UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER'S THESIS No. 91

Combining Adaptive Pretraining Techniques for the Task of Automated Fact-Checking

Ivan Rep

Zagreb, july 2023

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 10 March 2023

MASTER THESIS ASSIGNMENT No. 91

Student:	**Ivan Rep (0036517236)**
Study:	Computing
Profile:	Computer Science
Mentor:	prof. Jan Šnajder

Title: **Combining adaptive pretraining techniques for the task of automated fact-checking**

Description:

In today's time, which is defined by the abundance of available information, there is an ever-present need for quick, cheap, and effective fact-checking. This need led to the idea of automated fact-checking, for which purpose machine learning and especially natural language processing appears as the most promising candidate Currently, natural language processing is dominated by large pre-trained language models based on transformer architecture The idea of combining large language models with the fact-checking task was born especially with the use of adaptive pretraining techniques that promise better results The topic of the thesis is to develop models for automated fact-checking based on large pre-trained language models that combine different adaptive pre-training techniques. Models for two tasks need to be developed: claim check-worthiness detection and verdict prediction. It is necessary to study the relevant literature on automated fact-checking and adaptive pre-training techniques. A thorough evaluation of the developed model should be carried out on suitable datasets. Compare the developed model with other baseline models. All references must be cited, and all source code, documentation, executables, and datasets must be provided with the thesis.

Submission date: 23 June 2023

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

DIPLOMSKI ZADATAK br. 91

Pristupnik:	**Ivan Rep (0036517236)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Jan Šnajder

Zadatak: **Kombiniranje tehnika adaptivnoga predtreniranja za zadatak automatizirane provjere činjenica**

Opis zadatka:

U današnjem vremenu, koje je definirano mnoštvom dostupnih informacija, sve je prisutnija potreba za brzom, jeftinom i efektivnom provjerom činjenica. Ova potreba dovela je do ideje o automatiziranoj provjeri činjenica, za koju se svrhu strojno učenje, a posebice strojna obrada prirodnog jezika, pojavljuje kao najperspektivniji kandidat. Trenutačno strojnom obradom prirodnog jezika dominiraju veliki predtrenirani jezični modeli temeljeni na arhitekturi transformatora. Rađa se ideja za spajanjem velikih jezičnih modela sa zadatkom provjere činjenica, a posebice uz korištenje tehnika adaptivnog predtreniranja koje obećavaju bolje rezultate. Tema diplomskoga rada jest razviti modele za automatiziranu provjeru činjenica temeljene na velikim predtreniranim jezičnim modelima koji kombiniraju različite tehnike adaptivnog predtreniranja. Potrebno je razviti modele za dva zadatka: detekciju provjerivih tvrdnji i provjeru istinitosti provjerivih tvrdnji. Potrebno je proučiti relevantnu literaturu na temu automatizirane provjere činjenica i tehnika adaptivnog predtreniranja. Treba provesti iscrpno vrednovanje razvijenog modela na prikladnim skupovima podataka. Provesti usporedbu razvijenog modela s drugim modelima na istom zadatku. Radu je potrebno priložiti izvorni kod, opise skupova podataka i programsku dokumentaciju te citirati korištenu literaturu.

Rok za predaju rada: 23. lipnja 2023.

I would like to thank mag. ing. Fran Jelenić and prof. dr. sc. Jan Šnajder for numerous advice while writing my thesis.

TABLE OF CONTENTS

1. Introduction

Machine learning techniques have revolutionized the way numerical data is exploited. Combined with techniques natural language processing (NLP) provides, natural language understanding tasks (NLU) have become easier to solve. Notable findings in NLP include learning high-quality word vector representations known as *word2vec*. Machine learning techniques have also improved with the rise of deep learning, a subfield of machine learning which uses neural networks with hidden layers for modeling complex patterns. The first breakthrough in language understanding occurred with the recurrent neural network (RNN) architecture (Rumelhart et al., 1986). Newer techniques such as long short-term memory cells (LSTM) (Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRU) (Chung et al., 2014) further improved the beneficial effect of introducing recurrent connections into deep models.

Especially with the development of computing power, modern deep learning, and constant research in NLP, the performance in NLU tasks has skyrocketed. The most notable improvement happened with BERT, a transformer-based model (Vaswani et al., 2017) with a clever pretraining technique designed by Devlin et al. (2018), whose performance surpassed human performance on multiple benchmarking datasets.[1] The BERT pretraining procedure paved the way for many other pretrained architectures and pretraining became an important part of improving the performance of NLP models.

Due to the vast amount of information circulating on the internet, fact-checking has become increasingly important. Information and misinformation spreads easily in modern media outlets. For this reason, automated fact-checking has become a necessity. Researchers heavily rely on available textual data and techniques such as natural language processing, machine learning, and knowledge processing.

Fact-checking is the task of assessing whether claims made in written or spoken language are true, as stated by Guo et al. (2021). This task is an interdisciplinary area of artificial intelligence, journalism, and sociology. Manual fact-checking is of-

[1] https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html

ten conducted by organizations across the globe, although a big problem is that the number of verified articles is around half of the number of published articles (Lewis et al., 2008). Due to the rate at which new information spreads, manual fact-checking becomes insufficient.

Understanding language is crucial in fact-checking, as it is a prerequisite for creating highly accurate models. Fact-checkers rely on their world knowledge and language comprehension when assessing the truthfulness of the content they examine. When investigating the veracity or check-worthiness of a claim, it is important to recognize some words are more informative, e.g., named entities in a claim might be a strong indicator of check-worthiness of a claim. Pretraining techniques serve as a valuable proxy for understanding language in large language models (LLMs), meaning they capture linguistic patterns and contextual cues.

The rest of the thesis is organized as follows: section 2 covers the basics of machine learning theory, including the logistic regression algorithm. In section 3 the fundamentals of natural language processing and its applications are examined. Section 4 focuses on deep learning and more recent advances with applications in the field of NLP. Section 5 examines the pretraining techniques used in NLP. Finally section 6 provides insight into all the datasets and experiments covered in the thesis.

2. Machine learning

Machine learning is a subfield of artificial intelligence which "addresses the question of how to build computer programs that improve their performance at some task through experience" (Mitchell, 1997). Without being explicitly programmed to solve the problem, machine learning algorithms combine computational power and mathematical models to find patterns in the data. Due to the abundance of available data in this day and age, machine learning algorithms have revolutionized technology and are present in numerous fields such as bioinformatics, computer vision, natural language processing, and finance.

Machine learning can be divided into many subfields. The most common way to partition machine learning is into supervised and unsupervised learning. Supervised learning uses labeled data, meaning annotators have to manually assign labels corresponding to examples. It is important to note that the dataset is always a sample, which is a subset of all possible examples because it is either impossible to collect all examples or it is too expensive. The labels in supervised learning are defined using a set of classes in a classification task or real numbers in a regression task. More formally, a dataset for supervised learning can be defined as in (2.1), where $\mathbf{x}^{(i)}$ represents values of features for example, i, $y^{(i)}$ represents the label for example i, $\mathcal{X}$ represents the space of all possible examples and $\mathcal{Y}$ represents the space of all possible labels.

$$\mathcal{D}_{supervised} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N} \subseteq \mathcal{X} \times \mathcal{Y} \tag{2.1}$$

A dataset for unsupervised learning is simpler to define, as shown in (2.2), because it does not use labels. All the previously mentioned symbols in the equation have the same meaning.

$$\mathcal{D}_{unsupervised} = \{(\mathbf{x}^{(i)})\}_{i=1}^{N} \subseteq \mathcal{X} \tag{2.2}$$

As Alpaydin (2014) states, each machine learning algorithm has three aspects: the model, the loss function, and the optimization procedure. The model is a set of hypotheses (functions) parametrized by some free parameters. The equation (2.3) represents the model, where $\boldsymbol{\theta}$ represents some fixed values of the free parameters. This

means changing the values of the parameters changes the hypothesis as well.

$$g(\mathbf{x}|\boldsymbol{\theta}) \tag{2.3}$$

The second component of the machine learning algorithm is the loss function. The loss function defines the error on a single example, while the *approximation error* is the sum of losses over individual instances. The approximation error is shown in equation (2.4), where $L(\boldsymbol{\theta}|\mathbf{x}^{(i)}, y^{(i)})$ represents the loss calculated using parameters $\boldsymbol{\theta}$ over a single example in the dataset.

$$E(\boldsymbol{\theta}|\mathcal{D}) = \sum_{(\mathbf{x}^{(i)},y^{(i)})\in\mathcal{D}} L(\boldsymbol{\theta}|\mathbf{x}^{(i)}, y^{(i)}) \tag{2.4}$$

The most often used approach is deriving the approximation error using the negative log-likelihood, meaning the loss function can be calculated if the posterior probability distribution of the label is known. Another convention is that the sum is replaced with an average so that the approximation error does not depend on the size of the dataset.

The final component in a machine learning algorithm is the optimization procedure. The optimization procedure tries to find the parameters $\boldsymbol{\theta}^*$ that minimize the approximation error. Equation (2.5) represents the optimization procedure all the previously mentioned symbols have the same meaning.

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} E(\boldsymbol{\theta}|\mathcal{D}) \tag{2.5}$$

2.1. Logistic regression

One of the first and most important algorithms in supervised machine learning is the logistic regression. Besides being a machine learning algorithm, it was a staple algorithm for defining deep feedforward neural networks (FFNN). The binary logistic regression algorithm is used for binary classification, while the multinomial logistic regression is used for multi-class classification.

2.1.1. Binary logistic regression

The model of the binary logistic regression algorithm is a dot product of the feature vector and the weights, followed by an activation function called the logistic function. The logistic function is defined as in equation (2.6), while the function graph is shown in figure 2.1.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

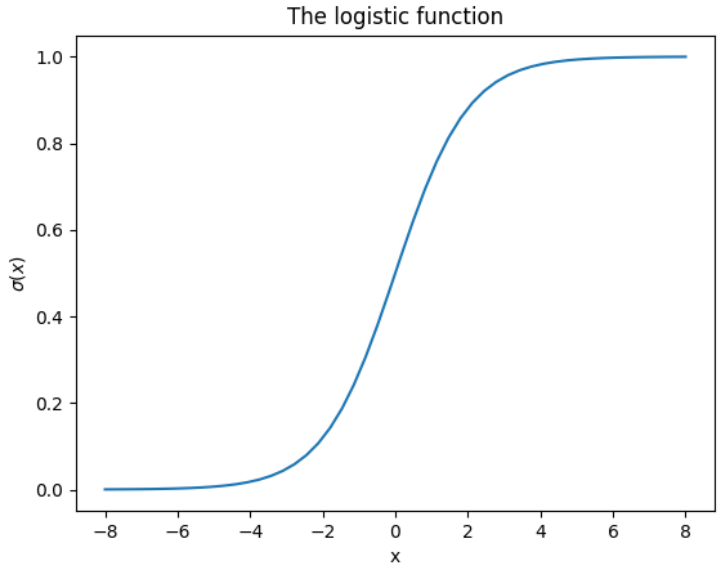

Figure 2.1: Values of the logistic function on the interval $(-8, 8)$.

A property the logistic regression relies on is that the output is bounded in the interval $(0, 1)$, meaning it can be regarded as a probability. The model of the binary logistic regression algorithm can be seen in equation (2.7), where $\mathbf{w}$ represents the learnable weights, and $\mathbf{x}$ represents the feature vector.

$$g(y = 1|\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^\intercal \mathbf{x}) \tag{2.7}$$

The loss function of the algorithm uses an assumption that the label follows a Bernoulli distribution. The Bernoulli distribution is a probability distribution where a random variable can take the value 1 with probability p, and value 0 with probability $1 - p$. The probability mass function of the Bernoulli distribution is shown in equation (2.8), where Y denotes a random variable, y represents an occurrence of the random variable Y, and p represents the probability that the value of the random variable Y is equal to 1.

$$P(Y = y) = p^y(1 - p)^{(1-y)} \tag{2.8}$$

The connection between the Bernoulli distribution and the logistic regression model is that the model is predicting the posterior probability of the label given the feature vector, meaning it can replace the probability term in the equation. Including the fact that the approximation error is proportional to the negative log-likelihood and that the approximation error should not depend on the size of the dataset, the final approxima-

tion error is shown in equation (2.9).

$$E(\mathbf{w}|\mathcal{D}) = -\frac{1}{N}\sum_{i=1}^{N} y^{(i)} \ln g(y^{(i)} = 1|\mathbf{x}^{(i)}; \mathbf{w}) + (1 - y^{(i)}) \ln(1 - g(y^{(i)} = 1|\mathbf{x}^{(i)}; \mathbf{w})) \tag{2.9}$$

The most often used optimization procedure is gradient descent, which calculates the derivative of the error function with respect to the parameters and takes a step in the opposite direction. It is an iterative procedure, meaning the current parameter values are calculated depending on the previous parameter values. The gradient descent optimization procedure is shown in equation (2.10), where η represents the learning rate, $\mathbf{w}^{(t+1)}$ represents the parameter values in the next iteration, and $\mathbf{w}^{(t)}$ represents the parameter values in the current iteration.

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial E}{\partial \mathbf{w}^{(t)}} \tag{2.10}$$

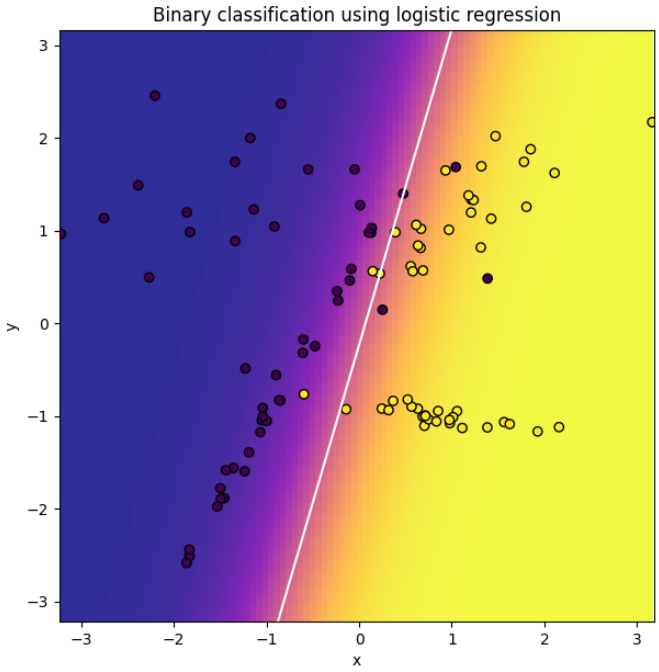

Figure 2.2: Binary classification of two-dimensional input data using a binary logistic regression classifier.

Figure 2.2 shows an example of solving a binary classification problem using the binary logistic regression classifier with two-dimensional input data. The color gradient represents the posterior probabilities. The decision boundary is linear, although it

can be modified with the help of feature mapping, where the input data is transformed using a predefined function.

2.1.2. Multinomial logistic regression

The multinomial logistic regression is more complicated than its binary predecessor. The logistic function accepts a scalar value, meaning it cannot be used in a multi-class setting. There exists a generalization function of the sigmoid and its name is the softmax function, shown in equation (2.11), where K represents the number of classes. The sigmoid function can be derived from the softmax function.

$$\text{softmax}(\mathbf{s})_i = \frac{\exp(s_i)}{\sum_{k=1}^{K} \exp(s_k)} \tag{2.11}$$

The model changes to accomodate the switch from the sigmoid function to the softmax function. The equation (2.12) shows the model that the multinomial logistic regression algorithm uses, where $\mathbf{p}$ denotes the vector of probabilities, $\mathbf{W}$ represents the weight matrix, $\mathbf{x}$ represents the feature vector, and $\mathbf{b}$ represents the bias vector. The weight matrix is of dimension $D \times C$, while the feature vector is of dimension $D \times 1$, indicating that the probabilties $\mathbf{p}$ and biases $\mathbf{b}$ will have dimension $C \times 1$.

$$\mathbf{p} = \text{softmax}(\mathbf{W}^\intercal \mathbf{x} + \mathbf{b}) \tag{2.12}$$

The loss function must change as well. The Bernoulli distribution is not appropriate anymore, since the problem in question is a multi-class problem. The solution to the problem is to use the categorical distribution, which is similar to a Bernoulli distribution but uses an arbitrary amount of classes. The probability mass function is defined as in equation (2.13).

$$P(\mathbf{Y} = \mathbf{y}) = \prod_{k=1}^{K} p_k^{y_k} \tag{2.13}$$

As with the Bernoulli distribution, the posterior probabilities p_k are exactly what the model is predicting, meaning they can be replaced. The final approximation error expression after applying the negative log-likelihood and averaging is shown in equation (2.14), where s_{ij} represents the logit for the correct class for example i.

$$E(\mathbf{W}|\mathcal{D}) = -\frac{1}{N} \sum_{i=1}^{N} \left(s_{ij} + \ln \sum_{k=1}^{K} \exp(s_{ik}) \right) \tag{2.14}$$

Unlike the other components, the optimization procedure stays the same as in the binary case, meaning the multinomial logistic regression can be optimized using gradient descent.

Figure 2.3 shows an example of classifying two-dimensional input data into three classes using a multinomial logistic regression classifier. As in the binary case, the decision boundary is linear, while the posterior probabilities are visualized using a color gradient.

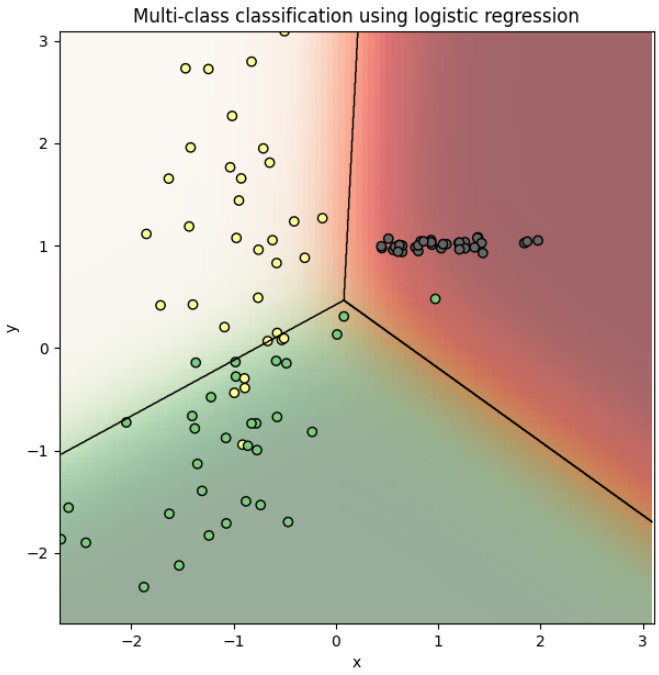

Figure 2.3: Three-way classification of two-dimensional input data using a multinomial logistic regression classifier.

3. Natural language processing

There are many definitions of natural language processing. The Oxford Languages[1] define it as "the application of computation techniques to the analysis and synthesis of natural language and speech", IBM's definition is "a branch of computer science – and more specifically, the branch of artificial intelligence or AI – concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.". Finally, Google Cloud's[2] definition is "Natural language processing (NLP) uses machine learning to reveal the structure and meaning of text. With natural language processing applications, organizations can analyze text and extract information about people, places, and events to better understand social media sentiment and customer conversations".

Applications of natural language processing cover a wide range of tasks. NLP tasks include, but are not limited to:

1. **Part-of-speech (POS) tagging** – Determining the part-of-speech tag for each word in the text. POS tagging can be formalized as a token-level classification task, where the classes depend on the language, e.g., for English some of the available classes are adjectives, adverbs, adpositions, nouns, punctuation, etc.

2. **Named entity recognition (NER)** – Identifying which words are named entities. NER is also a token-level classification task where the classes are universal, e.g., organization, location, person, currency, etc.

3. **Word sense disambiguation (WSD)** – WSD focuses on differentiating the meaning of a word with multiple meanings given some context. An example of a word with more meanings is "bank". "Bank" can refer to land alongside a river or lake, a financial establishment, or any other meaning of the word.

4. **Sentiment analysis** – Categorizing the opinion of the author given a text. More

[1] `https://languages.oup.com/google-dictionary-en/`
[2] `https://cloud.google.com/learn/what-is-natural-language-processing`

formally, sentiment analysis is a sequence classification task, where the classes are usually positive, neutral, or negative sentiment.

5. **Semantic textual similarity** – Represents measuring the similarity of meanings between two texts. Semantic textual similarity is not a sequence classification but rather a regression task with a bounded range. Labels and predictions are usually presented in the interval $[0, 5]$.

3.1. Automated fact-checking

A recently emerging application of natural language processing is automated fact-checking. Automated fact-checking consists of four tasks: claim detection, evidence retrieval, verdict prediction, and justification production, shown in figure 3.1.

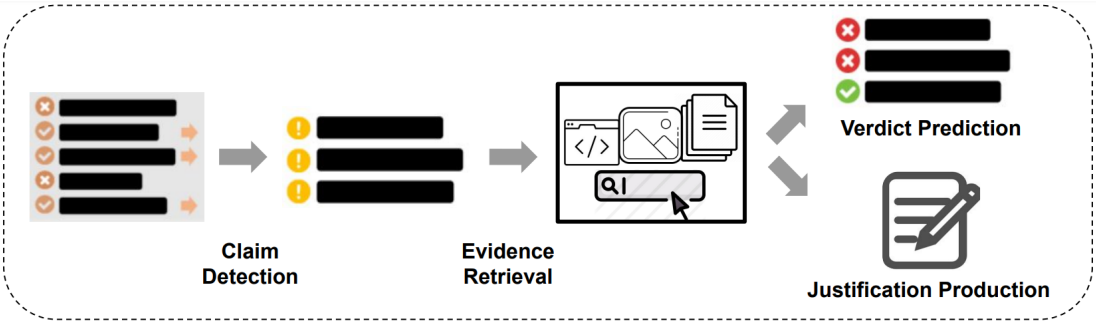

Figure 3.1: An image taken from (Guo et al., 2021) outlining the fact-checking pipeline.

Given a collection of texts that need to be checked, it is important to determine which texts are also claims. This task is called claim detection. A more important version of the claim detection task is claim check-worthiness detection. A check-worthy claim is defined as a claim for which the general public would be interested in knowing the truth (Hassan et al., 2015). It is important to distinguish the difference between claims and check-worthy claims. To illustrate the discrepancy between the two, three examples are provided:

1. "*Breathtaking Photos Capture Loss and Hope in the Age of Climate Change.*"
2. "*Worst yet to come? Experts say, 'Kerala rains match climate change forecasts'*"
3. "*CDC tells travelers to avoid China in expanded travel warning as coronavirus spreads*"

The first example illustrates a statement that is not a claim and not a check-worthy claim. The second statement is a claim but not a check-worthy claim. It is not a check-worthy claim because it does not specify which experts it refers to. Finally, the last statement is a check-worthy claim and a claim, as it targets CDC and China (Cheema et al., 2022). Claim check-worthiness detection is usually approached as a binary classification problem or an importance ranking of claims.

Claim check-worthiness detection is followed by evidence retrieval. Evidence retrieval aims to find information to indicate veracity regarding the claim in question, as defined by Guo et al. (2021). A big problem with this task is that some of the available information is not trustworthy. Usually, an assumption is made that some sources are trustworthy such as encyclopedias, Wikipedia[3], or search engine results. As this step is crucial for verifying the truthfulness of claims, evidence may be curated manually, automatically or using a combined approach, e.g., FullFact[4] uses data tables, legal documents and other primary sources if available, rather than press releases or executive summaries of statistical releases.[5]

Verdict prediction attempts to establish the veracity of a claim using the claim and the retrieved evidence. This problem is usually formulated as a binary classification problem, where the labels are true or false, or using a fine-grained setup. Fine-grained extensions include adding a label denoting the lack of information to predict the truthfulness or employing labels to represent the degree of truthfulness.

Finally, the last task of the fact-checking pipeline is justification production. Justification production is defined as convincing readers of the interpretation of the evidence regarding a claim's verdict prediction. It plays a vital role in persuading the reader of the predicted verdict. An interesting phenomenon called belief perseverance (Anderson, 1989) occurs when asserting a claim is false without providing evidence for it, as it can produce an opposite effect where confidence in the false claim is reinforced.

There exist four strategies for justification production, as stated in Guo et al. (2021). First, attention weights can be used to visualize the importance of tokens. Second, a decision-based model can be chosen for verdict prediction, e.g., choosing a logic-based system. Third, a summarization model can be used to output an explanation of the verdict given the claim and evidence. Finally, the last option is using a self-explanatory decision-making process such as knowledge graphs. It is important to consider that the model needs to correctly depict the reasoning process regardless of

[3] https://www.wikipedia.org/

[4] https://fullfact.org/

[5] https://fullfact.org/about/frequently-asked-questions/#sources

the veracity correctness.

3.2. Text features and preprocessing

Due to the inability of machine learning algorithms to deal with raw text of variable length, fixed-size features are created to represent text. But before creating features, it is crucial to preprocess the text to improve the quality and reliability of the data. By removing noise, inconsistencies, and irrelevancies, machine learning algorithms can more accurately recognize patterns within the data.

For text preprocessing, an NLP pipeline is usually considered. As an example, Spacy's[6] NLP pipeline is shown in figure 3.2.

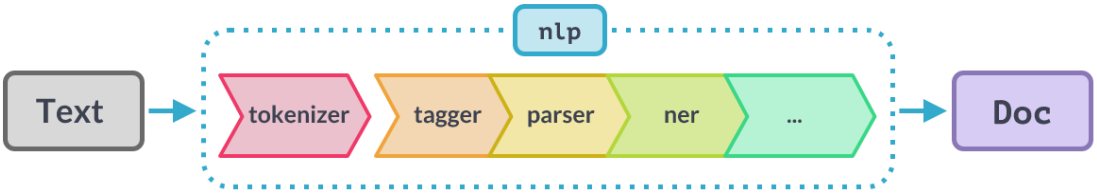

Figure 3.2: Overview of the Spacy NLP pipeline. The image is taken from Spacy's official website.

The first preprocessing step is tokenization. Tokenization splits a piece of text into smaller units called tokens. Tokens can represent words or units smaller than words called subwords. Once tokenization is done, the next step in the pipeline is POS tagging. The goal of POS tagging is to find the grammatical category of a word. Usually, POS tags are not language universal, but some well-known tags which are used in English are nouns (`NN`), verbs (`VB`), adjectives (`JJ`) and pronouns (`PRP`). After tagging, the next component in the pipeline is the dependency parser component. It jointly learns sentence segmentation and labeled dependency parsing. Sentence segmentation is the process of splitting texts into sentences, while labeled dependency parsing analyzes the structure of the sentence. Each word in the sentence is given a tag, while some examples of the tags are nominal subject (`nsubj`), root (`ROOT`), and direct object (`dobj`). Named entity recognition comes after the parser component, and it predicts whether a token is a named entity, such as a person (`PERSON`), a currency (`MONEY`), or an organization (`ORG`). Finally, after NER comes stemming or lemmatization. The goal of both procedures is to reduce words to their base form. Stemming removes

[6] `https://spacy.io/`

suffixes using a heuristic approach and it is a simpler approach than lemmatization. Lemmatization reduces the word to its dictionary form by removing the inflectional ending. A simple example to illustrate the difference is to consider the words "university" and "universe". The considered words will have the same stem, "univers", but different lemmas, "universe" and "university", respectively.

Besides finding base forms of words, further preprocessing steps can be taken. Another often employed step is removing stop words, which are words that do not provide valuable information. Examples of such words are "so", "as", "always", "be" and "no". In addition to removing stop words, another beneficial approach is spelling correction, which is a procedure that identifies misspelled words and corrects them.

Even though some feature extraction procedures are included in an NLP pipeline, such as NER and POS tagging, there are other standard ways to numerically represent text. The simplest option for presenting text to a machine learning model is using a one-hot representation. One-hot representations encode a document by representing each present token with value 1, while non-present tokens are denoted by value 0. An improvement of the sparse solution provided by one-hot representation are distributional vectors such as bag-of-words. The bag-of-words representation consists of tokens and their frequencies in the text which can be represented using a vector, meaning a collection of text documents can be represented using a matrix. An example of a collection of documents can be seen in equation (3.1), where N documents are represented using a vocabulary of size V. A problem that occurs with this setting is that longer texts will more often have higher frequencies for certain tokens, which can be remedied using normalization. An important observation regarding the bag-of-words approach is the order of words is ignored.

$$\mathbf{D} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1V} \\ w_{21} & w_{22} & \dots & w_{2V} \\ \vdots & & \ddots & \vdots \\ w_{N1} & w_{N2} & \dots & w_{NV} \end{bmatrix} \tag{3.1}$$

An improvement of the bag-of-words model can be achieved through the modification of weights, best known as the TF-IDF model. The main idea of the modification is that the relevance of a term is proportional to its frequency in the document, but also inversely proportional to the number of documents it occurs in. The final weight is the product of the term frequency (TF) and the inverse document frequency (IDF). The term frequency weighing used can be seen in equation (3.2), where t_i represents the token i in the vocabulary, d_j represents document j, and f_{t_i,d_j} represents the fre-

quency of token i in document j. The inverse document frequency is shown in equation (3.3), where N represents the number of documents, $DF(t_i, \mathbf{D})$ represents the document frequency of token i in a collection of documents $\mathbf{D}$ and finally, $\mathbf{D}$ represents the vectorized collection of documents.

$$TF(t_i, d_j) = \frac{f_{t_i,d_j}}{\sum_{t_k \in d_j} f_{t_k,d_j}} \tag{3.2}$$

$$IDF(t_i, \mathbf{D}) = \log \frac{1+N}{1+DF(t_i, \mathbf{D})} + 1 \tag{3.3}$$

3.3. Word embeddings

The aforementioned one-hot representations and distributional vectors have two weaknesses: sparsity and encoding structure rather than semantics. Dense representations that encode the meaning of words are known as word embeddings. The first notable improvement regarding embeddings are the *Continuous bag-of-words* method (Mikolov et al., 2013a), and the *Skip-gram* model (Mikolov et al., 2013b). The main idea behind these embedding techniques is connected to the distributional hypothesis (Firth and Palmer, 1968), which states that similar words appear in similar contexts.

3.3.1. The Skip-gram model

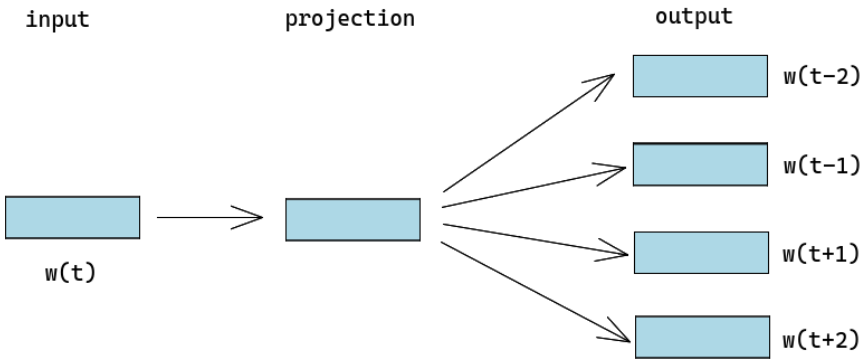

Figure 3.3: The *Skip-gram* model architecture with a context size $c = 2$. The figure was recreated using the original paper (Mikolov et al., 2013b).

The objective of the *Skip-gram* model is to find word representations that work well for predicting the previous and following words. A high level overview of the model architecture can be seen in figure 3.3. More formally, the goal is to maximize the

average log probability shown in (3.4), where T represents the number of words in the text, c represents half of the context size, and w_{t+j} the word at index $t+j$ respectively.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c}\log p(w_{t+j}|w_t) \tag{3.4}$$

The probability in equation (3.4) is calculated using a softmax function, where the denominator is a sum of as many elements as there are words in the vocabulary. Keeping in mind the complexity of calculating the probability exactly, it is possible to approximate it using hierarchical softmax. The authors abandon this approach and combine the *Skip-gram* model with negative sampling. Due to the inclusion of negative sampling the objective changes, as shown in (3.5), where ${v'}_{w_O}$ represents the embedding of the neighboring word w_O, v_{w_I} represents the embedding of the input word w_I, $\mathbb{E}_{w_i\sim P_n(w)}$ represents the expectation of the drawing distribution.

$$\log\sigma({v'}_{w_O}{}^{\intercal}v_{w_I}) + \sum_{i=1}^{k}\mathbb{E}_{w_i\sim P_n(w)}\left[\log\sigma(-{v'}_{w_i}{}^{\intercal}v_{w_I})\right] \tag{3.5}$$

The intuition behind the objective function is that the first component takes care of the positive sample, which is a neighbor of the input word and should be similar to it, suggesting the probability should be high. The second component takes care of the negative samples, which are expected to be words dissimilar to the input word, meaning their probabilities should be small. The reason for the negative sign in the second sigmoid term is that the objective function is being maximized, but the probability of non-neighboring words should be small, indicating that the effect should be inverted.

The architecture of the model is a two-layer feedforward neural network. The weight matrix $\mathbf{W}$ between the input layer and the projection layer is of dimension $\mathbf{V}\times\mathbf{D}$, where $\mathbf{V}$ denotes the vocabulary size, and $\mathbf{D}$ is a hyperparameter corresponding to the size of the embedding. The matrix between the projection layer and the output layer is the matrix $\mathbf{W}'$ of dimension $\mathbf{D}\times\mathbf{V}$, which is used for calculating the surrounding words. The final embeddings correspond to the matrix $\mathbf{W}$.

3.3.2. Continuous bag-of-words

The *Continuous bag-of-words* (CBOW) is the opposite approach compared to the *Skip-gram* model. CBOW uses the context words and predicts the current (middle) word. Given a context of size c, it encodes the words and averages them. The averaged representation is projected using a linear layer, which outputs a probability distribution over the vocabulary.

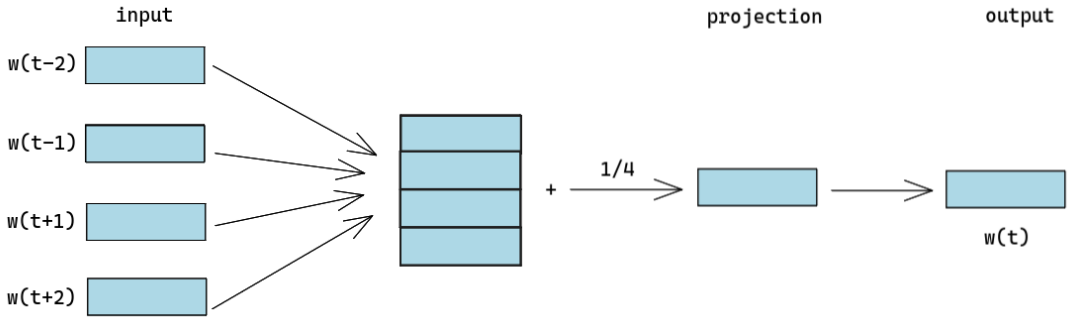

Figure 3.4: The *Continuous bag-of-words* model architecture with a context size $c = 2$. The figure was recreated using the original paper (Mikolov et al., 2013a).

Same as the *Skip-gram* model, CBOW uses a two-layer feedforward neural network, where the weight matrices have the same dimensions, but serve different purposes. The weight matrix $\mathbf{W}$ of dimension $V \times D$ is used for encoding the words to dense representations which are then averaged, and the weight matrix $\mathbf{W}'$ of dimension $D \times V$ is used for projecting the hidden representation to a probability distribution. As in the case of the *Skip-gram* model, the final embeddings correspond to the matrix $\mathbf{W}$.

4. Deep learning

With the advancement of GPUs and hardware, deep learning has gained popularity in the past decade, as depicted in figure 4.1. CPUs were insufficient for meeting the computational demands of deep learning, which involves numerous matrix multiplication operations. GPUs offered significant improvements in parallel processing, which researchers utilized to accelerate deep learning software. Additionally, specialized hardware called tensor processing units (TPUs) was developed for deep learning applications. Alongside hardware advancements, software libraries such as PyTorch[1] and TensorFlow[2] were created to leverage the computational speedups achieved by GPUs.

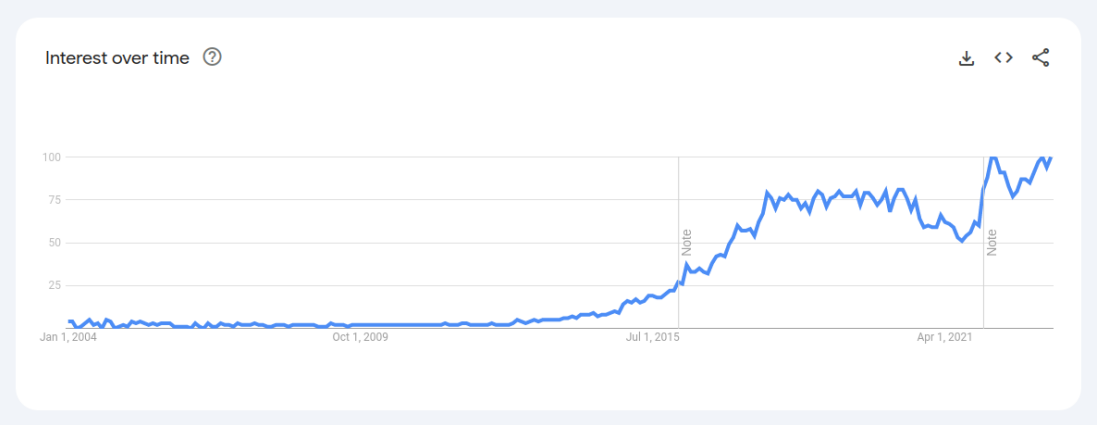

Figure 4.1: The interest over time for the term "deep learning" since January 2004 until 2023, collected by Google.

Since the performance of machine learning algorithms heavily depends on the representation of the data, shifting the focus to obtaining high-quality features was a crucial step for the rise of deep learning. Models require high-level features instead of low-level features, e.g., representing the nose, eyes, and mouth is crucial to detect a face, while using raw pixel values does not provide much information about the presence of a face. Automatically learning representations often results in much better per-

[1] https://pytorch.org/
[2] https://www.tensorflow.org/

formance than wasting time and effort on creating handcrafted features. Deep learning introduces expressing representations in terms of simpler representations, as explained by Goodfellow et al. (2016).

4.1. Feedforward neural networks

The simplest model in deep learning is a multilayer perceptron (MLP) or feedforward neural network. The name feedforward neural network comes from not using feedback connections between the layers. A feedforward neural network consists of a parametrized model $f(\mathbf{x}; \boldsymbol{\theta})$, which maps the input representation $\mathbf{x}$ to some value y using a set of parameters $\boldsymbol{\theta}$. The goal is to approximate the function $y = f^*(\mathbf{x})$. The model can be expressed in the form of a composition of K functions shown in equation (4.1) and (4.2).

$$f(\mathbf{x}; \boldsymbol{\theta}) = f_K(f_{K-1}(\dots f_1(\mathbf{x}; \theta_1)\dots); \theta_{K-1}); \theta_K) \tag{4.1}$$

$$= f_K \circ f_{K-1} \circ \cdots \circ f_1(\mathbf{x}) \tag{4.2}$$

A fully connected layer corresponds to an affine transformation of the input, which is the building block of a feedforward neural network. The fully connected layer can be concisely represented using matrix form, shown in equation (4.3), where $\mathbf{s}$ is a vector of dimension $n \times 1$, $\mathbf{W}$ denotes the parameter matrix of dimension $m \times n$, $\mathbf{x}$ represents the input vector of dimension $m \times 1$, and $\mathbf{b}$ represents the bias vector of dimension $n \times 1$. A more detailed look at the same equation is available in (4.4).

$$\mathbf{s} = \mathbf{W}^\intercal \mathbf{x} + \mathbf{b} \tag{4.3}$$

$$\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} w_{11} & w_{21} & \dots & w_{m1} \\ w_{12} & w_{22} & \dots & w_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1n} & w_{2n} & \dots & w_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{4.4}$$

Taking into account that two sequential affine transformation can be represented as a single affine transformation, it is important add nonlinearity. Adding nonlinear functions helps in modeling complex relationships and patterns in the data which cannot be captured by afffine transformations alone. Usually, the nonlinearities are added in the form of an activation function, such as the rectified linear unit (ReLU), tangent hyperbolic function or the previously mentioned sigmoid function. The general form of an

activation function is shown in equations (4.5) and (4.6), where an activation function $\mathbf{f}$ is applied elementwise to the vector $\mathbf{s}$ resulting in a new vector $\mathbf{h}$.

$$\mathbf{h} = \mathbf{f}(\mathbf{s}) \tag{4.5}$$

$$\begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix} = \begin{bmatrix} f(s_1) \\ f(s_2) \\ \vdots \\ f(s_n) \end{bmatrix} \tag{4.6}$$

The connection between a logistic regression model and a neural network is that the former is a neural network with a single layer and an activation function.

Another simple and beneficial improvement for a neural network is using normalization layers. Normalization layers are especially effective when using activation functions such as the ReLU activation function, because they ensure 50% of the activations are left as is, while 50% of the activations are cancelled. The normalization strategies depending on the dimensions of the considered representation. Some possible options for a normalization layer include batch normalization, layer normalization, instance normalization and group normalization. The equation for batch normalization are shown in (4.7), where γ and β are learnable parameters, and the expectation and variance are calculated over the batch dimension, as proposed by Ioffe and Szegedy (2015).

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}} \cdot \gamma + \beta \tag{4.7}$$

4.1.1. Loss functions

The loss functions used with a neural network depend on the task in question. Most often, the negative log-likelihood in combination with an assumption regarding the posterior distribution of the label $p(y|\mathbf{x};\boldsymbol{\theta})$ is used as a loss function, shown in equation (4.8).

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = -\sum \log p(y|\mathbf{x};\boldsymbol{\theta}) \tag{4.8}$$

The negative log-likelihood can be used for regression and classification tasks. For a regression task, the usual approach uses an assumption of normally distributed residuals with homogeneous variance, which leads to the mean squared error, shown in equation (4.9).

$$L_{MSE}(\mathbf{x}, y, \boldsymbol{\theta}) = (f(\mathbf{x};\boldsymbol{\theta}) - y)^2 \tag{4.9}$$

In the case of classification, the assumed posterior distribution is the categorical distribution, which leads to the cross entropy as in the case of multinomial logistic regression, which was already shown in equation (2.14).

4.1.2. Optimization procedures

The optimization procedure used with a neural network is typically a variant of gradient descent, previously mentioned in chapter 2. The default gradient descent algorithm comes in three variations: batch gradient descent (BGD), mini-batch gradient descent (MBGD), and stochastic gradient descent (SGD). The SGD variant calculates the gradients with respect to the parameters for a single example in the dataset and takes a step in the opposite direction. The memory requirements for SGD are low, and the gradients are inaccurate, but it is robust to the local minima of the error function. The most accurate gradients are computed using batch gradient descent, which calculates the mean of the gradients with respect to the parameters using the whole dataset. The memory requirements for BGD are often difficult to meet, and the gradients are more accurate, but the method is prone to get stuck in local minima. A hybrid method between the two is the mini-batch gradient descent, which uses mini-batches of examples and the average of the gradients with respect to the parameters. In practice, MBGD is the method of choice due to moderate memory requirements, and robustness to local minima, while increasing the accuracy of the gradients.

The adaptive moment estimation (**Adam**; Kingma and Ba, 2014) algorithm offers a more modern approach. The pseudocode of the algorithm taken from (Kingma and Ba, 2014) is shown in 1.

Algorithm 1 Adam optimization algorithm

Require: η: learning rate

Require: ϵ: constant for avoiding zero division

Require: β_1, β_2: exponential decay rates

Require: $f_t(\boldsymbol{\theta})$: objective function

Require: $\boldsymbol{\theta}_0$: the initial parameter value

$\mathbf{m}_0 \leftarrow 0$

$\mathbf{v}_0 \leftarrow 0$

$t \leftarrow 0$

while convergence is not reached **do**

  $t \leftarrow t + 1$

  $\mathbf{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$

  $\mathbf{m}_t \leftarrow \beta_1 \cdot \mathbf{m}_{t-1} + (1 - \beta_1) \cdot \mathbf{g}_t$

  $\mathbf{v}_t \leftarrow \beta_2 \cdot \mathbf{v}_{t-1} + (1 - \beta_2) \cdot \mathbf{g}_t \odot \mathbf{g}_t$

  $\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1 - \beta_1^t)$

  $\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1 - \beta_2^t)$

  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \cdot \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon)$

end while

The main idea of the Adam algorithm is to use previous knowledge about the error function to estimate how smooth it is. If the error function is smooth, the optimization procedure can use a larger step. The algorithm uses a learning rate η that influences the size of the step for updating the parameter values, exponential decay rate β_1 and β_2 that control the weights for using the first and second moment of the gradients, a parametrized function $f_t(\boldsymbol{\theta})$, and the initial parameter values $\boldsymbol{\theta}_0$. First, vectors $\mathbf{m}_0$ and $\mathbf{v}_0$, which represent starting values for the first and second moments of the gradients, are initialized to zeros. Then the algorithm iteratively calculates the gradient of the function $f_t(\boldsymbol{\theta}_{t-1})$, followed by calculating the first moment of the gradient $\mathbf{m}_t$ by including the moment from the previous iteration $\mathbf{m}_{t-1}$ and the current gradient $\mathbf{g}_t$, with both terms being scaled by β_1 and $(1-\beta_1)$, respectively. After calculating the first moment, the second moment of the gradient $\mathbf{v}_t$ is calculated as well, using the previous second moment $\mathbf{v}_{t-1}$ and the current second moment of the gradient $\mathbf{g} \odot \mathbf{g}$, with both terms scaled by β_2 and $(1 - \beta_2)$, respectively. The moments are rescaled, where the values of both scaling coefficients drop exponentially. The final update is done using the product of the learning rate and the ratio of the rescaled first moment $\hat{\mathbf{m}}_t$, and the square root of the rescaled second moment $\hat{\mathbf{v}}_t$.

An improvement of the Adam algorithm is the inclusion of decoupled weight decay regularization, known as AdamW, which was proposed by Loshchilov and Hutter (2017). When using adaptive gradient methods, L_2 regularization is not equal to weight decay, while the two terms are often used interchangeably. The algorithm is equal to the Adam algorithm except for a term shown in equation (4.10), added before calculating the first moment of the gradients, where λ represents the weight decay hyperparameter.

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} \cdot (1 - \eta\lambda) \tag{4.10}$$

4.1.3. Schedulers

Another often-used technique for improving model performance is using learning rate (LR) schedulers. LR schedulers dynamically adjust the learning rate during training, allowing the model to converge faster and reach higher metric values. The most often used schedulers are the linear scheduler and the cosine scheduler, but there exist more alternatives. The only difference they introduce in the optimization procedure is scaling the learning rate η by some factor. LR schedulers often offer a warmup period where the learning rate value gradually grows linearly. An example of a linear and cosine scheduler with a warmup ratio of 0.1 used over 1000 training steps with a learning rate of 1 can be seen in figure 4.2.

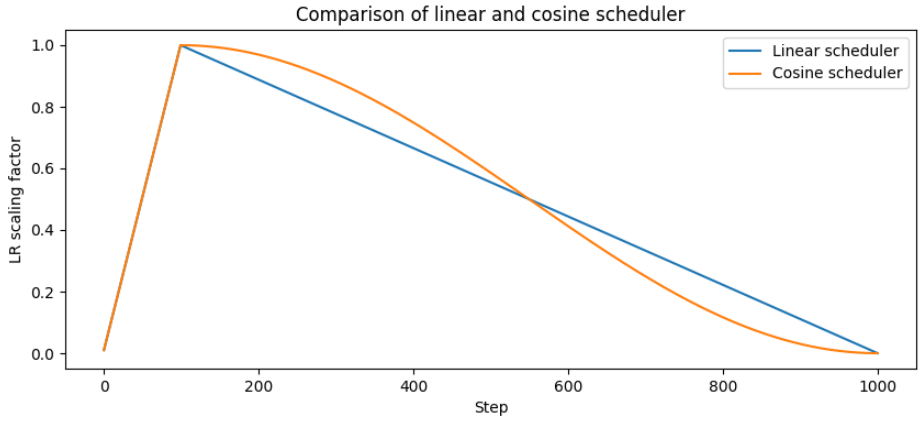

Figure 4.2: A comparison of linear and cosine learning rate schedulers over 1000 training steps with a warmup ratio of 0.1 and a learning rate of 1.

4.1.4. The backpropagation algorithm

An often overlooked but necessary component of deep neural networks is calculating the gradients with respect to the parameters. The algorithm responsible for this is the backpropagation algorithm, which is a fundamental concept in deep learning. First, the outputs of the model are computed in the forward pass, and the loss is calculated. Then, the derivative of the loss function with respect to the logits is computed. If the considered loss function L is the cross-entropy loss, $\mathbf{s}$ represents the logits, and $\mathbf{y}_{\text{OH}}$ represents the one-hot encoded labels, then the partial derivative of the loss function with respect to the logits can be calculated using equation (4.11).

$$\frac{\partial L}{\partial \mathbf{s}} = \text{softmax}(\mathbf{s})^\intercal - \mathbf{y}_{\text{OH}}{}^\intercal \tag{4.11}$$

Next, the gradients need to be backpropagated to the parameters. This can be achieved by using the chain rule. If the layer in question is fully connected, the activations of the previous layer are $\mathbf{h}_K$, the parameters of the layer are $\mathbf{W}$ and the biases are $\mathbf{b}$, then the derivatives of the parameters with respect to $\mathbf{W}$ and $\mathbf{b}$ are shown in equations, (4.12) and (4.13), respectively.

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{s}} \cdot \frac{\partial \mathbf{s}}{\partial \mathbf{W}} = \left[\frac{\partial L}{\partial \mathbf{s}}\right]^\intercal \cdot \mathbf{h}_K^\intercal \tag{4.12}$$

$$\frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{s}} \cdot \frac{\partial \mathbf{s}}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{s}} \cdot \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} = \frac{\partial L}{\partial \mathbf{s}} \tag{4.13}$$

The backpropagation algorithm uses the gradients of the loss with respect to the logits to backpropagate the errors deeper into the network, chaining them with the gradient of the logits with respect to the outputs of the previous layer. The general formula for calculating the gradients with respect to the parameters of some layer k can be represented using the equation (4.14).

$$\frac{\partial L}{\partial \boldsymbol{\theta}_k} = \frac{\partial L}{\partial \mathbf{h}_K} \cdot \frac{\partial \mathbf{h}_K}{\partial \mathbf{h}_{K-1}} \cdot \ldots \cdot \frac{\partial \mathbf{h}_{k+2}}{\partial \mathbf{h}_{k+1}} \cdot \frac{\partial \mathbf{h}_{k+1}}{\partial \boldsymbol{\theta}_k} \tag{4.14}$$

In most software libraries operations on tensors are recorded using directed acyclic graphs called computational graphs, while backpropagation is achieved through automatic differentiation. Automatic differentiation is a technique for calculating the gradients of the outputs with respect to the inputs or parameters in an automated fashion. There are two different modes of the technique: forward mode and reverse mode.

The forward mode calculates the gradients along with the result of the function. The other option is the reverse mode which calculates the result of the function first, while the gradients are calculated starting from the outputs. Pytorch, one of the best-known deep learning frameworks, uses reverse mode automatic differentiation. In the forward pass, the library runs the requested operation on a tensor and records the operation gradient function in the computational graph. Once the backward pass is initiated, the recorded gradient functions are executed starting from the root tensors (output tensors) and propagating the gradients towards the leaf tensors (input tensors) using the chain rule.

4.2. Recurrent neural networks

Feedforward neural networks are not designed for processing sequential data. Apart from using FFNNs, other architectural choices include convolutional neural networks (CNNs) and recurrent neural networks (RNNs). CNNs are often used in computer vision, because of their capability to model local interactions using the convolution operation. In contrast, RNNs specialize in sequence modeling, making them a perfect match for processing textual data.

The recurrent neural network is a very different framework than FFNNs. It introduces two ways to store data in a network: hidden states and inputs. To calculate the current hidden state of the recurrent neural network equation (4.15) is used, where $\mathbf{h}^{(t)}$ represents the current hidden state, $\mathbf{h}^{(t-1)}$ represents the previous hidden state and $\mathbf{x}^{(t)}$ represents the current input data. In the context of NLP, the current input data is usually a static embedding of a token, meaning each embedding gets processed sequentially. The idea behind using hidden states for a task is to include a memory component for remembering information relevant to the task. This way information contained in the static token embeddings is extracted and filtered for understanding the whole text sequence.

$$\mathbf{h}^{(t)} = f_h(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \mathbf{W}) \tag{4.15}$$

Besides transforming the hidden states, the original inputs at each time step are transformed using a parameter matrix $\mathbf{U}$, while the parameter matrix $\mathbf{W}$ is responsible for transforming the hidden states at each time step. Finally, RNNs calculate the output vector $\mathbf{o}^{(t)}$ at each time step using another parameter matrix $\mathbf{V}$ and the current hidden state $\mathbf{h}^{(t)}$, also shared between different time steps.

Since this kind of RNN uses only the left context, bidirectional RNNs were intro-

duced. The bidirectional aspect of an RNN introduces hidden states aware of the right context, shown in equation (4.16), where $\mathbf{g}^{(t+1)}$ represents the next hidden state. The difference can also be seen in equation (4.17), where the outputs are calculated using the hidden states that contain knowledge from the left context $\mathbf{h}^{(t)}$ and hidden states that contain knowledge from the right context $\mathbf{g}^{(t)}$.

$$\mathbf{g}^{(t)} = f_g(\mathbf{g}^{(t+1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta}_g) \tag{4.16}$$

$$\mathbf{o}^{(t)} = f_o(\mathbf{g}^{(t)}, \mathbf{h}^{(t)}; \boldsymbol{\theta}_o) \tag{4.17}$$

Motivated by the problems of vanishing gradients, exploding gradients, and learning long-term dependencies, modifications of the RNN were introduced. The first improvement was the long short-term memory (LSTM) model (Hochreiter and Schmidhuber, 1997), and the second improvement was the gated recurrent unit (GRU) (Chung et al., 2014).

4.2.1. Long short-term memory cell

The long short-term memory cell introduces more parameters to better separate their tasks. The cell uses an input gate, a forget gate, and an output gate.

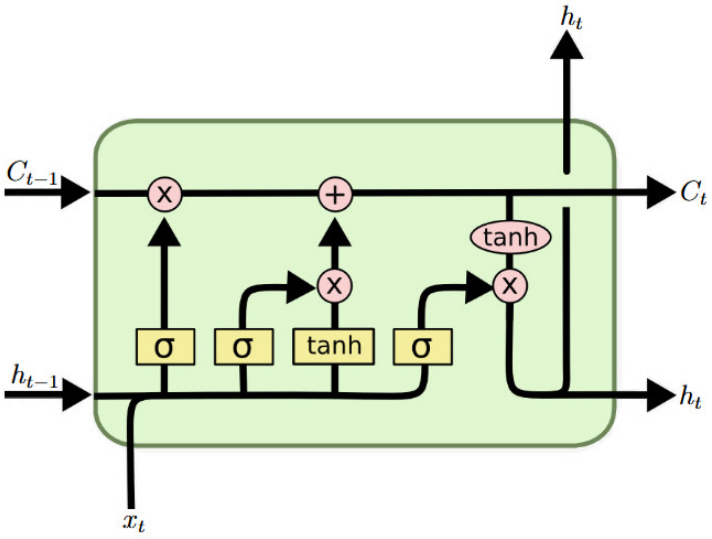

Figure 4.3: A diagram of the LSTM cell. The symbols C_{t-1} and C_t represent the previous and current cell states respectively, h_{t-1} and h_t represent the previous and current hidden states respectively, and x_t represents the current input. The original figure taken from a blog about LSTMs and GRUs (Olah, 2015) was altered.

The diagram of the LSTM cell is shown in figure 4.3. An LSTM cell depends on the previous cell state $\mathbf{c}_{t-1}$, the previous hidden state $\mathbf{h}_{t-1}$, and the current input data $\mathbf{x}_t$. First, the forget gate values $\mathbf{f}^{(t)}$ are computed, shown in equation (4.18), where the previously mentioned symbols have the same meaning, while $\mathbf{U}_f$, $\mathbf{W}_f$, and $\mathbf{b}_f$ represent the input weights, recurrent weights, and biases, respectively. The purpose of the forget gate is to filter out the information important for the task. The LSTM cell also uses an input gate $\mathbf{g}^{(t)}$, which utilizes its own set of input weights and recurrent weights, denoted by $\mathbf{U}_g$ and $\mathbf{W}_g$ (4.19), respectively. The input gate weights the influence of input data and its relevance for the task. Then, the internal state of the cell combines the information from the previous cell state and the newly acquired information. It can be computed using equation (4.20), where $\mathbf{U}_g$ and $\mathbf{W}_g$ represent the input weights and recurrent weights of the cell state, respectively. The output gate $\mathbf{q}^{(t)}$ influences the final hidden state $\mathbf{h}^{(t)}$, by using its own input weights $\mathbf{U}_o$ and recurrent weights $\mathbf{W}_o$, shown in equation (4.21). The final hidden state is represented using element-wise multiplication of the activated cell state and the output gate (equation 4.22).

$$\mathbf{f}^{(t)} = \sigma\left(\mathbf{U}_f{}^\intercal \mathbf{x}^{(t)} + \mathbf{W}_f{}^\intercal \mathbf{h}^{(t-1)} + \mathbf{b}_f\right) \tag{4.18}$$

$$\mathbf{g}^{(t)} = \sigma\left(\mathbf{U}_g{}^\intercal \mathbf{x}^{(t)} + \mathbf{W}_g{}^\intercal \mathbf{h}^{(t-1)} + \mathbf{b}_g\right) \tag{4.19}$$

$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + \mathbf{g}^{(t)} \odot \sigma\left(\mathbf{U}^\intercal \mathbf{x}^{(t)} + \mathbf{W}^\intercal \mathbf{h}^{(t-1)} \mathbf{b}\right) \tag{4.20}$$

$$\mathbf{q}^{(t)} = \sigma(\mathbf{U}_o{}^\intercal \mathbf{x}^{(t)} + \mathbf{W}_o \mathbf{h}^{(t-1)} + \mathbf{b}_o) \tag{4.21}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{c}^{(t)}) \odot \mathbf{q}^{(t)} \tag{4.22}$$

4.2.2. Gated recurrent unit

The gated recurrent unit aims to strike a balance between the parameter count and the cell's capacity, considering that the LSTM cell employs four times as many parameters as an RNN cell. Instead of using two different units for forgetting and updating the state, GRU uses only one.

An overview of the GRU is shown in figure 4.4. It introduces two gates: the update gate and the reset gate. The definition of the update gate can be seen in equation (4.23), where the newly introduced parameters $\mathbf{U}_u$, and $\mathbf{W}_u$ act as input weights, and recurrent weights, respectively. The reset gate is defined in equation (4.24), with its own set of parameters for the input weights and recurrent weights. The final hidden state is obtained by combining the previous hidden state and a mixture of information

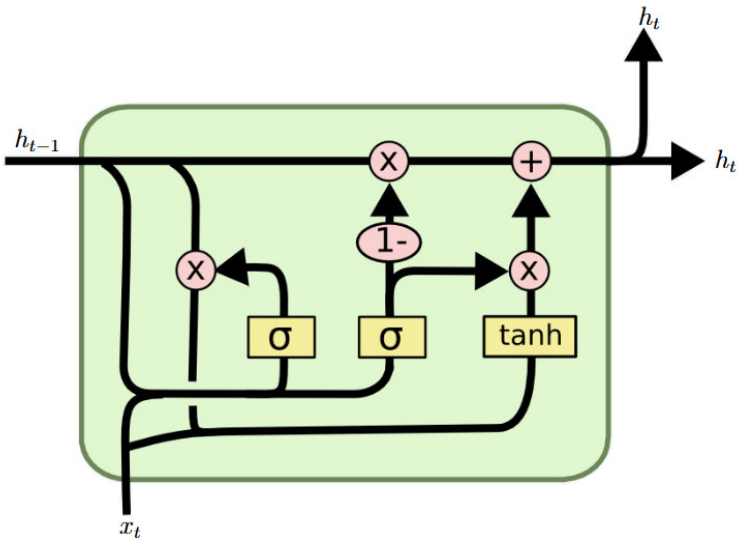

Figure 4.4: A diagram of the gated recurrent unit. The symbols h_{t-1} and h_t represent the previous and current hidden states respectively, and x_t represents the current input. The original figure taken from a blog about LSTMs and GRUs (Olah, 2015) was altered.

from the input, the last hidden state, and the reset gate, shown in equation (4.25). The hidden state calculation is also done using a new set of parameters $\mathbf{U}$ and $\mathbf{W}$, representing the input weights and recurrent weights, respectively. This means the GRU cell has three times more parameters than the RNN cell.

$$\mathbf{u}^{(t)} = \sigma\left(\mathbf{U}_u{}^\intercal \mathbf{x}^{(t)} + \mathbf{W}_u{}^\intercal \mathbf{h}^{(t-1)} + \mathbf{b}_u\right) \tag{4.23}$$

$$\mathbf{r}^{(t)} = \sigma\left(\mathbf{U}_r{}^\intercal \mathbf{x}^{(t)} + \mathbf{W}_r{}^\intercal \mathbf{h}^{(t-1)} + \mathbf{b}_r\right) \tag{4.24}$$

$$\mathbf{h}^{(t)} = \mathbf{u}^{(t-1)} \odot \mathbf{h}^{(t-1)} + (1 - \mathbf{u}^{(t-1)}) \odot \sigma\left(\mathbf{U}^\intercal \mathbf{x}^{(t)} + \mathbf{W}^\intercal(\mathbf{r}^{(t-1)} \odot \mathbf{h}^{(t-1)}) + \mathbf{b}\right) \tag{4.25}$$

4.3. The transformer architecture

The next important improvement in sequence modeling is the transformer architecture. Recurrent neural networks sequentially process text, while transformers process the whole text sequence at once. Instead of using hidden states as RNNs do, the transformer architecture relies on self-attention mechanisms to capture dependencies between tokens in a sequence, enabling it to effectively represent long-range dependencies. Transformers have proven to be highly effective in various fields like computer vision and natural language processing, achieving state-of-the-art results.

The staple mechanism in the transformer architecture is the self-attention mech-

anism (Vaswani et al., 2017). There are different self-attention mechanisms, but the version introduced in the previously mentioned paper is the scaled dot-product attention, shown in equation (4.26) and figure 4.5. It uses the query matrix $\mathbf{Q}$, a key matrix $\mathbf{K}$, both of dimension $d_k \times d_{model}$, and a value matrix $\mathbf{V}$ of dimension $d_v \times d_{model}$. First, the product of the query vectors in matrix $\mathbf{Q}$ and the key vectors in $\mathbf{K}$ is computed, resulting in a matrix of dimension $d_k \times d_k$. Since the results are not normalized, the dot products can be of large magnitude, meaning scaling is required. The authors illustrate this by assuming all entries in matrices $\mathbf{Q}$ and $\mathbf{K}$ are normally distributed, with mean 0 and variance 1, meaning the dot product of rows in matrices $\mathbf{Q}$ and $\mathbf{K}$ will have variance d_k. Because of the variance, the scores are normalized by a scaling factor of $\frac{1}{\sqrt{d_k}}$. The normalized scores can be interpreted as how much a token attends to other tokens. Then, the normalized scores are transformed using the softmax function, meaning large values become larger and small values become smaller. Finally, the output of dimension $d_k \times d_{model}$ is obtained by multiplying the transformed normalized scores by the value matrix $\mathbf{V}$.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d_k}}\right)\mathbf{V} \tag{4.26}$$

To further extend the self-attention principle, the authors propose combining it with a multi-headed mechanism. The multi-head attention mechanism splits the query, key, and value matrices into d_{model}/h submatrices, depending on the number of heads h, all of which are projected using another linear transformation. The split matrices are then again projected and transformed using scaled dot-product attention. The final output consists of concatenated output matrices multiplied by another matrix $\mathbf{W}^O$. The multi-head attention mechanism is shown in equations (4.27), (4.28), and figure 4.6.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_n)\mathbf{W}^O \tag{4.27}$$

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{4.28}$$

The transformer consists of an encoder-decoder architecture. The encoder maps an input sequence of symbols into a sequence of continuous representations. The decoder then uses the encoder's representations to generate an output sequence in an autoregressive fashion, meaning it uses the encoder's outputs and the previously generated symbols. An overview of the transformer architecture is shown in figure 4.7.

First, the input tokens are mapped into input embeddings in the encoder, which are vectors of dimension d_{model}. After embedding the tokens, information about the

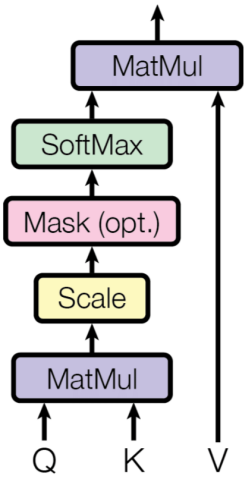

Figure 4.5: A diagram of the scaled dot-product attention mechanism, used in the transformers architecture. The figure is taken from (Vaswani et al., 2017).

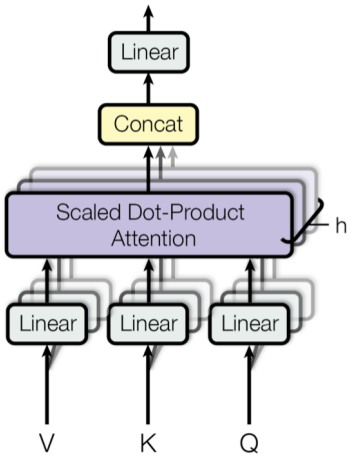

Figure 4.6: A diagram of the multi-head attention mechanism, used in the transformers architecture. The figure is taken from (Vaswani et al., 2017).

.

positions of each token is added using positional encodings. The positional encodings are based on sine and cosine functions shown in equations (4.29) and (4.30), where pos represents the position of a token in the sequence, while i represents the component of the embedding.

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right) \tag{4.29}$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right) \tag{4.30}$$

Then, multi-head scaled dot-product attention is calculated over the embeddings with added positional encodings. A residual connection is added between the embeddings with added positional encodings and the output of the multi-head attention. The two are added together and normalized using layer normalization. The outputs then pass through a two-layer feedforward neural network with a ReLU activation function. The inputs of the feedforward neural network are also connected using a residual connection and normalized using layer normalization.

The decoder is similar to the encoder with some minor differences. The masked multi-head attention takes care of masking the tokens that come after the current token. These tokens cannot be available to the model, as it generates the output in an autoregressive fashion. The second multi-head attention module differs from the usual

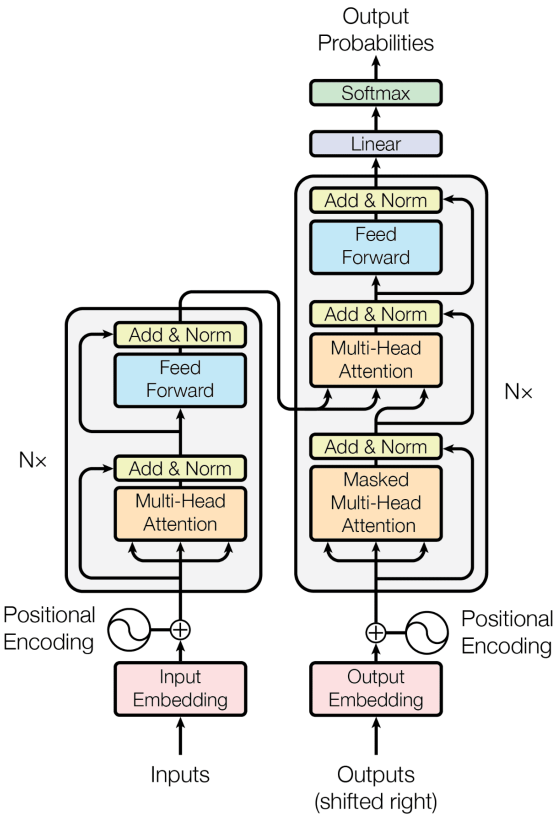

Figure 4.7: The transformer consisting of an encoder-decoder architecture. The figure is taken from Vaswani et al. (2017).

because the matrices $\mathbf{Q}$ and $\mathbf{K}$ are the encoder's outputs while the matrix $\mathbf{V}$ is the output of the masked multi-head attention module. A residual connection with layer normalization is also added.

The encoder and decoder are repeated $N = 6$ times. The final decoder output is fed into a fully connected layer and a softmax activation function which calculates probabilities over the vocabulary. For regularization purposes, the authors include dropout before the layer normalization and experiment with label smoothing.

5. Pretraining techniques

Feedforward neural networks, recurrent neural networks, and transformer-based architectures paved the way for using deep learning in natural language processing. Feedforward neural networks could not capture dependencies in a sequence. This limitation led to the development of recurrent neural networks, which partially addressed the problem but had issues with vanishing and exploding gradients, as well as modeling long sequences. By introducing the self-attention mechanism, the transformer presented a promising alternative for applying deep learning to textual data. The next notable advancement in NLP was focused on improving the representations of the text that the model uses. This progress had been made possible by using self-supervised learning techniques such as pretraining. Self-supervised learning is the same as supervised learning, but the labels for the training data are generated automatically. Pretraining refers to training a language model on a large corpus of unlabelled text data to learn general language understanding. This enabled models to grasp the subtleties of language and achieve state-of-the-art performance on a wide range of tasks. By leveraging large-scale pretraining and fine-tuning methodologies, NLP models have significantly advanced their ability to understand and generate text.

5.1. BERT

Previous methods such as ELMo (Peters et al., 2018), and the Generative pre-trained transformer (GPT) (Radford and Narasimhan, 2018), included using pre-trained language representations on downstream tasks. A weakness of both approaches is the unidirectionality of the pretraining procedure, e.g., GPT uses a left-to-right architecture, meaning attention is calculated only regarding the previous tokens. The “Bidirectional Encoder Representation from Transformers” (BERT) pretraining technique addresses this limitation by using two novel techniques: masked language modeling (MLM) and next sentence prediction (NSP).

Figure 5.1 represents an overview of pretraining and fine-tuning procedures for

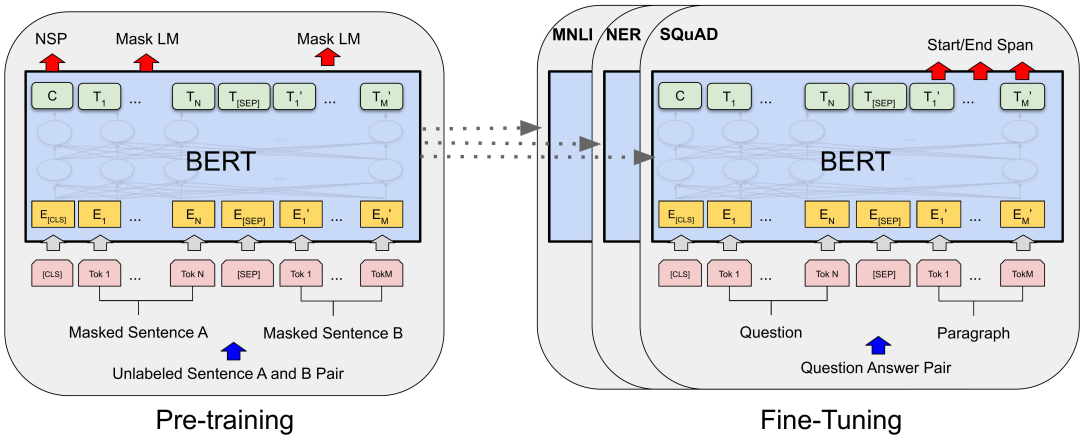

Figure 5.1: The pretraining and fine-tuning setup of the BERT model. The figure is taken from (Devlin et al., 2018).

BERT. The weights of the pretrained model are used for initializing the model for a downstream task. The vocabulary has a size of 30000 tokens. At the beginning of the sequence, the `[CLS]` token is added, whose last hidden state is used as a representation for the whole sequence. The separator token `[SEP]` is used for separating different parts of the sequence, e.g., when fine-tuning on a natural language inference dataset, the separator distinguishes between the hypothesis and the premise. Adding the separator token between two sentences changes the segment embeddings, shown in figure 5.2. The maximum length of the text sequence is 512 tokens.

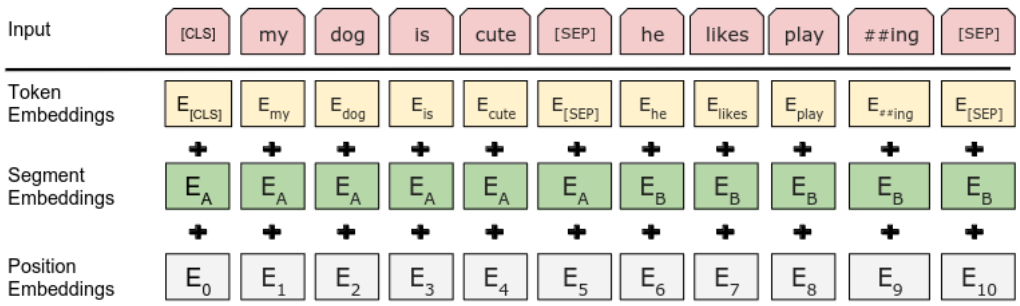

Figure 5.2: The embedding layer in the BERT model. The input representation is the sum of the token, segment, and position embeddings. The figure is taken from (Devlin et al., 2018).

5.1.1. Masked language modeling

Masked language modeling is a pretraining task that uses bidirectional context. The task consists of tokenizing the input text, which is then used for masking. First, 15% of

the tokens are chosen as candidates for masking. Out of that 15%, 80% of the tokens are replaced by the mask token, 10% of the tokens are replaced by a random word from the vocabulary, and 10% of the tokens are left as is. The model's goal is to find out which words were in place of the candidates for masking.

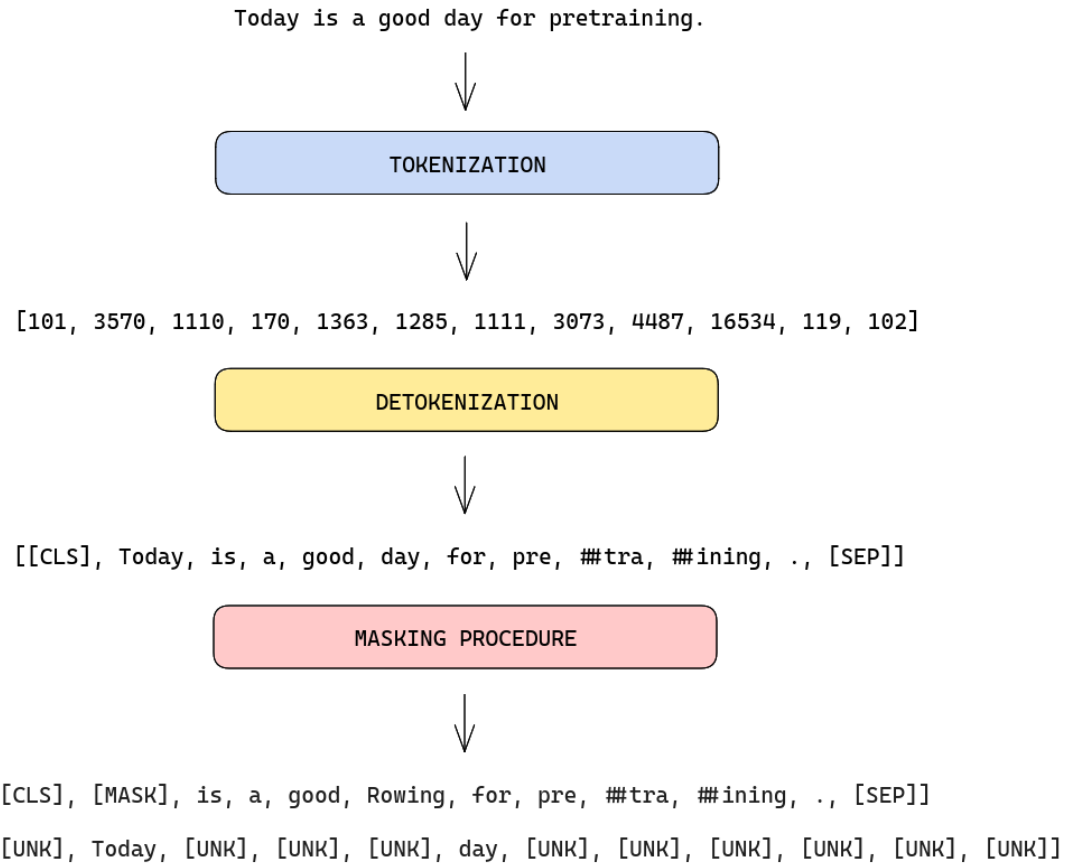

Figure 5.3: An example of sentence tokenization, followed by detokenization and masking. Detokenization serves for illustration purposes, while in the implementation all manipulation is done using the token identificators.

To illustrate the whole process, an example is provided in figure 5.3. The first sentence is an example sentence. This sentence is tokenized, meaning the words are converted to identificators of the corresponding tokens in the vocabulary. For longer words, this usually means they get split into subword tokens, such as the word "pre-training". The detokenization process shows how words from the given input got split into tokens. On an implementation level, the identificators are manipulated rather than detokenized tokens. Finally, the identificators go into the masking procedure, masking the word "Today", and changing it into the `[MASK]` token. The final modification is the replacement of the token "day" with the token "Rowing", a random token from the vocabulary. The last row represents the detokenized labels, where the only two

available labels are the original token that was masked, and the original token that was randomized.

More formally, masked language modeling is a classification task over the whole vocabulary. Optimization is done using a cross-entropy loss function, where the loss is calculated exclusively over the candidates for masking.

5.1.2. Next sentence prediction

Another pretraining task BERT introduces is the next sentence prediction. The task uses sentence A and sentence B, both taken from the corpus. In 50% of cases sentence B follows sentence A in the corpus, while in the other 50% of cases, sentence B is some other random sentence from the corpus. The model uses the same inputs as in the masked language modeling task, meaning the two objectives can be learned in parallel. More formally, the next sentence prediction task is a binary classification task. The output representation of the `[CLS]` token is used in combination with a fully connected layer with a single output logit. The loss used for the optimization is the binary cross-entropy loss.

5.2. RoBERTa

The BERT pretraining technique illustrated how important representation learning is. A replication study of BERT found that it was significantly undertrained, meaning longer pretraining further improves the downstream task results. RoBERTa (Liu et al., 2019) included simple modifications of the BERT pretraining procedure and achieved state-of-the-art results on some tasks. The changes RoBERTa introduces are: training the model longer and on more data, using a larger batch size, removing the next sentence prediction objective, and dynamically changing the masking pattern.

In the BERT implementation, the masking procedure is done only at the beginning and saved for later, meaning the masked tokens are the same in each epoch of pretraining. RoBERTa improves on this by changing the masked tokens randomly in each epoch. Besides using the same corpora BERT uses, RoBERTa is pretrained on CC-News, collected by the authors of RoBERTa, OpenWebText (Gokaslan and Cohen, 2019), and Stories (Trinh and Le, 2018). Apart from training on more datasets, they found improvements using larger batch sizes such as 2000, achieved through gradient accumulation. The authors of the RoBERTa paper also noticed that the next sentence prediction objective hurts the performance on downstream tasks, and they decided not

to use it. Finally, the last design change replaces the Wordpiece tokenizer with Byte-pair encoding (BPE), which is a hybrid between character and word-level representations with a larger vocabulary size of 50 000 tokens. Except for the change in the size of the vocabulary, any word can be encoded without using an "unknown" token.

5.3. ELECTRA

As masked language modeling techniques have highly demanding computational requirements, more efficient pretraining methods were introduced. One of these methods is the ELECTRA pretraining, which stands for "Efficiently Learning an Encoder that Classifies Token Replacement Accurately" (Clark et al., 2020). ELECTRA achieves state-of-the-art results, indicating that compute-efficient discriminative pretraining methods are essential for pushing the boundaries of natural language understanding.

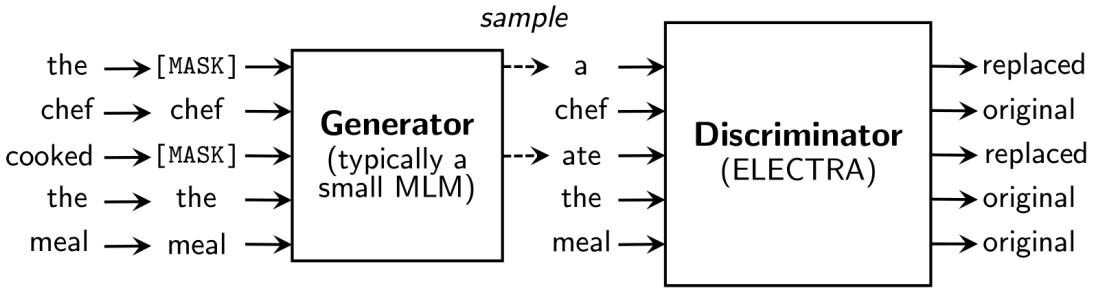

Figure 5.4: An overview of the ELECTRA pretraining procedure. The figure is taken from Clark et al. (2020).

Figure 5.4 displays an overview of the ELECTRA pretraining method. The ELECTRA method includes two models: a generator G, and a discriminator D. First, the input text is tokenized. After tokenization comes the masking procedure. 15% of all tokens are randomly chosen as masking candidates, where 85% of these tokens are replaced with mask tokens, and the remaining 15% are left as is. The tokenized masked inputs are fed into the generator G that outputs probabilities p_G for each token, representing the most suitable token for replacement. Then, a sampling step is done to determine the exact tokens which will replace the masked tokens. The modified text sequence goes through the discriminator D which outputs probabilities p_D, which correspond to a token being replaced. This means the discriminator tries to distinguish if the token is generated by the generator model G or if it is the original token.

More formally, the generator solves a multi-class classification task over the whole vocabulary, while the discriminator is doing a binary classification task. The models

are trained jointly using a combined loss function shown in equation (5.1), where $\mathcal{X}$ represents a corpus of text.

$$\mathcal{L}_{\text{Combined}} = \sum_{x \in \mathcal{X}} \mathcal{L}_{\text{MLM}}(\mathbf{x}, \boldsymbol{\theta}_G) + \lambda \mathcal{L}_{\text{Disc}}(\mathbf{x}, \boldsymbol{\theta}_D) \tag{5.1}$$

The combined loss function consists of two parts. The first part is the MLM loss, which is the expectation of the negative log-likelihood of the posterior distribution p_G of a token $x^{(i)}$, given the masked sequence of tokens $\mathbf{x}_{\text{masked}}$ and summed over all tokens $\mathbf{m}$. Equation (5.2) represents the MLM loss.

$$\mathcal{L}_{\text{MLM}} = \mathbb{E}\left(\sum_{i \in \mathbf{m}} - \log p_G(x^{(i)} | \mathbf{x}_{\text{masked}}) \right) \tag{5.2}$$

The second component of the combined loss is the discriminator loss (5.3), scaled by a factor of $\lambda = 50$. The loss function is equivalent to the expectation of the binary cross-entropy loss, where n is the number of tokens in the sequence and $\mathbf{x}_{\text{corrupt}}$ represents the input tokens with some tokens replaced by tokens sampled using the generator G.

$$\begin{aligned} \mathcal{L}_{\text{Disc}} = \mathbb{E}\Bigg(\sum_{i=1}^{n} & - \mathbf{1}\left\{ x^{(i)}_{\text{corrupt}} = x^{(i)} \right\} \log D(\mathbf{x}_{\text{corrupt}}, i) \\ & - \mathbf{1}\left\{ x^{(i)}_{\text{corrupt}} \neq x^{(i)} \right\} \log(1 - D(\mathbf{x}_{\text{corrupt}}, i)) \Bigg) \end{aligned} \tag{5.3}$$

The ELECTRA pretraining approach is similar to a generative adversarial network (GAN), but the two are not the same. The first difference is the situation where the generator guesses the correct token. The correctly guessed token is regarded as real, instead of fake. The second difference is the generator is trained using a maximum likelihood objective, meaning the generator is trying to replicate the text, rather than fooling the discriminator in an adversarial fashion. Finally, the GAN uses a noise vector as an input for generating new examples, while in ELECTRA, new tokens are generated based on the context.

5.4. Other pretraining techniques

There are many other pretraining techniques, besides masked language modeling, next sentence prediction, and replaced token detection. Inspired by the success of BERT, modified masked language modeling techniques emerged.

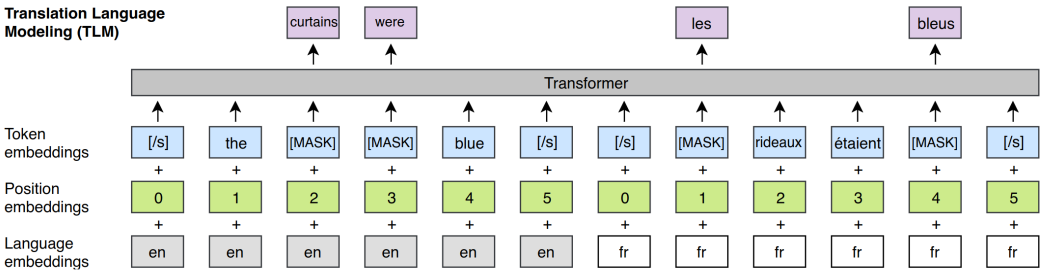

Figure 5.5: An overview of the translation language modeling technique used in XLM. The figure is taken from (Lample and Conneau, 2019).

Cross-lingual language modeling (XLM) enhances masked language modeling for multilingual models using the translation language modeling (TLM) technique shown in figure 5.5, where the MLM task is solved for multiple languages at once. The method requires N corpora of sentences with the same meaning but in different languages. Another difference is the addition of language embeddings in TLM, which is not needed in monolingual MLM tasks.

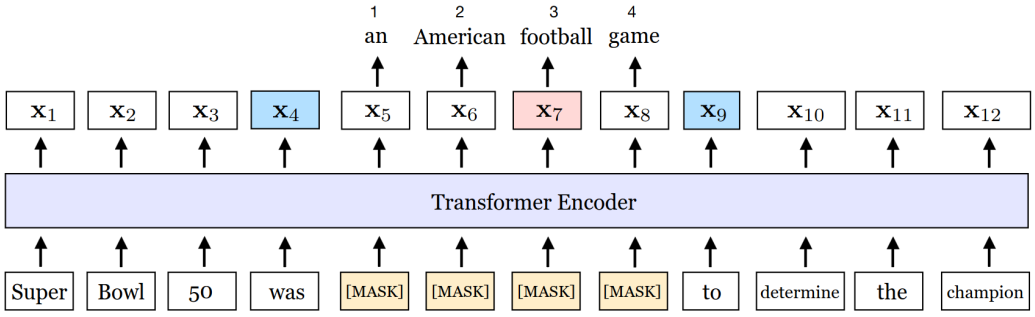

Figure 5.6: An overview of the span boundary objective used in SpanBERT. The figure is taken from (Joshi et al., 2019).

Joshi et al. (2019) introduced the SpanBERT model, which uses two pretraining objectives: masked language modeling and span boundary objective (SBO), both shown in figure 5.6. The difference in regards to the classic MLM technique is that the authors mask a continuous span of tokens. Apart from using the MLM objective, they introduce a span boundary objective, where a linear classifier is used in combination with the embedding of the last token before the span, the embedding of the first token after the span, and with the positional embedding of the token in question to predict the actual token.

The authors of StructBERT (Wang et al., 2019) also employ a similar span strategy.

They use the word structural objective, which includes masked language modeling and shuffling a span of non-masked tokens, but without shuffling the positional encodings. Apart from the modified MLM procedure, they make use of the sentence structural objective. The objective consists of sampling sentences with the same probability in three different ways: randomly, from the same document in the correct order, and the same document in reverse order, meaning the model has to distinguish the natural order of sentences.

Recently, even simpler alternatives for masked language modeling were created by Yamaguchi et al. (2021). They include five approaches: shuffled token detection, random token detection, shuffled token detection combined with random token detection, masked token type classification, and first character detection. An overview of the five mentioned tasks can be seen in 5.7.

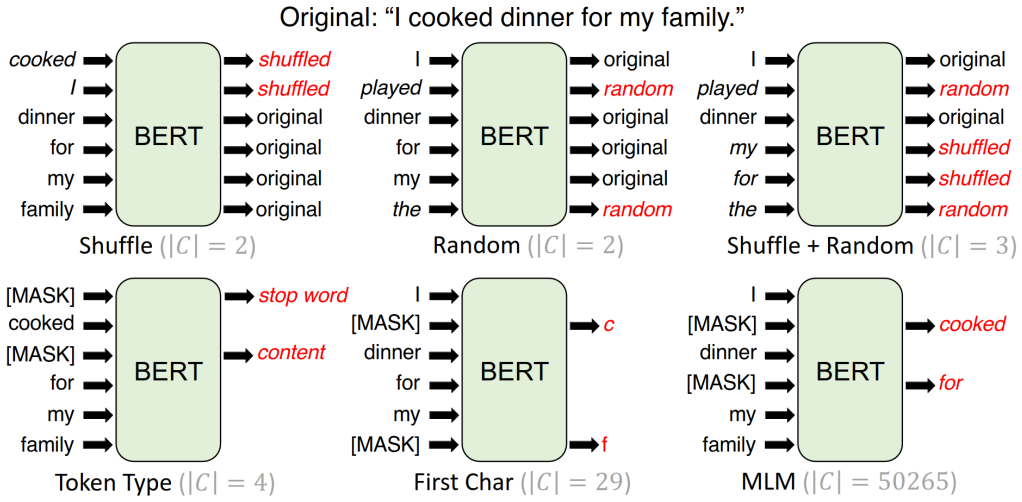

Figure 5.7: An overview of five pretraining methods introduced by Yamaguchi et al. (2021) and masked language modeling. The number of classes considered for a specific pretraining task is denoted below each scheme. The figure is taken from the same paper.

Out of the five pretraining tasks, the shuffle random task surpasses MLM on some downstream tasks, meaning it might be a good choice for fact-checking. The task is a three-way classification, where each token is classified as a random, original, or shuffled token. This way, the model learns to distinguish the natural positions of the tokens belonging to the same sentence, as well as distinguishing what the tokens belong to the same sentence. For each sample 10% of tokens are replaced with shuffled ones from the same sequence and another 10% of tokens are replaced with random ones from the vocabulary. The loss used in combination with the task is the cross-entropy loss averaged over all input tokens.

6. Experiments

The experiments cover two tasks: claim check-worthiness detection and fact-verification. Usually, both tasks are binary classification problems, but this varies depending on the dataset.

6.1. Datasets

Two datasets in total were chosen for the experiments. The dataset used for claim check-worthiness detection is the English dataset from *CLEF CheckThat! 2023, Task 1, Subtask B: Check-worthiness estimation from multi-genre content*[1] (CT23) (Barrón-Cedeño et al., 2023). The dataset chosen for fact-verification is *Fact extraction and verification* (FEVER) dataset with added adversarial claims (Atanasova et al., 2020). Both datasets are publicly available, while details regarding the dataset splits are available in Table 6.1.

Table 6.1: Counts of examples in the training, validation and test sets of the datasets chosen for the experiments after preprocessing.

Dataset	Train set	Validation set	Test set
CheckThat! 2023	16876	5587	316
FEVER with adversarial claims	15000 (228279)	16041	15937

The CT23 dataset for English consists of political debate transcripts, split into sentences. Each sentence is labeled with "Yes" or "No", indicating the claim check-worthiness of the sentence. The only preprocessing carried out on the dataset was removing 37 examples from the validation set present in the train set, removing 2 examples from the test set also present in the train set, and removing 1 example from the validation set also present in the test set. The label distributions are shown in table 6.2.

[1] https://gitlab.com/checkthat_lab/clef2023-checkthat-lab/-/tree/main/task1

The train set and the validation set has similar, imbalanced distributions, while the test set contains more positive examples, but is still imbalanced.

Table 6.2: The percentages of values in the label column for the CT23 dataset after preprocessing.

Label column value	Train set (%)	Validation set (%)	Test set (%)
No	75.954	75.818	65.822
Yes	24.045	24.182	34.178

The FEVER dataset consists of claims generated by modifying sentences extracted from Wikipedia articles, while the added adversarial claims were generated using a GPT-2 model by using the original claims, evidences, and universal triggers for changing the label from "SUPPORTS" to "REFUTES", and vice-versa. The values of the label in this dataset are "SUPPORTS", "REFUTES", and "NOT ENOUGH INFO". The reason this dataset was chosen in favor of the original FEVER dataset is because of simplicity, as it does not require further preprocessing. A stratified sample of 15000 examples was used for the training set in all of the experiments to keep the training time manageable. As there were no data leaks between the dataset split, no preprocessing was needed. The final label distribution for the FEVER with adversarial claims dataset is available in table 6.3. As with the CT23 dataset, the FEVER dataset is also imbalanced.

Table 6.3: The percentages of values in the label column for the FEVER with adversarial claims dataset.

Label column value	Train set (%)	Validation set (%)	Test set (%)
SUPPORTS	50.287	29.106	29.266
REFUTES	20.633	30.668	30.481
NOT ENOUGH INFO	29.080	40.226	40.252

6.2. Evaluation

All of the most important classification metrics such as accuracy, recall, precision, and F1 score can be defined using the confusion matrix, which consists of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Accuracy is defined as the number of true positives and negatives divided by all the examples in

the dataset, shown in equation (6.1).

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

Accuracy is a metric that is heavily dependent on the label distribution, e.g., if the dataset contains 90 examples labeled with 0, and 10 examples labeled with 1, always classifying the example as 0 would achieve 90% accuracy. A more robust way to evaluate the model is the F1 score, which is independent of the label distribution. The F1 score is defined as the harmonic mean of precision and recall, as shown in equation (6.2). The F1 score also does not depend on the number of true negatives.

$$\text{F1 score} = \frac{2PR}{P + R} = \frac{TP}{TP + \frac{1}{2}(FN + FP)} \tag{6.2}$$

Recall is a metric that focuses on evaluating the positively labeled instances in the dataset. It is defined as the number of true positives divided by the number of all positively labeled examples. High recall values can be crucial in use cases where the false negatives have severe consequences, such as tumor diagnosis. The definition is shown in equation (6.3).

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6.3}$$

On the other hand, precision is focused on the positively predicted instances in the dataset. It is defined as the ratio of true positives and all positively predicted examples. A use case where precision is convenient is e-mail spam filtering. The definition of precision can be seen in equation (6.4).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.4}$$

6.2.1. Multi-class classification

In the case of binary classification, it is simple to define the entries in the confusion matrix, because there are only two classes: the negative class and the positive class, but in multi-class classification, this is not the case. There are two ways to solve this problem: using the micro or macro estimator of a metric. For both estimators, the first step is to calculate the confusion matrices. To do this, one class is chosen to be the positive class, while all other classes serve as negative classes. Using this procedure, C confusion matrices of dimension 2×2 are obtained.

To calculate the micro estimator of a metric, all entries in C confusion matrices need to be summed up to acquire a single confusion matrix M. Finally, the micro

estimator of a metric is defined as the value of the metric using the entries in the confusion matrix M.

On the other hand, to calculate the macro estimator of a metric, each of the C confusion matrices is used separately to calculate the metric value. Finally, all the values are averaged. In all of the multi-class experiments, the macro estimator is reported, as it is usually smaller than the micro estimator. For simplicity, all of the reported metrics are multiplied by 100.

6.2.2. Hyperparameters

The hyperparameters for all experiments were kept the same. For model pretraining, each model was pretrained for 800 steps with an effective batch size of 2048. The effective batch size was achieved using gradient accumulation, by setting the actual training batch size to 32 and the number of steps for accumulating the gradients to 64. The AdamW optimizer was used with a learning rate of $1\mathrm{e}{-4}$, and with the weight decay hyperparameter set to $1\mathrm{e}{-2}$. A linear schedule was used in combination with a warmup ratio of 0.1. The seed was set to 42 and all of the experiments were carried out using a distributed setup. The hyperparameters for model fine-tuning were different. Each model was trained for 10 epochs, with a batch size of 16, and without using gradient accumulation. The learning rate was set to $2\mathrm{e}{-5}$. Models fine-tuned on the FEVER dataset used a distributed setup due to memory requirements while fine-tuning on the CT23 dataset was done using a single GPU. The final model chosen for evaluation on the test set was the model with the highest F1 score on the validation set. All other hyperparameters were left the same as in the pretraining setup.

6.3. Pretrained architectures

State-of-the-art architectures for NLP are publicly available. The available architectures also come with the weights learned using the pretraining objectives due to memory and time requirements.

6.3.1. Randomly initialized models

The simplest transformer model that can be used is a transformer model that is not pretrained. Three models are considered on both datasets: BERT base, RoBERTa base, and ELECTRA base. These backbones have a similar number of parameters.

ELECTRA and BERT models have around 108M parameters, with the discrepancy coming from a difference in the vocabulary size, while the RoBERTa model has around 124M parameters.

Table 6.4: The F1 scores achieved by randomly initialized baselines on the CT23 dataset. Maximum values in columns are shown in bold.

Model	Train set	Validation set	Test set
BERT base	91.75	**77.57**	81.07
RoBERTa base	87.74	77.05	81.07
ELECTRA base	**92.91**	76.62	**82.89**

Results on the CT23 dataset are shown in table 6.4. All of the models achieve similar F1 scores on the validation set, with the BERT base model being the best-performing model on the validation set. The test set illustrates a different situation where ELECTRA performs best, alhough the results on the test set might be misleading due to the small size. Given that the BERT base and ELECTRA base models have a similar number of parameters, the test set results suggest that the inductive bias of the ELECTRA model is slightly better suited for claim check-worthiness detection.

Table 6.5: The F1 scores achieved by randomly initialized baselines on the FEVER with adversarial claims dataset. Maximum values in columns are shown in bold.

Model	Train set	Validation set	Test set
BERT base	78.04	49.67	48.15
RoBERTa base	83.35	**51.12**	**49.24**
ELECTRA base	**83.64**	51.10	48.10

The results for the fact-verification dataset are available in table 6.5. The validation and test set F1 scores suggest the best-performing model is the RoBERTa model. In contrast to the test set, ELECTRA outperforms BERT on the validation set. The achieved results are generally much lower than on the claim check-worthiness detection task. This can be attributed to the task being a three-way classification task, instead of a binary classification task. Randomly initialized transformer-based models already achieve satisfying results on the claim check-worthiness task, while the use of pretrained representations is crucial for improving fact-verification.

6.3.2. Baselines

The next experiment displays the effect of using pretrained weights for the same transformer architectures on both datasets. Pretrained weights better represent the contextual information contained in the text, meaning the expected outcome is an improvement in the metric values.

Table 6.6: The F1 scores achieved by pretrained transformer models on the CT23 dataset. Maximum values in columns are shown in bold.

Model	Train set	Validation set	Test set
BERT base	99.00	83.49	85.61
RoBERTa base	99.55	84.13	**88.71**
ELECTRA base	**99.71**	**84.63**	85.10

The results of using models with pretrained weights on the CT23 dataset are shown in table 6.7. The RoBERTa model achieves the highest F1 score on the test set and improves the F1 score by the most in comparison to the non-pretrained models. The second best-performing model on the test set is the BERT model, which is only slightly better than the ELECTRA model. On the other hand, the validation set F1 scores indicate that ELECTRA works best. The F1 scores of the pretrained models on all dataset splits are generally greater than the non-pretrained models, illustrating how important pretraining on large corpora is for achieving performance boosts in claim check-worthiness detection. The test set improvements range from 2.21 on the ELECTRA model to 7.64 on the RoBERTa model.

Table 6.7: The F1 scores achieved by pretrained transformer models on the FEVER dataset. Maximum values in columns are shown in bold.

Model	Train set	Validation set	Test set
BERT base	**99.75**	79.75	78.13
RoBERTa base	97.24	**83.26**	**81.61**
ELECTRA base	97.16	83.17	81.11

The results of using models with pretrained weights on the FEVER with adversarial claims dataset are shown in table 6.7. The best-performing model on the test set is RoBERTa, which outperforms ELECTRA by a small margin. In comparison to the other models, the ELECTRA pretraining improved the model performance by the most on the test set (33.01), while the smallest improvement on the test set (29.98) is

seen for the BERT model. The improvements seen from using pretrained weights are much greater in the case of fact-verification rather than claim check-worthiness, suggesting high-quality representations easily connect the information in the claim with the information available in the evidence.

6.4. Adaptive pretraining

The experiments regarding adaptive pretraining were carried out using the same models with pretrained weights: BERT base, RoBERTa base, and ELECTRA base. The considered pretraining procedures were: masked language modeling, shuffle randomization, and ELECTRA pretraining. The masked language modeling and shuffle randomization objectives were tested using the BERT and RoBERTa models, while the ELECTRA pretraining objective was used exclusively with the ELECTRA model.

Table 6.8: The F1 scores achieved by fine-tuning adaptively pretrained transformer models on the CT23 dataset. Maximum values in columns are shown in bold.

Model and pretraining method	Train set	Validation set	Test set
BERT base + shuffle randomization	99.51	83.36	83.46
BERT base + MLM	**99.98**	84.27	85.43
RoBERTa base + shuffle randomization	99.81	**84.68**	86.62
RoBERTa base + MLM	97.89	84.51	85.43
ELECTRA base + ELECTRA	97.42	84.61	**88.64**

The results of combining pretrained models and adaptive pretraining procedures on the CT23 dataset are shown in table 6.8. Both BERT models perform worse on the test set than the original model, but adaptively pretraining BERT using MLM improves the validation F1 score by a small margin (0.78). Taking into account that the test set is of small size, this increase might indicate better generalization properties of the model, even coupled with a smaller test set metric. As well as BERT, RoBERTa suffers from worse test results but slightly improves on the validation set in the case of using shuffle randomization (0.55) and using MLM (0.38). Finally, the adaptively pretrained ELECTRA model achieves similar validation set F1 scores as the original, with the F1 score on the test set outperforming the original model by a fair margin (3.54). The indifference of the F1 scores to the adaptive pretraining can be attributed to the nature of the task at hand. Claim check-worthiness detection is a highly subjective task and it includes the need for knowledge about events, people, and organizations which is

difficult to achieve using pretraining exclusively on the dataset in question.

Table 6.9: The F1 scores achieved by fine-tuning adaptively pretrained transformer models on the FEVER dataset. Maximum values in columns are shown in bold.

Model and pretraining method	Train set	Validation set	Test set
BERT base + shuffle randomization	95.95	79.51	77.82
BERT base + MLM	**99.68**	79.67	77.89
RoBERTa base + shuffle randomization	99.26	83.45	81.21
RoBERTa base + MLM	95.90	83.20	81.18
ELECTRA base + ELECTRA	99.64	**83.65**	**82.01**

The results of combining pretrained models and adaptive pretraining procedures for fact-verification are shown in table 6.9. Results of the BERT model on the validation set and test set present a slightly negative effect on the F1 scores. RoBERTa achieves marginally better results on the validation set (0.19) when using the shuffle randomization pretraining method, but the test set suggests the generalization capabilities of the model are slightly worse. As in the case of claim check-worthiness, the biggest difference is observed in further pretraining the ELECTRA model, with improvements on the validation set (0.48) and the test set (0.9).

6.4.1. Selective masked language modeling

The masking procedure used in masked language modeling chooses the tokens at random. This approach might not be ideal since not all tokens contribute equally to the overall meaning of a sentence. Certain tokens, such as named entities, numbers, or verbs, often carry more informative value and it makes sense to prioritize them for masking. This approach allows the model to focus on predicting crucial information that is relevant to the downstream tasks. In the context of fact-checking, some of the more important parts of a sentence might be the subjects, objects, named entities, numbers, and predicates. The developed selective MLM method chooses the mentioned tokens as candidates for masking in 80% of cases, while in the other 20% of cases, the rest of the tokens are considered. This way the model focuses on the more important parts of the sentence, while not forgetting the previously learned information available in the less important parts of the sentence.

The results for the adaptively pretrained and finetuned models on the CT23 dataset are available in table 6.10. Adaptively pretraining the ELECTRA model using selective

Table 6.10: The F1 scores achieved by fine-tuning adaptively pretrained transformer models using the selective MLM method on the CT23 dataset. Maximum values in columns are shown in bold.

Model and pretraining method	Train set	Validation set	Test set
BERT base + selective MLM	**99.45**	**83.82**	**85.94**
RoBERTa base + selective MLM	99.36	83.48	83.24

MLM is omitted because the model is not pretrained using an MLM procedure. The BERT model achieves better results than the baseline on the validation set (0.33) and on the test set (0.33). In contrast, pretraining the RoBERTa model using the selective MLM method worsens the validation set and the test set results. Besides, the selective MLM method also improves on the test set results (0.51) achieved by adaptively pretraining the BERT model using the default MLM method.

Table 6.11: The F1 scores achieved by fine-tuning adaptively pretrained transformer models using the selective MLM method on the FEVER dataset. Maximum values in columns are shown in bold.

Model and pretraining method	Train set	Validation set	Test set
BERT base + selective MLM	**99.72**	79.94	78.51
RoBERTa base + selective MLM	99.11	**83.32**	**81.41**

The results for the adaptively pretrained and finetuned models on the FEVER dataset are available in table 6.11. As in the case of claim check-worthiness detection, the results for the BERT model and the selective MLM method suggest there are minimal improvements on the validation set (0.19) and test set (0.38) in comparison to the baselines. The results for the RoBERTa model are similar as in the case of no pretraining.

6.4.2. Multi-task pretraining

A weakness of the designed selective MLM method is that it only chooses the important parts of the sentence as candidates for masking, meaning the model does not use all the information available. For this reason, another task is designed to combine information the important words provide, called important word detection (IWD). The selective MLM method classifies tokens over the whole vocabulary taking into account the context, while the IWD is a token-level binary classification problem, where

each token is classified as important or non-important. The words chosen as important words were subjects, objects, predicates, named entities, and numbers. The loss is calculated excluding all special tokens and padding tokens. ELECTRA is omitted because it is not pretrained using MLM.

Table 6.12: The F1 scores achieved by fine-tuning adaptively pretrained transformer models using the selective MLM method combined with important word detection on the CT23 dataset. Maximum values in columns are shown in bold.

Model and pretraining method	Train set	Validation set	Test set
BERT base + selective MLM + IWD	99.43	83.73	83.13
RoBERTa base + selective MLM + IWD	**99.71**	**84.26**	**84.57**

The results of using multi-task pretraining on the CT23 dataset are shown in table 6.12. The results for the BERT model are similar to the baselines, with the validation set F1 score being slightly higher (0.24), but underperforming on the test set. The same is true for the RoBERTa model, which achieves an even smaller boost in performance on the validation set (0.13). Overall, the F1 scores indicate that combining the two methods results in a worse parameter initialization which deteriorates the performance of claim check-worthiness detection.

Table 6.13: The F1 scores achieved by fine-tuning adaptively pretrained transformer models using the selective MLM method combined with important word detection on the FEVER dataset. Maximum values in columns are shown in bold.

Model and pretraining method	Train set	Validation set	Test set
BERT base + selective MLM + IWD	91.17	79.83	77.56
RoBERTa base + selective MLM + IWD	**99.15**	**83.49**	**81.18**

The results of using multi-task pretraining on the FEVER dataset are shown in table 6.13. As in the case of claim check-worthiness detection, both models achieve higher F1 scores on the validation set, but lower F1 scores on the test set, indicating multi-task pretraining for fact-verification is not beneficial.

6.5. Fine-tuning strategies

This section covers approaches that modify the finetuning procedure. Three approaches are examined: adding information about important words, check-worthiness, and fact-

verification knowledge transfer and weighting the cross-entropy loss due to dataset imbalances.

6.5.1. Adding information about important words

The first mentioned approach is adding information about important words directly into the finetuning procedure. The considered approach transforms the claims by adding a special token after each important word, where the important words are the same as in the selective MLM pretraining procedure. It is important to note that for simplicity, the special token for important words was introduced in the finetuning procedure, meaning the representation is initialized at random. Nevertheless, it might give the model more insight into what words are crucial for better understanding the claim in question and solving the task. The approach adds one embedding vector in the embedding layer of the model, meaning the number of parameters is not the same.

Table 6.14: The F1 scores achieved by fine-tuning transformer models by adding information about important words on the CT23 dataset. Maximum values in columns are shown in bold.

Model	Train set	Validation set	Test set
BERT base	98.64	83.38	**88.03**
RoBERTa base	**99.34**	**84.39**	85.79
ELECTRA base	99.01	83.32	86.37

The results of directly adding the information about important words into the fine-tuning procedure on the CT23 dataset are shown in table 6.14. The achieved validation set results are slightly worse for the BERT and ELECTRA models, and slightly better for the RoBERTa model. The test set results show big improvements in comparison to the baseline BERT model (2.42), meaning it provides promising generalization properties. Other models underperform on the test set when compared to the corresponding baselines.

Table 6.15: The F1 scores achieved by fine-tuning transformer models by adding information about important words on the FEVER dataset. Maximum values in columns are shown in bold.

Model	Train set	Validation set	Test set
BERT base	93.20	79.03	77.53
RoBERTa base	99.10	82.86	81.20
ELECTRA base	**99.42**	**83.44**	**81.96**

The results of directly adding the information about important words into the fine-tuning procedure on the FEVER dataset are shown in table 6.15. BERT and RoBERTa performance suffers both on the validation set and on the test set, but ELECTRA improves slightly on the validation set (0.27) and on the test set (0.85). This means the results greatly depend on the model in question for both datasets.

6.5.2. Claim check-worthiness detection and fact-verification knowledge transfer

The second mentioned approach is inspired by Reimers and Gurevych (2019). The authors consider the task of semantic textual similarity with a focus on efficiency by introducing a siamese BERT network. They noticed an improvement in the performance of their model by finetuning on a natural language inference dataset and then using the finetuned model as an initialization for finetuning on the semantic textual similarity dataset. The next experiment displays how knowledge of claim check-worthiness transfers to fact-verification and vice versa.

Table 6.16: The F1 scores achieved by fine-tuning transformer models on the FEVER dataset first and then on the CT23 dataset. Maximum values in columns are shown in bold.

Model	Train set	Validation set	Test set
BERT base	**99.89**	83.27	83.90
RoBERTa base	99.68	84.20	**86.97**
ELECTRA base	99.72	**84.24**	86.63

Table 6.16 demonstrates the effect of finetuning on the FEVER dataset and then on the CT23 dataset. Knowledge learned from the fact-verification task deteriorates the performance of BERT and RoBERTa on the test set, but enhances the F1 score of the ELECTRA model on the test set by a fair margin (1.53).

Table 6.17: The F1 scores achieved by fine-tuning transformer models on the CT23 dataset first and then on the FEVER dataset. Maximum values in columns are shown in bold.

Model	Train set	Validation set	Test set
BERT base	**99.57**	78.50	77.20
RoBERTa base	99.08	82.82	**81.25**
ELECTRA base	99.38	**83.06**	80.80

The influence of finetuning on the CT23 dataset, followed by finetuning on the FEVER dataset is shown in 6.17. The results suggest that all of the models perform worse when using a claim check-worthiness detection model as an initialization point for fact-verification. Contrary to the case of finetuning on the FEVER dataset followed by finetuning on the CT23 dataset, the declining performance cannot be attributed to the choice of the model.

6.5.3. Weighted cross-entropy loss

The final approach takes a more conservative route of improving the model performance. Since the label distribution of the CT23 dataset is heavily imbalanced (table 6.2), and the label distribution of the FEVER dataset is slightly imbalanced (table 6.3), the method uses a weighting scheme when computing the loss function. The weighting added is the ratio of the number of examples and the value counts of the labels. This means an example from a class with a small number of examples corresponds to a larger weight, and an example from a class with a large number of examples corresponds to a smaller weight.

Table 6.18: The F1 scores achieved by fine-tuning transformer models on the FEVER dataset first and then on the CT23 dataset. Maximum values in columns are shown in bold.

Model	Train set	Validation set	Test set
BERT base	98.60	84.15	**86.79**
RoBERTa base	99.60	**84.62**	85.61
ELECTRA base	**99.81**	84.10	86.63

The results of finetuning on the CT23 dataset with a weighted loss function are shown in table 6.18. In comparison to the baseline, BERT results on the validation set (0.66) and on the test set (1.18) improve when using the weighting scheme. RoBERTa achieves slight improvements on the validation set (0.49), while the test set F1 score deteriorates. Finally, although ELECTRA demonstrates inferior performance on the validation set, there is a notable improvement in the test set F1 score (1.53).

Table 6.19 presents the results obtained from applying a weighted loss function during the finetuning process on the FEVER dataset. In the case of fact-verification, BERT and RoBERTa achieve subpar results in comparison to their baseline counterpart. On the other hand, ELECTRA achieves minor improvements on the validation set (0.38) and the test set (0.7), suggesting weighting has a positive effect on the performance.

Table 6.19: The F1 scores achieved by fine-tuning transformer models on the CT23 dataset first and then on the FEVER dataset. Maximum values in columns are shown in bold.

Model	Train set	Validation set	Test set
BERT base	**99.71**	79.71	78.27
RoBERTa base	99.25	82.88	81.56
ELECTRA base	97.11	**83.55**	**81.81**

6.6. Error analysis

An error analysis of a model for fact-checking plays an important role in understanding and uncovering limitations. Despite the effectiveness of large language models, they are not flawless, and they can make errors that may have intricate biases hidden in the data they were trained on. Conducting an error analysis is vital to identify and investigate the failure modes, as well as examining the specific scenarios or types of claims where they are more prone to errors. By understanding these failure modes, it is easier to refine and improve the models, leading to better performance of fact-checking systems. For simplicity, the error analysis only covers the adaptively pretrained BERT model and the baseline BERT model.

Ten claims where the baseline BERT mistakes the most are shown in table 6.20. A fundamental aspect of the claims with the highest error is the abundance of pronouns present in them. This makes it hard to identify whether the claim is check-worthy or not because the check-worthiness heavily depends on the context, e.g., if “my” in the third sentence refers to the president, then the claim is check-worthy, but if it refers to a person that is not a public figure, then the check-worthiness changes. Using a coreference resolution model might improve the performance in this specific case. On the other hand, some incorrectly labeled examples are present as well, e.g., the seventh claim is incorrectly labeled because no particular governors are mentioned, indicating that such mistakes might be present in the train set as well. The model might benefit from relabeling hard examples or completely removing them from the train set. The two previously mentioned problems demonstrate the subjectivity of the claim check-worthiness detection task and the need for high-quality labels and data.

Table 6.21 presents the ten examples where the adaptively pretrained BERT makes the biggest mistakes. On first glance, some examples which are hard for the baseline BERT model also appear in the mentioned table, indicating the existence of universally difficult examples. As previously mentioned, these might be mislabeled or heavily

Table 6.20: Ten claims from the test dataset ranked in descending order by the difference of the correct class label and the probability of the correct class of the baseline BERT model fine-tuned on the CT23 dataset.

Rank	Claim	Label
1	He thinks wind causes cancer, windmills.	Yes
2	The general who was with him said All he ever wants to do is divide people not unite people at all.	Yes
3	The Green New Deal is not my plan...	Yes
4	So why didn't he do it for 47 years?	Yes
5	And the fact is, I've made it very clear, within 100 days, I'm going to send to the United States Congress a pathway to citizenship for over 11 million undocumented people.	No
6	And we're in a circumstance where the President, thus far, still has no plan.	Yes
7	Many of your Democrat Governors said, President Trump did a phenomenal job.	Yes
8	Yeah, you did say that.	Yes
9	Some of these ballots in some states can't even be opened until election day.	Yes
10	Because it costs a lot of money to open them safely.	Yes

dependent on the context.

Table 6.21: Ten claims from the test dataset ranked in descending order by the difference of the correct class label and the probability of the correct class of the adaptively pretrained BERT model fine-tuned on the CT23 dataset.

Rank	Claim	Label
1	He thinks wind causes cancer, windmills.	Yes
2	We took away the individual mandate.	Yes
3	And we're in a circumstance where the President, thus far, still has no plan.	Yes
4	And it's gone through, including the Democrats, in all fairness.	Yes
5	The individual mandate – where you have to pay a fortune for the privilege of not having to pay for bad health insurance.	Yes
6	The places we had trouble were Democratic-run cities...	Yes
7	Manufacturing went in a hole-	Yes
8	Dr. Fauci said the opposite.	Yes
9	So why didn't he do it for 47 years?	Yes
10	Many of your Democrat Governors said, President Trump did a phenomenal job.	Yes

Five examples with the largest error made by the BERT baseline model fine-tuned on the FEVER dataset are showcased in table 6.22. First, the five most inaccurate predictions are made for examples labeled as "SUPPORTS". Unlike the case of claim check-worthiness detection, there is no apparent pattern in the claims. The possible failure modes include failure to link the surname and full name to the same person (example 1), failure to connect information contained in the predicate and object of the claim to an adverbial phrase from the evidence (examples 2 and 5), and having

problems with connecting information in the predicate and object of the claim to a participle phrase (example 3). The fourth example is the most alarming since there is no obvious reason for the misclassification.

Table 6.22: Five claims from the test dataset ranked in descending order by the difference of the correct class label and the probability of the correct class of the baseline BERT model fine-tuned on the FEVER dataset.

Rank	Claim	Evidence	Label
1	Stephen Moyer acted in a film adaptation of a comic.	In 1997, Moyer made his big-screen debut landing the lead role in the film adaptation of the long-running comic strip Prince Valiant by Hal Foster, working alongside Ron Perlman and Katherine Heigl.	SUPPORTS
2	Emmanuel Macron worked as a banker.	A former civil servant and investment banker, he studied philosophy at Paris Nanterre University, completed a Master's of Public Affairs at Sciences Po, and graduated from the École nationale d'administration (ENA) in 2004.	SUPPORTS
3	Jerome Flynn was born in March.	Jerome Patrick Flynn (born 16 March 1963) is an English actor and singer.	SUPPORTS
4	Kevin Bacon played a former child abuser.	Bacon is also known for taking on darker roles such as that of a sadistic guard in Sleepers and troubled former child abuser in a critically acclaimed performance in The Woodsman.	SUPPORTS
5	Mandy Moore is a musician.	Outside of her musical career, Moore has also branched out into acting.	SUPPORTS

Finally, five examples with the largest error made by the adaptively pretrained BERT model finetuned on the FEVER dataset are provided in table 6.23. Unlike the baseline BERT model, the most incorrect predictions are made for examples labeled as "REFUTES". In comparison to the baseline, it is easier to get an insight into what failure modes might be present. The misclassification into the "NOT ENOUGH INFO" class is somewhat justified in example one since evidence provides a similar amount of officially confirmed kills. Examples two and three show similar properties as in the previous model, except the issue arises in connecting information in the predicate and object to the information in the noun phrase. The fifth example requires external knowledge about the current year which is not available to the model, while the fourth claim does not show any obvious reason for the misclassification.

Table 6.23: Five claims from the test dataset ranked in descending order by the difference of the correct class label and the probability of the correct class of the adaptively pretrained BERT model fine-tuned on the FEVER dataset.

Rank	Claim	Evidence	Label
1	American Sniper (book) is about a sniper with 170 officially confirmed kills.	With 255 kills, 160 of them officially confirmed by the Pentagon, Kyle is the deadliest marksman in U.S. military history.	REFUTES
2	Lorelai Gilmore's father is named Robert.	The dynamic of single parenthood and the tension between Lorelai and her wealthy parents, Richard (Edward Herrmann) and especially her controlling mother, Emily (Kelly Bishop), form the main theme of the series story line.	REFUTES
3	Pearl (Steven Universe) projects a holographic butterfly.	She is a "Gem", a fictional alien being that exists as a magical gemstone projecting a holographic body.	REFUTES
4	Houses became Virginia's most valuable export in 2002.	Virginia's economy changed from primarily agricultural to industrial during the 1960s and 1970s, and in 2002 computer chips became the state's leading export by monetary value.	REFUTES
5	Temple of the Dog celebrated the 37th anniversary of their self-titled album.	The band toured in 2016 in celebration of the 25th anniversary of their self-titled album.	REFUTES

7. Conclusion

Automating the fact-checking process is crucial for this day and age, filled with an abundance of information from different sources. The development of such systems improves the efficiency of manual fact-checking organizations. The goal of this thesis was to see how different adaptive pretraining techniques affect transformer models on the tasks of claim check-worthiness and fact-verification.

The thesis describes the basic theory behind machine learning, natural language processing, and its applications, and how the two combine. Deep learning approaches are explained in depth, including recurrent neural networks and the transformer. An overview of the important pretraining techniques is given as well. First, the performance of non-pretrained transformer-based models is tested, followed by the pretrained models which serve as baselines. The experiments regarding pretraining cover masked language modeling, shuffle randomization, and ELECTRA pretraining. Besides, a new strategy for masked language modeling was developed, which focuses on masking more important words in the claims. Multi-task pretraining was experimented with, also using a newly developed method called important word detection. The method classifies each token as an important token or a non-important token. Aside from experimenting with pretraining methods, different finetuning strategies were tested: directly adding information about important words into the finetuning procedure using a special token, transferring knowledge from different subtasks in fact-checking and weighting the loss function to compensate for the dataset imbalances. The most notable improvements are visible in adaptively pretraining the ELECTRA model, directly adding information about the important words into the finetuning procedure in combination with BERT and ELECTRA, and weighting the cross-entropy loss function when using ELECTRA and BERT. The selective MLM method shows slight, but promising improvements in the context of fact-checking applications when compared to the baselines and the default MLM method.

Future research in the field regarding pretraining may explore guiding the masked language modeling in an unsupervised manner using a TF-IDF weighting scheme and

choosing words with a higher value than some predefined threshold as candidates for masking, as well as pretraining the models from scratch. Apart from pretraining, fine-tuning procedures can be further examined by using a mean of the token that represents the whole sequence and the tokens for each of the important words.

BIBLIOGRAPHY

Ethem Alpaydin. *Introduction to Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, edition 3, 2014. ISBN 978-0-262-02818-9.

Craig Anderson. Belief perseverance and self-defeating behavior. 01 1989. doi: 10.1007/978-1-4613-0783-9_2.

Pepa Atanasova, Dustin Wright, and Isabelle Augenstein. Generating label cohesive and well-formed adversarial claims. U *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3168–3177, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.256. URL https://aclanthology.org/2020.emnlp-main.256.

Alberto Barrón-Cedeño, Firoj Alam, Tommaso Caselli, Giovanni Da San Martino, Tamer Elsayed, Andrea Galassi, Fatima Haouari, Federico Ruggeri, Julia Maria Struss, Rabindra Nath Nandi, Gullal S. Cheema, Dilshod Azizov, and Preslav Nakov. The clef-2023 checkthat! lab: Checkworthiness, subjectivity, political bias, factuality, and authority. U Jaap Kamps, Lorraine Goeuriot, Fabio Crestani, Maria Maistro, Hideo Joho, Brian Davis, Cathal Gurrin, Udo Kruschwitz, and Annalina Caputo, editors, *Advances in Information Retrieval*, pages 506–517, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-28241-6.

Gullal S Cheema, Sherzod Hakimov, Abdul Sittar, Eric Müller-Budack, Christian Otto, and Ralph Ewerth. Mm-claims: A dataset for multimodal claim detection in social media. *arXiv preprint arXiv:2205.01989*, 2022.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: pre-training text encoders as discriminators rather than generators. *CoRR*, abs/2003.10555, 2020. URL https://arxiv.org/abs/2003.10555.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

J.R. Firth and F.R. Palmer. *Selected Papers of J.R. Firth, 1952-1959*. Indiana University Studies in the History and Theory of Linguistics. Longmans, 1968. ISBN 9780253351357. URL https://books.google.hr/books?id=OCC3oAEACAAJ.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus, 2019.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Zhijiang Guo, Michael Sejr Schlichtkrull, and Andreas Vlachos. A survey on automated fact-checking. *CoRR*, abs/2108.11896, 2021. URL https://arxiv.org/abs/2108.11896.

Naeemul Hassan, Chengkai Li, and Mark Tremayne. Detecting check-worthy factual claims in presidential debates. U *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 1835–1838, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450337946. doi: 10.1145/2806416.2806652. URL https://doi.org/10.1145/2806416.2806652.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. Spanbert: Improving pre-training by representing and predicting spans. *CoRR*, abs/1907.10529, 2019. URL http://arxiv.org/abs/1907.10529.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *CoRR*, abs/1901.07291, 2019. URL http://arxiv.org/abs/1901.07291.

Justin Lewis, Andy E. Williams, Robert Arthur Franklin, James Thomas, and Nick Mosdell. The quality and independence of british journalism. 2008.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. URL http://arxiv.org/abs/1711.05101.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomás Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013b. URL http://arxiv.org/abs/1310.4546.

Tom M Mitchell. *Machine learning*, svezak 1. McGraw-hill New York, 1997.

Cristopher Olah. Understanding lstm networks, 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018. URL http://arxiv.org/abs/1802.05365.

Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084, 2019. URL http://arxiv.org/abs/1908.10084.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. 1986.

Trieu H. Trinh and Quoc V. Le. A simple method for commonsense reasoning. *CoRR*, abs/1806.02847, 2018. URL http://arxiv.org/abs/1806.02847.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Wei Wang, Bin Bi, Ming Yan, Chen Wu, Zuyi Bao, Liwei Peng, and Luo Si. Structbert: Incorporating language structures into pre-training for deep language understanding. *CoRR*, abs/1908.04577, 2019. URL http://arxiv.org/abs/1908.04577.

Atsuki Yamaguchi, George Chrysostomou, Katerina Margatina, and Nikolaos Aletras. Frustratingly simple pretraining alternatives to masked language modeling. *CoRR*, abs/2109.01819, 2021. URL https://arxiv.org/abs/2109.01819.

Kombiniranje adaptivnih tehnika predtreniranja za zadatak automatizirane provjere činjenica

Sažetak

Razvoj tehnika strojnog učenja uvelike je promijenio ulogu numeričkih podataka i kako se ti podaci iskorištavaju. Kombiniranje tehnika obrade prirodnog jezika i arhitekture transformatora nadmašilo je ljudske performance na određenim zadacima razumijevanja prirodnog jezika. Jedan od zadataka razumijevanja prirodnog jezika je i automatizirana provjera činjenica, koja postaje sve više potrebna zbog brzine širenja dezinformacija u modernim medijima. Ovaj rad bavi se primjenom modela transformatora na zadatak detekcije tvrdnji koje je potrebno provjeriti i na zadatak provjere činjenica. Rad uključuje niz eksperimenata vezanih uz predtreniranje arhitektura transformatora i fino podešavanje parametara modela na spomenutim zadacima. Za predtreniranje modela isprobane su neke od postojećih metoda, osmišljena je i isprobana strategija ciljanog maskiranja riječi za zadatak maskiranog modeliranja jezika, i osmišljen je i isproban zadatak detekcije riječi značajnih za zadatak provjere činjenica. Nakon eksperimenata vezanih uz predtreniranje modela, za fino podešavanje parametara modela osmišljen je i isproban način direktnog ugrađivanja informacije o značajnim riječima, testiran je utjecaj korištenja otežavanja funkcije gubitka i međusobni prijenos znanja između zadataka provjere činjenica. Za svaki od naučenih modela prezentirani su i komentirani rezultati, a detaljnija analiza pogrešaka napravljena je za BERT model.

Ključne riječi: provjera činjenica, samonadzirano učenje, obrada prirodnog jezika, duboko učenje, transformator

Combining adaptive pretraining techniques for the task of automated fact-checking

Abstract

The development of machine learning has drastically changed the role of numerical data and how it can be exploited. Combining natural language processing techniques and transformer architecture has surpassed human performance on certain natural language understanding tasks. One of the natural language understanding tasks is automated fact-checking, which has become increasingly important due to the rapid spreading of misinformation and disinformation in modern media outlets. The thesis focuses on applying transformer models to the task of claim check-worthiness detection and fact-verification. It includes a series of experiments regarding pretraining transformer models and fine-tuning on the mentioned tasks. Besides experimenting with the well-known pretraining methods, a method for strategically masking words in the masked language modeling procedure was developed, as well as a method for detecting highly informative words. Aside from experimenting with pretraining, the thesis includes testing the inclusion of highly informative words directly into the fine-tuning procedure, as well as mutual knowledge transfer from each of the fact-checking tasks, and the use of loss function weighting. Each of the trained models is presented and the results are commented on, while a more detailed error analysis is carried out for the BERT model.

Keywords: fact-checking, self-supervised learning, natural language processing, deep learning, transformer