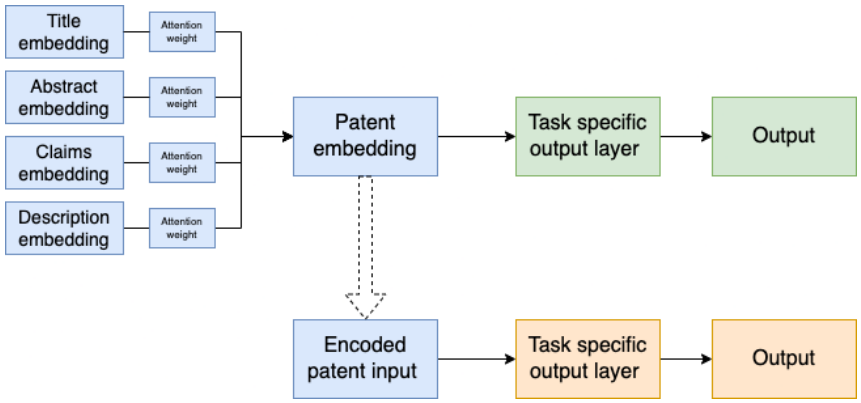


Figure 4.5: Visualization of the transfer learning task. The model is first trained on a specific task, shown in the top (blue) part. The model to encode the text input into a document embedding is then stored. This stored model is then used to create the inputs for a second task, the model itself is not trained again.

the models trained on the CPC classification are used as input for the Acceptance task. In the experiments described in Section 4.4.3, both types of models are applied to a third task. Here the goal is to predict the OECD patent quality indicator.

Only the best performing models in the earlier experiments were selected to be used in the transfer learning task. Therefore all experiments were only performed once and conclusions should be made carefully. As a baseline, the saved models were also used to perform the task for which they were trained.

4.4.1 Trained on Acceptance task

The results for the models trained on the Acceptance task, shown in Table 19, clearly show a decrease in performance on the CPC section classification task. The percentages show a comparison between the performance in this experiment and the performance in the previous experiments for that task.

The BERT and MSABERT w/o attention models have a decrease of around 60%. The Chunked BERT model also has a big decrease of almost 30%. The performance of the MSABERT model decreases the least, but it also performs significantly worse. The MSABERT model performs around 22% better in this task than the Chunked BERT and around 150% better than the other two models. Surprisingly, the MSABERT model also shows a significant increase on the acceptance task, which served as a baseline. The model, specifically the final fully connected layer, appears to

have learned a more general mapping from the embedding to the task-specific output than that in the first experiment.

Table 19: Results on both the Acceptance prediction and CPC classification tasks of the pre-trained models trained on the Acceptance task. The percentage shows the increase or decrease of the accuracy compared to the performance of the same model in a regular scenario where the model is fully trained.

Models trained on Acceptance task	**Acceptance task**	**CPC task**
BERT	0.630 (+4.3%)	0.259 (-63.0%)
Chunked BERT	0.621 (+1.2%)	0.525 (-28.2%)
MSABERT-4096 w/o attention	0.604 (+0.6%)	0.288 (-58.4%)
MSABERT-4096	**0.662** (+9.3%)	**0.643** (-11.1%)

4.4.2 Trained on CPC section

The models trained on the CPC section classification task, clearly outperform the models that were specifically trained for the Acceptance task. The performance of all models has increased by around 10%, except for the MSABERT w/o attention model which increased 3.5%. There is almost no difference between the models, with the exception of the MSABERT w/o attention model. This model performs significantly worse (around 8%) compared to the other models on the same task. The performance on the baseline task is similar to the original performance.

Table 20: Results on both the Acceptance prediction and CPC classification tasks of the pre-trained models trained on the CPC classifcation task. The percentage shows the increase or decrease of the accuracy compared to the performance of the same model in a regular scenario where the model is fully trained.

Models trained on CPC classification task	**Acceptance task**	**CPC task**
BERT	**0.676** (+11.9%)	0.699 (+0.1%)
Chunked BERT	0.673 (+9.7%)	**0.738** (+0.9%)
MSABERT-4096 w/o attention	0.622 (+3.5%)	0.670 (+1.2%)
MSABERT-4096	0.674 (+11.3%)	0.728 (+0.7%)

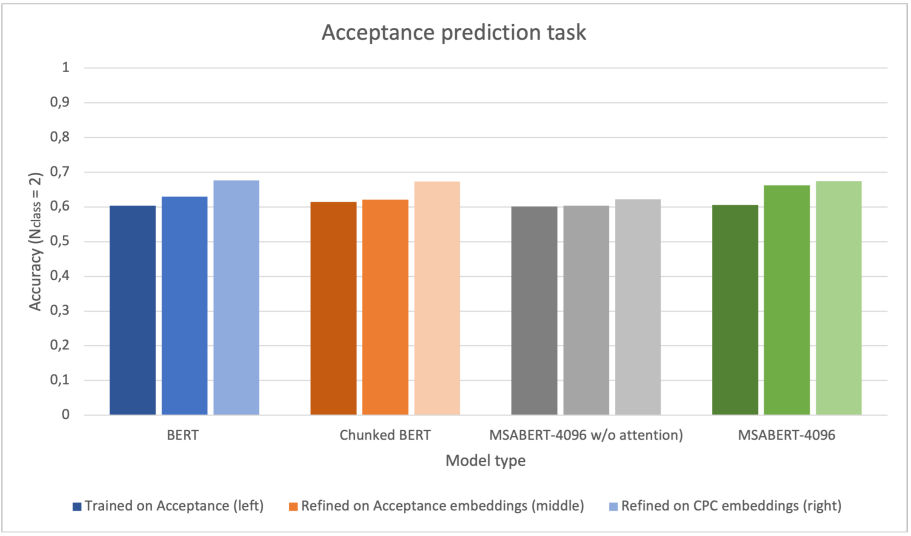


Figure 4.6: Bar chart visualization of the results on the acceptance prediction tasks for the original experiment and the transfer learning variations

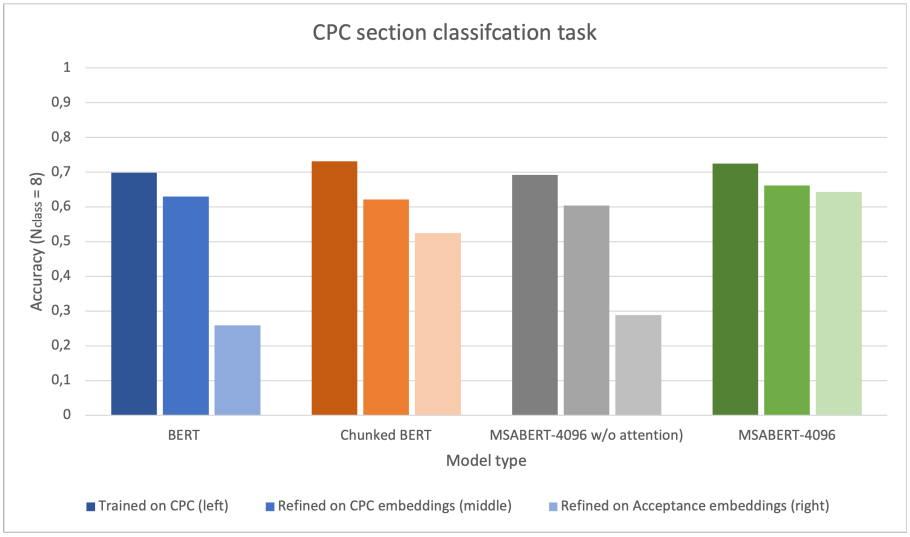


Figure 4.7: Bar chart visualization of the results on the CPC section classification tasks for the original experiment and the transfer learning variations

4.4.3 OECD quality indicator prediction

In the last experiment, the models are tested on a completely new task. This is the prediction of a quality indicator defined by the OECD. The models trained on the Acceptance task clearly outperform those trained on the CPC task. The Mean-Squared Error of the models trained on the CPC task is multiple times ($5 - 11\times$) higher than that trained on the Acceptance task. The performance of the models trained on the CPC task is significantly worse than a model that would simply predict the average over the training dataset for all patents. It is also apparent that the MSABERT model is performing much better than the other models. The Mean-Squared Error of the other models trained on the Acceptance task is at least $5\times$ as high, for the CPC trained models the MSE is around $3\times$ as high.

Table 21: Results on the Quality prediction task of the pre-trained models trained on the Acceptance prediction task.

Models trained on Acceptance task	**MSE on Quality prediction task**
BERT	0.124
Chunked BERT	0.157
MSABERT-4096 w/o attention	0.111
MSABERT-4096	**0.023**

Table 22: Results on the Quality prediction task of the pre-trained models trained on the CPC classifcation task.

Models trained on CPC task	**MSE on Quality prediction task**
BERT	0.988
Chunked BERT	0.890
MSABERT-4096 w/o attention	0.690
MSABERT-4096	**0.289**

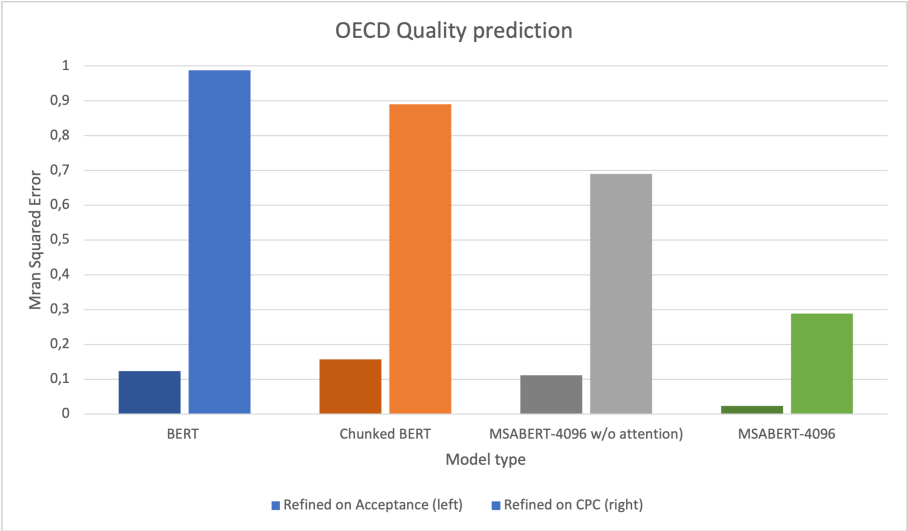


Figure 4.8: Bar chart visualization of the results on the OECD quality prediction task

5 Discussion

This thesis proposes a new language model specifically for patents. It extends the BERT Transformer model to be capable of handling longer texts and introduces a multi-section structure. The goal of this research is to test whether adding this multi-section structure and ability to handle longer texts is actually beneficial for the performance. This comparison is done by using three baseline models: A BERT model, a BERT model capable of handling longer text and an unweighted multi-section model.

The findings of Chapter 4 are summarized and interpreted in Sections 5.1 to 5.4. Some limitations of the newly introduced model and the study are discussed in Section 5.5. After this the contribution to the research fields are discussed in Section 5.6, after which some suggestions for future research are made in Section 5.7. The Chapter is concluded with a final conclusion in Section 5.8.

In general the performance of all four tested models on the original task is quite similar and differences are relatively small. The Chunked BERT model is the best performing model on both original tasks, with the MSABERT model as a close second with an insignificant difference. When we however use the models in a transfer learning scenario, the MSABERT model clearly outperforms the other models. In these experiments, the patent embedding learned by the MSABERT model generalizes better than that of the other models. The MSABERT model is better capable of capturing the underlying characteristics of the patents.

5.1 Multi-section adaptation

The MSABERT model introduces a multi-section structure to handle the differences between each section in the patent. In this structure each section is first handled separately, after which the embeddings of the different sections are combined. The sections are either averaged in an unweighted or weighted fashion, where the weights are learned in the training process. Theoretically, this has the advantage that the embedding of the title can be treated differently from that of the claims and the model can learn section-specific characteristics.

In the experiments, the Chunked Bert model without the multi-section structure slightly outperformed the MSABERT mode but the difference is insignificant. This suggest that it is not necessary to handle the patent section separately. It could very well be that the underlying predictors for the Acceptance or CPC class, are similar in all sections. In this case, it would not be needed to treat them separately.

An alternative explanation could be that there are temporal connections between the different sections that benefit the performance in the tasks. The MSABERT model excludes the temporal connections between sections as a result of treating them in a modular fashion. The improved performance of the multi-section fashion could be counterbalanced by the decreased performance of losing the temporal connections. Future research could use a model which first learns a section specific embedding, then concatenates them into a single document embedding and then use a Recurrent Neural Network to extract the most important parts of the text. Figures 5.1 and 5.2 show visualizations of the approach in MSABERT and a possible improvement respectively.

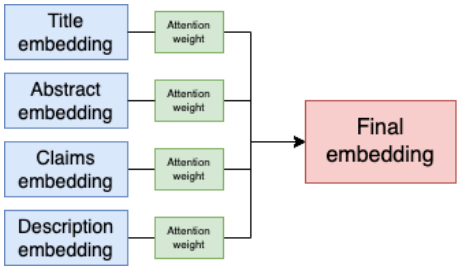


Figure 5.1: Visualization of how the MSABERT combines the section specific embeddings into a single final embedding.

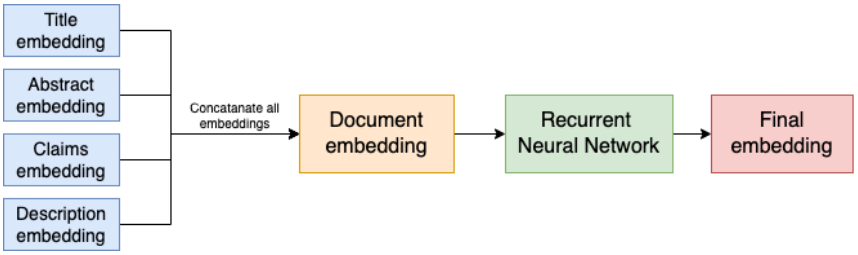


Figure 5.2: Visualization of how a new model could combine the section embeddings into a single vector using a Recurrent Neural Network.

5.2 Additional Explainability

One of the advantages of leveraging the multi-section structure of the patents within the model, is that it offers a way to compare the relative importance of the sections. The MSABERT model with attention is outperforming the model without attention in both tasks. This shows that not all sections are equally important for the task. Table 23 shows the learned relative importance, based on the computed attention weights, of the different sections for the Acceptance and CPC classification tasks. This additional explainability is a major advantage of this model and offers users to gain insight into what the model has learned.

Based on the learned attention weights, shown in Table 23, the title is the most relevant section. The abstract is almost completely irrelevant for the CPC task, whereas the claims section is less important for the acceptance task. It is important to note that the attention weights clearly differ between the two tasks. Where the claims section is the least important for the Acceptance task, it is the most relevant section for the CPC prediction task. This insight can benefit future work. Most previous work, using a restricted part of the text, uses the first part of the text. The attention weights suggest that selecting other parts of the text might benefit the performance.

Table 23: Computed attention weights per section within the MSABERT model. Averaged over the 5 experiments, performed in Sections 4.3 and 4.2

Section	Weight for Acceptance task	Weight for CPC task
Title	0.304	0.323
Abstract	0.268	0.093
Claims	0.146	0.375
Description	0.281	0.208

5.3 Extended BERT for longer documents

To compare the effect of using more of the patent text, the Chunked BERT model is compared to the original BERT model. The Chunked BERT model is able to leverage longer texts, similar to MSABERT, but does not include the multi-section adaptation.

In all tasks the adapted Chunked BERT model with longer texts is outperforming the original BERT model. The performance of the Chunked version is almost 5% better on the CPC classification task and around 1.5% (only minimally significant with p=0.13) for the Acceptance task. This shows that including more of the patent indeed improves the performance on these tasks.

As we have seen that there is a difference in the importance of each section, the improved performance could be explained by the number of tokens included per section. On average BERT uses 20 tokens for the title, 218 for the abstract and 274 tokens from the claims. The Chunked BERT model will use 20 tokens from the title, 218 abstract, 2037 from the claims and 4096 for the description. There is clearly a difference in how much each section is represented in the input for the models. One could get an indicator for how much this affects the performance by using the attention weights of the MSABERT model, as shown in Table 23. If the input for the model contains relatively many tokens of the sections that have a high attention weight, the performance will likely be better. To create the indicator, the number of tokens was multiplied by the attention weight and then divided by the total number of tokens. For the BERT model this gives us a value

of 20 for the Acceptance task and 25 for the CPC section task. The MSABERT model scores slightly higher for the Acceptance task at 24 but similar at the CPC section task at 26. The performance gap between the BERT model and the Chunked BERT model in the CPC section task was however quite big, almost 5%. The indicator suggests that this is not caused as an effect of which parts of the text are included and which are not, but more experiments focused on this are needed.

5.4 Transfer Learning

When we shift to the transfer learning capabilities of the different model, a completely different picture arises. Here the MSABERT model clearly outperforms all of the other models. For the quality prediction task, the MSE of the MSABERT model is a fraction of that of the others. Where for the other models the performance on the CPC task when trained on the Acceptance task drops a lot, the performance for the MSABERT model remains fairly stable.

The results show that the MSABERT model learns a representation of the patent that is much more general and can be used in a variety of tasks without retraining the model. This is significant, as most of the state-of-the-art performance in machine learning is based on pre-trained models. The results show that when using these models in a pre-trained scenario, the MSABERT model is clearly outperforming the other models. Where in a full training scenario the Chunked BERT model is performing slightly better, in a transfer learning setup the MSABERT model is the better candidate.

5.5 Limitations

As discussed before, patents are extremely long documents compared to the documents on which most research focuses. This has some major implications for the required processing time and the needed memory, both in storage and RAM/GPU. To limit this, some restrictions have been imposed on the MSABERT model.

The first limiting factor for this model is the computing memory. The models requires to keep the complete BERT embeddings for a patent in memory. Using the full patent quickly exceeded the computing memory of the hardware available for this experiment, which was 16GB, even when processing a single patent at a time. In the experiments the MSABERT model was restricted to 4096 tokens per section, with a maximum total of around eleven thousand tokens, to reduce this problem. Preferably the model would use the full patent text to optimally determine the effect of utilizing more of the document. Further research could optimize the model to be capable of handling the entire the full document and research the improved performance of using more tokens.

The MSABERT model takes as an input an embedded version of the text. To create an embedded document from the text, the text is first split into chunks and then passed through a BERT model. On average, the patents contains 19 chunks when using a maximum of 4096 tokens per section. Since the BERT model is stable the patent will always result in the same embedding. To save on processing time, each patent embedding is stored as a file on the machine. On average a patent results in a file of around 70MB. Reading the document into memory took up substantial time, as parallelization was limited due to the lack of RAM/GPU capacity, but was significantly faster than recalculating the embeddings. Using a similar approach for millions of patents would require high amounts of storage space.

To counterbalance these effects, some restrictions had to be made on the MSABERT model and the experiments. As mentioned before, the model has been restricted to 4096 tokens. Secondly, the number of patents in the dataset has been limited to nine thousand. Even though for the experiments fifty thousand patents were available, and comparable studies have used millions of patents.

The last limitation is that the hyperparameter testing and some parts of the experiments were not repeated multiple times and more variations could have been tested. This research therefore functions mostly as a proof-of-concept and proposes some model adaptations and potential research areas.

Even with this restrictions, running a single experiment took around around 3 days. The complete processing time of the testing, hyperparameter tweaking and experiments is in the order of months and exceeds half a year. Furthermore, a lot of time was spend to optimize the model

speed, be able to utilize multiple GPUs and to get the program working on AMD ROCm. [1].

5.6 Contributions

Most previous research in the field of patent quality prediction has focused on traditional NLP methods or applying pre-trained text models. This thesis contributes to this field by adapting BERT models to patents specifically. A new model was introduced to utilize the text length and the multi-section structure of patents. Due to limitations, this thesis should mainly be interpreted as a proof-of-concept and as inspiration for further research. This study does however already offers some new insights.

5.6.1 AI

The experiments have shown that, under the conditions of a small training set and a limited number of tokens, the adaptation of the models does not increase the performance in a training scenario. This research contributes by showing that learning a single embedding for the entire patent performs very similar to learning a section-specific embedding. It further shows a way to combine these section-specific embeddings into a single embedding using attention. Combining the tensors using attention is to be preferred over a simple unweighted averaging. Furthermore it shows that explainability can be added to the model without any cost to the performance.

On the other hand, in a pre-training scenario the MSABERT model significantly outperforms the other models. This suggests that this model has learned more general characteristics of patents, compared to the others. Most research focuses on the full training scenario and compares model performance in this way. This study has shown that testing the generalizability of the model can contribute to understanding which model should be preferred.

This research shows the importance for understanding the model behavior, and that it can be improved without a cost in performance. The model performance can vary heavily when using it for unseen tasks in a pre-trained fashion. This underlines the need for explainable AI (XAI) and clearly communicating its limits. More and more researchers are coming to similar conclusions (Angelov, Soares, Jiang, Arnold, & Atkinson, 2021; Das & Rad, 2020; Došilović, Brčić, & Hlupić, 2018), leading to the proposal of some principals for explainable AI by Phillips, Hahn, Fontana, Broniatowski, and Przybocki (2020). However there are many that still believe in a trade-off between performance and explainability (Gunning et al., 2019). This thesis shows that this is not necessarily the case and models can be developed that incorporate features for explainability without a cost in performance.

This study also offers a comparison of different methods for creating a fixed-size output from an input of arbitrary length using a GRU. Three methods were tested: using the last element in the output sequence, using the average of the output sequence and lastly using the final hidden state of the GRU. The comparison showed that using the last element in the output sequence performed considerably worse than the other two methods. The difference between the other two options is small, but using the average of the GRU output performs slightly better in our experiments. This is similar to the method proposed by Iyyer, Manjunatha, Boyd-Graber, and Daumé III (2015), which has been at the basis of many later models including Cer et al. (2018).

5.6.2 Managerial Insights

This research also contributes to the field of innovation management. Most importantly, this research shows the relative importance of the different patent sections. Some previous research, like Larkey (1999), has already differentiated between the sections. This has however mainly focused on improving the efficiency and performance of text mining techniques and not the relative importance of the sections for certain tasks.

This research not only sheds light on the relative importance of the sections, it also shows that this depends on the particular tasks. Where the abstract is useful for predicting the Acceptance rate of a patent, it is not needed for CPC classification. Similarly, where the claims are of critical importance for CPC classification, this is the least important section for Acceptance prediction.

[1] Next to the available machines on the HPC Peregrine cluster, a personal AMD ROCm GPU machine was available. Although Pytorch should run in this environment without any changes, an issue was found where autocast was causing extreme memory usage. See https://github.com/pytorch/pytorch/issues/77878

This research also directly contradicts previous research in the same field. Fall, Törcsvári, Benzineb, and Karetka (2003) found that using only the first 300 words improves performance as compared to the full text, but this thesis shows that when using more powerful models it is actually beneficial to use the full text. This is an important insight, as this means that future research should focus on developing powerful models that are capable of utilizing the full text, instead of models that limit the number of words.

Furthermore, this research shows that predicting if a patent will be accepted or not is an inherently difficult task. None of the models have achieved an particularly high accuracy. Even though the models have outperformed the model introduced in Arts et al. (2021), the increase in performance is considerably small. As all models appear to reach a ceiling, it is likely that there are significant limitations on how well computers can perform on this particular task. Employees of a patent office are likely affected by many factors that are not represented in the patent text alone. One major shortcoming of the models presented here, is that they do not include any knowledge of the overall field for which the patents are submitted. They do not include any knowledge of previous research and do not include anything to capture whether the patent improves on previous patents. It remains difficult to model all knowledge of a patent office employee.

This does not mean that text models are not capable of capturing a patents innovation value. The MSABERT model shows very promising results on predicting the OECD quality indicator.This was a small scale experiment, but the MSABERT model trained on the acceptance task was well able to predict the quality indicator. This shows that these models could be applied in practice, for instance to evaluate patent portfolios or identify promising internal patents.

5.7 Future Research

This thesis has explored a variety of models, but it only scratches the surface of what is to discover. Many of the experiments have been only repeated a few times, further research should validate the results found here. Future research should furthermore do experiments with bigger datasets. Most comparable research has used millions of patents, whereas this thesis has only used around ten thousand patents. This would require further optimizing the code and likely better hardware. To allow datasets of this size, GPUs with more memory would be needed. The newest A100 Nvidia GPU has a memory size of 80GB (Nvidia, 2020), compared to the 16GB available for these experiments. Future research could also take a different approach by using more data but less epochs. The hyperparameter tweaking showed that the model was able to achieve near-optimal performance in only a few epochs with a high learning rate. This approach could yield different results, while still limiting the required processing time. In general, this thesis offered a first proof of concept but future research could take it to the next level.

Alternative ways of handling long patents text could also be explored. Cer et al. (2018) introduced a model that encodes a sentence or paragraph into embedding vectors. Google's Universal Sentence does not have a hard limit on the input length, but a maximum length of 512 is advised. The difference between this encoder and BERT is that the sentence encoder outputs a single vector of 512 for an input of size t. BERT outputs a vector of 512 per input token, or a matrix of $t \times 512$. The MSABERT model can handle any input embedding. Using sentence embeddings instead of BERT embeddings for each chunk could help lower the memory and processing requirements.

Another approach could be to lower the number of tokens in the text. This could for instance be achieved by using text summarization models, for example the one provided by Bird, Klein, and Loper (2009). Valuable information might however get lost in the summarization. Similarly, the number of tokens could be reduced by removing frequently occurring words.

Neither the accuracy of the CPC section nor the prediction whether patents will be accepted is at a satisfactory level, such that it could be used in practice and replace human patent evaluators. The models could however already aid the human workers if some adaptations are made.

The first approach could be to only use the model for predictions when its very confident. A simple technique would be to only use the predictions if the softmax output is either above 0.9 or below 0.1 and ignore it in the other cases. Although the softmax output is an indicator of the certainty of the prediction, it is not always sufficient. Gal and Ghahramani (2016) propose an alternative mechanism using dropout as a Bayesian approximation of the uncertainty.

Another approach would be to apply the model to a different set of tasks. The OECD patent quality prediction already showed a lot of potential. Future research could actually more elaborately test the MSABERT model for this task. If the models are capable of accurately predicting

the OECD patent quality, this could be extremely useful for companies to evaluate patents prior to their application or even prior to developing the actual technology. Similarly, other task like citation prediction or CPC subclass prediction could be tested.

In the experiments presented here, the models were trained on a single task. Although the models were tested for their ability in a transfer learning scenario, one could also test the models in a multi-task fashion. In multi-task learning a model learns multiple tasks at the same time. The idea behind multi-task learning is that the model is regularized because it needs to encode knowledge relevant to both tasks, instead of only task-specific information (Caruana, 1997). This can improve the performance of the model on the tasks, but also on unseen tasks.

Lastly, further research could further test the transfer learning capabilities of the models. This thesis has shown that there are potentially big differences between the performance of the different models on unseen tasks. Only some small scale experiments have been performed, that should be validated by others. Next to this, others could try to find out why there is a difference between the models. Adding a layer of explainability to the models could increase our understanding on the model performance.

5.8 Conclusion

Statistics derived from patents have become the standard measurement for innovation. Although patents statistics have inherent biases and flaws and are only an indirect measure, they are widely available and are shown to correlate well with innovation and market value. The main drawback of patent statistics is however that most of them only become reliable after substantial time has passed. This thesis aims to solve this problem, by creating a text model that can predict the innovation and market value of patents based solely on the patent text. Current state of the art machine learning text models like BERT are however not a perfect fit for patents, as patents can contain very long text and they have a multi-section structure. This thesis proposes a new model, called MSABERT, that is able to handle longer texts and the multi-section structure. Each section is handled separately, after which they are combined using attentitive pooling. This attentitive pooling also adds a layer of explainability to the model, showing the relative importance of each section.

The results show that this newly introduced model achieves similar performance as existing models when the models are trained in an end-to-end fashion. Predicting whether patents will be accepted solely on the text is a generally hard task and the performance of the models seems to hit a ceiling. When the models are used in a transfer learning scenario, the MSABERT model clearly outperforms the other models. The MSABERT model pre-trained on the acceptance task is able to accurately predict the patent value measured as the OECD quality indicator. The performance of the other models lacks far behind with an error about 5 times as high. This is a very promising result, as it shows that the MSABERT is capable of extracting the patent value from the text alone.

The MSABERT model improves performance in a transfer learning scenario and adds explainability without a decrease in performance. This allows users to use this model in applications to predict patent value early in the process. It can be a useful tool for companies to evaluate internal patents, evaluate patents of other companies for a merger or acquisition or as a tool for research.

References

Abbas, A., Zhang, L., & Khan, S. U. (2014). A literature review on the state-of-the-art in patent analysis. *World Patent Information*, *37*, 3–13.

Alcacer, J., & Gittelman, M. (2006). Patent citations as a measure of knowledge flows: The influence of examiner citations. *The Review of Economics and Statistics*, *88*(4), 774–779.

Alcácer, J., Gittelman, M., & Sampat, B. (2009). Applicant and examiner citations in us patents: An overview and analysis. *Research Policy*, *38*(2), 415–427.

Allan, J., Wade, C., & Bolivar, A. (2003). Retrieval and novelty detection at the sentence level. In *Proceedings of the 26th annual international acm sigir conference on research and development in informaion retrieval* (pp. 314–321).

Altszyler, E., Sigman, M., Ribeiro, S., & Slezak, D. F. (2016). Comparative study of lsa vs word2vec embeddings in small corpora: a case study in dreams database. *arXiv preprint arXiv:1610.01520*.

Angelov, P. P., Soares, E. A., Jiang, R., Arnold, N. I., & Atkinson, P. M. (2021). Explainable artificial intelligence: an analytical review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *11*(5), e1424.

Arts, S., Hou, J., & Gomez, J. C. (2021). Natural language processing to identify the creation and impact of new technologies in patent text: Code, data, and new measures. *Research Policy*, *50*(2), 104144.

Ashtor, J. H. (2019). Investigating cohort similarity as an ex ante alternative to patent forward citations. *Journal of Empirical Legal Studies*, *16*(4), 848–880.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Balsmeier, B., Assaf, M., Chesebro, T., Fierro, G., Johnson, K., Johnson, S., ... others (2018). Machine learning and natural language processing on the patent corpus: Data, tools, and new measures. *Journal of Economics & Management Strategy*, *27*(3), 535–553.

Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Bergmann, I., Butzke, D., Walter, L., Fuerste, J. P., Moehrle, M. G., & Erdmann, V. A. (2008). Evaluating the risk of patent infringement by means of semantic patent analysis: the case of dna chips. *R&d Management*, *38*(5), 550–562.

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... others (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

Caceres, P. (2020). *The recurrent neural network - theory and implementation of the elman network and lstm.* Retrieved from https://pabloinsente.github.io/the-recurrent-net

Caruana, R. (1997). Multitask learning. *Machine learning*, *28*(1), 41–75.

Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., ... others (2018). Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Chen, H., Zhang, G., Zhu, D., & Lu, J. (2017). Topic-based technological forecasting based on patent data: A case study of australian patents from 2000 to 2014. *Technological Forecasting and Social Change*, *119*, 39–52.

Cheng, J., Dong, L., & Lapata, M. (2016). Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, *2*(4), 303–314.

Das, A., & Rad, P. (2020). Opportunities and challenges in explainable artificial intelligence (xai): A survey. *arXiv preprint arXiv:2006.11371*.

deGrazia, C. A., Frumkin, J. P., & Pairolero, N. A. (2020). Embracing invention similarity for the measurement of vertically overlapping claims. *Economics of Innovation and New Technology*, *29*(2), 113–146.

Deshpande, M. (2020). *Classification with support vector machines.* Retrieved from https://

pythonmachinelearning.pro/classification-with-support-vector-machines/
Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
Došilović, F. K., Brčić, M., & Hlupić, N. (2018). Explainable artificial intelligence: A survey. In *2018 41st international convention on information and communication technology, electronics and microelectronics (mipro)* (pp. 0210–0215).
Dumais, S. T., Letsche, T. A., Littman, M. L., & Landauer, T. K. (1997). Automatic cross-language retrieval using latent semantic indexing. In *Aaai spring symposium on cross-language text and speech retrieval* (Vol. 15, p. 21).
Education, I. C. (2020). *Convolutional neural networks.* Retrieved from https://www.ibm.com/cloud/learn/convolutional-neural-networks
Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179–211.
Eurostat, O. (2005). Oslo manual: guidelines for collecting and interpreting innovation data. *A joint publication of OECD and Eurostat. Paris: OECD*.
Falk, N., & Train, K. (2017). Patent valuation with forecasts of forward citations. *Journal of Business Valuation and Economic Loss Analysis*, *12*(1), 101–121.
Fall, C. J., Törcsvári, A., Benzineb, K., & Karetka, G. (2003). Automated categorization in the international patent classification. In *Acm sigir forum* (Vol. 37, pp. 10–25).
for Economic Co-operation, O., & Development. (2022). *Oecd, triadic patent families database, july 2021.* Retrieved from https://data.oecd.org/rd/triadic-patent-families.htm
Fukushima, K., Miyake, S., & Ito, T. (1983). Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE transactions on systems, man, and cybernetics*(5), 826–834.
Gal, Y., & Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning* (pp. 1050–1059).
Gault, F. (2018). Defining and measuring innovation in all sectors of the economy. *Research policy*, *47*(3), 617–622.
Gerken, J. M., & Moehrle, M. G. (2012). A new instrument for technology monitoring: novelty in patents measured by semantic patent analysis. *Scientometrics*, *91*(3), 645–670.
Giuri, P., Mariani, M., Brusoni, S., Crespi, G., Francoz, D., Gambardella, A., ... others (2007). Inventors and invention processes in europe: Results from the patval-eu survey. *Research policy*, *36*(8), 1107–1127.
Griliches, Z. (1990). *Patent statistics as economic indicators: a survey part 2.* NBER.
Guissous, A. E. (n.d.). *Figure 3.* Retrieved from https://www.researchgate.net/figure/Example-for-the-max-pooling-and-the-average-pooling-with-a-filter-size-of-22-and-a_fig15_337336341
Gunning, D., Stefik, M., Choi, J., Miller, T., Stumpf, S., & Yang, G.-Z. (2019). Xai—explainable artificial intelligence. *Science robotics*, *4*(37), eaay7120.
Guyot, J., Benzineb, K., Falquet, G., & Shift, S. (2010). myclass: A mature tool for patent classification. In *Clef (notebook papers/labs/workshops)*.
Hain, D., Jurowetzki, R., Buchmann, T., & Wolf, P. (2020). Text-based technological signatures and similarities: How to create them and what to do with them. *arXiv preprint arXiv:2003.12303*.
Hall, B. H., Jaffe, A. B., & Trajtenberg, M. (2001). *The nber patent citation data file: Lessons, insights and methodological tools* (Tech. Rep.). National Bureau of Economic Research.
Han, X., Zhang, Z., Ding, N., Gu, Y., Liu, X., Huo, Y., ... others (2021). Pre-trained models: Past, present and future. *AI Open*.
Hasan, M. A., Spangler, W. S., Griffin, T., & Alba, A. (2009). Coa: Finding novel patents through text analysis. In *Proceedings of the 15th acm sigkdd international conference on knowledge discovery and data mining* (pp. 1175–1184).
Hidden markov model graph. (n.d.). Retrieved from https://en.wikipedia.org/wiki/File:HMMGraph.svg
Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, *313*(5786), 504–507.
Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.
Iyyer, M., Manjunatha, V., Boyd-Graber, J., & Daumé III, H. (2015). Deep unordered composition

rivals syntactic methods for text classification. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)* (pp. 1681–1691).

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*.

Jordan, M. I. (1989). Serial order: A parallel distributed processing approach. *San Diego: University of California, Institute for Cognitive Science*.

Joshi, M., Levy, O., Weld, D. S., & Zettlemoyer, L. (2019). Bert for coreference resolution: Baselines and analysis. *arXiv preprint arXiv:1908.09091*.

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Kaplan, S., & Vakili, K. (2015). The double-edged sword of recombination in breakthrough innovation. *Strategic Management Journal*, *36*(10), 1435–1457.

Karpathy, A. (2015). *Sequences.* Retrieved from https://karpathy.github.io/2015/05/21/rnn-effectiveness/

Kelly, B., Papanikolaou, D., Seru, A., & Taddy, M. (2018). *Measuring technological innovation over the long run* (Tech. Rep.). National Bureau of Economic Research.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kitaev, N., Kaiser, Ł., & Levskaya, A. (2020). Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Kovács, B. (2017). Too hot to reject: The effect of weather variations on the patent examination process at the united states patent and trademark office. *Research Policy*, *46*(10), 1824–1835.

Larkey, L. S. (1999). A patent search and classification system. In *Proceedings of the fourth acm conference on digital libraries* (pp. 179–187).

Lazar, D. (n.d.). *Perceptron: Explanation, implementation and a visual example.* Retrieved from https://towardsdatascience.com/perceptron-explanation-implementation-and-a-visual-example-3c8e76b4e2d1

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lee, C., Kang, B., & Shin, J. (2015). Novelty-focused patent mapping for technology opportunity analysis. *Technological Forecasting and Social Change*, *90*, 355–365.

Lee, J.-S., & Hsiang, J. (2019). Patentbert: Patent classification with fine-tuning a pre-trained bert model. *arXiv preprint arXiv:1906.02124*.

Lemley, M. A., & Sampat, B. (2012). Examiner characteristics and patent office outcomes. *Review of economics and statistics*, *94*(3), 817–827.

Lemley, M. A., & Shapiro, C. (2005). Probabilistic patents. *Journal of Economic Perspectives*, *19*(2), 75–98.

Li, S., Hu, J., Cui, Y., & Hu, J. (2018). Deeppatent: patent classification with convolutional neural networks and word embedding. *Scientometrics*, *117*(2), 721–744.

Li, X., & Croft, W. B. (2005). Novelty detection based on sentence level patterns. In *Proceedings of the 14th acm international conference on information and knowledge management* (pp. 744–751).

Li, Y.-R., Wang, L.-H., & Hong, C.-F. (2009). Extracting the significant-rare keywords for patent analysis. *Expert Systems with Applications*, *36*(3), 5200–5204.

Liu, X., Yan, J., Xiao, S., Wang, X., Zha, H., & Chu, S. M. (2017). On predictive patent valuation: Forecasting patent citations and their types. In *Proceedings of the thirty-first aaai conference on artificial intelligence* (pp. 1438–1444).

Luhn, H. P. (1957). A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, *1*(4), 309–317.

Martín-de Castro, G., Delgado-Verde, M., Navas-López, J. E., & Cruz-González, J. (2013). The moderating role of innovation culture in the relationship between knowledge assets and product innovation. *Technological Forecasting and Social Change*, *80*(2), 351–363.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

The mcculloch-pitts neuron. (n.d.). Retrieved from https://www.oreilly.com/library/view/artificial-intelligence-by/9781788990547/97eeab76-9e0e-4f41-87dc-03a65c3efec3.xhtml

Medatwal, S. (2018). *Applying word2vec on our catalog data.* Retrieved from https://medium.com/arvind-internet/applying-word2vec-on-our-catalog-data-2d74dfee419d

Merialdo, B. (1994). Tagging English text with a probabilistic model. *Computational Linguistics*, *20*(2), 155–171. Retrieved from https://aclanthology.org/J94-2001

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Yih, W.-t., & Zweig, G. (2013). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies* (pp. 746–751).

Minsky, M., & Papert, S. A. (1988). *Perceptrons: An introduction to computational geometry.* MIT press.

Mitchell, T. (1997). Machine learning.

Moehrle, M. (2010). Measures for textual patent similarities: a guided way to select appropriate approaches. *Scientometrics*, *85*(1), 95–109.

Nagaoka, S., Motohashi, K., & Goto, A. (2010). Patent statistics as an innovation indicator. In *Handbook of the economics of innovation* (Vol. 2, pp. 1083–1127). Elsevier.

Noh, H., Jo, Y., & Lee, S. (2015). Keyword selection and processing strategy for applying text mining to patent analysis. *Expert Systems with Applications*, *42*(9), 4348–4360.

Nvidia. (2020). *Nvidia a100 tensor core gpu.* Retrieved from https://www.nvidia.com/en-us/data-center/a100/

Office, I. P. (n.d.-a). *Patent factsheets: Claims.* Retrieved from https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/873879/WS0057_Claims_Jan_2020.pdf

Office, I. P. (n.d.-b). *Patent factsheets: Description.* Retrieved from https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/873877/WS0055_Description_Jan_2020.pdf

OpenAI. (n.d.). *Openai gpt tokenizer.* Retrieved from https://beta.openai.com/tokenizer

Organization, W. I. P. (n.d.). *Patents.* Retrieved from https://www.wipo.int/patents/en/

Park, H., Yoon, J., & Kim, K. (2012). Identifying patent infringement using sao based semantic technological similarities. *Scientometrics*, *90*(2), 515–529.

Patent, T. U. S., & Office, T. (n.d.). *Bulk search and download.* Retrieved from https://developer.uspto.gov/api-catalog/bulk-search-and-download

Pennington, J., Socher, R., & Manning, C. D. (n.d.). *Glove: Global vectors for word representation.* Retrieved from https://nlp.stanford.edu/projects/glove/

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (pp. 1532–1543).

Phillips, P. J., Hahn, C. A., Fontana, P. C., Broniatowski, D. A., & Przybocki, M. A. (2020). Four principles of explainable artificial intelligence. *Gaithersburg, Maryland*.

Phillips v. awh corp. (2005). Retrieved from https://casetext.com/case/phillips-v-awh-corp-3

Prabhu. (2018). *Understanding of convolutional neural network (cnn) — deep learning.* Retrieved from https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

Qassemi, F. (2020). *A few remarks on gpt3.* Retrieved from https://www.linkedin.com/pulse/few-remarks-gpt3-farzad-qassemi/

Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., & Huang, X. (2020). Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 1–26.

Rabiner, L. R., & Juang, B. (1986). A tutorial on hidden markov models. *IEEE ASSP Magazine*, *3*(1), 4–16.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9.

Reitzig, M. (2004). Improving patent valuations for management purposes—validating new indicators by analyzing application rationales. *Research policy*, *33*(6-7), 939–957.

Retnasaba, G. (2008). Why it is easier to get a patent in september. *Available at SSRN 1121132*.

Robinson, T., Hochberg, M., & Renals, S. (1996). The use of recurrent neural networks in

continuous speech recognition. In *Automatic speech and speaker recognition* (pp. 233–258). Springer.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Rosenblatt perceptron. (n.d.). Retrieved from https://nl.wikipedia.org/wiki/Perceptron#/media/Bestand:Rosenblattperceptron.png

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Salton, G. (1971). *The smart retrieval system—experiments in automatic document processing.* Prentice-Hall, Inc.

Salton, G., & McGill, M. J. (1983). *Introduction to modern information retrieval.* mcgraw-hill.

Scherer, F. M., & Harhoff, D. (2000). Technology policy for a world of skew-distributed outcomes. *Research policy*, *29*(4-5), 559–566.

Schick, I. C. (2019). *Us patent filings: Peaking or false peak.* Retrieved from https://blog.specif.io/2019/03/06/us-patent-filings-peaking-or-false-peak/

Schuster, M., & Nakajima, K. (2012). Japanese and korean voice search. In *2012 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 5149–5152).

Sennrich, R., Haddow, B., & Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Smola, A., Song, L., & Teo, C. H. (2009). Relative novelty detection. In *Artificial intelligence and statistics* (pp. 536–543).

spacy tokenization. (n.d.). Retrieved from https://spacy.io/usage/linguistic-features#tokenization

Squicciarini, M., Dernis, H., & Criscuolo, C. (2013). Measuring patent quality: Indicators of technological and economic value.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

Teodoridis, F., Lu, J., & Furman, J. L. (2020). Measuring the direction of innovation: Frontier tools in unassisted machine learning. *Available at SSRN 3596233*.

Trappey, A. J., Hsu, F.-C., Trappey, C. V., & Lin, C.-I. (2006). Development of a patent document classification and search platform using a back-propagation network. *Expert Systems with Applications*, *31*(4), 755–765.

van Dongen, T., Wenniger, G. M. d. B., & Schomaker, L. (2020). Schubert: Scholarly document chunks with bert-encoding boost citation count prediction. *arXiv preprint arXiv:2012.11740*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*, 5998–6008.

Vig, J. (2019). A multiscale visualization of attention in the transformer model. *arXiv preprint arXiv:1906.05714*.

Wang, J., & Chen, Y.-J. (2019). A novelty detection patent mining approach for analyzing technological opportunities. *Advanced Engineering Informatics*, *42*, 100941.

Williams, R. J., & Zipser, D. (1995). Gradient-based learning algorithms for recurrent. *Backpropagation: Theory, architectures, and applications*, *433*, 17.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ... others (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, *9*(4), 611–629.

Yang, P., Sun, X., Li, W., & Ma, S. (2018). Automatic academic paper rating based on modularized hierarchical convolutional neural network. *arXiv preprint arXiv:1805.03977*.

Yang, S., Yu, X., & Zhou, Y. (2020). Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example. In *2020 international workshop on electronic communication and artificial intelligence (iwecai)* (pp. 98–101).

Younge, K. A., & Kuhn, J. M. (2016). Patent-to-patent similarity: a vector space model. *Available at SSRN 2709238*.

Zahran, M. (n.d.). *A hypothetical example of multilayer perceptron network.* Retrieved from https://www.researchgate.net/figure/A-hypothetical-example-of-Multilayer-Perceptron-Network_fig4_303875065

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the ieee international conference on computer vision* (pp. 19–27).

A Appendix

A.1 Patent Examples

Description: Typical Example

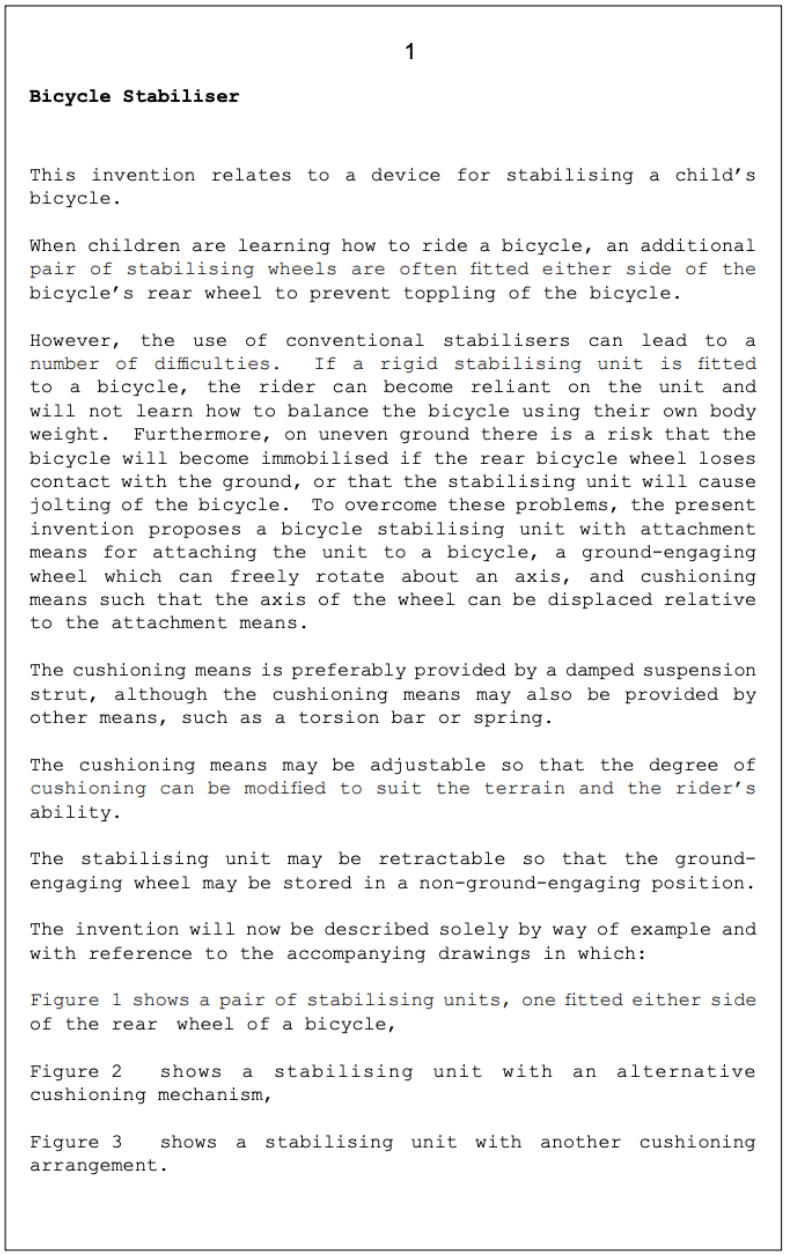
Page numbering

Title providing an indication of the invention

Introduction and background to the invention

Summary of the invention, also known as a 'statement of invention'

Optional features



1

Bicycle Stabiliser

This invention relates to a device for stabilising a child's bicycle.

When children are learning how to ride a bicycle, an additional pair of stabilising wheels are often fitted either side of the bicycle's rear wheel to prevent toppling of the bicycle.

However, the use of conventional stabilisers can lead to a number of difficulties. If a rigid stabilising unit is fitted to a bicycle, the rider can become reliant on the unit and will not learn how to balance the bicycle using their own body weight. Furthermore, on uneven ground there is a risk that the bicycle will become immobilised if the rear bicycle wheel loses contact with the ground, or that the stabilising unit will cause jolting of the bicycle. To overcome these problems, the present invention proposes a bicycle stabilising unit with attachment means for attaching the unit to a bicycle, a ground-engaging wheel which can freely rotate about an axis, and cushioning means such that the axis of the wheel can be displaced relative to the attachment means.

The cushioning means is preferably provided by a damped suspension strut, although the cushioning means may also be provided by other means, such as a torsion bar or spring.

The cushioning means may be adjustable so that the degree of cushioning can be modified to suit the terrain and the rider's ability.

The stabilising unit may be retractable so that the ground-engaging wheel may be stored in a non-ground-engaging position.

The invention will now be described solely by way of example and with reference to the accompanying drawings in which:

Figure 1 shows a pair of stabilising units, one fitted either side of the rear wheel of a bicycle,

Figure 2 shows a stabilising unit with an alternative cushioning mechanism,

Figure 3 shows a stabilising unit with another cushioning arrangement.

Figure A.1: Example of the description section of a patent. Image from Office (n.d.-b)

Description: Typical Example

Page numbering

Title providing an indication of the invention

Introduction and background to the invention

Summary of the invention, also known as a 'statement of invention'

Optional features

1

Bicycle Stabiliser

This invention relates to a device for stabilising a child's bicycle.

When children are learning how to ride a bicycle, an additional pair of stabilising wheels are often fitted either side of the bicycle's rear wheel to prevent toppling of the bicycle.

However, the use of conventional stabilisers can lead to a number of difficulties. If a rigid stabilising unit is fitted to a bicycle, the rider can become reliant on the unit and will not learn how to balance the bicycle using their own body weight. Furthermore, on uneven ground there is a risk that the bicycle will become immobilised if the rear bicycle wheel loses contact with the ground, or that the stabilising unit will cause jolting of the bicycle. To overcome these problems, the present invention proposes a bicycle stabilising unit with attachment means for attaching the unit to a bicycle, a ground-engaging wheel which can freely rotate about an axis, and cushioning means such that the axis of the wheel can be displaced relative to the attachment means.

The cushioning means is preferably provided by a damped suspension strut, although the cushioning means may also be provided by other means, such as a torsion bar or spring.

The cushioning means may be adjustable so that the degree of cushioning can be modified to suit the terrain and the rider's ability.

The stabilising unit may be retractable so that the ground-engaging wheel may be stored in a non-ground-engaging position.

The invention will now be described solely by way of example and with reference to the accompanying drawings in which:

Figure 1 shows a pair of stabilising units, one fitted either side of the rear wheel of a bicycle,

Figure 2 shows a stabilising unit with an alternative cushioning mechanism,

Figure 3 shows a stabilising unit with another cushioning arrangement.

Figure A.1: (continued) Example of the description section of a patent. Image from Office (n.d.-b)

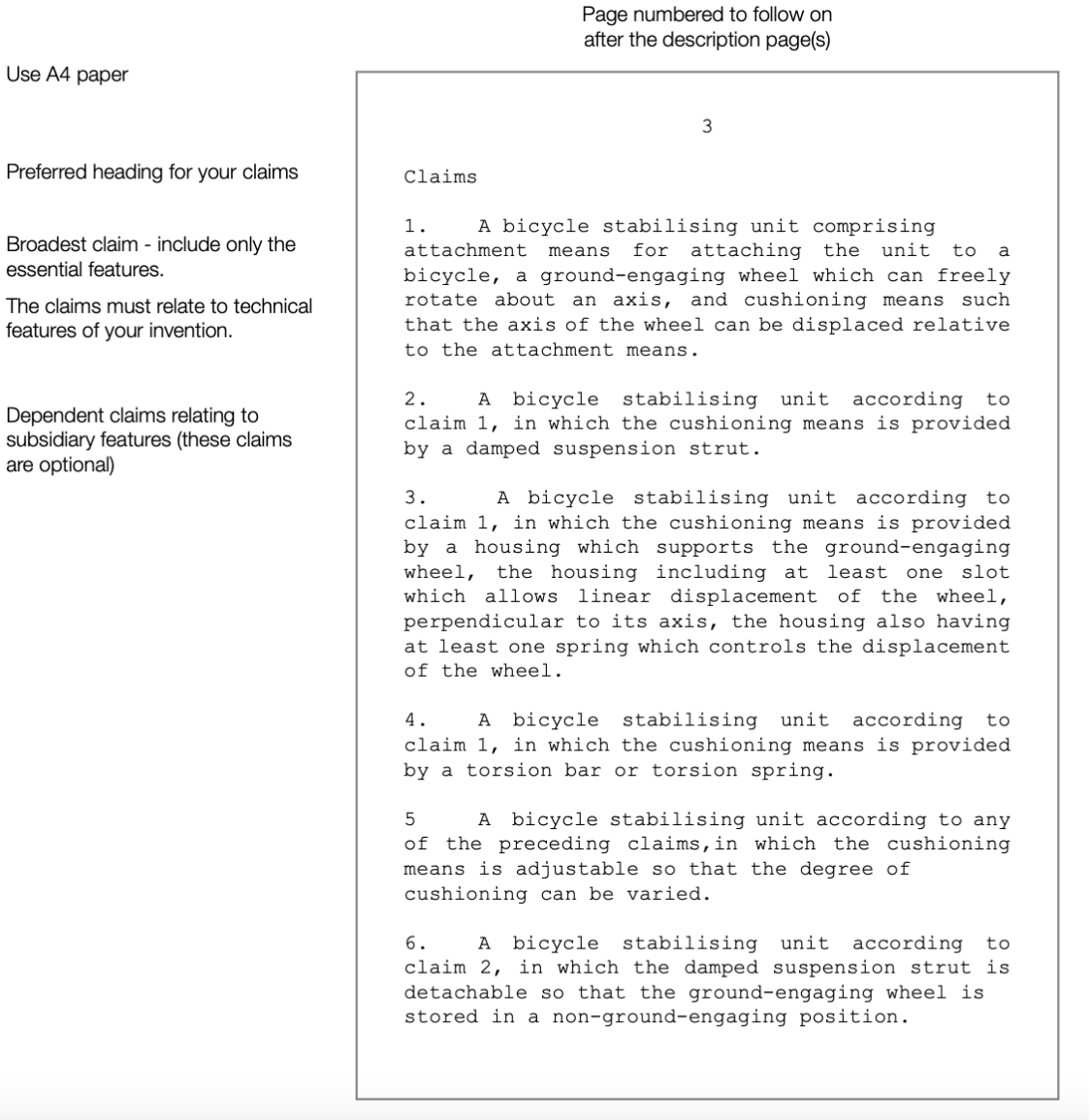


Figure A.2: Example of the claims section of a patent. Image from Office (n.d.-a)