Ludwig-Maximilians-Universität München

Department of Statistics

Master's Thesis

Exploration of fine-tuning and inference time of large pre-trained language models in NLP

Author:
Anna Korotkova

Supervisor:
Prof. Dr. Christian Heumann
Matthias Aßenmacher

17^{th} November 2020

Statement of authorship

I hereby declare that I have written the following master's thesis on my own without the use of any sources other than those cited in the following text, graphics, tables and formulas. Neither this nor a similar work has been presented to an examination committee.

Munich, 17^{th} November 2020, ...

Abstract

In this master's thesis, we perform an analysis based on the language models BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), DistilBERT (Sanh et al., 2020), ALBERT (Lan et al., 2019) and XLNET (Yang et al., 2019) whose architectures are based on the Transformer encoder which relies solely on an attention mechanism. We incorporate a measuring of the fine-tuning and inference time in the Python module `transformers` which was implemented by *huggingface* (Wolf et al., 2019). In the scope of this master's thesis, the models are fine-tuned on the General Language Understanding Evaluation tasks (Wang et al., 2019), a collection of nine sentence- or sentence-pair language understanding tasks. To monitor our analyses, we used the tool "Weights & Biases" (wandb) (Biewald, 2020) where a logging of hyperparameters and output metrics is possible. The goal of this master's thesis is to develop a diverse benchmark setting and evaluate the respective fine-tuning as well as inference times across different hyperparameters, language models and tasks. More precisely, a hyperparameter sweep with the maximum sequence length and the batch size are regarded, taking on the values {128, 256, 512} and {8, 16, 32}, respectively. We show that the maximum sequence length, the task size as well as the model have an effect on the fine-tuning time, whereas the batch size seems to be of secondary importance. Moreover, the maximum sequence length, the batch size and the choice of the model affect the inference time, whereas the task itself does not show such results.

Contents

List of Figures

List of Tables

List of Acronyms

AE	autoencoding
ALBERT	A Lite BERT
AR	autoregressive
BERT	Bidirectional Encoder Representations from Transformers
biRNN	bidirectional RNN
BOW	Bag of words
BPE	byte-pair encoding
CBOW	Continuous Bag of Words
CC-NEWs	Commom Crawl News data set
DistilBERT	a distilled version of BERT
GeLU	Gaussian Error Linear Unit
GloVe	Global Vectors for Word Representation
GLUE	General Language Understanding Evaluation
GRU	Gated Recurrent Unit
LM	language model
LSTM	Long Short-Term Memory
MLM	Masked Language Model
NLP	Natual Language Processing
NLU	natural language understanding
NSP	Next Sentence Prediction
RACE	ReAding Comprehension Dataset From Examinations
ReLU	rectified linear unit
RNN	Recurrent Neural Network

RoBERTa	Robustly optimized BERT approach
seq2seq	Sequence-to-Sequence
SOP	Sentence Order Prediction
SQuAD v1.1	Stanford Question Answering Dataset
SWAG	Situations With Adversarial Generations
TF-IDF	Term Frequency-Inverse Document Frequency

1 Introduction

With the introduction of Natual Language Processing (NLP) it became possible to create a direct communication between humans and computers. New techniques and methods for the processing of natural language in a computer-based manner were implemented and different methods and results from the linguistics were used and combined with Artificial Intelligence. This enables the usage of NLP for many tasks, such as speech recognition, segmentation of words or sentences or recognition of grammatical information (Luber, 2016). NLP's necessity is constantly growing with the increase of text production and improvement of technology. It turns out to be a challenging matter due to the fact that the natural language itself always changes and develops, is not well-defined and cannot be described as unambiguous (Goldberg, 2017). There are many attempts to solve these kinds of tasks with large and complicated architectures. A popular method that was implemented in the past years are large pre-trained language models which are subsequently fine-tuned. Five of these models are regarded in the following scientific work. A challenging issue that goes along with the usage of such models is high computational costs, specifically long training times. In the following, we analyze the fine-tuning time as well as the inference time of large pre-trained models.
In the second chapter (2) of this scientific work, some basic theory of document representations, word embeddings and language models is introduced. The subsequent chapter (3) illustrates the principle of Recurrent Neural Network (RNN), whereupon Attention and the Transformer architecture are described in more detail. Furthermore, we carry out an explanation of large pre-trained language model (LM) that are regarded in the later analysis, namely, BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), DistilBERT (Sanh et al., 2020), XLNET (Yang et al., 2019) and ALBERT (Lan et al., 2019). It follows a description of the regarded datasets, being the General Language Understanding Evaluation (GLUE) tasks (chapter 4), the experimental setup and implementation (chapter 5) and the analysis of fine-tuning and inference time itself (chapter 6 and 7, respectively). The last chapter (8) includes a conclusion and an outlook.

Note that in the following master's thesis mostly the following notation applies: Scalar values are denoted as uncased letters, e.g. a, vectors as lowercase, bold letters, e.g. $\mathbf{a}$ and matrices as uppercase letters, such as $\mathbf{C}$. Furthermore, we denote vector elements as lowercase letters with an index, for example a_i represents the i-th element of vector $\mathbf{a}$. A similar notation holds for matrices, with c_{ij} denoting the entry i, j in a matrix $\mathbf{C}$. Transposing a matrix results in a notation of the form $\mathbf{C^T}$.

2 From word representation to language model

Data preprocessing represents a crucial aspect in the field of Natural Language Processing. The input data in NLP is represented by textual data in the form of sequences that can exceed the length of a sentence, paragraphs or documents (Goldberg, 2017). Since machine learning algorithms cannot process textual data, the applicant has to use a feature extraction method and decide which one to use in order to perform a conversion of text into a matrix of features (Kothari, 2020). The methods which are described in the following section can be used to carry out this conversion. First, context-free document representations including Bag of words (BOW) and Term Frequency-Inverse Document Frequency (TF-IDF) are described, followed by word embeddings, such as Word2Vec and GloVe. Lastly, the principle of language modeling is introduced.

2.1 Document representations

One-hot encoding forms the basis of the document representations BOW and TF-IDF which are described in the following subsections. Each word that is included in a given text corpus is represented as a binary vector, i.e. a vector with the values 0 and 1. Hereby, all values are 0 except for one value which takes on the value 1. This value then represents the word that is encoded (Jeet, 2020).

2.1.1 Bag of words

One popular approach for feature extraction is named Bag of words. By using BOW, text can be transformed into vectors of a fixed length by taking the word count into account (Zhou, 2019). In the context of BOW, a text corpus, which can be a sentence or a document, can be referred to as a bag of words. When applying BOW, lists are generated in which the word order is not meaningful and the grammar is being ignored (Rouse, 2018).

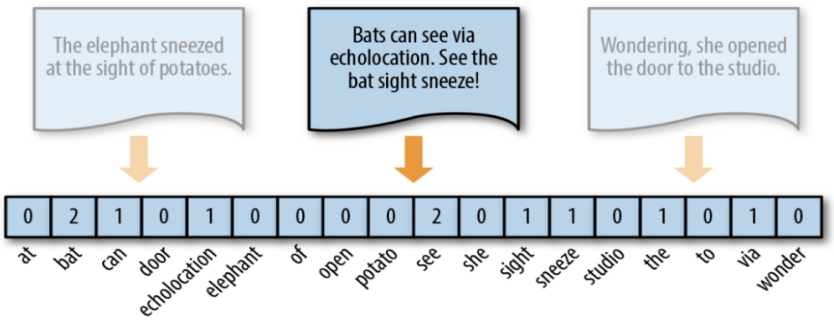

Figure 1: Example of a BOW representation, (Ghelani, 2019a)

In this example (figure 1), the text corpus is created on the basis of each unique word of the three sentences (documents). Generally speaking, the vector created in the BOW process can comply with unique words or n-grams[1] (also referred to as tokens) of the text corpus. The vector shown in figure 1 displays the representation of the second sentence. Hereby, each vector element incorporates the number of occurrences in the second sentence, e.g. the word "see" occurs two times[2]. Important to note is that in BOW, each word count is taken into account as a feature by regarding the histogram of words that appear in the text (Goldberg, 2017). This can be challenging since it is not necessarily an indicator for the importance of words. The word "the" and "and" for example show a high frequency in English texts but do not contribute much to gain a better understanding of it .

2.1.2 Term Frequency-Inverse Document Frequency

Another approach for encoding words is the Term Frequency-Inverse Document Frequency (TF-IDF) method. The main goal is the evaluation of the importance of a word to a document D which is included in a larger corpus V_{corpus}. Hereby, the term frequency (TF) measures how often the word w appears in the considered document ($\#_D(w)$). TF is then normalized with the document length to receive a

[1] an n-gram is a consecutive sequence of n elements from a text sample (Wikipedia, the free encyclopedia, 2020e)

[2] It seems that in this example word stemming and lemmatization were performed before constructing the vocabulary, which results in a reduced vocabulary (for example "bat" instead of "bats"). For more details about both methods, see https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

comparable measure. The inverse document frequency (IDF), on the other hand, quantifies how low the occurrence of the word is across documents. The IDF key figure gets higher, the rarer the word occurs (Ghelani, 2019a). The TF-IDF score can be quantified in the following manner:

$$\frac{\#_D(w)}{\sum_{w' \in D}(w')} * \log \frac{|V_{corpus}|}{|D \in V_{corpus} : w \in D|}, \tag{1}$$

whereas the first part of the equation refers to the term frequency (TF) and the second part to the inverse document frequency (IDF) (Goldberg, 2017).

A drawback of both presented methods is the fact that neither word similarities nor word meanings are encoded into the vectors (Ghelani, 2019a).

2.2 Word Embeddings

Another limitation of the two methodologies described above is that the vector representation of documents increases with increasing vocabulary size. As a consequence, we receive a vector that contains plenty of zeros (sparse vector) which makes use of more computational and memory resources during the fitting. A solution to this problem and the limitation of not being able to grasp word semantics are Neural Word Embeddings. The outstanding aspect of this methodology is the usage of a dense distributed representation for each word which requires much fewer dimensions than the sparse representation. Furthermore, applying contextual similarity results in more meaningful representations. These are vectors with real values that were learned from text in which words with the same meaning take on similar representations. Hence, word meanings can be captured (Ghelani, 2019a; Brownlee, 2017). The two most known word embeddings Word2Vec and GloVe are described in the following.

2.2.1 Word2Vec

A popular alternative to BOW is an algorithm named Word2Vec that was developed by Tomáš Mikolov and colleagues (Mikolov et al., 2013b). The primary goal is to learn a word embedding from a text corpus. Hereby, the context is determined

by a so-called window of neighboring words, e.g. how many words to the left and to the right side of the regarded word are included in the local usage context of the specific word. The advantage of this method is that it enables learning larger word embeddings with higher dimensions for larger text corpora which can include billions of words. This can be attributed to the algorithm's low space and time complexity (Mikolov et al., 2013b; Brownlee, 2017). Furthermore, computations, such as addition or subtraction, can result in meaningful word embeddings. A well-known example is the analogy solving task $w_{king} - w_{man} \approx w_{queen}$ with w as the word embedding (Goldberg, 2017). This indicates that by performing mathematical calculations, a certain degree of language understanding can be attained (Mikolov et al., 2013b). Word2Vec incorporates different optimization objectives, e.g. the Negative-Sampling objective. In this variant of the algorithm, a training of the network is performed with the goal to differentiate between word-context pairs that can be described as "good" from the "bad" ones. This objective aims for the estimation of $P(D = 1|w, c)$ which can be interpreted as the probability that the word-context pair is included in the set D of correct word-context pairs. A high probability should be assigned to pairs that come from D and a low one for pairs from $\bar{D}$ (set of incorrect pairs). The probability function can be quantified in the following manner with $r(w, c)$ being the score assigned to the word-context combination:

$$P(D = 1|w, c) = \frac{1}{1 + e^{-r(w,c)}}. \tag{2}$$

The objective of Word2Vec is the maximization of the following log-likelihood:

$$\mathcal{L}(\Theta; D, \bar{D}) = \sum_{(w,c)\in D} \log P(D = 1|w, c) + \sum_{(w,c)\in \bar{D}} \log P(D = 0|w, c). \tag{3}$$

Positive representations D are created from a corpus, whereas negative representations $\bar{D}$ are generated in the following way: "for each good pair $(w, c) \in D$, sample k words $w_{1:k}$ and each of (w_i, c) as a negative example to $\bar{D}$" (Goldberg, 2017, p. 124). As a consequence, $\bar{D}$ is k times larger than D with k being an algorithm parameter. Negative words w are sampled with the smoothed corpus-based frequency: $\frac{\#(w)^{0.75}}{\sum_{w'} \#(w')^{0,75}}$ (Goldberg, 2017, p. 124).

Word2Vec can be divided into two variants, namely Continuous Bag of Words

(CBOW) and Skip-Gram, which are described in the following.

2.2.1.1 CBOW The word embeddings retrieved from the CBOW model are generated by the prediction of the current word on the basis of its surrounding words, i.e. its context (Brownlee, 2017).
The CBOW version of Word2Vec specifies the context vector $\mathbf{c}$ as $\mathbf{c} = \sum_{i=1}^{k} c_i$. The word-context scoring function is $r(w, c) = \mathbf{w} \cdot \mathbf{c}$. Hence, the CBOW makes use of a continuous distributed representation of the context (Mikolov et al., 2013a). This results in

$$P(D = 1|w, c_{1:k}) = \frac{1}{1 + \exp(-\sum_{i=1}^{k} (\mathbf{w} \cdot c_i))}. \tag{4}$$

It should be noted that the CBOW version does not retain the information about the order of the context's elements whilst being able to use unbounded-length contexts, i.e. contexts with variable length. On the other hand, when considering bounded context length, keeping the order information is possible. This can be done by adding the relative position to the context element (Mikolov et al., 2013a; Goldberg, 2017).

2.2.1.2 Skip-Gram In contrast to the CBOW model, a prediction of the context on the basis of a current word is performed in order to learn the word embeddings of the Skip-Gram model (Mikolov et al., 2013a; Brownlee, 2017).

Within Skip-Gram the assumption is made that the elements c_i which are part of a context $c_{1:k}$ are independent from each other. Therefore, a word-context pair $(w, c_{1:k})$ within D can be seen as k different contexts $(w, c_1), \ldots, (w, c_k)$. The definition of $r(w, c)$ is very similar to the one in CBOW with the difference that each context is defined as a single embedding vector which takes on the following form:

$$P(D = 1|w, c_i) = \frac{1}{1 + \exp(-\mathbf{w} \cdot c_i)}$$

$$P(D = 1|w, c_{1:k}) = \prod_{i=1}^{k} P(D = 1|w, c_i) = \prod_{i=1}^{k} \frac{1}{1 + \exp(-\mathbf{w} \cdot c_i)} \tag{5}$$

$$\log P(D = 1|w, c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1 + \exp(-\mathbf{w} \cdot c_i)}.$$

Even though a strong independence assumption is made within Skip-Gram, it is a popular word embedding and effective in practice (Mikolov et al., 2013b; Goldberg, 2017, p. 125).

2.2.2 GloVe

The second word embedding methodology which was designed by Pennington et al. (2014) is named the Global Vectors for Word Representation (GloVe) and represents an extended version of the Word2Vec methodology. Matrix factorization techniques were used to design "classical vector space model representations of words" (Brownlee, 2017). They are proficient at the usage of global text statistics but do not perform that well at capturing the word meaning in comparison to learned methodologies like Word2Vec. With GloVe both worlds - the one of matrix factorization techniques and the one of Word2Vec - can be combined. The principle of GloVe is that an explicit word-context or a word co-occurence matrix is designed. It is based on global statistics, namely statistics across the entire corpus. One receives a learning model that may perform better than other models on several tasks, such as word similarity regarding the quality of word embeddings (Brownlee, 2017; Pennington et al., 2014).

To summarize, the Word2Vec model can be viewed as a predictive model, whereby GloVe represents a count-based model. Both models result in similar performances on downstream tasks (Ghelani, 2019a).

2.3 Language modeling

For the pre-trained word embeddings described in sections 2.1 and 2.2, the assumption that the meaning of a word remains stable across sentences is made, which presents a large limitation of these methods. Furthermore, document representations (section 2.1) perform an inclusion of their learnings only in the first layer of the model, which are referred to as embeddings. Hence, a training

of the remaining network needs to be performed when regarding a new task. Even though word embeddings (section 2.2) have the ability of grasping semantic word meanings, they cannot capture higher level characteristics such as long-term dependencies or negation. This can be attributed to the fact that the learning of word embeddings does not rely on sequential context, but on word concurrency (Ghelani, 2019b).
Before language modeling was performed in order to solve NLP problems, word embeddings were pre-trained on a large corpus, including unlabeled text data, with Word2Vec for example. Then, the first layer of a neural network was initialized, whereby the rest of the network was trained on data of a specific task. In the recent years, a new approach for supervised tasks was taken. Those models which are currently the state-of-the-art perform a pre-training based on language modeling with a subsequent fine-tuning based on labeled task data (Ghelani, 2019b). A detailed exploration of several such model follows in chapter 3

Language modeling can be understood as an unsupervised task with the ability to grasp many language characteristics, such as sentiment of long-term dependencies which are helpful for performing analyses on downstream tasks. The goal of language modeling is the estimation of the joint probability $p(\mathbf{x})$ of a sequence of tokens $\mathbf{x} = (x_1, \ldots, x_T)$. Often, an auto-regressive factorization of the following form is performed:

$$p(\mathbf{x}) = \prod_t p(x_t|\mathbf{x}_{<t}) \tag{6}$$

Hence, the joint probability can be simplified to modeling conditional probabilities (Dai et al., 2019). Generally speaking, there are two types of language models, namely statistical language models and neural language model. Statistical language models include for example n-gram models (Jurafsky et al., 2008). The aim of such models is the prediction of the next token given the prior (n - 1) tokens. It relies on the Markov assumption, i.e. the probability of a token depends only on the previous tokens (Kapadia, 2019). Neural language models (Bengio et al., 2003), on the other hand, apply different types of neural network to model language while simultaneously learning distributed representation for words to represent similarity between words (Dai et al., 2019; Rizvi, 2019).

3 Language models based on Transformer

In the following chapter, we describe methods that were used before Transformer, followed by a elaboration of the Transformer architecture. Subsequently, we illustrate five large pre-trained language models which rely on this architecture, namely BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), DistilBERT (Sanh et al., 2020), XLNet (Yang et al., 2019) and ALBERT (Lan et al., 2019).

3.1 Before Transformer

The present section elucidates methods that were used before Transformers for solving NLP tasks. These include RNNs and the attention mechanism which relies on encoder-decoder RNNs.

3.1.1 Recurrent Neural Networks

Processing NLP tasks such as sequence transduction[3] requires some sort of memory since there are many cases in which words in one sentence are relating to words from prior sentences. To solve this kind of tasks, RNNs (Elman, 1990; Rumelhart et al., 1986) have been used because of their properties. The special characteristic about an RNN is that it memorizes the previous inputs, which enables them to affect the ensuing predictions. This especially comes handy in NLP tasks, for example for predicting the next word in a sentence (Thomas, 2019).
Goldberg (2017) describes RNNs as functions, taking ordered sequences of T vectors $\mathbf{x} = (x_1, \ldots, x_T)$ with $x_t \in \mathbb{R}^{d_{in}}$ as inputs. Passing the input through an RNN results in the output $\mathbf{y} \in \mathbb{R}^{d_{out}}$, which represents a single vector (Goldberg, 2017, p 164). Usually, an RNN is applied, so that

$$h_t = f(h_{t-1}, x_t) \tag{7}$$

applies with $h_t \in \mathbb{R}^{f(d_{out})}$ representing a hidden state at time t and f being a non-linear function (Bahdanau et al., 2014; Goldberg, 2017, p. 164). The definition of RNN is a recursive one and holds for sequences of arbitrary length. The function f

[3] "Many machine learning tasks can be expressed as the transformation-or transduction-of input sequences into output sequences" (Graves, 2012)

stays the same across the sequence positions and a parameter sharing is performed across all time stamps since the same parameters are applied in every step. By applying RNNs, we do not need to rely on the Markov assumption that was used for modeling sequences with n-gram models. This is possible since the usage of RNNs enables a conditioning on $(x_1, \ldots, x_T)$. Note that RNNs also achieve high perplexity values[4] in comparison to n-gram models (Goldberg, 2017, p. 165).

In some NLP tasks, such as machine translation, not only regarding the past words x_{t-1}, but also having a look at the following words $(x_{t+1}, \ldots, x_T)$ may be of use for predicting x_t. The introduction of bidirectional RNN (biRNN) (Schuster et al., 1997) enables viewing past as well as future words of the sequence. Hereby, a biRNN contains two separate hidden states, namely the forward state h_t^f and the backward state h_t^b. Both are generated by a two RNNs, which are based on $(x_1, \ldots, x_t)$ and $(x_t, \ldots, x_1)$, respectively. The output y_t then forms a concatenation of the two RNNs at every position (Goldberg, 2017, p. 169).

Note that if the regarded context and the distance between the required information and the point where one wants to use it is large, RNNs become ineffective., i.e. it suffers from short-term memory. This can be attributed to the fact that the information is passed at each step and that the probability of losing information increases with the increasing chain length. While in theory this long-term dependencies could be learned, RNNs seem to not learn them in practice. Furthermore, RNNs do not distinguish between important and unimportant information (Giacaglia, 2019).

An approach to solve this issue is the usage of Long Short-Term Memory (LSTM) (Hochreiter et al., 1997) or Gated Recurrent Unit (GRU) (Cho et al., 2014a), each representing a special type of RNN. With information passing through so-called gates, a selective memorization or oblivion of important or less important information can be performed. Nonetheless, the drawbacks of RNNs also apply to LSTMs and GRUs, i.e. the performance of LSTMs and GRUs suffers if sentences are too long, since the probability of keeping the context of a word exhibits an exponential decline with the distance of the context from the word that is

[4] perplexity quantifies how well a probability model or distribution is predicting a sample (Wikipedia, the free encyclopedia, 2020f)

regarded (Phi, 2019; Giacaglia, 2019). Moreover, a parallelization of the process is hardly possible for RNNs, LSTMs and GRUs since sequential computation (word by word) is carried out. The last drawback that should be addressed is that long and short range dependencies are not explicitly modeled (Giacaglia, 2019).

An approach to overcome these issues is to use the attention mechanism within neural networks, specifically by applying encoder-decoders (Giacaglia, 2019).

The majority of models used in machine translation are part of a family of encoder-decoders (Cho et al., 2014a; Sutskever et al., 2014), also referred to as Sequence-to-Sequence (seq2seq) (Cho et al., 2014b). Generally speaking, the task of the encoder, which commonly is an RNN, is encoding the input sequence into a context vector c (Goldberg, 2017; Bahdanau et al., 2014). Usually, the task of the decoder is the prediction of the next word y_t on the basis of the context vector c and the prior word predictions $\{y_1, \ldots, y_{t-1}\}$. Since the decoder uses a language modeling objective, it can be formulated as a probability over translation $\mathbf{y}$ of the form:

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t | \{y_1, \ldots, y_{t-1}\}, c). \tag{8}$$

When applying an RNN, the conditional probabilities are each of the form

$$p(y_t | \{y_1, \ldots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c) \tag{9}$$

with g being a nonlinear function and s_t noting the hidden state of the RNN (Bahdanau et al., 2014). The encoder and the decoder are then jointly trained with the goal of maximizing the probability of a correct translation given a source sentence. Note that the encoder-decoder is helpful for mapping sequences of different lengths (Goldberg, 2017; Bahdanau et al., 2014).

3.1.2 Attention

Facing the problem of performance decline of basic encoder-decoder models for longer inputs, Bahdanau et al. (2014) present a new approach, namely an extension to the encoder-decoder. In contrast to the traditional encoder-decoder

implementation, the authors perform an adaptive choice of subsets of those vectors during the decoding. Hence, the model does not have to squeeze all the information of an input sentence into a vector of a fixed length, like in the basic encoder-decoder model and results in a performance improvement, especially for longer input sentences (Bahdanau et al., 2014).

The model architecture for neural machine translation introduced by Bahdanau et al. (2014) incorporates an encoder and a decoder. The encoder is a bi-RNN since the authors wish that the annotations of a word not only summarize the words occurring before the regarded word (like in the basic RNN), but also the subsequent ones. Furthermore, the decoder in this architecture searches through an input sentence while performing a decoding of a translation. Hereby, each conditional probability in equation 9 is of the form

$$p(y_i|\{y_1,\ldots,y_{i-1}\},\mathbf{x}) = g(y_{i-1},s_i,c_i) \tag{10}$$

with s_i being an RNN hidden state for time i:

$$h_i = f(s_{i-1},y_{i-1},c_i). \tag{11}$$

Here, a conditioning of the probability on the context vector c_i which depends on a sequence of annotations $(h_1,\ldots,h_T)$ generated by the encoder is performed. The h_is incorporate information about the whole input sequence, while focusing strongly on parts which lie in the surrounding of the i-th input word. The context vector c_i can be represented as the weighted sum of the annotations h_j:

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j. \tag{12}$$

The computation of the weight α_{ij} takes on the form

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T}\exp(e_{ik})} \tag{13}$$

with $e_{ij} = a(r_{i-1},h_j)$ being an alignment model which takes the hidden state of y_i, namely s_{i-1}, and the j-th annotation h_j into consideration. The goal of the model is to measure the matching between the inputs around position j and the output

at position i. The alignment model a is defined as a feed-forward neural network model and the training is carried out jointly with the rest of the model. α_{ij} can be interpreted as the probability of the output y_i being aligned to (translated from) the input word x_j. Note that α_{ij} can be interpreted as "the importance of the annotation h_j with respect to the previous hidden state s_{i-1} in deciding the next state s_i and generating y_i" (Bahdanau et al., 2014, p. 4). With this, a so-called attention mechanism is integrated in the decoder. By the implementation of this mechanism, the encoder is not forced to encode all information of the input sentence into a vector of fixed length. A spreading of information throughout the sequence of annotations can be performed. This information can be selectively acquired (Bahdanau et al., 2014).

Some mentioned drawbacks could still not be solved with the help of RNNs combined with attention mechanism, e.g. the parallelization of input processing, which increases the training time. To solve this issue, Transformers were introduced which enable a parallelization of the regarding computations (Giacaglia, 2019). This is described in the following section.

3.2 Transformer

With the introduction of an increasing amount of language models in the past few years, the model architecture Transformer stands out in particular[5]. Model variants of Transformer form the current state-of-the-art models, achieving impressive performance results on a wide range of NLP tasks. This model architecture solely relies on an attention mechanism with which global dependencies between the input and the output can be created. Apart from solely relying on an attention mechanism, the special feature about Transformer is that significantly more parallelization is possible and new state-of-the-art results in machine translation were reached in 2017 (Ashish Vaswani et al., 2017).

[5]the implementation is available at https://github.com/tensorflow/tensor2tensor

3.2.1 Model architecture

The majority of transduction models is based on an encoder-decoder structure. Consequently, the model is auto-regressive at each step, i.e. when creating the next symbol, Transformer uses the symbols that were previously created as additional input. But in contrast to the encoder-decoder models described earlier, no recurrent networks are used.

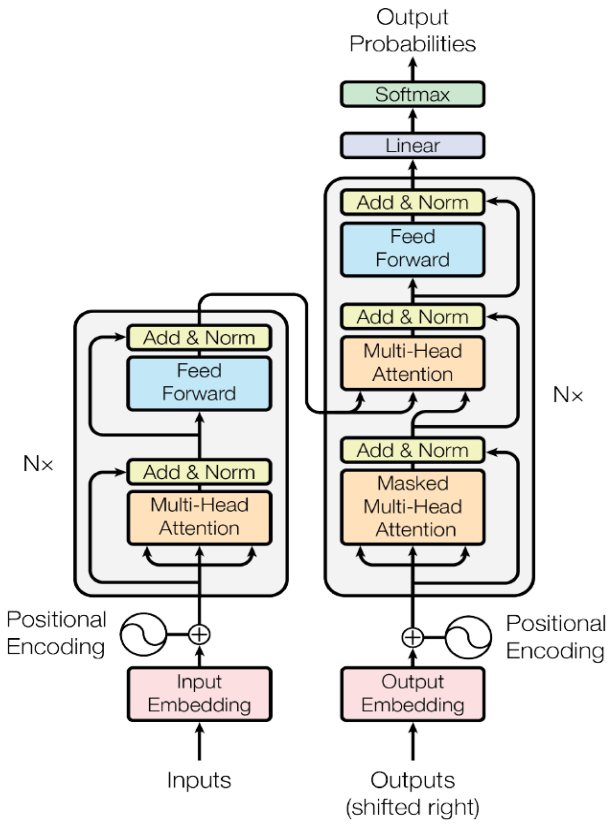

Figure 2: The Transformer - Model architecture (Ashish Vaswani et al., 2017, p. 3)

The encoder consists of a stack of six identical layers ($N = 6$). Each encoder layer is composed of two sub-layers, namely a multi-head self-attention mechanism and a simple feed-forward network that is position-wise fully connected. Additionally, a residual connection and afterwards a layer normalization are inserted around each sub-layer (see figure 2). The output of each sub-layer can therefore be described as $LayerNorm(x + Sublayer(x))$. In order to make the residual

connections easier, all sub-layers and the embedding layers generate outputs that have a dimensionality of $d_{model} = 512$.
Like the encoder, the decoder also consists of six identical layers. Each decoder layer incorporates the two encoder sub-layers and a third sub-layer, in which multi-head attention is carried out over the encoder stack. Residual connections and layer normalization are inserted in a similar manner as in the encoder. In order to avoid that positions attend subsequent positions, the self-attention sub-layer within the decoder is modified. Because of this modification (also referred to as masking) and the output embeddings being shifted by one position, one can state that the predictions for position i only depend on the output at positions $< i$ (Ashish Vaswani et al., 2017).

As strings cannot be applied directly to the model, the first step is to embed the input and output tokens (with dimension d_{model}) by applying learned embeddings. Ashish Vaswani et al. (2017) performed a conversion of the decoder output to predicted next-token probabilities by applying a linear transformation and a softmax function[6]. Note that the same weight matrix is shared between two embedding layers and the linear transformation before applying the softmax.

Unlike RNNs or CNNs, the Transformer does not incorporate recurrence or convolution. Consequently, to avail oneself of the order of the sequence, so-called positional encodings were added at the bottoms of the decoder and encoder stacks in the input embeddings. Their dimension equals the one of the embeddings (d_{model}) to enable an addition of both. Ashish Vaswani et al. (2017) made use of sine as well as cosine functions of different frequencies which can be expressed as

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{14}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right). \tag{15}$$

.

Equation 14 can be interpreted as each dimension i of PE complying with a

[6] the softmax is defined as $\text{softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$ (Goldberg, 2017)

sinusoid, with *pos* being the position. The authors applied the displayed function because they assumed "it would allow the model to easily learn to attend by relative positions, since for any offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}" (Ashish Vaswani et al., 2017, p. 6). Moreover, they used the sinusoidal version "because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training" (Ashish Vaswani et al., 2017, p. 6).

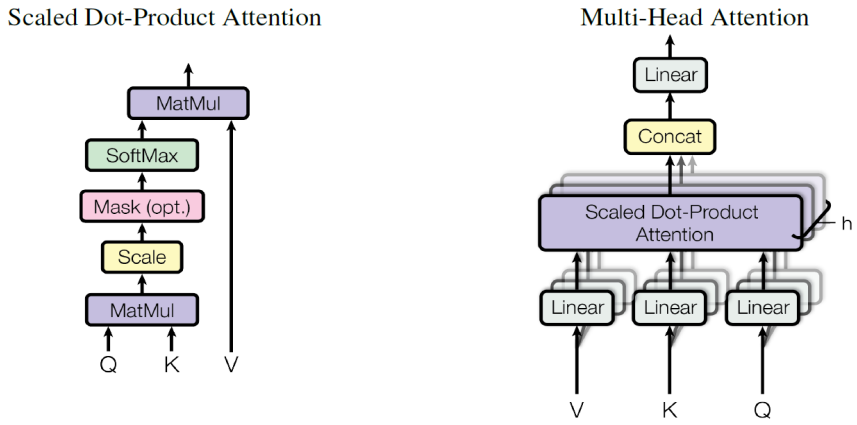

Figure 3: Left: Scaled Dot-Product Attention; Right: Multi-Head Attention consists of several attention layers running in parallel (Ashish Vaswani et al., 2017, p. 4).

Ashish Vaswani et al. (2017) refer to the attention implemented in their work as the "Scaled Dot-Product Attention" (see left side of figure 3). Hereby, queries and keys with the dimensions d_k and d_v form the input. A set of queries is quantified by the matrix $\mathbf{Q}$, whereas a query can be described as a vector representation of one word in the sequence. Keys that are represented by $\mathbf{K}$ are "vector representations of all the words in the sequence" (Allard, 2019) and $\mathbf{V}$ are values that can be defined in the same way as $\mathbf{K}$. The output matrix (also referred to as Attention) is computed as follows:

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^{\mathbf{T}}}{\sqrt{d_k}}\mathbf{V}\right). \tag{16}$$

Ashish Vaswani et al. (2017) hypothesize that when regarding large key dimen-

sionality d_k, the dot product takes on large values, which results in a softmax function with very small gradients. To prevent this from happening, the scale factor $\frac{1}{\sqrt{d_k}}$ was added.

Furthermore, multi-head attention is applied in the Transformer architecture (as can be seen in figure 2). This means that keys, values and queries are projected h times while using different learned linear projections. In the next step, the attention function is applied in parallel to each of the projected versions of keys, values and queries. After performing a concatenation and a projection, we receive the values of the multi-head attention. The above described procedure is visualized on the right side of figure 3. The special characteristic of multi-head attention in comparison to a single attention head is that joint attendance "to information from different representation subspaces at different positions" is possible (Ashish Vaswani et al., 2017, p. 4). The authors formulate multi-head attention as

$$\begin{aligned} \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_\text{h})\mathbf{W^O} \\ \text{where} \quad \text{head}_\text{i} &= \text{Attention}(\mathbf{QW_i^Q}, \mathbf{KW_i^K}, \mathbf{VW_i^V}). \end{aligned} \tag{17}$$

Hereby, the projections are defined as the parameter matrices $\mathbf{W_i^K} \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W_i^V} \in \mathbb{R}^{d_{model} \times d_v}$, $\mathbf{W_i^Q} \in \mathbb{R}^{d_{model} \times d_k}$ and $\mathbf{W^O} \in \mathbb{R}^{hd_v \times d_{model}}$. The authors incorporated 8 attention layers or heads, where $d_k = d_v = d_{model}/8 = 64$ is applied for each head. Since a dimensionality reduction was performed on each head, the computational costs resemble those of single-head attention with non-reduced dimensions (Ashish Vaswani et al., 2017).

The multi-head attention is incorporated in three different places in the Transformer model. First, the encoder-decoder attention layers contain queries that origin from the prior decoder layer, whereas the keys and values stem from the output of the encoder. Therefore, all positions in the decoder have the permission to attend over all input sequence positions. Furthermore, self-attention layers which can also be found in the encoder, contain queries, keys and values that all stem from the output of the prior encoder layer. Hence, all positions of the encoder are allowed to attend all positions from the prior encoder layer. Lastly, the self-attention layers in the decoder follow a similar principle as the

ones in the encoder, except for the fact that the positions are only permitted to attend each position up to (including) the position that is currently regarded. To maintain the auto-regressive property of the decoder (and hence avoid left-side information flow), a masking of input values of the softmax complying with undesired connections is carried out. This masking equals setting the values to $-\infty$ (Ashish Vaswani et al., 2017).

A fully connected feed-forward network can be found in all of the layers in the encoder and decoder which is connected by a ReLU activation function[7]. The application is carried out in an identical and separate manner to each position with x as the input.

$$\mathrm{FNN}(\mathbf{x}) = \max(\mathbf{0}; \mathbf{x}\mathbf{W_1} + \mathbf{b_1})\mathbf{W_2} + \mathbf{b_2}. \tag{18}$$

Note that the linear transformations are equal to each other across the various positions, whereas the parameters of each layer differ from each other. While $d_{model} = 512$ holds, the dimensionality of the inner layers is $d_{ff} = 2048$.

Ashish Vaswani et al. (2017) point out three main advantages of self-attention compared to recurrent or convolutional layers. The first advantage they address is that the connection of all positions is carried out with a constant number of sequential operations, whereby the recurrent layer is in need of $O(n)$ of those. Hence, the computational complexity of self-attention is lower compared to recurrent layers if $n < d$ (n: sequence length; d: representation dimensionality) applies. When regarding sentence representations that are utilized by state-of-the-art models in machine translation, this usually applies. For long sequences, the computational complexity of self-attention could be ameliorated by making the restriction to only take an r-sized neighborhood in the input sequence into account. This way, the maximum path length would be reduced to $O(n/r)$.
To connect all pairs of input and output positions, $O(n/k)$ convolutional layers are needed when regarding contiguous kernels (for $k < n$), which extends the maximum path length between any two positions within the network. Note that the

[7]The rectified linear unit (ReLU), also known as the rectifier activation function, takes on the form $\mathrm{ReLU}(x) = \max(0, x)$ (Agarap, 2018, p. 45f))

shorter the paths are, the easier the learning of long-range dependencies is. Even though convolutional layers have higher computational expenses than recurrent layers, separable convolutions have a much lower complexity. Nonetheless, even when considering $k = n$, the complexity equals a self-attention layer combined with a point-wise feed-forward layer, which is applied by the authors.
Additionally, the self-attention could provide models that have a higher interpretability (Ashish Vaswani et al., 2017).

3.2.2 Training

In the scope of this scientific work, the Transformer should solve two tasks, namely an English-to-German and an English-to-French translation task. For the English-German training of the Transformer, an encoding of the sentences with byte-pair encoding (BPE) whose source-target vocabulary contains approx. 37,000 tokens was performed (Ashish Vaswani et al., 2017)[8]. The English-French training made use of the WordPiece embeddings (Wu et al., 2016) with 32,000 tokens[9].
For the training of the Transformer, one machine with 8 NVIDIA P100 GPUs was used. The base models were trained for 12 hours, whereby the training step time was ca. 0.4 seconds. For the large models, the training step took about 1.0 seconds, resulting in a total training time of 3.5 days (Ashish Vaswani et al., 2017).

Within the Transformer architecture the Adam optimizer with the hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$ was used[10]. During training the learning rate was increased for the warmup training steps in a linear way. Later, it was decreased proportionally to $\frac{1}{\sqrt{step\_num}}$.

To prevent the model from overfitting, the authors implemented the regularization types residual dropout and label smoothing during training. Dropout[11] is applied

[8] For more insights on BPE, see appendix, section 8

[9] For more insights on WordPiece, see appendix, section 8

[10] The Adam optimizer is "an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments" (Kingma et al., 2014). For more details and the pseudo-code, see Kingma et al. (2014).

[11] Within the dropout methodology neurons (incl. their connections) are randomly dropped

to the output of every sub-layer, whereupon an addition to the sub-layer input and a normalization is carried out. Furthermore, it is applied to the embedding sums as well as the positional encodings in the decoder and encoder. The dropout rate for the base model is 0.1. For label smoothing[12] a value of $\epsilon_{ls} = 0.1$ was used. As a consequence, the uncertainty of the model increases while simultaneously showing an improvement in accuracy and BLEU score [13].

All in all, the Transformer models achieved better results than the previous state-of-the-art models, such as ByteNet (Kalchbrenner et al., 2016) or ConvS2S (Gehring et al., 2017) on machine translation tasks, e.g. WMT 2014 English-to-German translation task. These results are achieved with only a fraction of the training costs the competitive single models need.

3.3 BERT

The concept of pre-training a language model on a unlabeled text corpus with a subsequent supervised fine-tuning on a specific task was introduced by Radford et al. (2018). It represents the predecessor of a wide range of large pre-trained language models. One of them forms the basis of three of the following models (RoBERTa, DistilBERT and ALBERT) and is referred to as the Bidirectional Encoder Representations from Transformers (BERT). Devlin et al. (2019) describe it as a language representation model where the goal is to "pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers" (Devlin et al., 2019, p. 1). After the pre-training, a fine-tuning of the model with only one ancillary layer is possible. With this model, a state-of-the-art performance on a large variety of tasks, without crucially changing the architecture depending on the task, can be achieved. BERT outperforms the former introduced models on eleven NLP tasks.

The architecture of BERT consists of two main parts, namely pre-training and

from the network during training time (Srivastava et al., 2014)

[12] Hereby, the classifier layer is regularized by performing an estimation of the "marginalized effect of label-dropout during training" (Szegedy et al., 2015)

[13] the BLEU score is applied for evaluating the quality of text that was translated by a model from one natural language into another (Wikipedia, the free encyclopedia, 2020a)

fine-tuning. The pre-training is performed on the basis of unlabeled data, whereby different pre-training tasks are taken into account. The first step in fine-tuning is to initialize the model with the already pre-trained parameters. Thereupon, a fine-tuning of all parameters is carried out, whereas labeled data from downstream tasks is used. For each downstream task, we receive a separate fine-tuned model despite the fact that the initialized parameters are the same for every model.

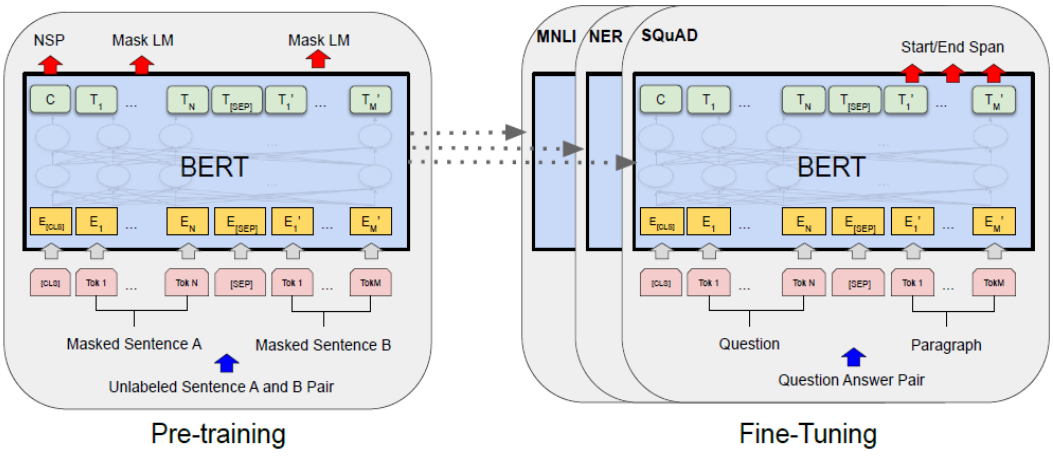

Figure 4: Pre-training and fine-tuning procedures for BERT. Question-Answering example (Devlin et al., 2019)

Figure 4 displays the two procedures for BERT. In the following, this example is referred to several times. The unified architecture of BERT across diverse tasks is a notable feature of BERT. Note that the architecture used for pre-training and the one used for fine-tuning differ only in the output-layers.

"BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described in Ashish Vaswani et al., 2017 and released in the tensor2tensor library[14]" (Devlin et al., 2019, p. 3).
In the scientific work from Devlin et al. (2019) the number of layers which represents the Transformer blocks is symbolized with L, H refers to the hidden size and A to the number of self-attention heads. Furthermore, it is noted that

[14]see https://github.com/tensorflow/tensor2tensor

the feed-forward/filter size is specified as $4H$. The results are presented for the two models $\text{BERT}_{\text{BASE}}$ with the parameters $L = 12$, $H = 768$, $A = 12$ and a total number of 110 mil. parameters and $\text{BERT}_{\text{LARGE}}$ with $L = 24$, $H = 1024$ and $A = 16$ and a total amount of 304 mil. parameters.
To draw a good comparison, $\text{BERT}_{\text{BASE}}$'s model size was set to the same model size as OpenAI GPT (Radford et al., 2018). Nevertheless, when comparing those two models it should be noted that the BERT Transformer is based on bidirectional self-attention, whereas constrained self-attention, which only enables attendance to its left, is used in the GPT Transformer.

A single sentence, i.e. "an arbitrary span of contiguous text", as well as sentence pairs, e.g. question-answer pairs, can be unequivocally represented in one token sequence by the input representation. Hereby, the goal is to enable the processing of a wide range of down-stream tasks for BERT. Furthermore, WordPiece embeddings (Wu et al., 2016) with a vocabulary of 30,000 tokens are used. Each representation always starts with the special classification token `[CLS]`. For classification tasks, the associated final hidden state is then used as the sequence representation. To distinguish the sentences of a sentence pair, they are separated with the token `[SEP]` in the first step. Afterwards, a learned embedding is added to every token, pointing out in which sentence it is contained (`A` or `B`). The input embedding is referred to as E, `[CLS]`'s final hidden vector as $C \in \mathbb{R}^H$ and the final hidden vector corresponding to the i-th input token as $T_i \in \mathbb{R}^H$ (see figure 4). Given a token, we can form the input representation by summing the given token, segment as well as position embeddings. This is visualized in the following figure 5.

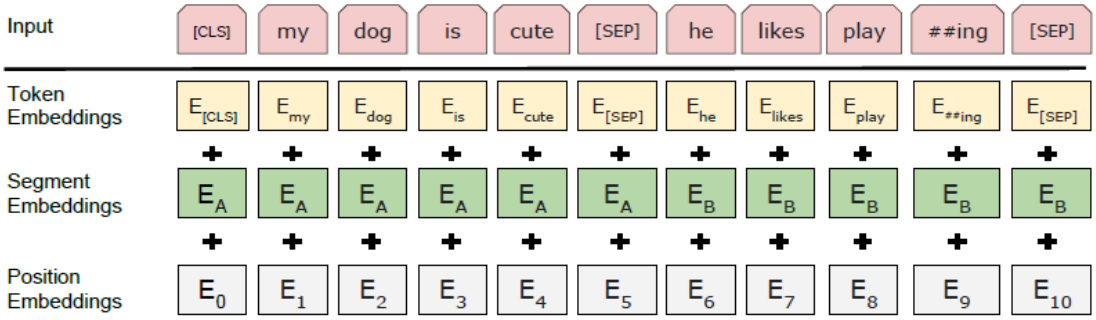

Figure 5: BERT input representation (Devlin et al., 2019)

3.3.1 Pre-training BERT

For pre-training BERT two unsupervised tasks are used which are visualized on the left side of figure 4. The first task is called Masked Language Model (MLM) task or alternatively Cloze task (Taylor, 1953). In order to enable training a deep bidirectional model without allowing each word to "view itself" and therefore simply learn a copy and paste technique, Devlin et al. (2019) perform a random masking of 15% of all tokens in each input sequence and a subsequent prediction of the masked tokens. Hereby, the final hidden layers complying with the mask tokens `[MASK]` serve as the input to an output softmax over the vocabulary. Note that before performing the masking, WordPiece tokenization is carried out. A drawback of this method is the generation of a discrepancy between the pre-training and the fine-tuning because the `[MASK]` token is not present during the fine-tuning. In order to reduce it, words to be masked are only substituted with `[MASK]` in 80% of the cases and in 10% of the time with a random token. The remaining 10% do not undergo a change. T_i is then used for predicting the original token with cross entropy[15], assuming the i-th token was chosen to be masked.

The second pre-training task is the Next Sentence Prediction (NSP) task and has

[15] Koech (2020) formulate the cross entropy as $L_{CE} = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$ with y_i as the true label and $\hat{y}_i$ as the softmax probability for the ith class. It "measures the dissimilarity between the true label distribution $\mathbf{y}$ and the predicted label distribution $\hat{\mathbf{y}}$" (Goldberg, 2017, p. 27)

the goal of training a model which comprehends sentence relationships between two sentences. This task can be easily created on the basis of a monolingual corpus. If sentences `A` and `B` are chosen for each example in the pre-training, 50% of the cases already contain examples in which `B` represents the next sentence following `A` and is labeled as `IsNext`. In the remaining 50%, the next sentence is a random sentence from the corpus which is then labeled as `NotNext`. For predicting the next sentence, C is used (see figure 4). The sampling of both sentences is performed in such a way that maximum sequence length resembles 512 tokens, whereby a batch size of 256 sequences was selected (resulting in 128,000 tokens per batch).

The model was trained for a million steps, which equals about 40 epochs. Moreover, Adam optimization was used with a "learning rate of 1-e4, $\beta_1 = 0.9$, $\beta_2 = 0.999$, learning rate warmup over the first 10,000 steps, and linear decay of the learning rate" (Devlin et al., 2019, appendix A.2). Furthermore, the authors applied a dropout probability of 0.1 to all layers for regularization purposes as well as a GeLU activation function[16]. The pre-training time for $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$ amounts to four days, whereby the prior was trained on four and the latter on 16 Cloud TPUs. Since longer sequences are way less cheap in a computational sense than shorter ones, the model was pre-trained with a sequence length of 128 in 90% of the steps. For the remaining 10%, Devlin et al. (2019) used a sequence length of 512 with the goal of learning the positional embeddings (Devlin et al., 2019). The pre-training corpus consists of the BooksCorpus with 800 mil. words (Zhu et al., 2015) as well as English Wikipedia with 2,500 mil. words, for which only Wikipedia text passages are regarded (Devlin et al., 2019).

3.3.2 Fine-tuning BERT

Due to the self-attention methodology used in the Transformer, BERT is capable of modeling various downstream tasks by swapping the corresponding inputs and outputs, which makes fine-tuning easy. This procedure involves feeding BERT

[16] Gaussian Error Linear Unit (GeLU) is defined as $\text{GeLU}(x) = xP(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2}(1+\text{erf}(\frac{x}{\sqrt{2}})) \approx 0.5x(1+\tanh(2\pi(x+0.044715x^3)))$ with erf as the error function (Hendrycks et al., 2016)

with inputs and outputs of the respective task and fine-tuning all parameters end-to-end.

The token representations are plugged into an output layer for tasks on token-level, e.g. question answering or sequence tagging, whereby the `[CLS]` representation serves as an input to an output classification layer, e.g. sentiment analysis or entailment. Note that fine-tuning a model is relatively "cheap" in a computational sense compared to pre-training (Devlin et al., 2019).

Most of the hyperparameter values of the model are taken over from the pre-training, except for the learning rate, number of training epochs and the batch size. Devlin et al. (2019) note that the optimal values for the mentioned hyperparameters depend on the task, whereby a certain range of values can result in a good performance. For the batch size, possible candidates for optimal values are 16 and 32, for the learning rate 2e-5, 3e-5 and 5e-5 and for the number of epochs 2, 3 and 4. Furthermore, the authors mention that small datasets are way less robust towards hyperparameter changes than large datasets that contain over 100,000 representations for example.

3.3.3 Experiments

Devlin et al. (2019) carried out a fine-tuning of BERT on 11 NLP tasks which are described in the following.

The General Language Understanding (GLUE) benchmark (Wang et al., 2019) represents a compilation of nine natural language understanding (NLU) tasks. We present a more in-depth description of the respective tasks in chapter 4. For conducting a fine-tuning on GLUE, the input sequence is represented as described above. Furthermore, $C \in \mathbb{R}^H$ - the final hidden vector - complying to `[CLS]` is used as the aggregated representation. Hence, the classification layer weights $W \in \mathbb{R}^{K_{label} \times H}$ with K_{label} being the number of labels, are the only new parameters initiated during fine-tuning. Furthermore, the authors compute a standard classification loss $\log(\text{softmax}(CW^T))$.

For each GLUE task, the fine-tuning is performed for 3 epochs over the data with a batch size of 32. The best fine-tuning learning rate was chosen on the development set for each task and can take on a value in {2e-5, 3e-5, 4e-5, 5e-5}. In addition, the fine-tuning based on small datasets can be volatile for $\text{BERT}_{\text{LARGE}}$.

Therefore, several random restarts were conducted, and the best model based on the development set was selected.

Devlin et al. (2019)'s results show that $\text{BERT}_{\text{BASE}}$ as well as $\text{BERT}_{\text{LARGE}}$ perform better than all regarded systems on all GLUE tasks by a considerable span and improve the average accuracy of OpenAI GPT (Radford et al., 2018), the former state-of-the-art. Interesting to note is the fact that the architecture of OpenAI GPT's and $\text{BERT}_{\text{BASE}}$ resemble each other except for the attention masking. Furthermore, the authors underline $\text{BERT}_{\text{LARGE}}$'s performance steadily being better than the one of $\text{BERT}_{\text{BASE}}$ across every GLUE task. This applies especially for GLUE tasks with small training datasets.

The authors additionally performed a fine-tuning experiment on the basis of the Stanford Question Answering Dataset (SQuAD v1.1), which includes 100,000 crowdsourced question/answer pairs (Rajpurkar et al., 2016a) and on SQuAD v2.0 which is an extension of the prior one. Both experiments show that the BERT models outperform the former state-of-the-art. The last fine-tuning experiment undertaken by Devlin et al. (2019) is fine-tuning BERT based on the dataset Situations With Adversarial Generations (SWAG) (Zellers et al., 2018) which includes 113,000 examples of sentence-pair completion that have the goal of evaluating so-called "grounded common sense inference". Again, BERT outperforms the former state-of-the-art models.

3.4 RoBERTa

Large pre-trained language models such as XLM (Lample et al., 2019), GPT (Radford et al., 2018), BERT (Devlin et al., 2019) or XLNet (Yang et al., 2019) contributed to a significant performance improvement. However, it turns out to be difficult to identify the most important influencing factors of the method due to their expensive training and the usage of private training data. Therefore, Liu et al. (2019) conduct a replication study of the BERT pre-training, where they investigate certain modifications, being changes to the training time and batch size, elimination of the NSP task, using longer sentences for training and modifying the masking pattern.

Due to the discovery of BERT being undertrained, a new method for training

BERT was suggested which is named RoBERTa [17]. These new changes to BERT result in a new state-of-the-art on four out of nine GLUE tasks and comply with the SQuAD and RACE state-of-the-art results when controlling for training data.

3.4.1 Experiments

Liu et al. (2019) mostly apply BERT's optimization hyperparameters with the exception of the peak learning rate and the number of warmup steps. Those are being tuned for each setting separately. Furthermore, a tuning of Adam's epsilon improved the performance in some cases, whereby using $\beta_2 = 0.98$ led to a stability improvement when using large batch sizes. Contrarily to the original implementation of BERT (Devlin et al., 2019), RoBERTa is only trained on sequences with the full length, whereas the maximum sequence length is 512 tokens (Liu et al., 2019).

As shown in Baevski et al. (2019), the increase of data size can lead to an improvement of the performance of downstream-tasks. Hence, Liu et al. (2019) collected as much data as possible which enables adapting the qualitative and quantitative aspects of data in an appropriate way for each comparison. Additional to the BooksCorpus and English Wikipedia used in the BERT pre-training, the Commom Crawl News data set (CC-NEWs) (Nagel, 2016), OpenWebText (Gokaslan et al., 2019) which contains open-source web text, and STORIES (Trinh et al., 2018) which includes a subset of CC-NEWS with a story-like schema, were used for pre-training RoBERTa. As can already be seen, RoBERTa uses much more data for pre-training than BERT. To be exact, BERT only uses 16GB, whereby RoBERTa uses 161GB which is ten times more data.

The pre-trained models are evaluated on downstream tasks of the benchmarks GLUE (Wang et al., 2019), SQuAD (Rajpurkar et al., 2016a) and ReAding Comprehension Dataset From Examinations (RACE) (Lai et al., 2017).

In the following, the factors which could influence BERT's pre-training are de-

[17] code and models can be found at https://github.com/pytorch/fairseq/tree/master/examples/roberta

scribed. Hereby, the model architecture is kept fixed. Liu et al. (2019) start with training the models with the same hyperparameter settings as $\text{BERT}_{\text{BASE}}$.

3.4.2 Training Procedure Analysis

First, BERT's MLM objective is regarded. BERT's architecture relies on a single static mask, whereby the masking itself is performed once during the data preprocessing. A duplication of the training data was performed ten times which leads to each sequence being masked in ten different ways over 40 training epochs. This means that each sequence has been viewed four times with the same mask during training. Liu et al. (2019) draw a comparison between this strategy and the dynamic masking. In the latter, a masking pattern is produced each time a sequence is given to the model. Especially for pre-training with more steps or more data, this gains importance. The dynamic masking matches or slightly improves the performance of RoBERTa's static re-implementation of BERT. Hence, the dynamic masking is used in their further analysis.

Even though Devlin et al. (2019) state that the deletion of the NSP task leads to a performance deterioration on some GLUE tasks and SQuAD V1.1, the imperative of this task was put into question (Lample et al., 2019; Yang et al., 2019; Joshi et al., 2019). To get to the bottom of the two diverging statements, the following four training techniques are regarded.
`SEGMENT-PAIR+NSP` is implemented according to BERT's input format (Devlin et al., 2019) and contains the NSP loss, i.e. the inputs contain segment pairs, whereas each can consist of multiple sentences. The combined length cannot exceed 512 tokens.
In `SENTENCE-PAIR+NSP`, natural sentence pairs which are sampled contiguously from a portion of one or more documents are regarded. Because they are way shorter than 512 tokens, the batch size is increased in order to match the total number of tokens of `SEGMENT-PAIR+NSP`. The NSP loss is also taken into consideration.
The third training format that is analyzed is called `FULL-SENTENCES`. Here, the contiguous sampling of full sentences from either one or several documents is carried out. The total length should not exceed 512 tokens. When the end of a

document is reached, a sampling of sentences from the next document starts and additional separator tokens are added between documents. In this scenario the NSP loss is eliminated.
The last considered input format is called `DOC-SENTENCES`. The input resembles `FULL-SENTENCES` with the exception that a crossing of the document boundaries is not permitted. The sequence length of inputs sampled near to the end of a document is often smaller than 512 tokens. Hence, a dynamical increase of the batch size is applied in order to approximately match the number of tokens of `FULL-SENTENCES`. The NSP loss is not regarded in this case (Liu et al., 2019).

Firstly, the authors draw a comparison between the performance of BERT's input format, `SEGMENT-PAIR`, and the format `SENTENCE-PAIR`. The evaluation shows that the usage of individual sentences has a negative effect on the performance of downstream tasks.Liu et al. (2019) hypothesize that this negative effect is caused by the model not being able to learn long-range dependencies in this case. Furthermore, applying `DOC-SENTENCES` results in a better performance than $\text{BERT}_{\text{BASE}}$ and the removal of the NSP loss exhibits the same or slightly better downstream task performance. The last finding in this experiment is that the constraint for a sequence to originate from one document (`DOC-SENTENCES`) shows a slightly better performance than regarding sequences from multiple documents (`FULL-SENTENCES`). Nonetheless, to facilitate the comparison to related work, the authors decide on using `FULL-SENTENCES` in the remaining analysis.

Recent work has shown that application of very large mini-batches can lead to faster optimization and better performance in the BERT training (You et al., 2019). By using gradient accumulation[18], BERT's training of one million steps with a batch size of 256 sequences can be transformed into a training of 125,000 steps with a batch size of 2,000 sequences or 31,000 steps with a batch size of 8,000. The results in Liu et al. (2019) show that the increase of batch sizes leads to an improvement of perplexity for the MLM objective and of the end-task accuracy. Another advantage of larger batches is that their parallelization is easier. The authors further use a batch size of 8,000.

[18] a mechanism where batches are splitted into mini-batches that will be run sequentially (for more details see Haleva (2020))

Furthermore, the authors regard Byte-Pair-Encoding (BPE) which can handle large vocabularies. Subword units which are determined by statistically analyzing the training corpus form the basis of BPE. Its vocabulary size usually varies from 10,000 to 100,000 subword units, whereby unicode characters can make up a large part of the vocabulary. In the scientific work of Radford et al. (2019) an implementation of BPE is presented in which bytes form the base subword units. The usage of bytes leads to a subword vocabulary of smaller size (50,000 units), whereby an encoding is still possible without the application of "unknown" tokens. BERT's implementation WordPiece (Wu et al., 2016) makes use of a character-level BPE vocabulary with 30,000 tokens, which requires an input preprocessing beforehand. In contrary, the byte-level BPE vocabulary containing 50,000 subword units which is used in Liu et al. (2019) does not require previous tokenization or preprocessing. Even though the byte-level BPE performs slightly worse on some tasks than the character-level BPE, the authors decided on using the byte-level BPE for further experiments because in their opinion the advantages of this method, namely being a universal encoding scheme, predominate the small performance decline.

3.4.3 Evaluation of RoBERTa

The previously described modifications of the implementation of BERT are aggregated in the following. More precisely, a training based on dynamic masking with the application of `FULL-SENTENCES`, without the usage of an NSP loss is performed with large mini-batches and a larger byte-level BPE. This configuration carries the name Robustly optimized BERT approach (RoBERTa). In order to be able to distinguish between the importance of the two addressed aspects and modeling choices, Liu et al. (2019) start their investigation with training RoBERTa according to $\text{BERT}_{\text{LARGE}}$'s architecture. The pre-training is performed for 100,000 steps on the basis of a similar book corpus and Wikipedia dataset. The pre-training was carried out on 1024 V100 GPUs for about one day.

RoBERTa outperforms $\text{BERT}_{\text{LARGE}}$, when controlling for training data. This underlines the importance of the design choices that were analyzed in the section above. In the next step, a training of RoBERTa is performed based on this

data and the datasets CC-NEWS, OPENWEBTEXT and STORIES, keeping the same number of training steps. The performance continues to improve across all downstream tasks which emphasizes the relevance of data diversity as well as size within the pre-training. In the last step, the duration of the pre-training is set to be much longer, implicating an increase of training steps to 300,000 and then to 500,000. Again, a large performance increase can be seen, whereby the models with 300,000 and with 500,000 steps even outperform $\text{XLNet}_{\text{LARGE}}$ for the majority of tasks. The authors state that even the models that were trained the longest do not seem to overfit the data. For the remaining analysis on the three benchmarks, the best RoBERTa model, i.e. the one which was trained for 500,000 steps based on the five datasets, is considered.

For the GLUE benchmark, two fine-tuning settings are taken into account. In the first scenario, RoBERTa's fine-tuning is based solely on training data of the respective task, i.e. the fine-tuning is performed separately for each of the tasks. Hereby, the hyperparameter sweep with learning rates taking on the values {1e-5, 2e-5, 3e-5} and batch sizes the values {16, 32} is regarded with a linear warmup for the first 6% of the steps, which is then replaced by a linear decay to 0. The fine-tuning is performed for 10 epochs. Furthermore, early stopping, which depends on the performance of each task, is applied. The remaining hyperparameters equal the ones from the pre-training. Hereby, RoBERTa outperforms $\text{BERT}_{\text{LARGE}}$ and $\text{XLNet}_{\text{LARGE}}$ on all nine tasks on the development set. Even though RoBERTa and $\text{BERT}_{\text{LARGE}}$ use the MLM objective and their architectures resemble each other, RoBERTa still achieves the state-of-the-art. This underlines the fact that formats like data size and training steps (training time) could be more important than initially assumed. In the second setting, Liu et al. (2019) deal with the comparison of RoBERTa and other models on the test set via the GLUE leaderboard. Hereby, a wide range of approaches are based on multi-task fine-tuning, whereas the author's approach only relies on single-task fine-tuning. Note that the fine-tuning for the tasks RTE, STS and MRPC started from the MNLI single-task model. In the second setting a wider hyperparameter sweep with learning rate $\in$ {1e-5, 2e-5, 3e-5}, batch size $\in$ {16, 32}, weight decay of 0.1, 10 being the maximal number of epochs, a linear learning rate decay and a warmup ratio of 6% is regarded. Furthermore, ensembles of five to

seven models per task are built. It should be noted that for the tasks QNLI and WNLI task-specific fine-tuning approaches are performed in order to meet the performance of the leaderboard approaches[19]. The second setting of RoBERTa was uploaded to the GLUE leaderboard, where in four out of nine tasks it reached state-of-the-art results.

RoBERTa's performance on the SQuAD v1.1 development set equals XLNet's performance and outperforms $\text{BERT}_{\text{LARGE}}$. Having a look at the SQuAD v2.0 results, one can see that RoBERTa shows new state-of-the-art results. Additional to this analysis, the authors uploaded RoBERTa to the SQuAD v2.0 leaderboard. RoBERTa shows a better performance than all but one of the single models and has a score located at the top among models which renounce the usage of additional data.

For the RACE benchmark, a concatenation of each candidate answer with the according question and paragraph was performed, which can be seen as a modification of RoBERTa. In the next step, an encoding of each sequence is performed and the resulting `[CLS]` result is then fed to a full-connected layer with the goal of predicting the correct answer. Question-answer pairs that exceed 128 tokens are truncated. The maximum sequence length cannot exceed 512 tokens, so a truncation of the passage is also performed if necessary. The analysis of fine-tuning based on RACE tasks shows that RoBERTa outperforms the other models on middle- as well as high-school settings.

3.5 DistilBERT

Large pre-trained language models are widely used nowadays. On one hand, such models show a large performance improvement and, on the other hand, the model sizes become larger and larger, which implicates some reservations. One of them is "environmental cost of scaling exponentially computing requirements of these models" (Schwartz et al., 2019; Strubell et al., 2019; Sanh et al., 2020). Furthermore, the increasing computational and memory costs may lead to an obstruction

[19] for more details, see Liu et al., 2019, "Task-specific modifications"

of a broad implementation of those models. In the scientific work of Sanh et al. (2020), the authors introduce a new approach, namely, a distilled version of BERT (DistilBERT). Within the scope of the model's pre-training, knowledge distillation is carried out. The authors have shown that an achievement of 97% of BERT's performance and simultaneous reduction of BERT's model size by 40% as well as training time by 60% is possible. Moreover, the model shows a good performance on a wide range of down-stream tasks when fine-tuned. The model size is also small enough, so that the model can even be run on mobile devices [20].

3.5.1 Knowledge distillation

Classification models in the field of supervised learning are ought to be generally trained to predict a class based on the maximization of the estimated probability of the "gold labels". Hereby, the cross-entropy loss between the predicted distribution of the model and the (one-hot) distribution of the training labels should be minimized. Often, a model that performs well results in an output distribution where high probabilities are assigned to the correct class and probabilities around zero to the remaining ones. Hereby, small probabilities also differ from each other, which is an indicator for the generalization abilities of the model. The compression procedure knowledge distillation (Bucila et al., 2006, Hinton et al., 2015) is about training a compact model (the student) to behave like a larger model (the teacher) or an ensemble of models by matching the output distribution (Sanh et al., 2020). A training of the student is performed based on a distillation loss of the form $L_{ce} = \sum_i t_i \log(r_i)$ with t_i and r_i being a probability estimated by the teacher and the student, respectively. Hence, a cross-entropy is performed on soft targets (probabilities of the teacher) rather than hard targets (one-hot encoding of the gold class) (Sanh, 2019). Applying this objective leads to a "rich training signal" based on making use of the whole teacher distribution. Further, the softmax-temperature of the form $p_i = \frac{\exp(z_{score;i}/T_{smooth})}{\sum_j \exp(z_{score;j}/T_{smooth})}$ (Hinton et al., 2015), with T_{smooth} controlling the smoothness of the output distribution and $z_{score;i}$ being the model score of class i, is applied. The same T_{smooth} is used for the student as well as the teacher at training time. At inference, T_{smooth} is defined as 1, leading to the

[20] pre-trained weights as well as the training code can be viewed in the transformers library by *huggingface* at https://github.com/huggingface/transformers

standard softmax. The resulting training objective is then a linear combination of the masked language model loss (L_{mlm}) (Devlin et al., 2019) and the distillation loss. Furthermore, a cosine embedding loss L_{cos} with the tendency of aligning the student's and teacher's hidden states vectors was added.

3.5.2 Model architecture

DistilBERT is based on the same general architecture as BERT (Devlin et al., 2019), whereas an elimination of the token-type embeddings and the pooler is performed. At the same time, Sanh et al. (2020) halved the number of layers[21] [22].
Another aspect which the authors focus on is to figure out what the right initialization of the student is, so that it converges. For the initialization of DistilBERT from BERT the authors make use of the common hidden size of the student and the teacher and take one of two layers (Sanh, 2019).
The authors' suggested methods for training BERT are applied, i.e. a distillation of DistilBERT is performed on very large batches without the usage of the NSP objective while applying dynamic masking.
Furthermore, DistilBERT was trained on the same corpus as BERT. The training was performed on eight 16GB V100 GPUs for about 90 hours.

3.5.3 Experiments

DistilBERT's generalization capabilities and language understanding is analyzed on the basis of the GLUE benchmark. For each task, the model is fine-tuned without ensembling models or applying a multi-task scheme, resulting in scores based on the development set. DistilBERT's performance on all nine tasks is either similar to ELMo's (Peters et al., 2018) performance or outperforms it. In comparison to BERT, DistilBERT maintains 97% of the performance while using only 60% of the parameters (DistilBERT: 66 mil. parameters; BERT: 110 mil. parameters) (Sanh et al., 2020).
Moreover, the model shows only a slightly worse performance than BERT on the

[21] modifications of the tensor's last dimension (hidden dimension) did not show to have a high influence on computational efficiency (Sanh, 2019)

[22] L2 distance was used as the distillation loss on the downstream tasks in some work, but the research from Sanh et al. (2020) indicates that the cross-entropy loss performs better (Sanh, 2019)

downstream tasks SQuAD v1.1 (Rajpurkar et al., 2016a) and "IMDb sentiment classification task" (Maas et al., 2011), despite having a 40% smaller model size. Sanh et al. (2020) further perform a study of whether to add another distillation step during the adaption phase. Hereby, BERT which was fine-tuned on SQuAD performs as a teacher for an additional loss term and DistilBERT is fine-tuned on SQuAD. Two distillation steps, namely during pre-training and during fine-tuning, are undertaken. Given the model size, DistilBERT takes on evaluation values within three points of BERT.
An investigation of the speed-up/size trade-off of DistilBERT based on a full pass on the STS-B development set again shows that the model only has 60% of BERT's parameters while also being 60% faster than BERT.
Lastly, DistilBERT's ability to be used for on-the-edge-applications is analyzed by the implementation of a mobile application designed for answering of questions. Hereby, a comparison of the average inference time on recent smartphones of DistilBERT and $\text{BERT}_{\text{BASE}}$ fine-tuned for question answering is carried out. DistilBERT needs 71% less inference time (if not taking the tokenization step into account) and the weight of the model amounts to 207GB.

3.6 XLNet

The two most common pre-training objectives are autoregressive (AR) language modeling and autoencoding (AE), such as BERT. The goal of autoregressive models is the estimation of the probability distribution of a corpus (Dai et al., 2015; McCann et al., 2017; Peters et al., 2018; Radford et al., 2018; Devlin et al., 2019). It is only capable of encoding a uni-directional context and because downstream NLU tasks are usually in need of bidirectional context information, AR language modeling turns out as a sub-optimal choice for effective pre-training. Pre-training based on AE has the goal of rebuilding the original data from damaged input. This reconstruction can be performed due to the fact that density estimation is not incorporated. Consequently, the "bidirectional information gap in AR language model" (Yang et al., 2019, p. 2) mentioned earlier can be closed. Nevertheless, the AE based pre-training implies a pre-training - fine-tuning discrepancy. Artificial symbols which are used to damage the corpora in the first place (e.g. `[MASK]` in BERT) do not appear in real data when fine-tuning. Additionally, BERT makes

the oversimplified assumption that predicted tokens are independent from each other, given the unmasked tokens.

Yang et al. (2019) introduce XLNet, a generalized autoregressive language model[23]. The model incorporates the maximization of the expected log likelihood of a sequence w.r.t. all possible permutations of the factorization order. This makes it possible to use bidirectional context. The model does not make use of corrupted data, and hence is not exhibited to the fine-tuning - pre-training discrepancy. Furthermore, it does not rely on the independence assumption used in BERT. In order to improve the architectural design of XLNet, the segment recurrence mechanism and the relative encoding scheme of Transformer-XL (Dai et al., 2019) are included in the pre-training, whereby a reparameterization of the Transformer network is performed.

Analyses show that XLNet performs better than BERT on a wide range of downstream tasks, such as GLUE, SQuAD, RACE, Yelp, IMDB and the ClueWeb09-B document ranking task.

3.6.1 Model architecture

Objective: Permutation Language Modeling

Yang et al. (2019) follow the ideas incorporated in the orderless NADE (Uria et al., 2016) and introduce the permutation language modeling objective which is implemented in XLNet's pre-training. This objective keeps the advantages of AR models as well as BERT and also avoids the weaknesses of both. Given a text sequence $\mathbf{x} = (x_1, \ldots, x_T)$, the objective can be formulated in the following manner:

$$\max_{\theta} \mathbb{E}_{z \sim \mathcal{Z}_T} \left[\sum_{t=1}^{T} \log p_\theta(x_{z_t} | \mathbf{x}_{\mathbf{z}_{<t}}) \right]. \tag{19}$$

$\mathcal{Z}_T$ is defined as the set of all possible permutations of the index sequence $(1, 2, \ldots, T)$, z_t and $\mathbf{z}_{<t}$ as the "t-th element and the first $t-1$ elements of a permutation $\mathbf{z} \in \mathcal{Z}_T$" (Yang et al., 2019, p. 3).

Note that only the factorization is permuted and not the sequence order which is a necessary choice since the model only takes sequences with a natural order into account during fine-tuning.

[23] the implementation can be found at https://github.com/zihangdai/xlnet

Two Stream Self-Attention for Target-Aware Representations

An implementation of the standard Transformer parameterization is not possible due to the fact that if it is implemented naively, a prediction of the same distribution is performed without taking the target position into account. In order to bypass this issue, a re-parameterization of the next-token distribution is introduced:

$$p_\theta\left(X_{z_t} = x | \mathbf{x}_{z<t}\right) = \frac{\exp(e(x)^\intercal g_\theta(\mathbf{x}_{\mathbf{z}<t}, z_t))}{\sum_{x'} \exp(e(x')^\intercal g_\theta(\mathbf{x}_{\mathbf{z}<t}, z_t))}. \tag{20}$$

Hereby, $e(x)$ denotes the embedding of x and $g_\theta(\mathbf{x}_{\mathbf{z}<t}, z_t)$ the context representations produced by neural models which additionally incorporate z_t as input and are therefore target position aware.

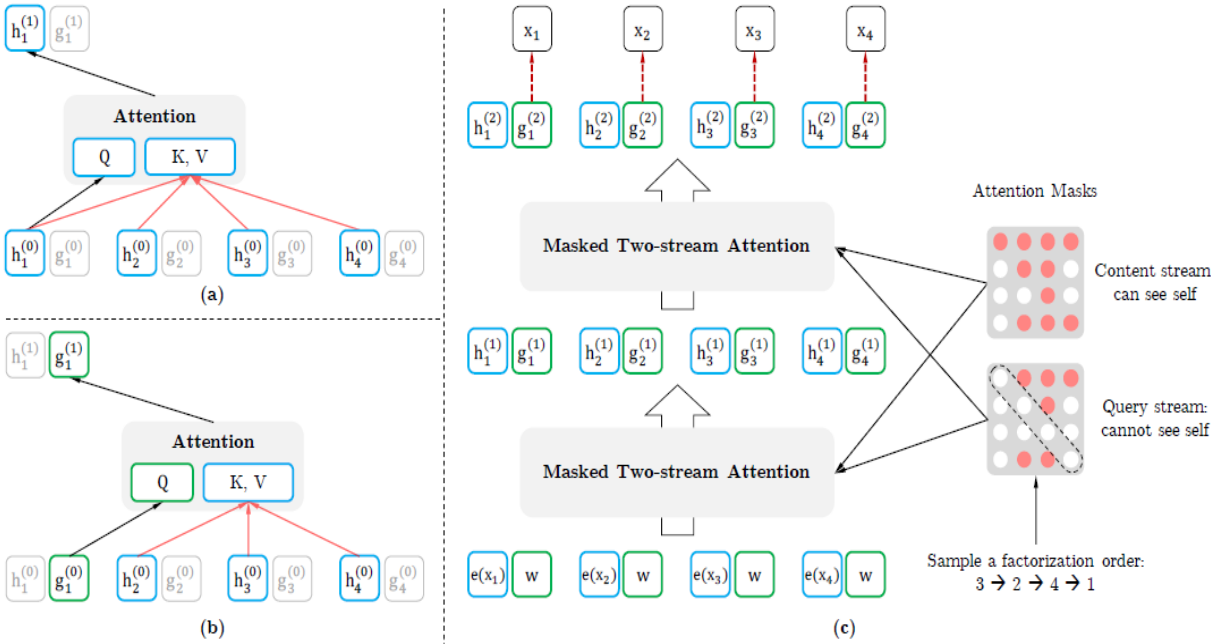

Figure 6: (a) Content stream attention. (b): Query stream attention. (c): Training of permutation language modeling with two-stream attention. (Yang et al., 2019, p. 4)

Furthermore, the authors make use of two hidden representations instead of one (like in the standard Transformer architecture) which are the following: $h_\theta(\mathbf{x}_{\mathbf{z}\leq t})$, short h_{z_t} is the content representation that carries out the encoding of the context as well as x_t. The second representation is the query representation $g_\theta(\mathbf{x}_{\mathbf{z}<t}, z_t)$,

short g_{z_t}. For this representation, only information about the context $\mathbf{x}_{\mathbf{z}<t}$ and the position z_t is available. The update rules of the query stream as well as the one of the content stream, which are carried out with a shared set of parameters, are displayed in figure 6. In the upper left corner (a), the update rule of the content stream, which resembles the standard self-attention, can be seen. Hence, during fine-tuning we can utilize only this stream as a Transformer. In (b) the update rule of the query stream is shown. Hereby, the content w_{z_t} is not accessible. The right side of figure 6, (c), gives an "overview of the permutation language modeling with two-stream attention" (Yang et al., 2019, p. 4). Hereby, a trainable vector w serves as initialization ($g_i^{(0)} = w$) for the query stream and the according embedding for the content stream ($h_i^{(0)} = e(x_i)$). The update of each stream is performed for each self-attention layer $m = 1, \ldots, M$ in the following way:

$$g_{z_t}^{(m)} \leftarrow \text{Attention}(\mathbf{Q} = g_{z_t}^{(m-1)}, \mathbf{KV} = \mathbf{h}_{\mathbf{z}<t}^{(m-1)}; \theta) \tag{21}$$

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\mathbf{Q} = h_{z_t}^{(m-1)}, \mathbf{KV} = \mathbf{h}_{\mathbf{z}\leq t}^{(m-1)}; \theta) \tag{22}$$

To solve equation 20, the last-layer query representation $h_{z_t}^{(M)}$ can then be used.

In order to simplify the difficult optimization problem, which is caused by permutation, the authors only carry out a prediction of the last tokens in a factorization order. Hereby, a splitting of $\mathbf{z}$ into a non-target subsequence and a target subsequence is performed. The log-likelihood of the target subsequence given the non-target subsequence should then be maximized.

Incorporating Ideas from the Transformer-XL

XLNet includes two methodologies from Transformer-XL (Dai et al., 2019) which are the relative positional encoding scheme and the segment recurrence mechanism. The relative positional encoding scheme is applied without modification, using the original sequence. The integration of the recurrence mechanism into the permutation setting is not straight-forward and needs to be modified in order for the model to be able to reuse hidden states from prior segments [24].

[24] for a more detailed explanation, see Yang et al., 2019, p. 5

Modeling Multiple Segments

In order to enable XLNet to model multiple segments, it follows BERT's pre-training scheme by randomly sampling two segments and handle its concatenation as one. The only memory that is being reused stems from the same context. XLNet's input takes the same form as BERT's input, namely `[CLS, A, SEP, B, SEP]`. Note that XLNet does not incorporate the NSP task (Yang et al., 2019).

In contrast to BERT, XLNet applies the relative encoding idea from Transformer-XL which is to only model relationships between positions to the encoding of segments. This means that it is only taken into account whether two positions in a sequence are located within the same segment and not which specific segment they originate from (like in BERT). By applying relative positional encoding, the inductive bias of relative encodings is introduced, which leads to an improvement of the generalization. Another benefit is that thereby a fine-tuning on tasks with more than two input segments is enabled (Yang et al., 2019).

When comparing BERT and XLNet, it becomes clear that both models apply partial prediction. Not only does it represent a necessity for BERT since no prediction is possible if a masking of all tokens is performed, but it also helps reducing the optimization difficulty for both models. Important to note is the fact that due to the independence assumption of BERT, the model is not able to observe dependencies between targets. Hence, "XLNet always learns more dependency pairs given the same target and contains 'denser' effective training signals" (Yang et al., 2019, p. 6).

3.6.2 Experiments

XLNet's pre-training is based on the same text corpora as BERT (Devlin et al., 2019) as well as Giga5 (Parker et al., 2011), ClueWeb 2012-B (extended from Callan et al., 2009) and CommonCrawl (Crawl, 2019), whereby short articles or the ones with a low quality are not included. Furthermore, a tokenization with SentencePiece was performed, resulting in subword pieces (For a more detailed explanation of SentencePiece, see appendix, section 8).
In a fair comparison - pre-training both models and hyperparameters on the same

data - the authors show that XLNet performs better than BERT on all regarded downstream tasks. Furthermore, XLNet was compared to RoBERTa, whereby additional data and the same hyperparameters as in RoBERTa were used for XLNet's pre-training to ensure a fair comparison. XLNet shows a better performance than RoBERTa and BERT on all considered down-stream tasks. Furthermore, Yang et al. (2019) notice that for explicit reasoning tasks such as SQuAD and RACE that contain a longer context, XLNet's performance gain is larger which could be attributed to the fact that XLNet is additionally based on Transformer-XL. XLNet also results in a considerable performance improvement for classification tasks.
The ablation study from Yang et al. (2019) underlines the importance of the Transformer-XL backbone and the permutation LM which make a large contribution to XLNet outperforming BERT on a large margin. Additionally, when eliminating the memory caching methodology, a performance decrease can be observed. Furthermore, the span-based prediction as well as the bidirectional input pipeline represent an important component of XLNet. Lastly, the NSP objective does not contribute to a performance improvement and was therefore removed from XLNet (Yang et al., 2019).

3.7 ALBERT

As already described in the prior sections, the concept of pre-training full networks and afterwards fine-tuning them on downstream tasks showed state-of-the-art results, whereby it has emerged that larger models perform better. Computational constraints, e.g. memory limitations aggravate performing experiments with large models. In order to overcome the obstacles of large memory consumption and training time, Lan et al. (2019) present a modified version of BERT called A Lite BERT (ALBERT)[25].

3.7.1 Model architecture

ALBERT's and BERT's architecture resemble each other in the sense that both are based on a Transformer encoder (Ashish Vaswani et al., 2017) and GELU

[25]the implementation of the model can be found at https://github.com/google-research/ALBERT

non-linearities (Hendrycks et al., 2016). Furthermore, the feed-forward/filter size is fixed as $4H$ (H being the hidden size) and the number of attention heads as $H/64$, like in Devlin et al. (2019).

Lan et al. (2019) present the following three main modifications of BERT's configurations.

Factorized embedding parameterization

In models like BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019) or XLNet (Yang et al., 2019), the size of the WordPiece embedding E equals H ($E \equiv H$). On one hand, this is subpar from a modeling perspective since WordPiece embeddings should learn representations that are context-independent while hidden-layer embeddings are ought to learn context-dependent ones. Lan et al. (2019) state that untying E from H leads to using the total model parameters more efficiently corresponding to the modeling need itself, which indicates that $H \gg E$. On the other hand, from a practical point of view, the vocabulary size V_{LM} is ought to be large in NLP. Hence, if $E \equiv H$ applies, an increase of H results in an increase of the embedding matrix of size $V_{LM} \times E$, which again can lead to a model incorporating billions of parameters. For this reason, a factorization of the embedding parameters, resulting in two matrices, is performed for ALBERT. Hereby, the one-hot vectors are first projected into an embedding space of size E and in the next step into the hidden space of size H. This causes a reduction of the embedding parameters from $O(V_{LM} \cdot H)$ to $O(V_{LM} \cdot E + E \cdot H)$, which is significant in the case of $H \gg E$. The same E is used for all word pieces (Lan et al., 2019).

Cross-layer parameter sharing

The authors further implement a sharing of all parameters across layers which ensures that the model parameters do not increase with the network's depth. It is used as a default design choice for ALBERT. Having a look at the L2 distances and cosine similarity (in terms of degree) of the input and the output embedding for each layer, it can be stated that ALBERT's crossover from layer to layer is far more smooth than the one of BERT. This indicates that the cross-layer

parameter sharing has an impact on the stabilization of network parameters. Even though ALBERT's metrics decline in comparison to BERT's, they do not show a convergence towards zero after 24 layers. This underlines the fact that ALBERT's embeddings are oscillating (Lan et al., 2019).

These two parameter reduction techniques lead to a significant reduction of the number of BERT's parameters while not seriously deteriorating the performance. They also serve to improve the generalization of the model by stabilizing the training.

Inter-sentence coherence loss

BERT's pre-training is based on the MLM and the NSP task (Devlin et al., 2019). In recent scientific papers (Liu et al., 2019, Yang et al., 2019) NSP was found ineffective which was substantiated by the performance improvement across a variety of downstream tasks and as a consequence it was removed. Lan et al. (2019) hypothesize that the ineffectiveness of the task is based on the simplicity of the task compared to the MLM task. NSP combines *topic prediction* with *coherence prediction* in one task, even though *topic prediction* represents an easier task which also shows bigger overlaps with what is learned when applying the MLM loss. The authors introduce a loss which is based mainly on modeling inter-sentence coherence, the Sentence Order Prediction (SOP) loss. For creating positive examples, it follows BERT's technique and samples two consecutive segments originating from the same document. Negative examples are based on the same two segments with a reversed order. This way, "finer-grained distinctions about discourse-level coherence properties" are learned (Lan et al., 2019, p. 5). Later on, it is shown that the NSP task is not able to solve the SOP task. In contrast to that, SOP is able to solve the NSP task to an appropriate extent. Consequently, ALBERT shows an improved performance for multi-sentence encoding tasks.

Important to note is the fact that ALBERT has much less parameters in comparison to BERT. $\text{ALBERT}_{\text{LARGE}}$ for example incorporates 18 times less parameters than $\text{BERT}_{\text{LARGE}}$. Lan et al. (2019) note that the most important advantage of ALBERT's design choices is the improved parameter efficiency.

3.7.2 Experiments

In the following, the experimental setup is displayed in more detail in order to quantify the earlier addressed parameter efficiency.

The baseline models are pre-trained based on the same corpora as BERT (Devlin et al., 2019). Furthermore, the maximum sequence length is set to 512, whereby input sequences shorter than 512 tokens are randomly generated with a probability of 10%. The vocabulary size is 30,000 tokens which are generated by applying SentencePiece (Kudo et al., 2018) like in XLNet (Yang et al., 2019). Moreover, the creation of masked inputs for the MLM task is carried out with n-gram masking (Joshi et al., 2019). The probability for the maximum length n of n-gram can be formulated in the following manner:

$$p(n) = \frac{1/n}{\sum_{k=1}^{N} 1/k}. \tag{23}$$

The authors set n to the value of 3. Note that all model updates apply a batch size of 4,096 and use a LAMB optimizer[26] for which the learning rate is set to 0.00176 (You et al., 2019). All models are trained for 125,000 steps on Cloud TPU V3, whereby the used number of TPUs depends on the model size and takes on values from 64 to 512. The described experimental setup is applied to all of the versions of ALBERT and BERT models from Lan et al. (2019). The evaluation of the model takes place based on the three common benchmarks GLUE (Wang et al., 2019), SQuAD (Rajpurkar et al., 2016a) and RACE (Lai et al., 2017). Similar to Liu et al. (2019), an early stopping is performed based on the development sets.

The authors' research shows that $\text{ALBERT}_{\text{XXLARGE}}$ only uses 70% of $\text{BERT}_{\text{LARGE}}$'s parameters while outperforming it on the development set of various downstream tasks, such as SQuaD v1.1 and v2.0, MNLI, SST-2 and RACE (Lan et al., 2019).

[26] You et al. (2019) introduced LAMB, a layerwise adaptive large batch optimization strategy which enables a stable training of large batches. The underlying logic comes from LARS (You et al., 2017), which performs poorly on attention models. For a detailed explanation of LAMB, see You et al. (2019)

Furthermore, it takes approximately 1.7 less time for $\text{ALBERT}_{\text{LARGE}}$ to iterate through data than $\text{BERT}_{\text{LARGE}}$, indicating ALBERT models having a higher data throughput than its associated BERT models. Changing the vocabulary embedding size E on $\text{ALBERT}_{\text{BASE}}$ under the non-shared condition, the largest E ($= 768$) results in a better performance. When regarding this effect under the all-shared condition, $E = 128$ shows the best performance and is therefore used in further analyses. Lan et al. (2019) further compare cross-layer parameter-sharing strategies applied to $\text{ALBERT}_{\text{BASE}}$ with $E = 128$ and $E = 768$. In general, a cross-layer sharing strategy can be described as performing a division of L layers into N groups of size M where each group is sharing the parameters. The authors show that the performance improves with a smaller M. Since decreasing M results in a large increase of the number of parameters, the all-shared strategy is chosen to be the default choice of ALBERT. A further experiment that is conducted is a comparison between the following inter-sentence losses: none (like in XLNet and RoBERTa), NSP (like in BERT) and SOP (like in ALBERT) applied to $\text{ALBERT}_{\text{BASE}}$. Hereby, the performance on the MLM, NSP and SOP task is regarded where it can be seen that the NSP loss does not solve the SOP task well. On the other hand, the SOP loss shows a good performance for the NSP and the SOP task. Furthermore, an improvement of the downstream task performance can be monitored if the SOP loss is applied. Another question to be answered is about the performance of $\text{ALBERT}_{\text{XXLARGE}}$ compared to $\text{BERT}_{\text{LARGE}}$ if both are trained for approximately the same amount of time. The authors display that $\text{ALBERT}_{\text{XXLARGE}}$ shows significant better performance results on the downstream tasks than $\text{BERT}_{\text{LARGE}}$. It is further shown that training with additional data has a large positive effect when regarding the development set MLM performance. This can also be observed for the downstream tasks apart from the SQuAD benchmarks. Due to the fact that even after a training of one million steps, the authors' largest model does not show overfitting the dropout was removed which resulted in a performance increase of the MLM task. For a performance comparison between state-of-the-art models and ALBERT, the models were trained on the BERT vocabulary as well as the additional data used in XLNet and RoBERTa. Furthermore, the fine-tuning was performed only on a single task and the results which are based on the development set represent median results of five runs. For ALBERT, the earlier described best

design choices were applied. Furthermore, the checkpoints for the final ensemble model are chosen based on the performance on the development set, whereby the number can vary depending on the task. Both ALBERT model variants - the single model and the ensemble model - outperform the state-of-the-art models on all benchmarks.

The Appendix in Lan et al. (2019) contains a detailed table about the chosen hyperparameters for downstream tasks. Since the other presented scientific works mention a learning rate range, but do not specify concrete learning rate values and the hyperparameters in Lan et al., 2019 are adapted from Devlin et al. (2019), Liu et al. (2019) and Yang et al. (2019), we set the respective learning rates in our analyses according to the table in Lan et al. (2019).

GLUE task	Learning rate
CoLA	1e-5
STS	2e-5
SST-2	1e-5
MNLI	3e-5
QNLI	1e-5
QQP	5e-5
RTE	3e-5
MRPC	2e-5
WNLI	2e-5

Table 1: Learning rates for ALBERT in GLUE tasks, Lan et al. (2019), A.4, p. 17

4 GLUE

Since we fine-tune the large pre-trained language models described above on the the GLUE tasks in order to conduct a fine-tuning and inference time measurement, the GLUE tasks are shortly described in this chapter. Hereby, it should be noted that we always use the training set for the analysis of the fine-tuning time and the development set to analyze the inference time. We also added the performance measures referring to the respective GLUE task. The comparison of performance measures of our analysis and the GLUE benchmark should be handled with care, since the GLUE benchmark incorporates performance results on the test set, whereas the performance results of our analyses are based on the development sets.

GLUE is a benchmark that consists of nine tasks in the field of sentence understanding in the English language. With the help of those tasks, a wide variety of domains, quantities of data as well as challenges can be covered. This benchmark is built in such a manner that "good performance should require a model to share substantial knowledge (e.g. trained parameters) across all tasks, while still maintaining some task-specific components" (Wang et al., 2019). This statement underlines the fact that applying models with a Transformer architecture such as BERT, ALBERT, etc. - like we do in the scope of this thesis - is not only permitted, but also recommended. In the following, each of the GLUE tasks is described, respectively.

Since we fine-tune the earlier described models on each of these tasks, we regard the respective training dataset in our analysis. The following structure is adopted from Wang et al. (2019).

4.1 Single-sentence Tasks

4.1.1 CoLA

Firstly, the task CoLA which is also called "The Corpus of Linguistic Acceptability" (Warstadt et al., 2018) is described. This task contains English decisions on acceptability which are obtained from journal articles and books dealing with theory on linguistic. Each element of the dataset consists of a sequence of words

which is commented with whether this element represents a grammatical English sentence or not (0: not a grammatical English sentence; 1: grammatical English sentence) (Wang et al., 2019). For evaluation purposes, the Matthews correlation coefficient is applied (Matthews, 1975). This coefficient has the goal to evaluate the performance of unbalanced binary classification tasks. It can take values from -1 to 1, whereas 0 can be interpreted as guessing, +1 as perfect prediction and -1 as total disagreement between observation and prediction (Warstadt et al., 2018; Wikipedia, the free encyclopedia, 2020d).
The CoLA training dataset consists of 8,550 observations. Approximately 29.56% of them are labeled as 0 and can therefore be interpreted as "not a grammatical English sentence". About 70.44% of the observations are "grammatical English sentences". "One more pseudo generalization and I'm giving up." represents an example for a grammatical English sentence (label=1), whereas "Mary's criticism him was cruel." is an example from the CoLA dataset for a sentence that is not grammatically correct (label=0).

4.1.2 SST-2

The dataset SST-2 that stands for "the Stanford Sentiment Treebank" (Socher et al., 2013) has the goal to predict the sentiment of a sentence. The sentences stem from movie recensions and human comments on the respective sentiment. In this case, it is a binary classification problem where the values are "positive" (label = 1) or "negative" (label = 0) (Wang et al., 2019).
The training dataset consists of 67,349 sentences, whereby 44.22% are labeled as "not good"/"negative" (label = 0) and the remaining 55.78% are labeled as "good"/"positive". An example for a sentence that was annotated with the label 0 is the following: "the movie's major and most devastating flaw". On the other hand, the sentence "most certainly has a new career ahead of him" was marked with the label 1.

4.2 Similarity and Paraphrase Tasks

4.2.1 MRPC

The next task that represents a part of the GLUE tasks is the dataset "The Microsoft Research Paraphrase Corpus" (Dolan et al., 2005), which can be abbreviated by MRPC. It contains sentence pairs which origin from online news sources. Humans commented on whether two sentences that make a sentence pair can be seen as equivalent in a semantic manner. Due to the fact that both classes are not balanced, the F1 score is reported additionally to the accuracy (Wang et al., 2019).
The training dataset contains 3,668 sentence pairs of which 32.55% are labeled as not semantically equivalent (label = 0) and 67.44% as semantically equivalent (label = 1). The sentences "It was called mandatory, but Dupont said authorities did not force people to leave." and "While it was being called mandatory, Dupont said authorities were not forcing people from their homes." are labeled as semantically equivalent (label 1) and the two sentences "Istomin later married Casals' widow, Marta, after Casals' death in 1973." and "In 1975 he married Marta Casals, the widow of Pablo Casals." as not semantically equivalent (label 0).

4.2.2 QQP

The QQP - Question Question Pairs[27] - task contains question pairs that stem from Quora which is a community question-answering website. Here, the goal of distinguishing between semantically equivalent and non-equivalent question pairs is pursued. The classes are imbalanced. Consequently, the F1 score as well as the accuracy are reported. Wang et al. (2019) used the test set for computing performance measures. Important to note is hereby the fact that the class distribution of the labels differs between the test and the training set.
The QQP training dataset contains 363,870 question pairs. 63.07% of these observations are not semantically equivalent and the remaining 36.93% form the positive class which contains semantically equivalent questions. An example for the positive class is the question pair "What is the most beautiful memory you have?" and "What is your most beautiful memory?". In contrast to this, the question pair

[27] https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs

"How long does it take to walk a marathon?" and "How long is a marathon?" is regarded as semantically non-equivalent.

4.2.3 STS-B

The next task that is regarded is the Semantic Textual Similarity Benchmark, short STS-B (Cer et al., 2017). The sentence pairs that form the corpus are gathered from natural language inference data, image and video captions as well as news headlines. Additionally, humans assigned a similarity score to the respective sentence pairs which ranges from 1 to 5, whereas the goal is the prediction of those scores. The evaluation is carried out by applying the correlation coefficients from Pearson and Spearman (Wang et al., 2019). The training dataset consists of 5,749 sentence pairs. The score distribution can be viewed in the following graphic:

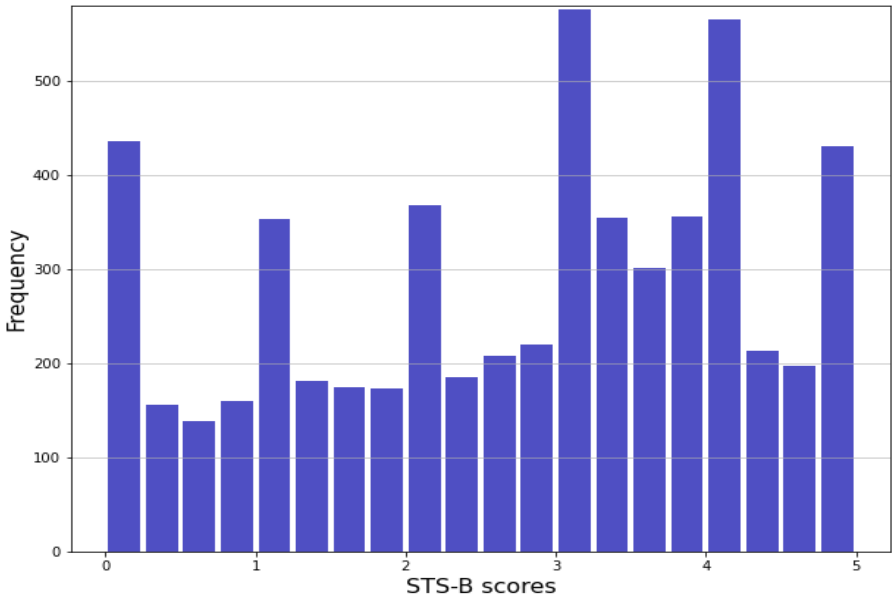

Figure 7: Distribution of similarity scores in STS-B training data

Figure 7 shows that score 3 and 4 represent the most widely represented similarity score in the STS-B training dataset. A sentence pair that was annotated with a similarity score of 0 is "A boy is crawling into a dog house." and "A cat is playing on

the floor.". Another example is the sentence pair "Six dead in Philippine restaurant blast" and "6 killed in Philippines restaurant blast." which was assigned with the score 5.

4.3 Inference Tasks

4.3.1 MNLI

The inference tasks include, amongst other tasks, MNLI, which is also called the "Multi-Genre Natural Inference Corpus" (Williams et al., 2018). It contains sentence pairs that consist of a premise and a hypothesis sentence. Additionally, entailment annotations are included. The goal of the task is the entailment prediction, i.e. whether the premise sentence represents an entailment of the hypothesis sentence, a contradiction or none of the above (is neutral). In the MNLI task, the premise sentences are drawn from ten different sources in total, e.g. government reports and fiction.
In the training set, which consists of 392,702 examples, each class has nearly the same frequency, i.e. each class contains approximately 33.33% of the examples. An example for a sequence pair that represents a contradiction is the following: "You've got to look in the mirror every morning and ask 'What am I organizing for ?'" and "You don't need to know why you are organizing.". Note that the test set is split into a matched and a mismatched version. The matched section refers to in-domain data, whereas mismatched refers to cross-domain data. Furthermore, Wang et al. (2019) advise to utilize the SNLI data (Bowman et al., 2015) which consists of 550 thousand observations as additional data.

4.3.2 QNLI

The next dataset that is regarded is the "Stanford Question Answering Dataset" (Rajpurkar et al., 2016b). It represents a question-answering dataset that includes pairs consisting of questions and paragraphs. Hereby, the answer (one sentence) to the according question is included in the paragraph which is gathered from Wikipedia.
The task is modified to a sentence pair classification by Wang et al. (2019). This is undertaken by generating pairs of each question and each sentence in the according

context. Furthermore, question-sentence pairs that evince a modest lexical overlap are screened out. The goal of this task is the determination of whether the context sentence includes the answer to the according question. On one hand, by using the amended task, the requirement of selecting the exact answer does not have to be fulfilled. On the other hand, the assumption that the input always contains the answer and hence the lexical overlap is a reliable indicator cannot longer be presumed. The modified task is named QNLI (Wang et al., 2019).
The QNLI training dataset contains 104,743 examples, whereas the dataset is balanced. An 'entailment' example is the following: "What age ran from 1700 to 1200 BCE?" as the question and "The Late Bronze Age (from 1700 to 1200 BCE)" as the sentence. A 'not entailment' example is the following: "What type of weather is a rarity in Houston?" and "Houston has mild winters in contrast to most areas of the United States.".

4.3.3 RTE

The "Recognizing Textual Entailment" task, short RTE, was generated by combining four textual entailment datasets, namely data from RTE1 (Dagan et al., 2006), RTE2 (Bar Haim et al., 2006), RTE3 (Giampiccolo et al., 2007) and RTE5 (Bentivogli et al., 2009). These datasets are based on Wikipedia text as well as news. In order to ensure consistency, all four datasets were transformed to a two-class split. Datasets with three classes were modified in such a manner that the classes 'neutral' and 'contradiction' were combined to one class named 'not entailment' (Wang et al., 2019). The RTE training dataset consists of 2,490 sentence pairs. The dataset is balanced with a class frequency of 49.84% for 'entailment' and the remaining 50.16% for 'not entailment'. An example of an RTE sentence pair labeled as 'entailment' is "Catastrophic floods in Europe endanger lives and cause human tragedy as well as heavy economic losses." and "Flooding in Europe causes major economic losses.". In contrary, "The Chicago White Sox are a major league baseball team based in Chicago, Illinois." and "The Bulls basketball team is based in Chicago, Illinois." was labeled as 'not entailment'.

4.3.4 WNLI

The last dataset that is considered in this setting is the "Winograd Schema Challenge" (Levesque et al., 2011). This task "is a reading comprehension task in which a system must read a sentence with a pronoun and select the referent of that pronoun from a list of choices" (Wang et al., 2019, p. 5). A manual construction of the examples was undertaken. Each of these examples depends on the contextual information that is allocated by a phrase or word within the sentence itself. The ambiguous pronoun is substituted by each possible referent in order to form sentence pairs. The goal of this sentence pair classification task is the prediction of whether the transformed sentence is entailed by the original sentence. The evaluation set of this WNLI task is based on data from fiction books (Wang et al., 2019). Note that whilst the training dataset is balanced (label = 0: 50.87% and label = 1: 49.13%), the test dataset, which was used by Wang et al. (2019) for evaluation, is imbalanced with 65% of examples labeled as 'not entailment'. Furthermore, there exists a data idiosyncrasy, which means that there are cases where the development set as well as the training set contain equivalent sentences. Hence, if the training examples are learned by heart by the model, a wrong label prediction will be made on the according development set example (Wang et al., 2019).
The training dataset contains 635 examples. One of these examples is the following: "Dan had to stop Bill from toying with the injured bird. He is very cruel." as sentence1 and "Dan is very cruel." as sentence2. This sentence pair is labeled as zero which is equivalent to 'not entailment'. An entailment example (with label = 1) is "Sara borrowed the book from the library because she needs it for an article she is working on. She reads it when she gets home from work." as sentence1 and "She reads the book when she gets home from work." as sentence2.

5 Experimental setup and implementation

In the following chapter, we make some important notes on the experimental as well as technical setup, along with a comparison of the Python modules `time` and `timeit`.

5.1 Experimental setup

The focus of the following analysis (chapter 6 and 7) is the measurement of the fine-tuning and inference time of LM models on the above presented GLUE tasks. The regarded LM models are the uncased and cased versions of $\text{BERT}_{\text{BASE}}$ as well as $\text{DistilBERT}_{\text{BASE}}$, $\text{RoBERTa}_{\text{BASE}}$, $\text{AlBERT}_{\text{BASE}}$ and the cased version of $\text{XLNet}_{\text{BASE}}$ (the uncased version is not implemented in the `transformers` module we use). We fine-tuned the Transformer models which are integrated in the Python module `transformers` (Wolf et al., 2019) on the GLUE tasks WNLI, RTE, MRPC, CoLA, SST-B and STS-2. These models include bert-base-cased and bert-base-uncased which represent the cased and uncased version of $\text{BERT}_{\text{BASE}}$, respectively. Furthermore, distilbert-base-cased and distilbert-base-uncased are included in the analysis. These are the cased and uncased version of $\text{DistilBERT}_{\text{BASE}}$. Within the uncased versions of BERT and DistilBERT, the text has been lowercased before tokenization was performed, for example *New York* turns into *new york*. Furthermore, accent markers are eliminated in the uncased version. In the cased versions, the text is not pre-processed before tokenization [28] (Devlin et al., 2019). Additionally, we fine-tuned roberta-base, which represents $\text{RoBERTa}_{\text{BASE}}$. The model albert-base-v1 represents $\text{ALBERT}_{\text{BASE}}$ and xlnet-base-cased is the cased version of $\text{XLNet}_{\text{BASE}}$ [29]. Moreover, it is important to note that we did not fine-tune large versions of the described models due to computational limitations. For the same reason, we fine-tuned the respective models only for one epoch (instead of two or three that are recommended the presented LM papers).
We use different hyperparameter sweeps for the fine-tuning itself. Here, we follow the respective hyperparameter recommendations included in the scientific work about ALBERT (see chapter 3.7) since it incorporates a more detailed hyperpa-

[28] for more information, see https://github.com/google-research/bert
[29] The exact model configurations can be found at https://huggingface.co/

rameter table for fine-tuning on GLUE which relies on the hyperparameter configurations from BERT. The weight decay[30] was set to 0.01 and Adam epsilon to 1e-6. The learning rate ranges from 1e-5 to 5e-5, depending on the regarded GLUE dataset (see table 1). Furthermore, a variation of the hyperparameters *maximum sequence length* and *batch size* is included since they have the highest potential, along with the dataset and the model, to have an effect on the respective fine-tuning and inference time. The training batch size (or evaluation batch size) is set to {8, 16, 32} and the maximum sequence length to {128, 256, 512}. Note that when measuring the fine-tuning time, we regard different training batch sizes since the measurement is included in the fine-tuning (training) of the model itself. For measuring the inference time, different evaluation batch sizes are applied since the measurement is included in the prediction function which is used during evaluation. In the following analysis, we refer to the training batch size when mentioning the batch size in the fine-tuning time setting and to the evaluation batch size when addressing it in the inference time setting.

5.2 Technical setup

We carry out the analyses with Python (Van Rossum et al., 2009), using Visual Studio Code and Jupyter Notebooks (Kluyver et al., 2016). The large pre-trained language models which are fine-tuned in the scope of our analyses are implemented in the Python module `transformers` by *huggingface* (Wolf et al., 2019). When fine-tuning, we go through the hyperparameter combinations via grid search, so that all hyperparameter combinations are regarded. We integrate the time measurements in the `transformers` v.3.0.0 module (Wolf et al., 2019), more precisely in the script *trainer.py*[31]. Furthermore, we include an additional argument defining the number of fine-tuning iterations in the *training_args.py* script[31] [32]. The fine-tuning was carried out using the script *run_glue.py*[33], which is also incorporated in the `transformers` module and is based on the machine learning framework PyTorch (Paszke et al., 2019). For the implementation of the fine-tuning time, we follow McCormick's and Ryan's implementation (McCormick et al., 2019a) and

[30] also referred to as L2 regularizer (Goldberg, 2017, p. 30)
[31] the scripts can be found in the folder *src/transformers*
[32] the modified scripts can be retrieved at https://github.com/annakorotkova/transformers
[33] this script can be found in the folder /examples/text-classification/

include it in the *train()* function. The fine-tuning time measurement starts before defining the total train loss and ends after updating the learning rate. All in all, we measured how long the training (here fine-tuning) takes for one epoch. While we perform the fine-tuning five times for the tasks WNLI, RTE and MRPC, it is carried out three times for the CoLA, SST-B and STS-2 to get stable values of the fine-tuning time. The number of repetitions is reduced for the tasks CoLA, SST-B and STS-2 since they contain more observations and therefore have higher computational costs. Inference in the field of machine learning can be defined as "the process of using a trained machine learning algorithm to make a prediction" (DeBeasi, 2019). Thus, we integrate the time measuring of the inference in the prediction loop function (*_prediction_loop()*) that is also used by the *evaluate()* function which is applied when evaluating the model. Hereby, passing one input to the model is measured, whereby one input equalized one batch. This results in multiple measurements, depending on the number of batches, which is why we perform the fine-tuning once instead of multiple times. Both analyses are carried out on the virtual machine Ubuntu 18.04.1 LTS, incorporating a Tesla V100 PCIe 16GB GPU. The respective runs are logged with wandb (Biewald, 2020), a tool which enables tracking machine learning experiments.

5.3 Time modules in Python

In order to measure the fine-tuning as well as the inference time of large pre-trained LMs in Python, it is necessary to decide which time measuring function to use. As noted before, we implement this measurement in Python (Van Rossum et al., 2009), whereby there are two main modules that should be considered. In the following, we describe the Python modules `time` (Python Software Foundation, 2020b) and `timeit` (Python Softwarer Foundation, 2020) as well as the regarded functions and illustrate our decision on the function we use.

First, it is important to note that the Python module `timeit` is generally speaking more suitable for conducting performance tests/comparisons than the `time` module due to the fact that it measures the time multiple times without the need to additionally implement a for loop (Ardit, 2019). Nonetheless, the functions implemented in the *timeit* module are rather suitable for measuring the time per-

formance of short code snippets or whole functions than long code snippets. This holds since the module operates based on a different namespace than the Python script itself. Important to note is that there are two types of namespaces, namely the local namespace and the global one. The local namespace for a function is created when the function is called, whereas the creation of the global namespace for a module is carried out when the module definition is read in (Python Software Foundation, 2020a). The time measurement should be integrated in functions that not only require passing the global namespace but also the local namespace as they refer to prior defined functions and arguments. Consequently, the namespaces should be merged and passed to the function in order to enable the time measuring. By following this procedure, it is possible to provide the variables/modules that were already implemented via the argument *globals* that can be found within the functions in the `timeit` module and which determines the namespace the function should operate on. However, it is rather advised against to perform such a merging. Because of the reason described above, we decided to work with the `time` module of Python and integrate an iteration of the time measurement (Python Software Foundation, 2020b; Python Softwarer Foundation, 2020). The next decision that needs to be made is the question of which time function to use. Hereby, it has to be decided whether the system time (also called process time) of the code snippet or the wall-clock time - which equals the real time - should be measured. Since interpretability is the first priority and the measurement runs on a virtual machine which is only used by us, we decided to measure the wall-clock time, even though it is less precise than the system time. For this purpose we use the *time()* function that is integrated in the `time` module (Python Software Foundation, 2020b).

6 Analysis of fine-tuning time

The following chapter contains the analysis of fine-tuning time. First, we perform a descriptive analysis where the fine-tuning time is grouped by the respective hyperparameters, tasks as well as models. This is followed by a correlation matrix, ANOVAs, where the fine-tuning time is grouped by the respective hyperparameters, tasks and models and a log-linear regression. Lastly, we illustrate the relation between the fine-tuning time and the accuracy.

6.1 Descriptive analysis

We perform a fine-tuning based on the tasks WNLI, RTE, MRPC, CoLA, STS-B and SST-2, whereas a fine-tuning of the tasks QNLI, QQP and MNLI failed. Due to the fact that these are the three largest tasks containing the highest numbers of examples and that fine-tuning large pre-trained language models requires a lot of memory, we hypothesize that the fine-tuning failed due to a lack of RAM [34]. This issue is also addressed by Devlin et al., 2019. Since the fine-tuning which was performed in their scientific work used a Cloud TPU with 64GB RAM, the authors state that the usage of a GPU with 12GB - 16GB RAM could lead to out-of-memory issues when using the same hyperparameters as in the paper[35].

The final dataset that is used for further analyses contains 291 observations in total. The main variable of interest is the fine-tuning time. As mentioned earlier, the fine-tuning was carried out multiple times for each hyperparameter combination in order to receive more stable results. In the following, the minimum fine-tuning time per hyperparameter combination is regarded. With this, we specify how long a fine-tuning would take at least. In the following, the fine-tuning time refers to the minimal fine-tuning time if not stated otherwise.

[34] Random Access Memory (RAM) can be defined as a short-term computer memory in which all running processes and programs are stored temporarily (Aschermann, 2017)

[35] for more insights, see https://github.com/google-research/bert/blob/master/README.md

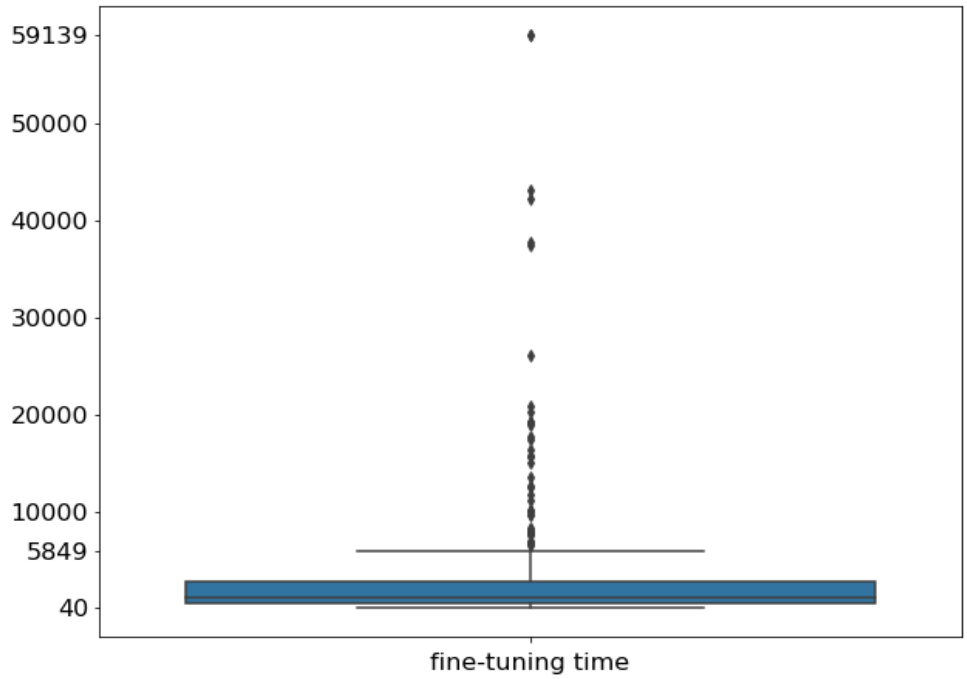

Figure 8: Boxplot of fine-tuning time in seconds

Figure 8 displays the boxplot of the fine-tuning time in seconds. Hereby, the weld point of the distribution lies between 40.3 and 5,848.8[36]. Both values represent the lower and the upper whisker, respectively, while the lower whisker is also the minimum in this case. The median takes on the value 1,195.4. Furthermore, some outliers can be detected which can take on values up to 5,9139.1 (maximum). The mean of the fine-tuning time is 3,670.5 and the standard deviation 7,650.9, which underlines the high spread in the data. A more detailed table of the ten highest as well as the ten lowest fine-tuning times can be found in the appendix (see table 31 and 32). Note that models that are fine-tuned on the SST-2 task, most of which incorporate a maximum sequence length of 512, take on the highest fine-tuning times. Models that were fine-tuned on the WNLI task (in five of ten cases it is the distilbert-base-uncased models), most of which show a maximum sequence length of 128, take on the lowest fine-tuning times. Both observations intuitively make

[36]the values for the fine-tuning time are rounded up to one decimal point

sense as the SST-2 task contains the highest amount of examples, while WNLI represents the smallest dataset. This gives a first intuition about which factors might influence the fine-tuning time.

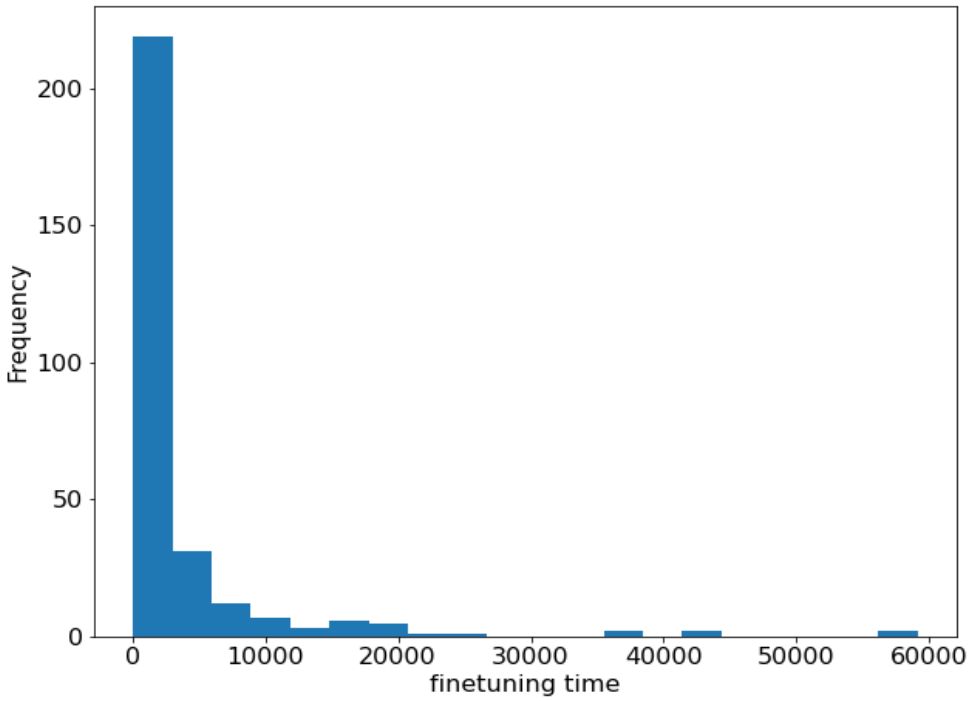

Figure 9: Histogram of fine-tuning time in seconds

The first bin in figure 9 contains 219 data points (out of 291) which represent the majority of the data. This means that, as stated before, the majority of the fine-tuning times takes on values between approximately 40 and 2,995. The second bin, which ranges from about 2,995 to 5,950, contains 31 data points, followed by the third bin ranging from approximately 5,950 to 8,905 which contains 12 values. The following bins show a frequency of 7 at maximum. This underlines the fact that the fine-tuning time shows a right skewed distribution. Note that due to computational limitations higher fine-tuning time values are not possible because a fine-tuning of very large datasets with models that require a long fine-tuning would go beyond the scope of this work. This should always be kept in mind during the analysis.

6.1.1 Comparison across GLUE tasks

As described earlier, we regard the GLUE tasks WNLI, RTE, MRPC, CoLA, SST-B and STS-2 in this analysis. The total number of observations per task should be 63, which results from the number of models (7) multiplied with the number of regarded values of the maximum sequence length (3) and of the training batch size (3). RTE is the only dataset with 63 observations because no (technical) problems occurred when fine-tuning on RTE. Note that a fine-tuning of bert-base-cased and distilbert-base-cased on WNLI, STS-B and SST-2 could not be performed due to errors[37]. This results in a reduced dataset with 45 observations ($63 - 2 \cdot 3 \cdot 3 = 45$) for WNLI and STS-B. Furthermore, the XLNet models were not fine-tuned on the SST-2 task due to computational limitations and runs crashed for albert-base-v1 when applying a maximum sequence length of 521 tokens, probably also due to lack of RAM. Hence, we receive 33 observations for SST-2 ($63 - 2 \cdot 3 \cdot 3 - 1 \cdot 3 \cdot 3 - 3 = 33$). For MRPC, fine-tuning distilbert-base-cased and bert-base-cased failed when applying a sequence length of 128 and 256, whereby it worked for a maximum sequence length of 512. This results in 51 observations ($63 - 2 \cdot 2 \cdot 3 = 51$). For CoLA, we did not fine-tune XLNet because the run time was again too high. Therefore, the dataset for this task incorporates 54 observations ($63 - 1 \cdot 3 \cdot 3 = 54$). This complicates a fair comparison and should always be kept in mind during the analysis.

As described in the chapter 4 of this thesis, the tasks themselves also contain different numbers of examples. The following figure shows the fine-tuning time grouped by the number of observations of the respective GLUE task. The colors of the boxplots correspond to the tasks. The training dataset of WNLI is the smallest task with 635 observations, followed by RTE with 2,490 examples, MRPC with 3,668, SST-B with 5,749 and CoLA with 8,550 observations. The training dataset of STS-2 is with a dataset size of 67,349 the largest GLUE task that we regard in this analysis and incorporates a clearly larger amount of data compared to the other GLUE training sets.

[37] an issue was also created for this case which can be found at https://github.com/huggingface/transformers/issues/4828

The following figure displays the fine-tuning time grouped by the respective task size number that corresponds to a specific task. Note that the colors of the boxplots correspond to the respective tasks.

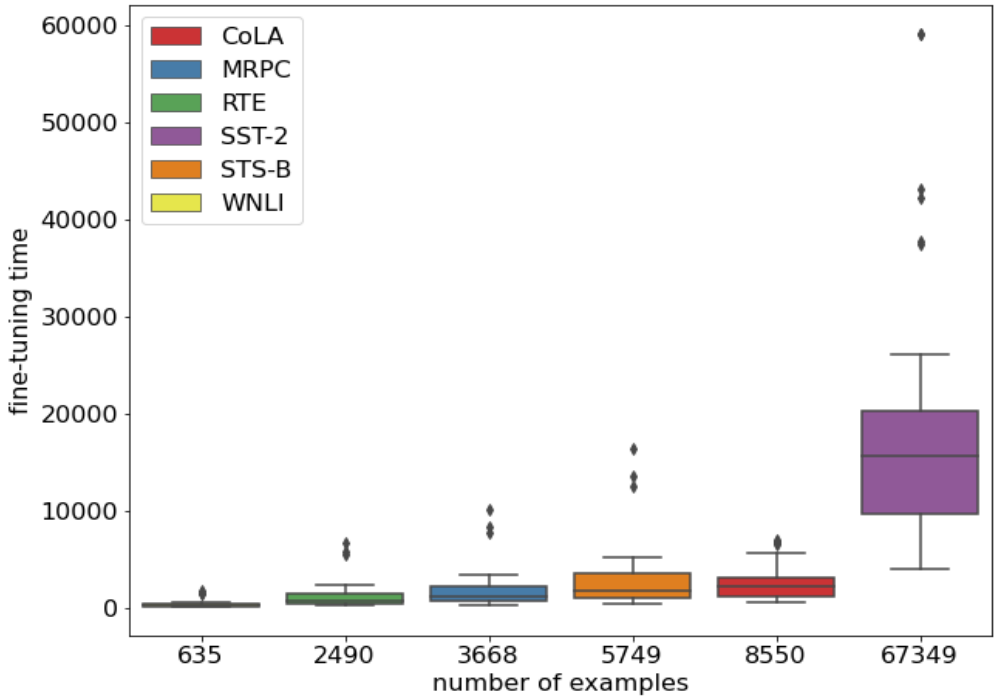

Figure 10: Boxplot of fine-tuning time grouped by number of examples per task

A notable trend can be viewed in figure 10, indicating that the number of examples that a dataset contains plays an important role for the fine-tuning time.

To further investigate this trend, a more detailed view is displayed in table 2. All numerical values in this table, except for the task size, refer to the fine-tuning time.

task name	task size	count	mean	std	min	median	max
WNLI	635	45	308.2	359.0	40.3	179.5	1648.8
RTE	2490	63	1062.1	1247.7	154.8	636.7	6668.0
MRPC	3668	51	1764.9	1970.5	229.9	1050.3	10063.2
STS-B	5749	45	2812.3	3362.9	368.4	1655.7	16341.8
CoLA	8550	54	2555.2	1858.1	523.5	2100.4	6988.0
SST-2	**67349**	33	**19175.3**	14623.1	4017.2	**15637.1**	59139.1

Table 2: Descriptive analysis of fine-tuning time grouped by tasks

We can see that the higher the task size is, the higher the key figures of the fine-tuning time is, such as mean, standard deviation and median. The only exception is the task CoLA. As stated above, it should be noted that for this task we did not fine-tune the model XLNet, which might be the reason for this exception.

Therefore, it is necessary to regard the respective fine-tuning times grouped by task without xlnet-base-cased, which is displayed in table 3:

task name	task size	count	mean	std	min	median	max
WNLI	635	36	204.5	151.5	40.3	154.5	589.1
RTE	2490	54	765.3	573.8	154.8	597.8	2280.4
MRPC	3668	42	1258.8	867.5	229.9	998.0	3387.7
STS-B	5749	36	1826.6	1286.5	368.4	1422.5	5132.8
CoLA	8550	54	2555.2	1858.1	523.5	2100.4	6988.0
SST-2	**67349**	33	**19175.3**	14623.1	4017.2	**15637.1**	59139.1

Table 3: Descriptive analysis of fine-tuning time grouped by tasks without $\text{XLNet}_{\text{BASE}}$

Table 3 exhibits a clear trend since all statistical figures increase with higher task size. Hence, the more sequences a task contains, the higher the fine-tuning time is, which is intuitively plausible.

6.1.2 Comparison across large pre-trained LMs

As already hinted, the fine-tuning time could also depend on the model. Thus, the fine-tuning time grouped by the models is regarded in the next step. The following figure 11 shows the according boxplots.

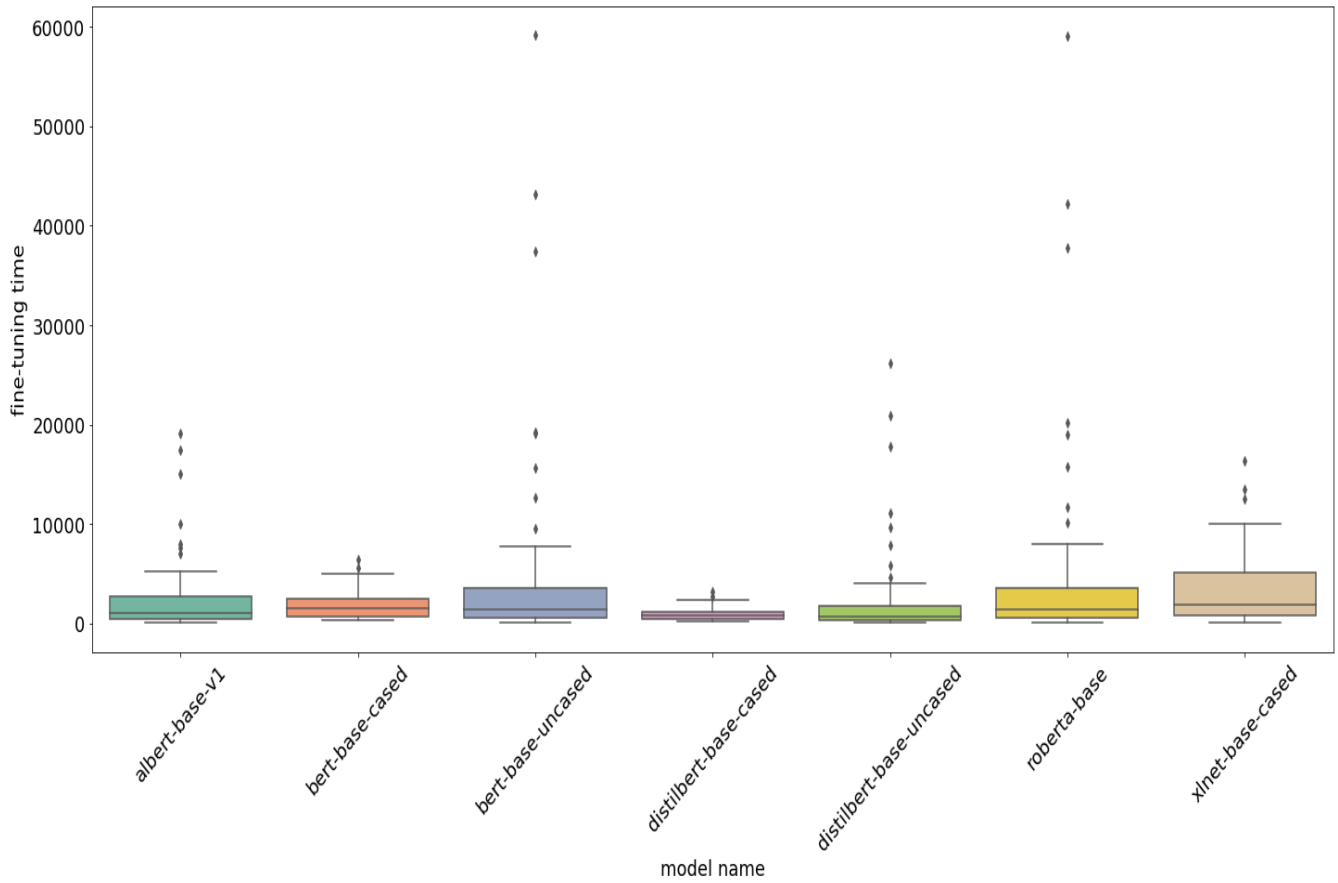

Figure 11: Boxplot of fine-tuning time grouped by models

At first glance, the model distilbert-base-cased exhibits the smallest fine-tuning times with only a few outliers. The model bert-base-cased shows a similar pattern. This can be attributed to the fact that, as mentioned before, it was not possible to perform a fine-tuning on all tasks for these two models. Therefore, it is helpful to have a closer look at the descriptive analysis.

model	count	mean	std	min	median	max
distilbert-base-cased	21	1004.0	810.4	154.8	817.5	3164.9
distilbert-base-uncased	54	2633.1	5248.1	40.3	659.5	26147.2
albert-base-v1	51	2801.7	4291.8	75.5	1096.4	19153.6
bert-base-cased	21	2116.3	1718.9	323.2	1567.9	6485.4
bert-base-uncased	54	**5448.9**	11216.6	79.5	1388.9	59139.1
roberta-base	54	5430.3	11186.5	79.6	1410.4	59108.7
xlnet-base-cased	36	3611.9	4097.2	147.4	**1848.7**	16341.8

Table 4: Descriptive analysis of fine-tuning time by model

In table 4, it can be viewed that distilbert-base-cased has the smallest fine-tuning time mean and standard deviation. However, when having a look at the fine-tuning time minimum or median, this statement cannot be confirmed. This leads to the hypothesis that the fact that the model incorporates only 21 runs could lead to the small values. We can make a very similar assertion for bert-base-cased. When comparing the fine-tuning times of the remaining models, we see that distilbert-base-uncased has the smallest average fine-tuning time and the smallest median and minimum. Nonetheless, when regarding the standard deviation and the maximum, albert-base-v1 shows smaller values, which could also stem from the difference in the counts of both models. Hence, it can be said that both models evince small fine-tuning time values in comparison to bert-base-uncased, roberta-base and xlnet-base-cased. Bert-base-uncased has the largest average fine-tuning time as well as standard deviation. Roberta-base displays very similar values. This could originate from the large outliers, which can be seen in figure 11. In contrary to this, xlnet-base-cased has the highest median, a small standard deviation and only few outliers. Hereby, it should be noted that xlnet-base-cased only contains 36 runs, while bert-base-uncased and roberta-base each contain 54 runs. Therefore, the smaller average fine-tuning time of xlnet-base-cased could stem from the fact that we did not perform a fine-tuning on the largest tasks CoLA and STS-B.

In the next step, we only regard the fine-tuning time for the tasks WNLI, RTE, MRPC and STS-B to generate an equal data basis for the models distilbert-base-

uncased, bert-base-uncased, roberta-base and xlnet-base-cased, which enables a fair comparison of the fine-tuning time among these models.

model	count	mean	std	min	median	max
distilbert-base-uncased	36	572.3	532.3	40.3	392.0	2285.6
albert-base-v1	36	1075.3	1070.6	75.5	627.2	4442.8
bert-base-uncased	36	1179.4	1164.5	79.5	733.1	5132.8
roberta-base	36	1153.7	1045.5	79.6	780.4	4056.4
xlnet-base-cased	36	**3611.9**	4097.2	147.4	**1848.7**	16341.8

Table 5: Descriptive analysis of fine-tuning time by model without SST-2 and CoLA

In table 5, distilbert-base-uncased exhibits the smallest statistical figures for the fine-tuning time. All displayed key values of albert-base-v1 are approximately two times larger than the one of distilbert-base-uncased. The fine-tuning time of roberta-base is the third largest among the displayed models, followed by bert-base-uncased and xlnet-base-cased. It should be noted that in this table (table 5), in contradiction to the one before (table 4), the model xlnet-base-cased exhibits the largest values for the fine-tuning time mean, median and standard deviation. This leads to the assumption that xlnet-base-cased takes the longest fine-tuning time (the fine-tuning time median is more than twice as high as the fine-tuning time median of roberta-base), when keeping the other variables constant. Furthermore, the differences in fine-tuning times between the models roberta-base, bert-base-uncased and albert-base-v1 are not as large, when comparing them to the fine-tuning times of distilbert-base-uncased and xlnet-base-cased. Hence, it could be stated that the use of XLNet leads to rather high and applying DistilBERT to rather low fine-tuning times.

In order to compare the cased model versions of BERT and DistilBERT with their respective uncased versions, we only regard a fine-tuning of the models on the tasks RTE and CoLA in order to enable a fair comparison. This results in a count of 18 observations (see table 6).

model	count	mean	std	min	median	max
distilbert-base-cased	18	984.6	876.5	154.8	**732.4**	3164.9
distilbert-base-uncased	18	**990.6**	901.4	166.2	714.2	3180.0
bert-base-cased	18	**2032.2**	1835.9	323.2	**1491.7**	6485.4
bert-base-uncased	18	2021.6	1883.0	311.8	1462.7	6851.5

Table 6: Descriptive analysis of fine-tuning time by model only with RTE and CoLA

Having a look at table 6, we can see that the fine-tuning times of the cased and the uncased version of both, DistilBERT and BERT, respectively, do not exhibit large differences. In both cases the cased version shows higher a higher median. Nonetheless, no clear statement can be made about whether the uncased or the cased version leads to larger fine-tuning times since all other statistical key figures for distilbert-case-uncased for example are higher than the ones of the cased version. All in all, this leads to the assumption that it makes little difference whether the cased or uncased version of a model is used.

6.1.3 Comparison across maximum sequence length

Another important hyperparameter that could have an effect on the fine-tuning time is the maximum sequence length. The following plot (figure 12) displays the boxplots of fine-tuning time grouped by the maximum sequence length, which can take on the values 128, 256 or 512 in the scope of the analysis.

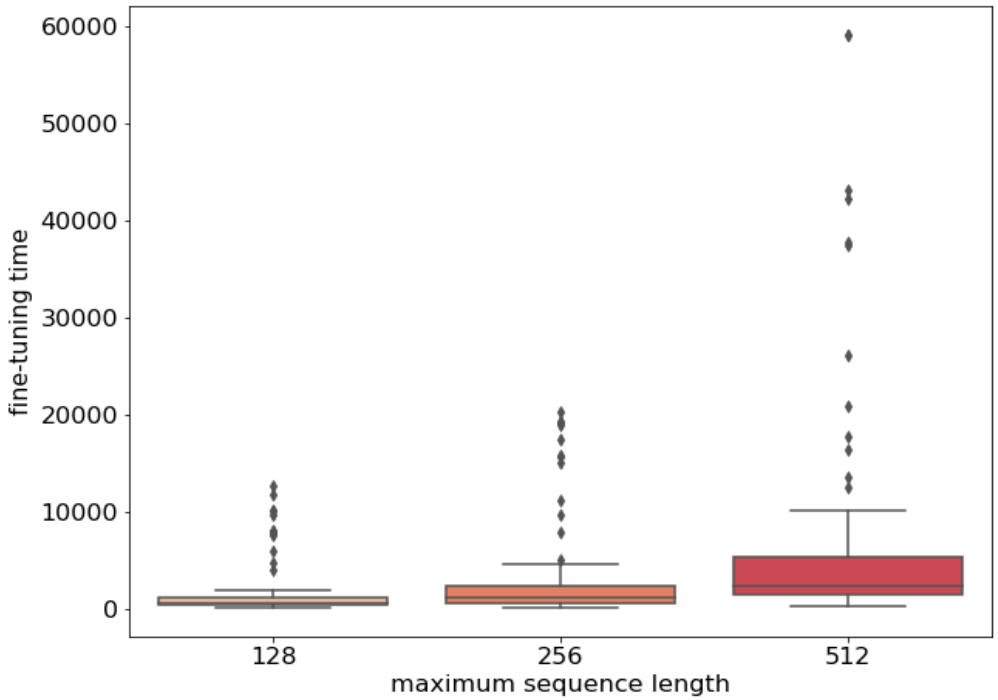

Figure 12: Boxplot of fine-tuning time grouped by maximum sequence length

The boxplots show a clear trend which indicates that the larger the maximum sequence length is, the larger the fine-tuning time is. Next, a more detailed look at the statistical key figures of the fine-tuning time grouped by the maximum sequence length is illustrated.

max. seq. len.	**count**	**mean**	**std**	**min**	**median**	**max**
128	96	1620.7	2744.9	40.3	627.2	12611.0
256	96	3020.0	5133.4	73.7	1133.5	20238.7
512	99	**6325.7**	11336.0	169.2	**2314.6**	59139.1

Table 7: Descriptive analysis of fine-tuning time by maximum sequence length; max. seq. len. in table stands for maximum sequence length

Table 7 shows that when doubling the maximum sequence length, the fine-tuning time also demonstrates an approximately doubled value. Hence, the fine-tuning

time is the smallest for a maximum sequence length of 128 and the largest for 512, which indicates a linear effect of the maximum sequence length on the fine-tuning time.

6.1.4 Comparison across batch size

Another hyperparameter that could have an effect on the fine-tuning time is the batch size. A visualization of the fine-tuning time grouped by the batch size is displayed in the following plot.

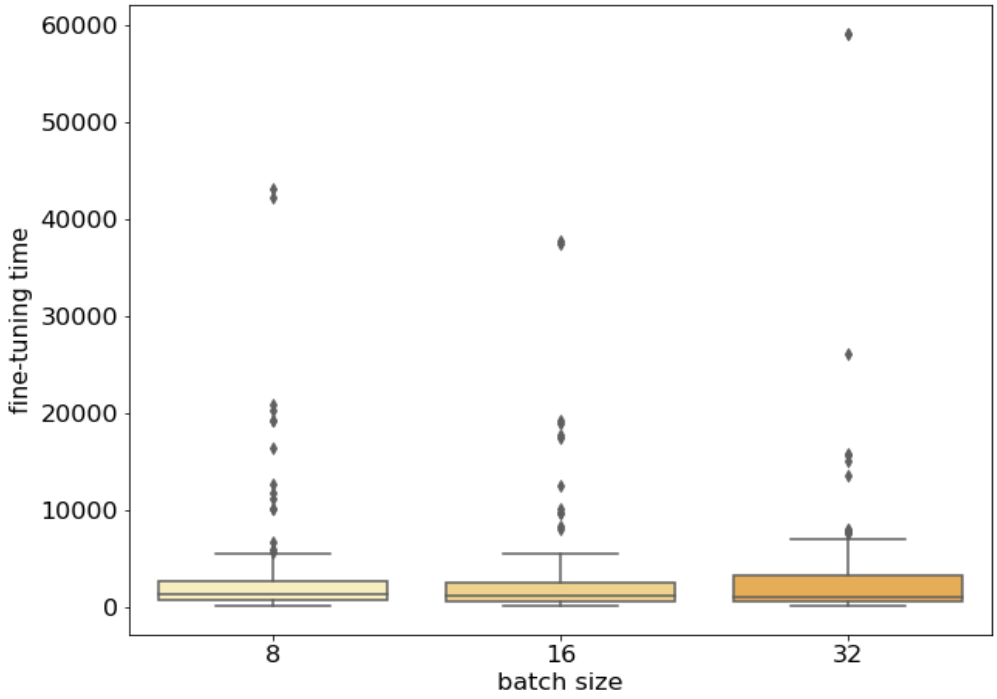

Figure 13: Boxplot of fine-tuning time grouped by batch size

Having a look at figure 13, the regarded groups do not exhibit any large differences. Nonetheless, one can see that the median of the fine-tuning time decreases with an increasing batch size. In the next step, we investigate the fine-tuning time grouped by batch size in more detail (table 8).

batch size	count	mean	std	min	median	max
8	97	3793.0	7286.2	64.3	1318.7	43090.0
16	97	3341.0	6447.1	48.0	1135.6	37749.2
32	97	3886.4	9055.7	40.3	1033.9	59139.1

Table 8: Descriptive analysis of fine-tuning time by batch size

Table 8 exhibits a decline of the fine-tuning time median with increasing batch size. Interesting to note is that this does not apply for the remaining values. The fine-tuning time mean, for example for a batch size of 16, is the smallest, followed by the fine-tuning time for a batch size of 8 and 32. Since a larger batch size results in less batches, which lead to less parameter and learning rate updates and hence, less calculations, the monitored effect does not match with the intuition. We would expect to monitor a decline of the fine-tuning time with increasing batch size.

To further investigate the observed effect, we consider a more detailed table, in which we grouped the fine-tuning time additionally by the maximum sequence length. Moreover, we do not include the GLUE tasks MRPC and SST-2 since the fine-tuning based on these tasks was not performed for all maximum sequence lengths. Hereby, only the mean, the median and the standard deviation are displayed (the count is for each row 23).

max. seq. length	batch size	mean	std	median
128	8	738.9	547.5	**614.0**
	16	563.2	411.5	431.7
	32	536.5	454.4	368.4
256	8	1257.0	1002.3	**1000.8**
	16	1269.8	1151.5	815.1
	32	1095.4	1018.2	665.2
512	8	3219.4	3467.5	1916.3
	16	745.9	2741.0	1542.9
	32	3587.6	3107.9	**2285.6**

Table 9: Descriptive analysis of fine-tuning time by batch size and maximum sequence length. The tasks MRPC and SST-2 are excluded

When regarding the fine-tuning time median for each batch size per maximum sequence length, it becomes clear that the median decreases with an increasing batch size for the maximum sequence length. Having a closer look at the maximum sequence length 512, we can detect that when applying a batch size of 32, the highest fine-tuning time median across the different batch sizes appears. The smallest fine-tuning time median occurs when using a batch size of 16, followed by the median for a batch size of 8 (for a maximum sequence length of 512). This effect contradicts the addressed intuition, which can also be observed for the remaining maximum sequence lengths (128 and 256), namely that the fine-tuning time decreases with an increasing batch size. This could be attributed to the fact that with a larger sequence length, the frequency of padding tokens[38] increases, which might lead to a higher convergence time, which then might lead to irritating effects. However, this is a vague hypothesis which cannot be verified and should definitely be scrutinized.

[38]Padding, i.e. adding padding tokens [PAD], is performed by the tokenizers in *huggingface* when the actual sequence is shorter than the maximum sequence length. For more insights, see https://huggingface.co/transformers/main_classes/tokenizer.html

6.2 Correlation matrix

In the next step, we regard the Pearson correlations between the batch size, the maximum sequence length, the task size and the fine-tuning time (figure 14).

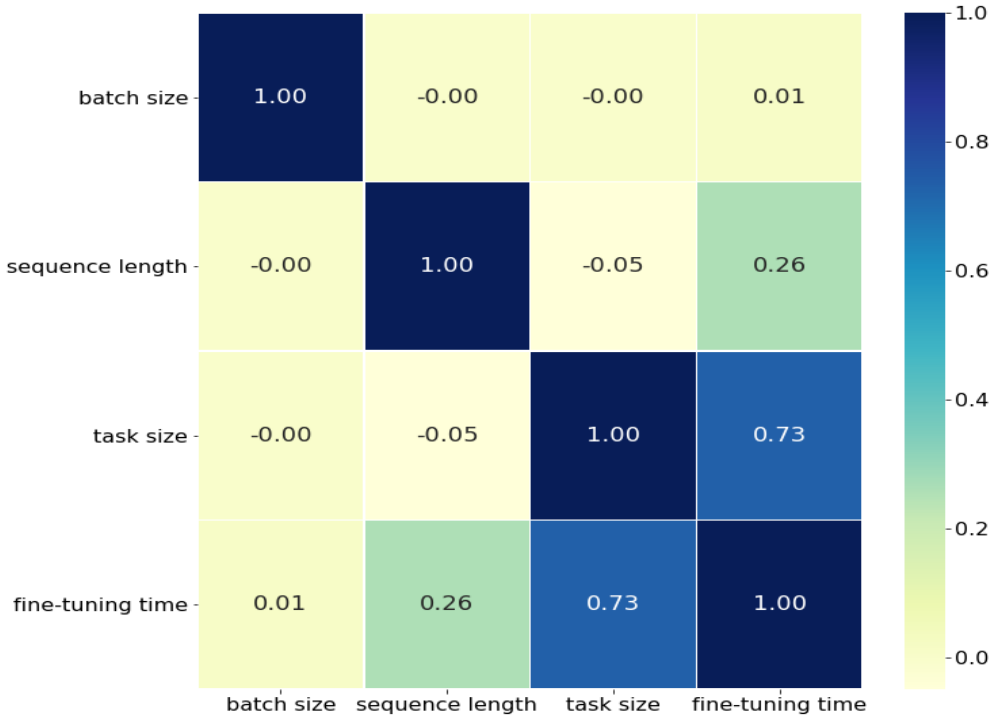

Figure 14: Correlation plot of fine-tuning time, number of examples per task, maximum sequence length and batch size

We can see that the correlation between the fine-tuning time and the task size is, with a positive Pearson correlation of 0.73, the highest among the displayed correlations. This indicates a strong positive relationship between the fine-tuning time and the task size. Furthermore, a rather moderate positive correlation can be detected between the maximum sequence length and the fine-tuning time. The remaining correlations indicate no or very weak relationships between the variables. Due to the fact that the model represents a categorical variable, it is not included in the correlation matrix. To determine if there exists a relationship between the fine-tuning time and the model, an ANOVA is generated in the following analysis (section 6.3). This is also carried out for the other variables.

In summary, we can state that the task size, the maximum sequence length and model have an effect on the fine-tuning time. The batch size seems to be of secondary importance. In order to make further investigations, we generate a one-way ANOVA for each hyperparameter and perform a log-linear regression (section 6.4 and 6.3).

6.3 ANOVA

In the next step, we regard ANOVAs for the fine-tuning time with the respective hyperparameters as groups. An ANOVA tests if the mean of a metric variable (here: fine-tuning time) shows significant differences across three or more groups. Hereby, a test for overall difference is performed, meaning that at least one of the groups shows statistical difference from the other groups but it does not tell which group it is. The requirements to actually conduct an ANOVA, being a normal distribution of the residuals, homoscedasticity and independence of samples, could not be met[39]. But since the ANOVAs we present in the following serve solely as an additional indicator for the effect of hyperparameters on the fine-tuning time, the violation of the assumptions is regarded as not severe. Furthermore, the unequal data basis for the respective groups should always be kept in mind. Hence, the following statements should be taken with a grain of salt. Due to the fact that we only regard three values for the hyperparameters batch size and maximum sequence length, respectively, we transform those hyperparameters to categorical variables. This enables performing an ANOVA for the fine-tuning time for each variable, which we analyze in the following in more detail.

First, we have a look at the ANOVA for the fine-tuning time grouped by the GLUE tasks, which is displayed in table 10.

	sum of squares	**df**	**F**	**PR(>F)**
between samples	9.156+09	5	66.742	5.535e-46
within samples	7.820e+09	285	-	-

Table 10: ANOVA of fine-tuning time, groups: GLUE tasks

[39]see appendix section 8 for more details

The ANOVA shows a significant result. Hence, at least two of the six task groups (WNLI, RTE, MRPC, STS-B, CoLA and SST-2) differ from each other, i.e. the fine-tuning time mean shows significant differences for at least two of the tasks. For a more accurate statement, post-hoc tests need to be performed, which are skipped since their reliability would be questionable.

The next ANOVA (table 11) regards the fine-tuning time grouped by the models.

	sum of squares	**df**	**F**	**PR(>F)**
between samples	6.348e+08	6	1.839	0.0915
within samples	1.634e+10	284	-	-

Table 11: ANOVA of fine-tuning time, groups: models

This ANOVA displays a p-value of 0.0915. Depending on how the significance level is determined, this indicates a significant (if $\alpha = 0.1$) or not significant (if $\alpha = 0.05$) result. The possible non-significance could be attributed to the unbalanced data which especially occurs when regarding the model groups. Hence, it is difficult to make a clear statement.

In the next step, we have a look at the ANOVA for the fine-tuning time grouped by the maximum sequence length.

	sum of squares	**df**	**F**	**PR(>F)**
between samples	1.157e+09	2	10.537	0.000038
within samples	1.582e+10	288	-	-

Table 12: ANOVA of fine-tuning time, groups: maximum sequence lengths

Table 12 shows a significant result, which means that at least two of the three groups (128, 256 and 512) significantly differ from each other. Therefore, the fine-tuning time mean differs for at least two levels of the variable maximum sequence length.

Lastly, an ANOVA for the fine-tuning time and the batch sizes is examined.

	sum of squares	**df**	**F**	**PR(>F)**
between samples	1.630e+07	2	0.138	0.8708
within samples	1.696e+10	284	-	-

Table 13: ANOVA of fine-tuning time, groups: batch sizes

The ANOVA (table 13) shows non-significant results, which leads to the statement that there is no statistically significant difference in the fine-tuning time mean for the different levels of batch size (8, 16 and 32). Even though this does not match the general intuition, as already addressed, it matches the results from the descriptive analysis of the fine-tuning time.

All in all, we could see that the results match the observations that were made during the descriptive analysis, namely that the tasks as well as the maximum sequence length seem to have a large effect on the fine-tuning time. In contradiction to that, the batch size does not seem to have a high effect on the fine-tuning time. Lastly, it is difficult to make a statement for the model since we assume that the disbalance which is present in the data influences the model data in a non-negligible manner.

6.4 Log-Linear Regression

In the next step, we set up a log-linear regression model as an add-on to the descriptive analysis. Additional test of the assumptions, which should be met to perform a linear regression, can be viewed in the appendix, section 8. Not all assumptions could be met but since this regression serves as an explanatory basis and the effect size is of main interest, meeting all of the assumptions is of secondary importance. In the conducted regression, the logarithmic fine-tuning time is the dependent variable and the hyperparameters are included in the model as independent variables. The model variables are included as dummy variables with bert-base-uncased as the reference category. Hence, the interpretations of the β-coefficients are carried out with respect to this model. The same procedure is done for the task variable, RTE being the reference category. Furthermore, the hyperparameters batch size and maximum sequence length are included in the model.

As already hinted, the dependent variable fine-tuning time was logarithmized[40] due to its right skewed distribution, whereby an approximately normal variable is aspired. Another goal which is hereby pursued, is to approach linearity between the independent variables and the dependent variable. When regarding a linear regression without transforming the variables, the R^2 results in 0.659, whereas the model with the variable transformations leads to an R^2 of 0.984. Hence, the log-linear model is regarded in the following.

In the first step, we examine some key values of the transformed variable in order to have a better intuition for it.

Transformed variable	**mean**	**std**	**min**	**median**	**max**
log(fine-tuning time)	7.146	1.420	3.696	7.086	10.988

Table 14: Descriptive analysis of log fine-tuning time

The logarithmic fine-tuning time ranges from approximately 3.7 to 11.0 and its median takes on the approximate value of 7.1.

While keeping in mind that the log-linear regression should not be taken as a face value, the respective coefficients are regarded.

[40] all logarithmic transformations in this context refer to the natural logarithms

variable	coeff	standard error	p-value
Intercept	5.5126	0.045	0.000
model[albert-base-v1]	-0.0953	0.036	0.009
model[bert-base-cased]	0.0127	0.050	0.779
model[distilbert-base-cased]	**-0.7139**	0.050	0.000
model[distilbert-base-uncased]	-0.6955	0.036	0.000
model[roberta-base]	0.0075	0.036	0.835
model[xlnet-base-cased]	**0.9598**	0.041	0.000
task[WNLI]	**-1.3674**	0.038	0.000
task[MRPC]	0.3593	0.036	0.000
task[STS-B]	0.8380	0.038	0.000
task[CoLA]	1.2047	0.035	0.000
task[SST-2]	**3.2659**	0.042	0.000
batch size	-0.0041	0.001	0.000
max. sequence length	0.0040	6.86e-05	0.000

Table 15: Log-linear regression model: β-coefficients

Table 15 displays the β-coefficient of the log-linear regression model. The according p-values of the coefficients should be regarded with caution and are not necessarily reliable since not all assumptions for conducting a linear regression could be satisfied. Having a look at the model coefficients, one can see that xlnet-base-cased has the largest positive effect on the logarithmic fine-tuning time among the models and compared to the model bert-base-uncased for the task RTE (c.p.). It can be interpreted in the following manner: The model xlnet-base-cased results in a fine-tuning time that is approximately $\exp(0.9589) - 1 = 1.611$ times larger[41] than the one of bert-base-uncased for the task RTE, while holding the other variables constant. The contrary applies when fine-tuning distilbert-base-cased. It results in a fine-tuning time decrease compared to bert-base-uncased for RTE (c.p.). More precisely the fine-tuning of distilbert-base-cased on RTE results in a $1 - \exp(-0.7139) = 1 - 0.4897 = 0.5103$ times smaller fine-tuning time than for bert-base-uncased and RTE (c.p.). The other coefficients can be interpreted in a similar manner. Interesting to note is that the coefficients for bert-base-cased

[41] or 0.9589 times larger than the logarithmic time

and roberta-base take on low values and are not significant, which matches the observations from the descriptive analysis since these fine-tuning times did not show a high difference in comparison to bert-base-uncased. Furthermore, we can see that the only task coefficient with a negative sign is WNLI, which is reasonable since WNLI contains less examples than RTE. The other tasks all incorporate more observations than RTE and therefore contribute to a larger fine-tuning time for bert-base-uncased, in comparison to the GLUE task RTE (c.p.). The coefficient with the highest value among the task coefficients (and among all β-coefficients) is the one of SST-2, the largest task in this setting. Hence, if a fine-tuning bert-base-uncased on the SST-2 task is performed, the fine-tuning time is approximately $\exp(3.2659) - 1 = 26.204$ times larger than a fine-tuning of bert-base-uncased performed on RTE (c.p.). Furthermore, the batch size shows a negative β-coefficient, indicating that when increasing the batch size by one unit for the fine-tuning of bert-base-uncased on RTE, the fine-tuning time is about $1 - \exp(-0.0041) = 1 - 0.9959 = 0.0041$ times smaller (c.p). When regarding a 32-unit increase (which makes sense in our case), then the fine-tuning time is about $1 - \exp(-0.0041 \cdot 32) = 0.123 = 12.3\%$ smaller for bert-base-uncased and RTE (c.p.). When increasing the maximum sequence length by 128 units (which is reasonable with regard to the context), the fine-tuning time for bert-base-uncased and RTE shows an approximate increase of about $1 - \exp(0.004 \cdot 128) = 0.6686 = 66.86\%$ (c.p.).

All in all, we can state that the β-coefficients of the log-linear regression match the assumptions drawn from the descriptive analysis, namely that the batch size only has a small effect on the fine-tuning time, whereas the task size as well as the maximum sequence length play an important role for the fine-tuning time. The larger the task size is, the higher the fine-tuning time is. The same relation applies for the maximum sequence length. While some models do not show high differences in the fine-tuning time (roberta-base, albert-base-v1 and both BERT model versions), others show high differences (xlnet-base-cased and both DistilBERT model versions). We additionally implemented a Random Forest in order to get an intuition of the variable importance, which displays similar results (see appendix 8 for more details).

6.5 Accuracy and fine-tuning time

Lastly, an interesting aspect to examine is the relation between fine-tuning time and performance measure. In the following, the performance measure accuracy is regarded which was calculated for the tasks WNLI, RTE, MRPC and SST-2. For CoLA, Matthews correlation coefficient and for STS-B, the Spearman and the Pearson correlation coefficient were computed. Since we only have 54 and 45 data points for these measures, respectively, it is hard to make assumptions on the relation between them and the fine-tuning time. Since the accuracy is calculated for 192 observations, we regard the relation between the accuracy and the fine-tuning time in the following (figure 15).

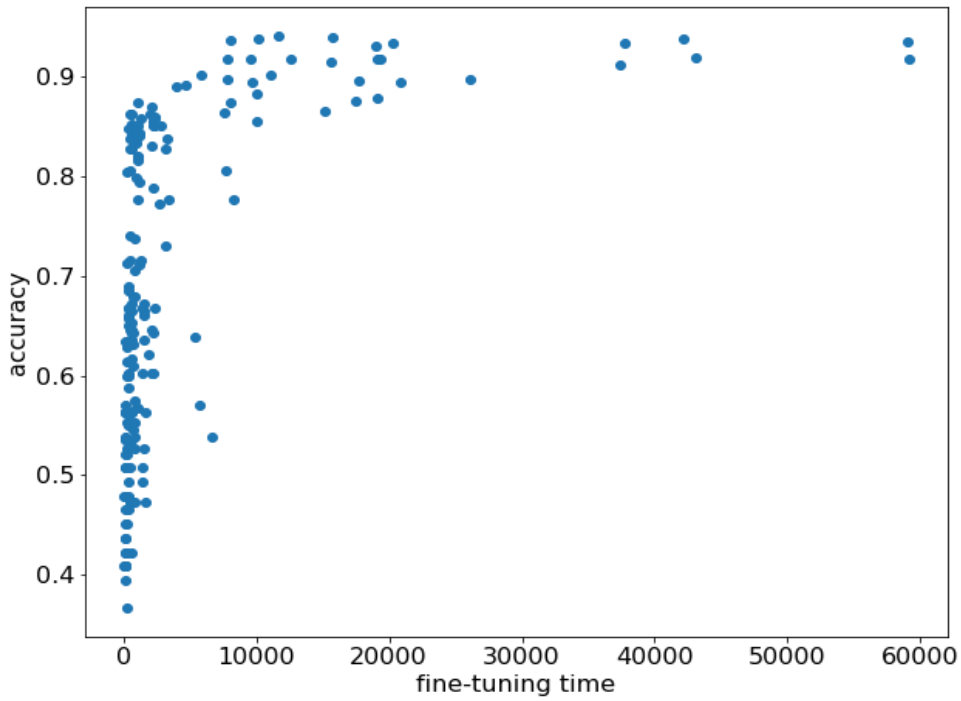

Figure 15: Scatterplot between fine-tuning time and accuracy

The Scatterplot in figure 15 displays a clear trend, namely that the accuracy increases with higher fine-tuning time. Furthermore, the Pearson correlation between the fine-tuning time and the accuracy amounts to approximately 0.52, which

indicates a moderate positive relationship between both. This statements should be treated with caution since not the fine-tuning time itself but factors like the model complexity and task size which lead to an increased fine-tuning time could attribute to this trend. Nonetheless, the displayed trend matches the intuition that longer training leads to higher accuracy.

7 Analysis of inference time

The following chapter covers the analysis of inference time. For this, a fine-tuning of the same large pre-trained language models that were described in section 6.1 was performed. First, a descriptive analysis of the inference time grouped by the respective hyperparameters, is carried out. Secondly, a correlation matrix is displayed, followed by ANOVAs, for which the inference time is grouped by the hyperparameters. The penultimate section consists of a log-linear regression. Lastly, the relation between the inference time and the accuracy is regarded.

7.1 Descriptive analysis

For this analysis, a fine-tuning based on the tasks WNLI, RTE, MRPC, CoLA, and STS-B was performed, whereby a fine-tuning of the tasks QNL, QQP and MNLI failed (like in the analysis of the fine-tuning time), probably due to lack of RAM. Due to computational costs and temporal limitation of this master's thesis, a fine-tuning of the language models based on SST-2 could not be carried out. The final dataset that we use for further analyses contains 267 observations in total. The main variable of interest is the inference time. Hereby, the fine-tuning was not carried out multiple times any more due to the fact that by performing the fine-tuning one time, inference is carried out multiple times (dependent on the number of batches) in order to perform an evaluation of the regarded model. The input size for the evaluation and hence, the inference depends on the regarded batch size. Consequently, the number of inference times also depends on the batch size. The smaller the batch size is, the more batches are generated and the higher the number of performed inferences is. Hence, one then receives a higher number of measurements for the inference time. The same argument applies for the task size. A bigger task size creates more batches (assuming one regards the same batch size) and hence more measurements of the inference time. It should be noted that the input is based on the development set of the respective task. For example, when regarding the development set of WNLI which contains 71 observations and applying a batch size of 32 for the evaluation, we receive approximately 2.22 batches, which result in three batches. The first two batches contain 32 observations, while the last batch contains 7 observations. The three

resulting inference times differ mainly due to the different input size. Since the last batch usually contains less observations, we do not regard the last inference time and take the minimum of the prior inference times (for one specific hyperparameter combination). In the following, we analyze the minimum fine-tuning time per hyperparameter combination. The inference time refers to the minimal inference time if not stated otherwise.

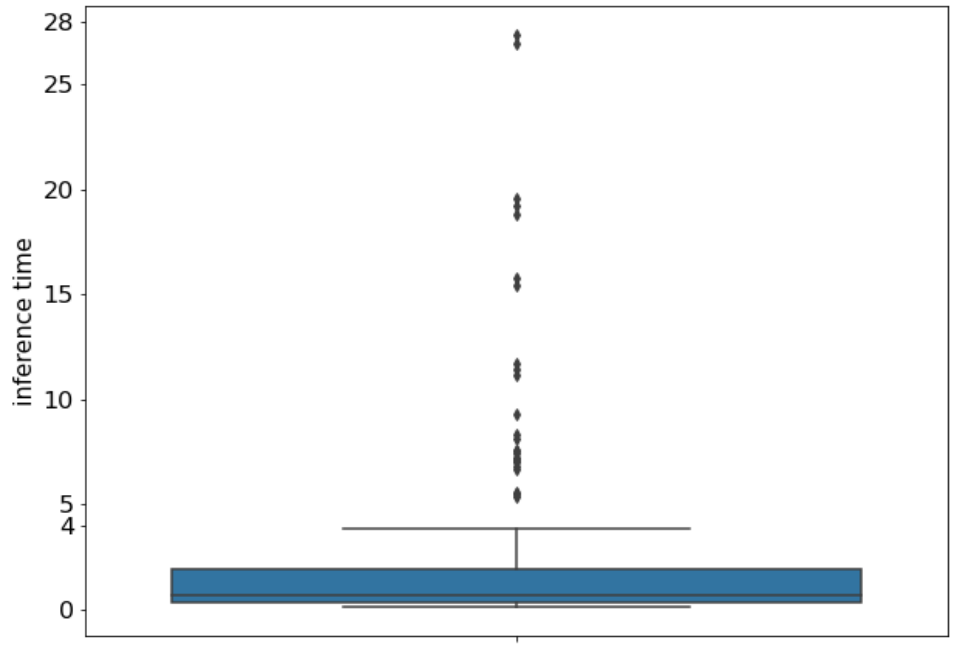

Figure 16: Boxplot of inference time in seconds

In figure 16, the boxplot of the inference time, measured in seconds, is visualized. The weld point of the distribution lies between the lower whisker (0.092 [42]), which is also the minimum in this case, and the upper whisker (3.832). The median takes on the value 0.628. The boxplot displays some outliers which reach values up to the maximum of 27.374. Further important statistical key figures are the mean and the standard deviation which take on the values 1.974 and 3.734, respectively. A more detailed table of the ten highest as well as the ten lowest inference times can

[42] the values are rounded up to three decimals for the inference time

be found in the appendix (see table 34 and 35). Note that the table with the ten lowest inference times solely exhibits the model xlnet-base-cased with a maximum sequence length of 512. Furthermore, the batch size is 32 for the five highest inference time and later takes on the value 16. On the other hand, the table with the ten lowest inference times exhibits the two variants of the DistilBERT$_{\text{BASE}}$, a maximum sequence length of 128 and a batch size that mostly takes on the value 8. Both observations intuitively makes sense and give a first intuition about which factors might have an impact on the inference time.

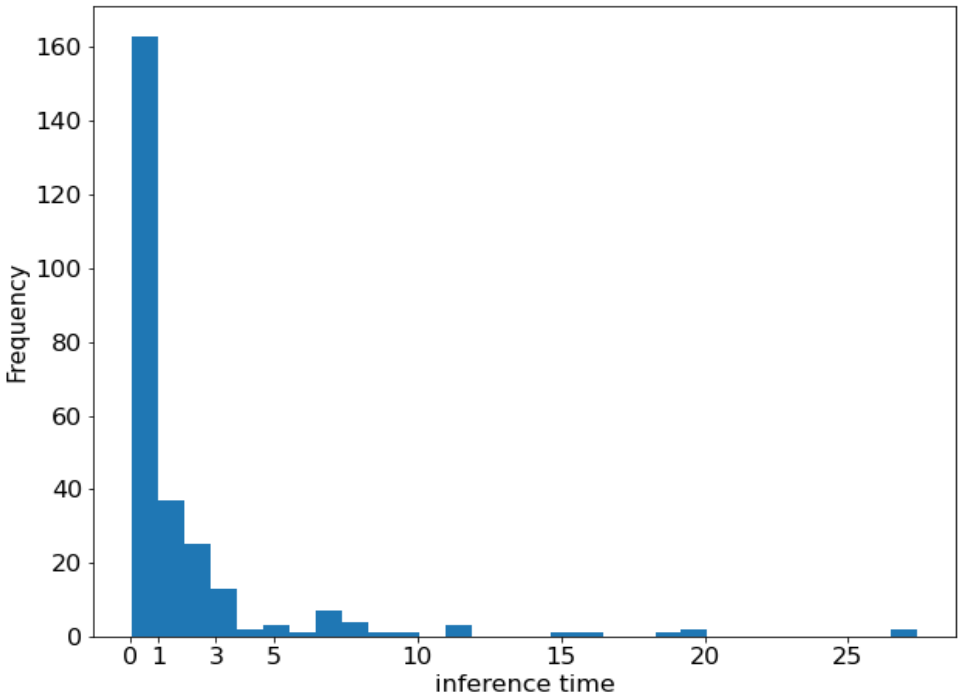

Figure 17: Histogram of inference time in seconds

At first glance, we detect a right skewed distribution of the inference time in figure 17. Hereby, the first bin contains 193 observations (out of 267) and hence, most of the inference times take on values between about 0.092 and 1.456. The second bin, ranging from 1.456 to 2.820, incorporates 32 observations, followed by the third bin which contains values up to 4.183 with 15 data points. The remaining bins contain mostly up to three data points, except for one which contains nine. This

underpins the already addressed right skewed distribution. We should keep in mind that due to computational limitations, more hyperparameter combinations with higher sequence lengths and batch sizes could not be regarded, which also partly leads to this distribution.

The same computational problems/limitations that are described in section 6.1 apply here. Due to the fact that a fine-tuning time of the task SST-2 requires a long computation time, we did not perform a fine-tuning on this task in order to analyze the inference time. Furthermore, we performed a fine-tuning of XLNet on CoLA additional to the already described training. This results in a dataset with 267 examples.

7.1.1 Comparison across GLUE tasks

Since the measurement of inference time is incorporated in the prediction loop function, which is then utilized for the evaluation, a different dataset is regarded than during the fine-tuning itself, namely the development set of each task. Note that the respective dataset size is different from the according training data. WNLI still represents the smallest dataset with a task size of 71 examples, followed by RTE with 277 examples. MRPC contains 408 sequences, whereby CoLA contains 1,042 and STS-B is the largest dataset in this setting with 1,500 examples.

In the following graphic, the inference time is grouped by the respective task size number that corresponds to a specific task. The colors of the boxplots correspond to the respective tasks.

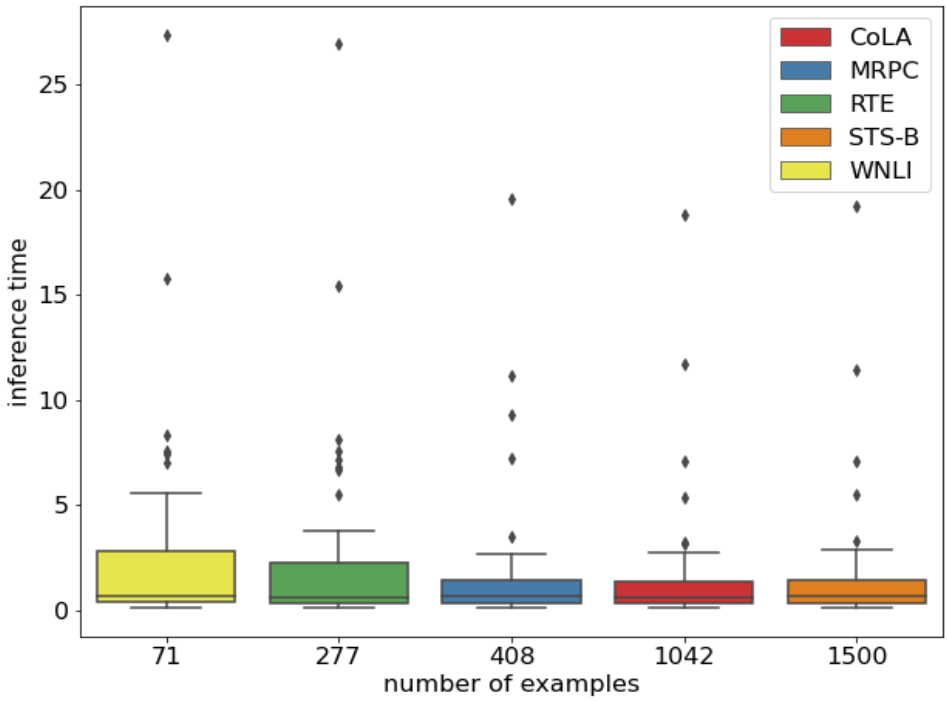

Figure 18: Boxplot of inference time grouped by number of examples per task

In figure 18, we can see that with a larger task size, the standard deviation of the inference time decreases. This might stem from the fact that the larger a task is, the more batches are generated which leads to an increased number of inference time measurements, making the minimum inference time more stable. The smallest tasks WNLI and RTE incorporate the largest outliers. Important to note is the fact that one cannot rely on the accuracy of the decimal points when measuring time (Python Software Foundation, 2020b) which aggravates concrete statements about the inference time. This should be kept in mind during the whole analysis.
In the following table, the inference time grouped by task is investigated in more detail with the help of various statistical key figures.

task name	task size	count	mean	std	min	median	max
WNLI	71	45	**2.684**	4.836	0.093	**0.691**	27.374
RTE	277	63	2.218	4.147	0.095	0.618	26.956
MRPC	408	51	1.791	3.334	0.096	0.649	19.563
CoLA	1500	54	1.491	2.883	0.092	0.564	18.843
STS-B	1042	45	1.810	3.366	0.094	0.653	19.205

Table 16: Descriptive analysis of inference time by tasks

Having a look at the key figures in table 16, we see that the inference time mean is the largest for the smallest tasks WNLI and RTE which can be attributed to the large outliers. The standard deviation is higher for smaller batch sizes. When having a look at the median, this trend cannot be clearly read out, which aggravates a clear statement about the effect of the task on the inference time.

Since the models albert-base-cased, distilbert-base-cased and bert-base-cased could not be fine-tuned on some datasets and some maximum sequence lengths (see section 6.1), we exclude them for the following comparison, which is displayed in table 17, to enable a fair comparison of the inference time across the GLUE tasks.

task name	task size	count	mean	std	min	median	max
WNLI	71	36	**2.899**	5.278	0.093	0.715	27.374
RTE	277	36	2.855	5.177	0.097	**0.726**	26.956
MRPC	408	36	2.133	3.904	0.096	0.648	19.563
CoLA	1500	36	2.028	3.692	0.092	0.586	18.843
STS-B	1042	36	2.040	3.718	0.094	0.674	19.205

Table 17: Descriptive analysis of inference time by tasks, without the models albert-base-cased, distilbert-base-cased and bert-base-cased

Table 17 exhibits that the task size appears to not necessarily have an effect on the inference time, which makes sense because the input size depends on the batch size and not on the task size since the examples are split into batches (of a certain size). RTE shows the highest inference time median, whereby CoLA incorporates the smallest inference time median. In summary, it can be said that no clear pattern can be detected at first glance.

7.1.2 Comparison across large pre-trained LMs

An important aspect influencing the inference time could be the choice of the model. Note that the measuring of inference time is performed on models which were fine-tuned beforehand on the different GLUE tasks. The following figure visualizes the inference time grouped by model.

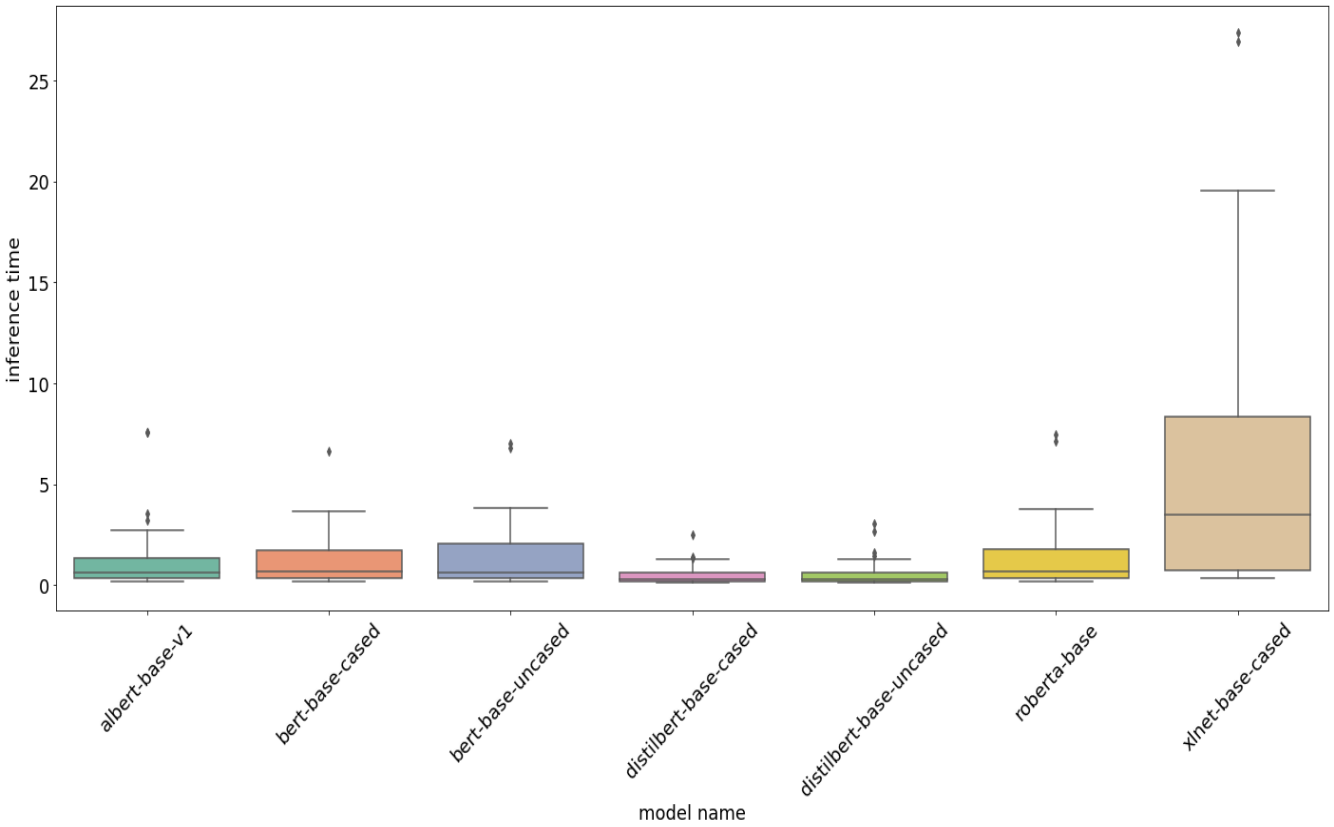

Figure 19: Boxplot of inference time grouped by models

Figure 19 displays a large inference time with a large standard deviation of xlnet-base-cased compared to the other models. The two versions of the DistilBERT$_{\text{BASE}}$ model show the smallest values for the inference time with the smallest standard deviation. Table 18 demonstrates the respective key figures in more detail.

model	count	mean	std	min	median	max
distilbert-base-cased	21	0.565	0.586	0.092	0.297	2.465
distilbert-base-uncased	45	0.561	0.636	0.092	0.287	3.062
albert-base-v1	45	1.238	1.616	0.182	0.633	7.570
bert-base-cased	21	1.387	1.529	0.181	0.658	6.643
bert-base-uncased	45	1.342	1.548	0.183	0.631	7.027
roberta-base	45	1.351	1.610	0.183	0.648	7.433
xlnet-base-cased	45	**6.310**	7.164	0.331	**3.474**	27.374

Table 18: Descriptive analysis of fine-tuning time by model

At first glance, we see that in comparison to the other models, both versions of the DistilBERT$_{\text{BASE}}$ model illustrate small inference times, whereas XLNet$_{\text{BASE}}$ incorporates large values. The inference times of the BERT$_{\text{BASE}}$ models, the ones of albert-base-v1 and roberta-base, are all situated in a similar value range. We made a similar observation during the analysis of the fine-tuning time and state that the use of XLNet leads to rather high inference times, whereas applying DistilBERT results in rather low inference times.

To enable a fair comparison of the inference time for the cased model versions of BERT and DistilBERT with their respective uncased versions, we solely regard the models on the tasks RTE and CoLA, which result in a count of 18 observations.

model	count	mean	std	min	median	max
distilbert-base-cased	18	0.540	0.609	0.092	0.279	2.465
distilbert-base-uncased	18	0.616	0.740	0.092	**0.319**	3.062
bert-base-cased	18	1.366	1.627	0.181	**0.622**	6.643
bert-base-uncased	18	1.416	1.650	0.183	0.618	6.800

Table 19: Descriptive analysis of fine-tuning time by model for RTE and CoLA

Table 19 shows that the statistical key figures of the uncased version of DistilBERT$_{\text{BASE}}$ regarding the inference time are higher than the ones of the cased version. The same can be stated for BERT$_{\text{BASE}}$, except for the median. Nonetheless, the differences (especially regarding the median) are little, which is why they

could also be attributed to measurement instabilities. This leads to the assumption that (as for the fine-tuning time) it makes litte difference whether we use the case or the uncased version.

7.1.3 Comparison across maximum sequence length

In the next step, we have a look at the inference time grouped by the maximum sequence length. This is visualized in the following figure (figure 20).

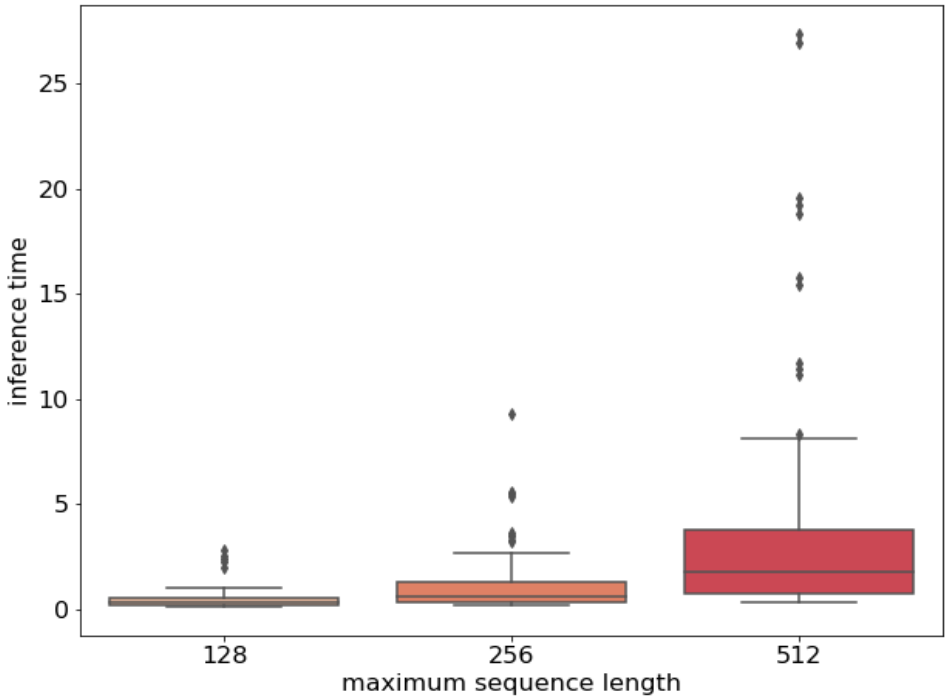

Figure 20: Boxplot of inference time grouped by maximum sequence length

The visualization exhibits a clear connection between the maximum sequence length and the inference time. This intuitively makes sense. The larger the sequences are (the more tokens they contain), the more there is to process during inference, which leads to a larger inference time. A more detailed view is represented below.

maximum sequence length	count	mean	std	min	median	max
128	87	0.458	0.517	0.092	0.322	2.769
256	87	1.242	1.553	0.150	0.616	9.252
512	93	**4.078**	5.537	0.287	**1.751**	27.375

Table 20: Descriptive analysis of fine-tuning time by batch size

This table underlines the statements above since all key figures of the inference time increase with a higher maximum sequence length. A doubling of the maximum sequence length 128 leads to an approximate doubling of the inference time median. Note that increasing the maximum sequence length from 256 to 512 results in statistical key figure of inference time that are even higher than the doubled value of their predecessors, e.g. the inference time mean in now more than three times higher.

In order to carry out a fair comparison across different maximum sequence lengths, we regard all values, except for the ones of the task MRPC since a fine-tuning of distilbert-base-cased and bert-base-cased failed for the maximum sequence lengths 128 and 256. This can be viewed in the following table.

maximum sequence length	count	mean	std	min	median	max
128	72	0.458	0.532	0.0915	0.316	2.769
256	72	1.174	1.35	0.150	0.601	5.581
512	72	**4.422**	5.764	0.287	**2.161**	27.374

Table 21: Descriptive analysis of inference time by maximum sequence length, excluding MRPC

Table 21 shows that the larger the maximum sequence length is, the higher the inference time is. The more tokens a sequence incorporates, the more there is to analyze, which results in a higher inference time. The observations stated above are exhibited even more in this table, which speaks for a clear connection between the inference time and the maximum sequence lenght.

7.1.4 Comparison across batch size

Finally, the inference time grouped by batch size is visualized in the following plot.

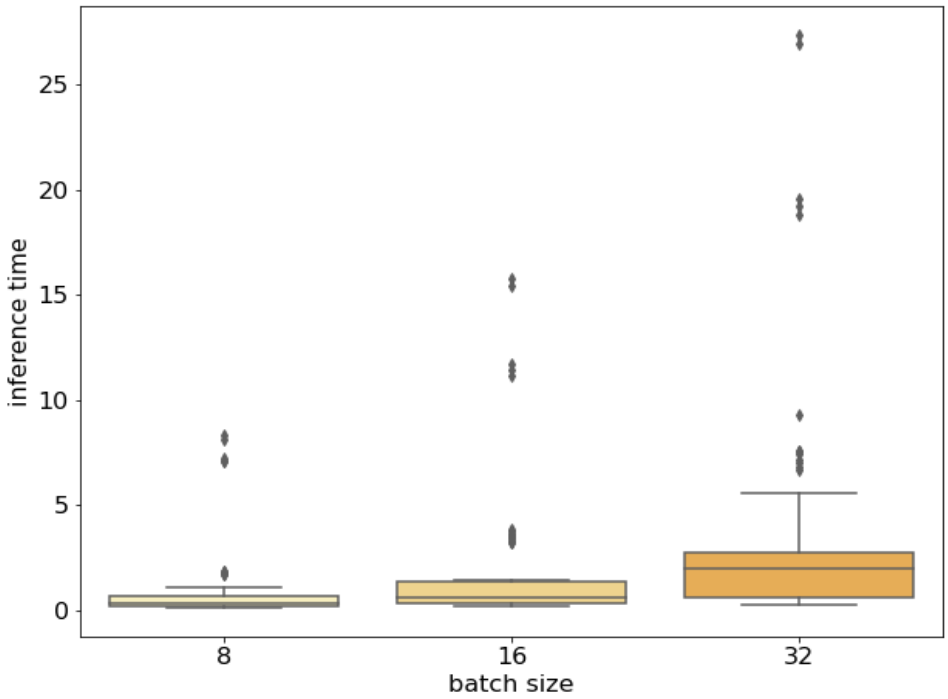

Figure 21: Boxplot of inference time grouped by batch size

Graphic 21 exhibits a clear trend, indicating that the batch size has a large effect on the inference time. A higher batch size results in a higher inference time. For a more in depth investigation of this statement, we examine the according statistical key figures in the following table.

batch size	count	mean	std	min	median	max
8	89	0.854	1.706	0.092	0.319	8.347
16	89	1.729	3.022	0.144	0.608	15.804
32	89	**3.340**	5.185	0.249	**1.942**	27.374

Table 22: Descriptive analysis of fine-tuning time by model

Note that a larger batch size means that the model evaluation is based on a larger input size, e.g. a batch size of 8 contains 8 sequences for which inference is performed. A clear effect can be seen in figure 21 and table 22, indicating that the inference time increases with larger batch sizes since all statistical key figures show this pattern. Having a look at the respective inference median one can see that the inference mean for batch size 16 is approximately twice as high as the one for batch size 8. The median inference time for a batch size of 32 is about three times larger than the one for batch size 16.

7.2 Correlation matrix

In the next step, we have a look at the Person correlations. The following correlation plot (figure 22) shows the Pearson correlation between the batch size, the sequence length, the task size and the inference time.

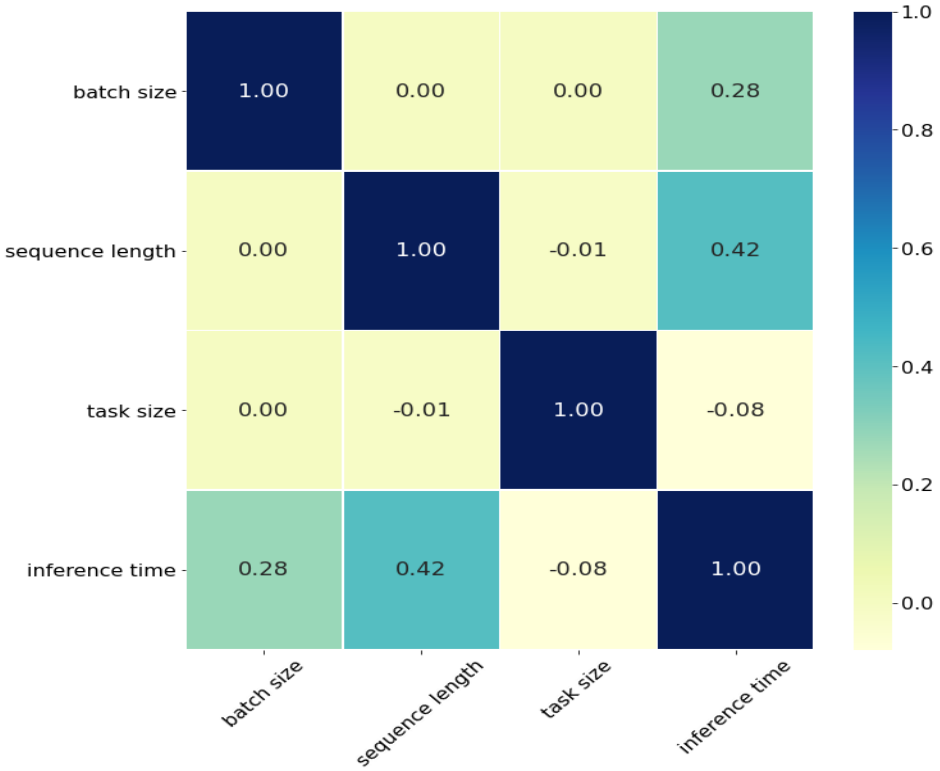

Figure 22: Correlation plot of inference time, number of examples per task, maximum sequence length and batch size

The correlation between the inference time and the sequence length takes on the highest value (0.42) among the presented correlations. This indicates a moderate positive relationship between both variables. Furthermore, we detect a positive moderate to small correlation between the inference time and the batch size. All other correlations are either zero or take on very low values. Since calculating correlations on categorical variables is not possible, the variable *model* is not displayed in this correlation matrix. To determine if there is a relationship between this variable and the inference time, we perform a one-way ANOVA in the following (section 7.3). This analysis is also performed for the other variables.

After carrying out the descriptive analysis for the inference time, we can state that the hyperparameters batch size and maximum sequence length have the largest influence on the inference time. While the choice of the model also has an effect on the inference time, the task itself seems to be of secondary importance. In order to further investigate these effects, an ANOVA and a log-linear regression are calculated in the following. These represent an add-on to the descriptive analysis and should not be taken at face value since not all of the assumptions for conducting ANOVAs and the log-linear regression, respectively, could not be met[43].

7.3 ANOVA

In the first step, an ANOVA of inference time grouped by the tasks (WNLI, RTE, MRPC, STS-B and CoLA) is regarded (table 23).

	sum of squares	**df**	**F**	**PR(>F)**
between samples	44.057	4	0.787	0.534
within samples	3664.997	262	-	-

Table 23: ANOVA of inference time, groups: GLUE tasks

This ANOVA shows non-significant results, which leads to the conclusion that no significant differences in the inference time mean can be detected for the different

[43]The test of the assumptions for performing a linear regression and an ANOVA, respectively, can be found in the appendix, section 8 and 8, respectively

tasks. This matches the findings of the descriptive analysis since clear trends could not be detected.

The next ANOVA regards the inference time grouped by the models.

	sum of squares	**df**	**F**	**PR(>F)**
between samples	1044.694	6	16.991	1.500e-16
within samples	2664.359	260	-	-

Table 24: ANOVA of inference time, groups: models

This ANOVA (table 24) exhibits significant results, which indicate a difference in the inference time mean between at least two of the seven models (the respective cased and uncased versions of $\text{BERT}_{\text{BASE}}$ and $\text{DistilBERT}_{\text{BASE}}$, albert-base-v1, roberta-base and xlnet-base-cased). To investigate concretely which groups differ from each other, post-hoc tests need to be performed, which are not carried out in this analysis since already the expressiveness of the ANOVAs should be scrutinized.

The following table displays an ANOVA of inference time grouped by the maximum sequence length.

	sum of squares	**df**	**F**	**PR(>F)**
between samples	658.221	2	28.479	6.317e-12
within samples	3050.832	264	-	-

Table 25: ANOVA of inference time, groups: maximum sequence lengths

The displayed ANOVA (table 25) exhibits significant results. Hence, the means of the inference time differ significantly for at least two of the three maximum sequence lengths (128, 256 and 512).

Lastly, the inference time grouped by batch size is examined.

	sum of squares	**df**	**F**	**PR(>F)**
between samples	282.980	2	10.903	0.000028
within samples	3426.07	264	-	-

Table 26: ANOVA of inference time, groups: batch sizes

The ANOVA (table 26), which shows the inference time with respect to the batch sizes, points out a significant p-value. Consequently, we can make the statement that the inference time means differ significantly at least for two of the three batch sizes (8, 16 and 32). Interesting to note is the fact that in comparison to the p-values of the two prior ANOVAs, this p-value is larger, which could be attributed to the fact that the distances between the batch sizes themselves are not very large.

7.4 Log-Linear Regression

In the present section a log-linear regression with the logarithmized inference time as the target variable is regarded as an add-on to the descriptive analysis. This regression is very similar to the one introduced in section 6.4, with the only exception that a different target variable, namely the logarithmized inference time, is regarded.
Similar to the log-linear regression that we carried out for the fine-tuning time, the target variable inference time was logarithmized[44]. We performed this transformation for the same reasons, namely, to ensure linearity and approach normality. When comparing a linear regression in which the inference time was not logarithmized with this log-linear regression, we can detect that the log-linear regression exhibits a better performance ($R^2 = 0.553$ vs $R^2 = 0.920$). Hence, we elucidate the log-linear regression in the following.

[44]all logarithmic transformations in this context refer to the natural logarithm

Before having a look at the β-coefficients of the log-linear regression, we regard the statistical key values of the transformed variables (table 27).

Transformed variable	**mean**	**std**	**min**	**median**	**max**
log(inference time)	-0.206	1.240	-2.391	-0.465	3.310

Table 27: Descriptive analysis of log inference time

Since taking the log of values between 0 and 1 results in negative values and the inference time exhibits many values within this range, some statistical figures take on negative values. The logarithmized inference time ranges from approximately -0.47 to 3.31 and its median takes on the approximate value of -0.46.

Keeping in mind that the log-linear regression serves as an add-on, the β-coefficients are examined in the following (table 28).

variable	**coeff**	**standard error**	**p-value**
Intercept	-2.6125	0.089	0.000
model[albert-base-v1]	-0.0795	0.076	0.295
model[bert-base-cased]	-0.0824	0.098	0.400
model[distilbert-base-cased]	**-0.9253**	0.098	0.000
model[distilbert-base-uncased]	-0.8215	0.076	0.000
model[roberta-base]	0.0037	0.076	0.961
model[xlnet-base-cased]	**1.2977**	0.076	0.000
task[WNLI]	0.0104	0.073	0.886
task[MRPC]	**-0.2899**	0.069	0.000
task[CoLA]	-0.2377	0.064	0.000
task[STS-B]	-0.2560	0.073	0.000
batch size	0.0585	0.002	0.000
max. sequence length	0.0049	0.000	0.000

Table 28: Log-linear regression model: β-coefficients

Table 28 exhibits the respective β-coefficients of the log-linear regression. As in section 6.4 already noted, the respective p-values should be regarded with

caution since not all assumptions for conducting a linear regression could be satisfied. The model reference category is bert-base-uncased and the task reference category is RTE. When regarding the β-coefficients of the models, we can see that xlnet-base-cased has the largest positive effect on the logarithmic inference time among all models, compared to the inference time bert-base-uncased on RTE (c.p.). More precisely, when fine-tuning the xlnet-base-cased on the task RTE, the inference time is about $\exp(1.2977) - 1 = 3.6609 - 1 = 2.6609$ times larger than the one of bert-base-uncased and RTE (c.p.). Furthermore, the model distilbert-base-cased for example has the opposite effect on the inference time (c.p.). When applying distilbert-base-cased on RTE, the inference time is about $1 - \exp(-0.9253) = 0.6036$ times smaller than the one of bert-base-uncased fine-tuned on RTE (c.p.). The other model coefficients can be interpreted in a similar manner. Furthermore, it is interesting to note that the β-coefficients of albert-base-v1, bert-base-cased and roberta-base take on smaller values compared to the other model β-coefficients and are not significant (although the significance should be regarded with caution). This matches the observations made in the descriptive analysis since the inference times of these models do not differ much in general and in comparison to bert-base-uncased. When having a look at the task β-coefficients, it becomes clear that the task size does not have an impact on the inference time (as already observed in the descriptive analysis and the ANOVA). While the task WNLI - the one with the smallest task size - has an increasing effect on the inference time for bert-base-uncased compared to the task RTE (c.p.)[45], the other tasks exhibit negative coefficients, which indicate a decrease in inference time for bert-base-uncased compared to RTE (c.p.). A clear trend is hereby difficult to distinguish. When increasing the batch size by eight units (which is reasonable in this context), the inference time is $\exp(0.0585 \cdot 8) - 1 = 0.5967$ times larger for bert-base-cased and RTE (c.p.). Having a look at the effect of the maximum sequence length, one can detect that increasing the maximum sequence length by 128 leads to a $\exp(0.0049 \cdot 128) - 1 = 0.8724$ larger inference time for bert-base-uncased and RTE, while holding the other coefficients constant.

All in all, the observations made during the analysis of the log-linear regression

[45] attention: the impact is not significant

match the observations which result from the descriptive analysis of the inference time. More precisely, the maximum sequence length and the batch size play an important role for the inference time. With a larger maximum sequence length as well as batch size, the inference time shows an increase. Depending on which model is regarded, the model could also have an effect on the inference time (e.g. xlnet-base-cased and both DistilBERT model versions). Moreover, we implemented a Random Forest, which indicates that the variable maximum sequence length is the most important variable among the regarded variables, followed by the model and the batch size. The task exhibits the lowest variable importance (see appendix 8 for more details).

7.5 Accuracy and inference time

Furthermore, an interesting aspect to examine is the relation between the inference time and the performance measure. In the following, we present the performance measure accuracy, which was calculated for the tasks WNLI, RTE and MRPC. For CoLA, Matthews correlation coefficient and for STS-B the Spearman and the Pearson correlation coefficient were computed. Since we only have 63 and 45 data points for these measures, respectively, it is hard to make assumptions for the relation between them and the inference time. Since the accuracy is calculated for 159 observations, we regard the relation between the accuracy and the inference time in the following figure.

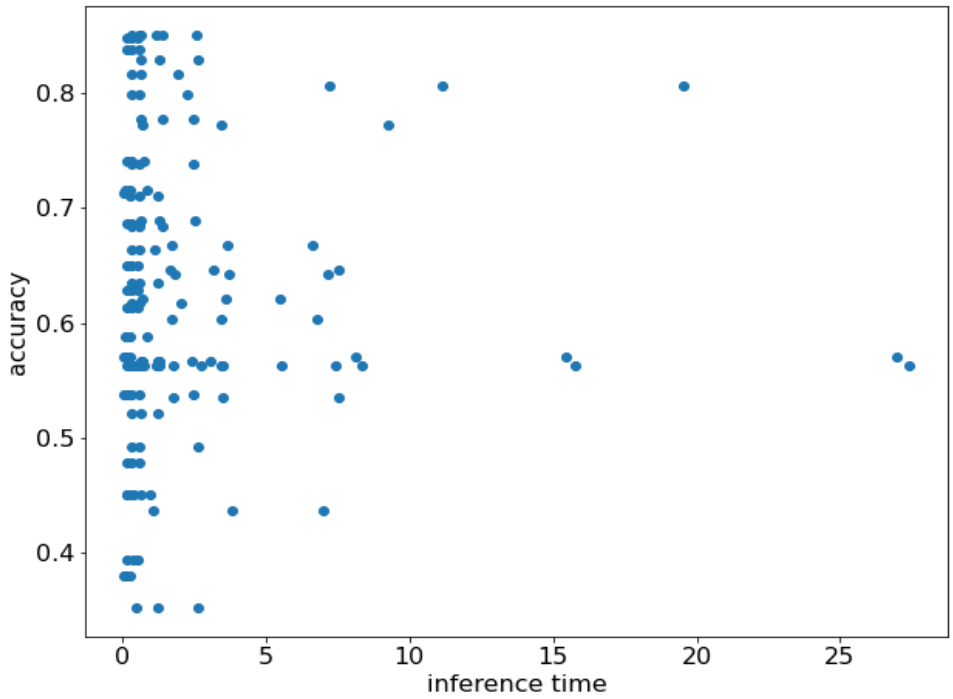

Figure 23: Scatterplot between inference time and accuracy

Note that, in contrary to the relation between fine-tuning time and the accuracy, no clear trend can be seen in this scatterplot (figure 23). The pattern seems rather random and the Pearson correlation between both variables amounts to -9.99e-4, which indicates that there is no relationship between both.

7.6 Comparison to the results by *huggingface*

A direct comparison of our results and the ones from the benchmark study by *huggingface* is not possible since we integrate other models into our analysis than *huggingface*. More precisely, we take the pre-trained LMs which were fine-tuned on GLUE tasks into consideration, whereas the models regarded in the benchmark study by *huggingface* are fine-tuned on SQuAD v2. Furthermore, the BERT model displayed in table 29 and 30 incorporates $L = 10$ layers (i.e. Transformer blocks) and a hidden size of $H = 512$, which represents a smaller architecture than the one

of $\text{BERT}_{\text{BASE}}$. Nonetheless, it is interesting to have a look at the results of their study, which are displayed in table 29 and 30[46] and draw a comparison between the observed trends.

model name	batch size	seq length	time in s
aodiniz/bert_uncased_L-10_H-51	256	8	0.033
aodiniz/bert_uncased_L-10_H-51	256	32	0.119
aodiniz/bert_uncased_L-10_H-51	256	128	0.457
aodiniz/bert_uncased_L-10_H-51	256	512	2.21
deepset/roberta-base-squad2	256	8	0.064
deepset/roberta-base-squad2	256	32	0.25
deepset/roberta-base-squad2	256	128	1.01
deepset/roberta-base-squad2	256	512	4.65

Table 29: Inference speed results by *huggingface* with varying sequence lengths, https://github.com/huggingface/transformers/blob/master/notebooks/05-benchmark.ipynb

The table displays that when holding the hyperparameters constant, $\text{RoBERTa}_{\text{BASE}}$ shows higher inference time than the BERT model. This trend could not be detected in our analysis, which could be attributed to the fact that the BERT architecture in our analysis is larger that the one that is regarded by *huggingface*. Furthermore, the inference time increases with increasing sequence length. Not only the same trend could be detected in our analysis, but the retrieved inference times take on values which lie in a similar value range. Hence, this table helps verifying the findings of the inference time analysis in chapter 7.1.

model name	batch size	seq length	time in s
aodiniz/bert_uncased_L-10_H-51	8	512	0.056
aodiniz/bert_uncased_L-10_H-51	64	512	0.402
aodiniz/bert_uncased_L-10_H-51	256	512	1.591

Table 30: Inference speed results by *huggingface* with varying batch sizes, https://github.com/huggingface/transformers/blob/master/notebooks/05-benchmark.ipynb

[46] The benchmark study can be viewed at https://github.com/huggingface/transformers/blob/master/notebooks/05-benchmark.ipynb; following their notation, the name of the BERT model is shortened in both tables and stands for "aodiniz/bert_uncased_L-10_H-512_A-8_cord19-200616_squad2"

Table 30 displays an increase of the inference time, when enlarging the batch size for the BERT model. This trend can also be seen in our analysis (section 7.1 to 7.4).

Consequently, both observations, namely that the inference time increases with larger maximum sequence length and with an increase of the batch size, could be seen in both analysis, which verifies our hypotheses made in section 7.1 to 7.4.

8 Conclusion and Outlook

The main goal of this scientific work is the measurement and exploration of fine-tuning and inference time of large pre-trained LMs. In order to conduct an analysis of these, different hyperparameter combinations were regarded, including the hyperparameters maximum sequence length and batch size. We fine-tuned a wide range of pre-trained LMs based on numerous GLUE tasks. Subsequently, we analyzed the resulting fine-tuning and inference times across the regarded variables and underlined the effects by integrating ANOVAs and log-linear regression models. Moreover the relation between the accuracy and the fine-tuning time was regarded. For the inference time, we further compared our results to the benchmark study from *huggingface*.
The performed analyses show that the task itself, or better the task size, has the largest effect on the fine-tuning time. The larger the task is, the higher the fine-tuning time is. While the batch size appears to only have a small effect on the fine-tuning time, the effect it has on the inference time is shown to be large. Furthermore, the maximum sequence length is shows to have an effect on the fine-tuning time and the inference time, respectively. The cased model version of $\text{XLNet}_{\text{BASE}}$ exhibits the largest fine-tuning times. Applying $\text{RoBERTa}_{\text{BASE}}$, both versions of $\text{BERT}_{\text{BASE}}$ and the light version of BERT, namely $\text{ALBERT}_{\text{BASE}}$ results in inference times which lie in a similar value range, respectively. The same statement holds for the fine-tuning time when applying $\text{RoBERTa}_{\text{BASE}}$ and both versions of $\text{BERT}_{\text{BASE}}$, followed by a slightly lower fine-tuning time for $\text{ALBERT}_{\text{BASE}}$. As Sanh et al. (2020) noted, $\text{DistilBERT}_{\text{BASE}}$ incorporates about 40% less parameters than $\text{BERT}_{\text{BASE}}$, which causes a much lower fine-tuning and inference times.

Since the effect of the batch size on the fine-tuning time that we detected in our analysis does not follow the general intuition, it should be further analyzed. Hereby, a larger value range, for example the existing sweep plus the values {64, 128, 256, 512} could be regarded. Moreover, the hyperparamters epoch, taking on the recommended values from Devlin et al. (2019) in the range of {2,3,4} and learning rates, taking on the values {1e-5, 2e-5, 3e-5, 5e-5} (just like in table 1), could be included in the sweep.

Furthermore, all runs that crashed due to technical limitations, such as lack of RAM (which probably occurred when fine-tuning of QNLI, QQP and MNLI), should be performed. Hereby, the usage of a more "powerful" GPU with more RAM should be considered. The fine-tuning for the cased version of $\text{BERT}_{\text{BASE}}$ and $\text{DistilBERT}_{\text{BASE}}$ could not be performed, presumably due to a bug. Since this bug might be fixed in a newer version of the `transformers` package of *huggingface* (Wolf et al., 2019), it should be used for further analyses. Furthermore, it is interesting to note that when we fine-tuned WNLI on those models with a maximum sequence length of 64 and 80, the fine-tuning succeeded. Hence, a different value range for the maximum sequence length could be considered. Moreover, to enable a more exhaustive research, multiple GPUs with more RAM could be used.
To distinguish if the task type has an effect on the fine-tuning and inference time, an analysis with a constant task size across the different tasks could be performed. This could also be done the other way around, namely by changing the task size while keeping the task type constant (or performing the fine-tuning on the same task with different task sizes).
Since the research was only performed based on PyTorch, it would be interesting to distinguish if the usage of TensorFlow would result in different fine-tuning and inference times. A comparison of inference times on CPU and GPU for PyTorch and TensorFlow was already carried out by *huggingface*[47]. Nonetheless, no such research exists for language models which were fine-tuned on GLUE tasks. Moreover, to our knowledge, an investigation of the fine-tuning time was not performed up to this point of time.
Note that due to computational limitations caused by the usage of one GPU and the computationally expensive architecture of large pre-trained LMs, this analysis reaches its limits. Generally speaking, a more exhaustive analysis should be performed in order to be able to make more concrete statements about the effect size of the different parameters on the fine-tuning and inference time. Furthermore, more time measurements, especially for the measurement of fine-tuning time could be performed in order to receive more stable results. Hence, there is still much potential for future work. Moreover, this master's thesis shows that some language models, especially $\text{XLNet}_{\text{BASE}}$, but also $\text{RoBERTa}_{\text{BASE}}$ and $\text{BERT}_{\text{BASE}}$ exhibit

[47] For more details, see https://medium.com/huggingface/benchmarking-transformers-pytorch-and-tensorflow-e2917fb891c2

large fine-tuning times. This applies especially for fine-tuning those LMs on large datasets, as is common in NLP to achieve good performance values. With this statement, we underline the fact that there is still much potential to improve the computational costs of pre-trained LMs in terms of training times, but also memory and invest more in researching both issues. With further pursuing the conducted analysis of fine-tuning and inference time of LMs, the main origins for higher values (i.e. specific hyperparameter combinations) of both could be discovered and concrete recommendations could be made.

Bibliography

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). Retrieved November 1, 2020, from https://arxiv.org/pdf/1803.08375

Allard, M. (2019). What is a transformer?. an introduction to transformers and sequence-to-sequence learning for machine learning. *Inside Machine Learning*. Retrieved August 16, 2020, from https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04

Ardit. (2019). How to measure the execution time of a python script.

Aschermann, T. (2017). Ram einfach erklärt - was ist das? Retrieved September 12, 2020, from https://praxistipps.chip.de/ram-einfach-erklaert-was-ist-das_35097

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, & Illia Polosukhin. (2017). Attention is all you need. Retrieved July 22, 2020, from https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Baevski, A., Edunov, S., Liu, Y., Zettlemoyer, L., & Auli, M. (2019). Cloze-driven pretraining of self-attention networks. Retrieved August 21, 2020, from https://arxiv.org/pdf/1903.07785

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. Retrieved November 2, 2020, from https://arxiv.org/pdf/1409.0473

Bar Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B., & Szpektor, I. (2006). The second PASCAL recognising textual entailment challenge.

Bedre, R. (2018). Test anova assumptionspermalink. Retrieved November 9, 2020, from https://reneshbedre.github.io/blog/anova.html

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research 3*, 1137–1155. Retrieved October 25, 2020, from https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

Bentivogli, L., Dagan, I., Dang, H. T., Giampiccolo, D., & Magnini, B. (2009). The fifth PASCAL recognizing textual entailment challenge.

Biewald, L. (2020). Experiment tracking with weights and biases [Software available from wandb.com]. Retrieved September 1, 2020, from https://www.wandb.com/

Bostrom, K., & Durrett, G. (2020). Byte pair encoding is suboptimal for language model pretraining. Retrieved November 2, 2020, from https://arxiv.org/pdf/2004.03720

Bowman, S. R., Angeli, G., Potts, C., & Manning, C. D. (2015). A large annotated corpus for learning natural language inference, In *Proceedings of the 2015 conference on empirical methods in natural language processing*, Lisbon, Portugal, Association for Computational Linguistics. https://doi.org/10.18653/v1/D15-1075

Brownlee, J. (2017). What are word embeddings for text? *Machine Learning Mastery*. Retrieved August 18, 2020, from https://machinelearningmastery.com/what-are-word-embeddings/

Bucila, C., Caruana, R., & Niculescu-Mizil, A. (2006). Model compression. *KDD*.

Callan, J., Hoy, M., Yoo, C., & Zhao, L. (2009). Clueweb09 data set.

Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., & Specia, L. (2017). SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation, In *Proceedings of the 11th international workshop on semantic evaluation (SemEval-2017)*, Association for Computational Linguistics. https://doi.org/10.18653/v1/S17-2001

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014a). On the properties of neural machine translation: Encoder-decoder approaches. Retrieved November 2, 2020, from https://arxiv.org/pdf/1409.1259

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014b). Learning phrase representations using rnn encoder-decoder for statistical machine translation. Retrieved October 25, 2020, from https://arxiv.org/pdf/1406.1078

Crawl, C. (2019). Common crawl. Retrieved September 18, 2020, from https://commoncrawl.org/

Dagan, I., Glickman, O., & Magnini, B. (2006). The PASCAL recognising textual entailment challenge, In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*. Springer.

Dai, A. M., & Le V, Q. (2015). Semi-supervised sequence learning. Retrieved September 17, 2020, from https://arxiv.org/pdf/1511.01432

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le V, Q., & Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. Retrieved September 18, 2020, from https://arxiv.org/pdf/1901.02860

DeBeasi, P. (2019). Training versus inference. Retrieved September 10, 2020, from https://blogs.gartner.com/paul-debeasi/2019/02/14/training-versus-inference/#:~:text=Inference%3A%20Inference%20refers%20to%20the,algorithm%20to%20make%20a%20prediction.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding, In *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)*, Minneapolis, Minnesota, Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1423

Dolan, W. B., & Brockett, C. (2005). Automatically constructing a corpus of sentential paraphrases, In *Proceedings of the international workshop on paraphrasing*.

Elman, J. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211. https://doi.org/10.1016/0364-0213(90)90002-E

Gage, P. (1994). A new algorithm for data compression. Retrieved August 26, 2020, from http://www.pennelynn.com/Documents/CUJ/HTML/94HTML/19940045.HTM

Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. Retrieved August 21, 2020, from https://arxiv.org/pdf/1705.03122

Ghelani, S. (2019a). From word embeddings to pretrained language models — a new age in nlp — part 1. *Towards Data Science*. Retrieved August 21, 2020, from https://towardsdatascience.com/from-word-embeddings-to-pretrained-language-models-a-new-age-in-nlp-part-1-7ed0c7f3dfc5

Ghelani, S. (2019b). From word embeddings to pretrained language models — a new age in nlp — part 2. *Towards Data Science*. Retrieved August 25, 2020, from https://towardsdatascience.com/from-word-embeddings-to-pretrained-language-models-a-new-age-in-nlp-part-2-e9af9a0bdcd9

Giacaglia, G. (2019). How transformers work. Retrieved October 20, 2020, from https://towardsdatascience.com/transformers-141e32e69591

Giampiccolo, D., Magnini, B., Dagan, I., & Dolan, B. (2007). The third PASCAL recognizing textual entailment challenge, In *Proceedings of the acl-pascal workshop on textual entailment and paraphrasing*. Association for Computational Linguistics.

Gokaslan, A., & Cohen, V. (2019). Openwebtext corpus. Retrieved August 21, 2020, from http://Skylion007.github.io/OpenWebTextCorpus

Goldberg, Y. (2017). Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, *10*(1), 1–309. https://doi.org/10.2200/S00762ED1V01Y201703HLT037

Graves, A. (2012). Sequence transduction with recurrent neural networks. Retrieved August 27, 2020, from https://arxiv.org/pdf/1211.3711

Haleva, R. (2020). What is gradient accumulation in deep learning? Retrieved September 12, 2020, from https://towardsdatascience.com/what-is-gradient-accumulation-in-deep-learning-ec034122cfa

Hendrycks, D., & Gimpel, K. (2016). Gaussian error linear units (gelus). Retrieved August 16, 2020, from https://arxiv.org/pdf/1606.08415

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. Retrieved September 13, 2020, from https://arxiv.org/pdf/1503.02531

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Jeet. (2020). One hot encoding of text data in natural language processing. Retrieved October 20, 2020, from https://medium.com/analytics-vidhya/one-hot-encoding-of-text-data-in-natural-language-processing-2242fefb2148

Joshi, M., Chen, D., Liu, Y., Weld, D. S., Zettlemoyer, L., & Levy, O. (2019). Spanbert: Improving pre-training by representing and predicting spans. Retrieved September 11, 2020, from https://arxiv.org/pdf/1907.10529

Jurafsky, D., & Martin, J. (2008). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition* (Vol. 2).

Kalchbrenner, N., Espeholt, L., Simonyan, K., van den Oord, A., Graves, A., & Kavukcuoglu, K. (2016). Neural machine translation in linear time. *CoRR*,

abs/1610.10099. Retrieved August 20, 2020, from http://arxiv.org/abs/1610.10099

Kapadia, S. (2019). Language models: N-gram. Retrieved October 25, 2020, from https://towardsdatascience.com/introduction-to-language-models-n-gram-e323081503d9

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. Retrieved August 28, 2020, from https://arxiv.org/pdf/1412.6980

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., & Willing, C. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows (F. Loizides & B. Schmidt, Eds.). In F. Loizides & B. Schmidt (Eds.), *Positioning and power in academic publishing: Players, agents and agendas*. IOS Press.

Koech, K. E. (2020). Cross-entropy loss function. Retrieved October 11, 2020, from https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

Kothari, M. (2020). Feature extraction techniques – nlp. *GeeksforGeeks*. Retrieved August 4, 2020, from https://www.geeksforgeeks.org/feature-extraction-techniques-nlp/

Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple subword candidates. Retrieved November 2, 2020, from https://arxiv.org/pdf/1804.10959

Kudo, T., & Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. Retrieved September 16, 2020, from https://arxiv.org/pdf/1808.06226

Lai, G., Xie, Q., Liu, H., Yang, Y., & Hovy, E. (2017). Race: Large-scale reading comprehension dataset from examinations. Retrieved September 10, 2020, from https://arxiv.org/pdf/1704.04683

Lample, G., & Conneau, A. (2019). Cross-lingual language model pretraining. Retrieved September 11, 2020, from https://arxiv.org/pdf/1901.07291

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. Retrieved August 23, 2020, from https://arxiv.org/pdf/1909.11942

Levesque, H. J., Davis, E., & Morgenstern, L. (2011). The Winograd schema challenge, In *AAAI spring symposium: Logical formalizations of commonsense reasoning*.

Liu, Y., Ott, M., Goyal, N., Du Jingfei, Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. https://arxiv.org/pdf/1907.11692

Luber, S. (2016). Definition. was ist natural language processing? *BigData-Insider*. Retrieved July 22, 2020, from https://www.bigdata-insider.de/was-ist-natural-language-processing-a-590102/

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis, In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, Portland, Oregon, USA, Association for Computational Linguistics. Retrieved September 15, 2020, from https://www.aclweb.org/anthology/P11-1015

Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et biophysica acta*, *405*(2), 442–451. https://doi.org/10.1016/0005-2795(75)90109-9

McCann, B., Bradbury, J., Xiong, C., & Socher, R. (2017). Learned in translation: Contextualized word vectors. Retrieved September 17, 2020, from https://arxiv.org/pdf/1708.00107

McCarty, K. (2018). Interpreting results from linear regression – is the data appropriate? Retrieved November 12, 2020, from https://www.accelebrate.com/blog/interpreting-results-from-linear-regression-is-the-data-appropriate

McCormick, C., & Ryan, N. (2019a). Bert fine-tuning tutorial with pytorch. Retrieved September 10, 2020, from https://mccormickml.com/2019/07/22/BERT-fine-tuning/

McCormick, C., & Ryan, N. (2019b). Bert word embeddings tutorial. Retrieved September 12, 2020, from https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint 1301.3781*. Retrieved July 23, 2020, from https://arxiv.org/pdf/1301.3781.pdf

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *arXiv preprint 1310.4546v1*. Retrieved August 17, 2020, from https://arxiv.org/pdf/1310.4546v1.pdf

Nagel, S. (2016). Cc-news. Retrieved August 21, 2020, from https://commoncrawl.org/2016/10/news-dataset-available/

Parker, R., Graff, D., Kong, J., Chen, K., & Maeda, K. (2011). *English gigaword fifth edition, linguistic data consortium*. [Philadelphia, PA], Linguistic Data Consortium.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett, Eds.). In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32*. Curran Associates, Inc. Retrieved October 20, 2020, from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pennington, R., Jeffrey nd Socher, & Manning, C. (2014). GloVe: Global vectors for word representation, In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, Doha, Qatar, Association for Computational Linguistics. https://doi.org/10.3115/v1/D14-1162

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. Retrieved September 14, 2020, from https://arxiv.org/pdf/1802.05365

Phi, M. (2019). Illustrated guide to lstm's and gru's: A step by step explanation. Retrieved August 10, 2020, from https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Python Software Foundation. (2020a). 9. classes. Retrieved September 19, 2020, from https://docs.python.org/3/tutorial/classes.html

Python Software Foundation. (2020b). Time — time access and conversions. Retrieved July 31, 2020, from https://docs.python.org/3/library/time.html

Python Softwarer Foundation. (2020). Timeit - measure execution time of small code snippets. https://docs.python.org/3/library/timeit.html

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. Retrieved August 20, 2020, from https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. Retrieved August 20, 2020, from https://www.ceid.upatras.gr/webpages/faculty/zaro/teaching/alg-ds/PRESENTATIONS/PAPERS/2019-Radford-et-al_Language-Models-Are-Unsupervised-Multitask-%20Learners.pdf

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016a). Squad: 100,000+ questions for machine comprehension of text, In *Proceedings of the 2016 conference on empirical methods in natural language processing*, Association for Computational Linguistics. https://doi.org/10.18653/v1/D16-1264

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016b). SQuAD: 100,000+ questions for machine comprehension of text, In *Proceedings of emnlp*, Austin, Texas, Association for Computational Linguistics.

Rizvi, M. S. Z. (2019). A comprehensive guide to build your own language model in python! Retrieved October 25, 2020, from https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-language-model-nlp-python-code/

Rouse, M. (2018). Bag of words model (bow model). Retrieved August 16, 2020, from https://searchenterpriseai.techtarget.com/definition/bag-of-words-model-BoW-model

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0

Sanh, V. (2019). Smaller, faster, cheaper, lighter: Introducing distilbert, a distilled version of bert. Retrieved September 14, 2020, from https://medium.com/huggingface/distilbert-8cf3380435b5

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. Retrieved September 1, 2020, from https://arxiv.org/pdf/1910.01108v4

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681. https://doi.org/10.1109/78.650093

Schuster, M., & Nakajima, K. (2012). Japanese and korean voice search. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5149–5152. Retrieved October 1, 2020, from https://static.googleusercontent.com/media/research.google.com/de//pubs/archive/37842.pdf

Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2019). Green ai. https://arxiv.org/pdf/1907.10597

Sennrich, R., Haddow, B., & Birch, A. (2016). Neural machine translation of rare words with subword units, In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, Berlin, Germany, Association for Computational Linguistics. https://doi.org/10.18653/v1/P16-1162

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank, In *Proceedings of emnlp*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958. Retrieved August 28, 2020, from http://jmlr.org/papers/v15/srivastava14a.html

Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. Retrieved September 13, 2020, from https://arxiv.org/pdf/1906.02243

Sutskever, I., Vinyals, O., & Le V, Q. (2014). Sequence to sequence learning with neural networks. Retrieved November 2, 2020, from https://arxiv.org/pdf/1409.3215

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the inception architecture for computer vision. Retrieved August 30, 2020, from https://arxiv.org/pdf/1512.00567

Taylor, W. L. (1953). "cloze procedure": A new tool for measuring readability. *Journalism Quarterly*, *30*(4), 415–433. https://doi.org/10.1177/107769905303000401

Thomas, C. (2019). Recurrent neural networks and natural language processing. Retrieved October 20, 2020, from https://towardsdatascience.com/recurrent-neural-networks-and-natural-language-processing-73af640c2aa1

Trinh, T. H., & Le V, Q. (2018). A simple method for commonsense reasoning. Retrieved September 10, 2020, from https://arxiv.org/pdf/1806.02847

Uria, B., Côté, M.-A., Gregor, K., Murray, I., & Larochelle, H. (2016). Neural autoregressive distribution estimation. Retrieved September 18, 2020, from https://arxiv.org/pdf/1605.02226

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. Scotts Valley, CA, CreateSpace.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2019). GLUE: A multi-task benchmark and analysis platform for natural language understanding [In the Proceedings of ICLR.]. *arXiv preprint 1804.07461*.

Warstadt, A., Singh, A., & Bowman, S. R. (2018). Neural network acceptability judgments. *arXiv preprint 1805.12471*.

Wikipedia, the free encyclopedia. (2020a). Bleu. Retrieved August 26, 2020, from https://en.wikipedia.org/wiki/BLEU

Wikipedia, the free encyclopedia. (2020b). Breusch-pagan test. Retrieved November 12, 2020, from https://en.wikipedia.org/wiki/Breusch%E2%80%93Pagan_test

Wikipedia, the free encyclopedia. (2020c). Durbin–watson statistic. Retrieved November 12, 2020, from https://en.wikipedia.org/wiki/Durbin%E2%80%93Watson_statistic

Wikipedia, the free encyclopedia. (2020d). Matthews correlation coefficient. Retrieved October 10, 2020, from https://en.wikipedia.org/wiki/Matthews_correlation_coefficient

Wikipedia, the free encyclopedia. (2020e). N-gram. Retrieved August 20, 2020, from https://en.wikipedia.org/wiki/N-gram

Wikipedia, the free encyclopedia. (2020f). Perplexity. Retrieved August 15, 2020, from https://en.wikipedia.org/wiki/Perplexity

Williams, A., Nangia, N., & Bowman, S. R. (2018). A broad-coverage challenge corpus for sentence understanding through inference, In *Proceedings of naacl-hlt*.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., ... Rush, A. M. (2019). Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, *abs/1910.03771*. Retrieved September 20, 2020, from https://github.com/huggingface/transformers

Wu, Y., Schuster, M., Chen, Z., Le V, Q., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., ... Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. Retrieved August 16, 2020, from https://arxiv.org/pdf/1609.08144

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le V, Q. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. https://arxiv.org/pdf/1906.08237

You, Y., Gitman, I., & Ginsburg, B. (2017). Large batch training of convolutional networks. https://arxiv.org/pdf/1708.03888

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., & Hsieh, C.-J. (2019). Large batch optimization for deep learning: Training bert in 76 minutes. Retrieved September 12, 2020, from https://arxiv.org/pdf/1904.00962

Zellers, R., Bisk, Y., Schwartz, R., & Choi, Y. (2018). Swag: A large-scale adversarial dataset for grounded commonsense inference. Retrieved August 16, 2020, from https://arxiv.org/pdf/1808.05326

Zhou, V. (2019). A simple explanation of the bag-of-words model. Retrieved August 28, 2020, from https://victorzhou.com/blog/bag-of-words/

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, In *2015 ieee international conference on computer vision*, Piscataway, NJ, IEEE. https://doi.org/10.1109/ICCV.2015.11

Appendix

Subword tokenization methods

Large pre-trained language models make use of several subword tokenizers, which contain two elements, namely, the generation of a vocabulary construction and the tokenization. During the vocabulary construction procedure a text corpus is taken as an input and a new vocabulary of a desired size is produced. The new vocabulary is then applied to new text during the tokenization procedure, resulting in a sequence of tokens (Bostrom et al., 2020).

Note that BERT uses the WordPiece method (Schuster et al., 2012). RoBERTA and DistilBERT apply the BPE method (Gage, 1994; Sennrich et al., 2016) over raw bytes. XLNET and ALBERT apply the SentencePiece tokenizer.

BPE can be described as a simple data compression algorithm implemented by Philip Gage, where a replacing of the most common pairs of consecutive bytes in the data with a single byte not included in the data is performed (Gage, 1994). Sennrich et al. (2016) adjust the BPE for the segmentation of words. Hereby, the authors regard character or character sequences instead of bytes. In the first step, each word is represented by a sequence of characters with an end-of-word symbol ".". The authors perform an iterative counting of all symbol pairs with a subsequent merging of the most common pairs, whereby each merging results in a new symbol (a character n-gram). This results in a symbol vocabulary of a size that resembles the initial vocabulary with addition of the merge operations. As the desired size represents a hyperparameter, it can be defined by the user[48].

The WordPiece model (WPM) deals with the generation of a vocabulary which includes individual characters, subwords as well as words. Words that are not included in the vocabulary are split into the largest possible subwords. Therefore, these words do not need to be treated in any special way, e.g. out-of-word-tokens, which leads to an improved accuracy (McCormick et al., 2019b; Wu et al., 2016; Schuster et al., 2012). The WordPiece model represents a language-modeling

[48]For a more detailed explanation plus pseudo-code, see Sennrich et al. (2016)

based variant of BPE, where "each potential merge is scored based on the likelihood of an n-gram language model trained on a version of the corpus incorporating that merge" (Bostrom et al., 2020).[49]

SentencePiece is a "language-independent subword tokenizer and detokenizer designed for Neural-based text processing" (Kudo et al., 2018, p. 1). This tokenizer has the ability to train subword models from raw sentences, without any prior tokenization into word sequences, whereas the creation of a language independent end-to-end system is possible. Furthermore, it contains the BPE (Sennrich et al., 2016 and the unigram language model (Kudo, 2018). In contrast to both methods, SentecePiece can perform a training from raw text data (Kudo et al., 2018)[50].

Fine-tuning time

Ten highest fine-tuning times

The following table displays the ten highest fine-tuning times.

task	model	seq. len.	b.s.	f. time	acc
SST-2	bert-base-cased	512	32	59139.1	0.92
SST-2	roberta-base	512	32	59108.7	0.94
SST-2	bert-base-uncased	512	8	43090.0	0.92
SST-2	roberta-base	512	8	42174.8	0.94
SST-2	roberta-base	512	16	37749.2	0.93
SST-2	bert-base-uncased	512	16	37449.8	0.91
SST-2	distilbert-base-uncased	512	32	26147.2	0.90
SST-2	distilbert-base-uncased	512	8	20886.2	0.89
SST-2	roberta-base	256	8	20238.7	0.93
SST-2	bert-base-uncased	256	16	19283.4	0.92

Table 31: Ten highest fine-tuning times; b.s. stands for batch size, seq. len. for maximum sequence length, f. time for fine-tuning time and acc. for accuracy

[49]For a more detailed explanation plus pseudo-code, see Schuster et al., 2012 and Bostrom et al., 2020

[50]For a more detailed explanation plus pseudo-code, see Kudo et al., 2018

Ten lowest fine-tuning times

The following table displays the ten lowest fine-tuning times.

task	model	seq. len.	b.s.	f. time	acc
WNLI	distilbert-base-uncased	128	32	40.3	0.41
WNLI	distilbert-base-uncased	128	16	48.0	0.48
WNLI	distilbert-base-uncased	128	8	64.3	0.42
WNLI	distilbert-base-uncased	256	32	73.7	0.42
WNLI	albert-base-v1	128	32	75.50	0.4
WNLI	bert-base-uncased	128	32	79.5	0.48
WNLI	roberta-base	128	32	79.6	0.52
WNLI	albert-base-v1	128	16	86.5	0.46
WNLI	distilbert-base-uncased	256	16	90.2	0.39
WNLI	roberta-base	128	16	92.1	0.56

Table 32: Ten lowest fine-tuning times; b.s. stands for batch size, seq. len. for maximum sequence length, f. time for fine-tuning time and acc. for accuracy

Check ANOVA assumptions

In order to conduct an ANOVA some assumptions have to be met. These are elaborated in the following.

Since the measurement of fine-tuning time is performed separately for each regarded hyperparameter combination for different models and on different GLUE tasks, they do not depend on the prior or later trainings. Hence, we assume that the regarded time measurements do not depend on each other. Nonetheless, this assumption follows a general logic and cannot be verified or rejected with certainty.

The residuals have to follow a normal distribution. This can be tested by performing the Shapiro-Wilk test. The null-hypothesis assumes that the data (here: the residuals) is drawn from a normal distribution. For all ANOVAs that were carried out, this test results in small p-values (5.28e-27, 1.26e-28, 9.14e-27 and 1.56e-26). Hence, we reject the null-hypothesis to a significance level of 0.05 and a

normal distribution for the residuals cannot be assumed. Note that the test does not exhibit a high power if small data sizes are regarded (Bedre, 2018). Hence, we have a look at the according Q-Q-plot to gain a better understanding about whether the groups exhibit a normal distribution or not.

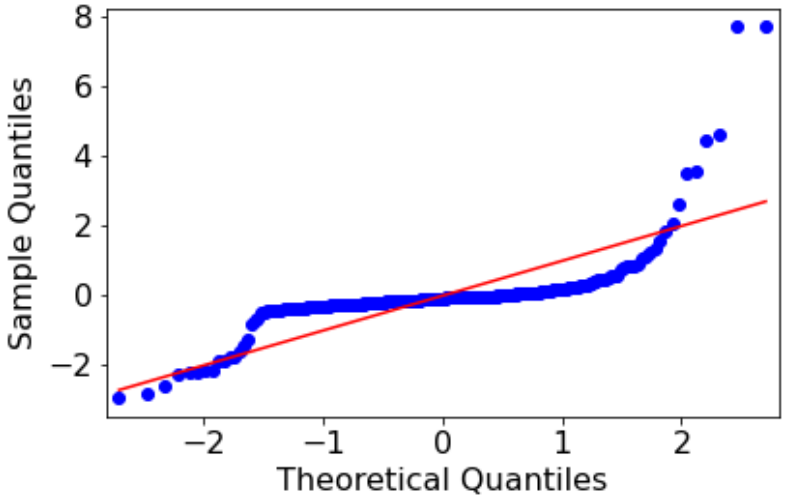

Figure 24: Q-Q plot for fine-tuning time grouped by task

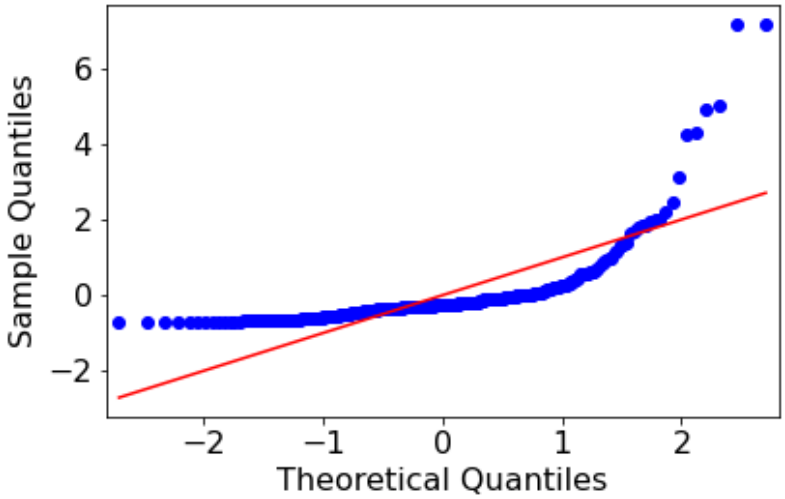

Figure 25: Q-Q plot for fine-tuning time grouped by model

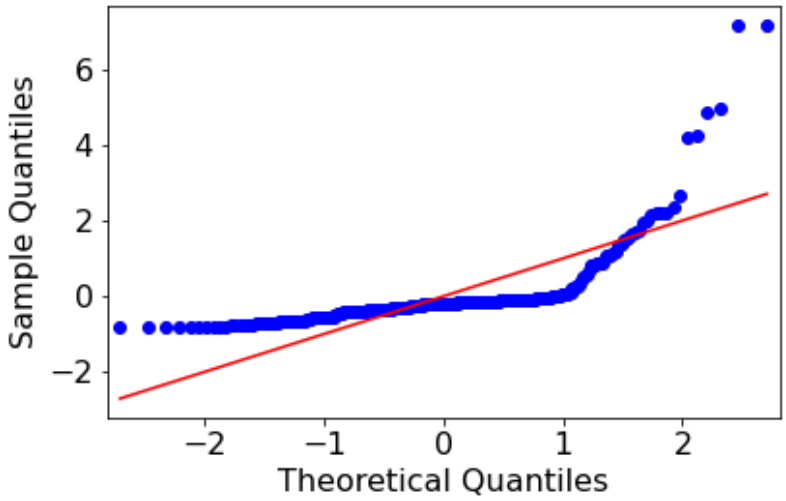

Figure 26: Q-Q plot for the fine-tuning time grouped by maximum sequence length

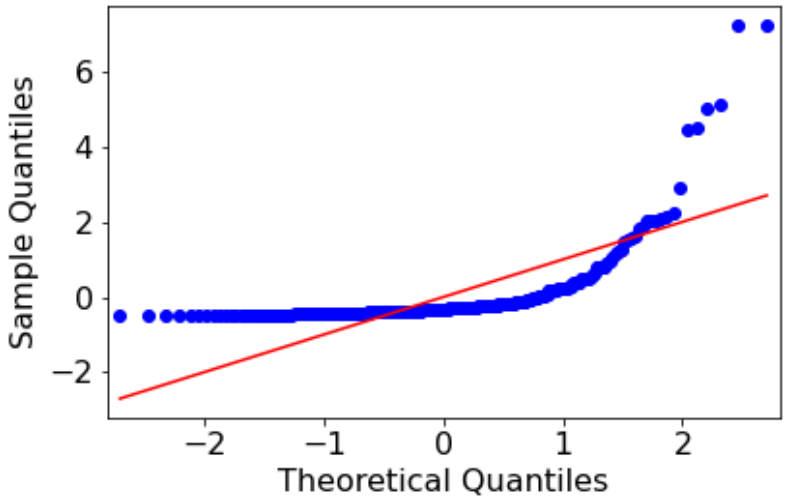

Figure 27: Q-Q plot for the fine-tuning time grouped by batch size

All Q-Q plots indicate a non-normal distribution of the respective residuals, which leads to the conclusion that the assumption about the normal distribution cannot be met.

In order to check if the assumption of the homogeneity of variances can be met, we carry out Levene's test since this test can be conducted if the data is not drawn from a normal distribution (Bedre, 2018). The null-hypothesis states that samples from populations have equal variances. The test results in very small p-values for the maximum sequence length and the task (0.0009 and 4.250e-24). Hence, we reject the null hypothesis to a significance level of 0.05 for both variables. Consequently, the homogeneity of variances cannot be assumed. The test results in a p-value of 0.867 for the batch size, which is why the null hypothesis cannot be rejected to a significance level of 0.05, indicating that homogeneity of variances can be assumed. For the model, Levene's test results in a p-value of 0.085. Depending on the significance level (e.g. 0.05 or 0.1), the null hypothesis of homogeneity of variances can either be rejected or not.

All in all, the tests should be regarded with caution since not only the between group independence should be questioned, but the respective groups do not always incorporate the same number of examples and are therefore not balanced. Since some assumptions could not be met, the values resulting from the conducted ANOVAs should be regarded with caution.

Check assumptions for linear regression

In order to conduct a linear regression, following five assumptions have to be fulfilled:

- Linear relationship
- Multivariate normality
- No or little multicollinearity
- No autocorrelation
- Homoscedasticity

Since the model and the task are included as dummy variables in the log-linear regression, they meet the linearity assumption. Since the variables maximum sequence length and batch size only include three values and no clear trend could

be detected, we can also assume linearity.

To further check the normality assumptions, we display a Q-Q plot of the residuals.

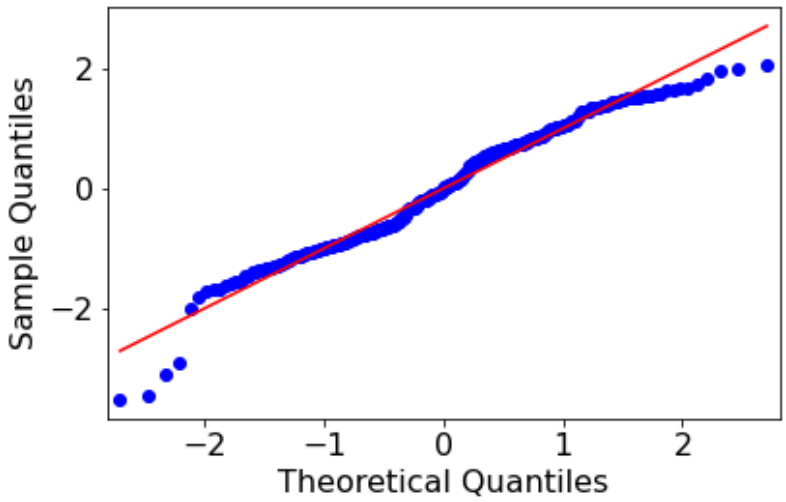

Figure 28: Q-Q plot for the residuals of the log-linear regression for log. fine-tuning time

Having a look at figure 27, we can see that the normality assumption can be met. This is probably caused by taking the logarithm of the fine-tuning time.
We have a look at condition number[51] to distinguish if multicollinearity is present in the data. Since this number takes on a high value (2.51e+03), multicollinearity might be present and hence, the assumption cannot be met.
In order to ascertain if autocorrelation of lag 1 of the error terms is present, we conduct a Durbin-Watson test. The null hypothesis is that no autocorrelation is present in the data. If the test statistic takes on a value of 2 and since in our case it amounts to 1.821, no problems with autocorrelation should occur (Wikipedia, the free encyclopedia, 2020c). Note that this statement only follows a rule of thumb.
To test if homoscedasticity is present, we conduct a Breusch-Pagan test. The null

[51]The condition number "measures the sensitivity of a function's output as compared to its input [...]. When we have multicollinearity, we can expect much higher fluctuations to small changes in the data, hence, we hope to see a relatively small number, something below 30." (McCarty, 2018)

hypothesis is that homoskedasticity is present (Wikipedia, the free encyclopedia, 2020b). In our case the p-value is 1.363e-7. Hence, we can reject the null hypothesis and therefore homocedasticity of the residuals. Hence, the fifth assumption can be fulfilled in our case.

Due to the fact that not all assumptions for conducting a linear regression could be met, the conducted log-linear regression should be regarded with a grain of salt.

Random Forest Regression - Variable importance

In order to get better insights about the variable importance of variables which could influence the logarithmic fine-tuning time, we perform a Random Forest Regression, in which the logarithmic fine-tuning time is the dependent variable and the remaining variables are the independent ones. The following table displays the respective variable importances. Hereby, it should be noted that the categorical variables task and model were encoded in order to perform this analysis.

Variable	**Variable Importance**
task	0.656
max. sequence length	0.233
model	0.096
batch size	0.015

Table 33: Random Forest Regression for log. fine-tuning time: Variable Importance

Inference time

Ten highest inference times

The following table displays the ten highest inference times.

task	model	seq. len.	b.s.	inf. time	acc	spear	mcc
WNLI	xlnet-base-cased	512	32	27.37	0.56	-	-
RTE	xlnet-base-cased	512	32	26.96	0.57	-	-
MRPC	xlnet-base-cased	512	32	19.56	0.81	-	-
STS-B	xlnet-base-cased	512	32	19.21	-	0.83	-
CoLA	xlnet-base-cased	512	32	18.84	-	-	0
WNLI	xlnet-base-cased	512	16	15.80	0.56	-	-
RTE	xlnet-base-cased	512	16	15.45	0.57	-	-
CoLA	xlnet-base-cased	512	16	11.72	-	-	0
STS-B	xlnet-base-cased	512	16	11.42	-	0.83	-
MRPC	xlnet-base-cased	512	16	11.17	0.81	-	-

Table 34: Ten highest inference times; b.s. stands for batch size, seq. len. for maximum sequence length, inf. time for inference time and acc. for accuracy; spear for Spearman correlation coefficient and mcc for Matthews correlation coefficient

Ten lowest inference times

The following table displays the ten lowest inference times.

task	model	seq. len.	b.s.	inf. time	acc	spear	mcc
CoLA	distilbert-base-cased	128	8	0.0915	-	-	0.23
CoLA	distilbert-base-uncased	128	8	0.092	-	-	0
WNLI	distilbert-base-uncased	128	8	0.093	0.38	-	-
STS-B	distilbert-base-uncased	128	8	0.094	-	0.84	-
RTE	distilbert-base-cased	128	8	0.095	0.54	-	-
MRPC	distilbert-base-uncased	128	8	0.096	0.71	-	-
RTE	distilbert-base-uncased	128	8	0.097	0.57	-	-
CoLA	distilbert-base-uncased	128	16	0.144	-	-	0
STS-B	distilbert-base-uncased	128	16	0.146	-	0.84	-
CoLA	distilbert-base-cased	128	16	0.147	-	-	0.23

Table 35: Ten lowest inference times; b.s. stands for batch size, seq. len. for maximum sequence length, inf. time for inference time and acc. for accuracy; spear for Spearman correlation coefficient and mcc for Matthews correlation coefficient

Note that in both tables (34 and 35), the Matthews correlation coefficient takes on the value 0 in some cases. This might indicate that the performance could not be measured properly. Nonetheless, this data is used in the analysis since the inference times take on reasonable values and the failure of measuring the performance is not an indicator for an inference time measurement failure.

Check ANOVA assumptions

In order to conduct an ANOVA some assumptions have to be met. These are elaborated in the following.

Since the measurement of inference time is performed separately for each regarded hyperparameter combination for different models and on different GLUE tasks, they do not depend on the prior or later trainings. Hence, we assume that the regarded time measurements do not depend on each other. Nonetheless, this

assumption follows a general logic and cannot be verified or rejected with certainty.

The residuals have to follow a normal distribution. As stated before, when checking the ANOVA assumptions for the fine-tuning time (see 8), an appropriate test to check this assumption is the Shapiro-Wilk test. For all ANOVAs that were carried out, this test results in small p-values (1.054e-23, 1.300e-25, 2.914e-22 and 3.383e-26). Hence, we reject the null-hypothesis to a significance level of 0.05 and a normal distribution for the residuals cannot be assumed. Due to the low power of the test for small datasets, we further regard the according Q-Q-plot.

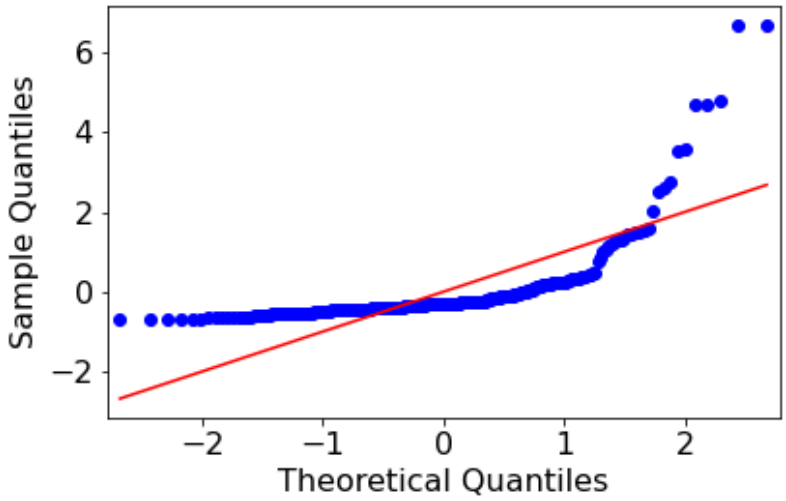

Figure 29: Q-Q plot for inference time grouped by task

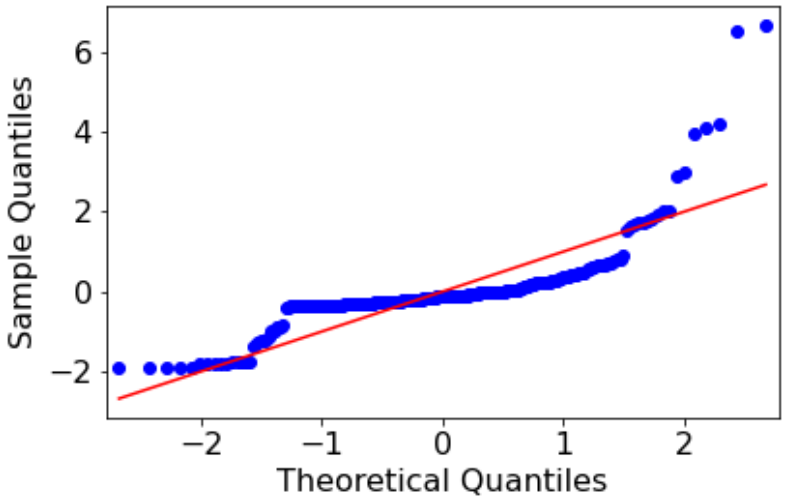

Figure 30: Q-Q plot for inference time grouped by model

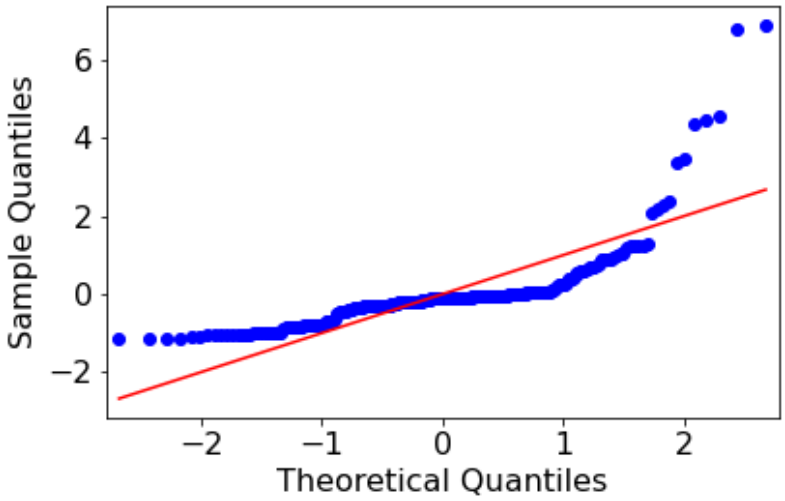

Figure 31: Q-Q plot for the inference time grouped by maximum sequence length

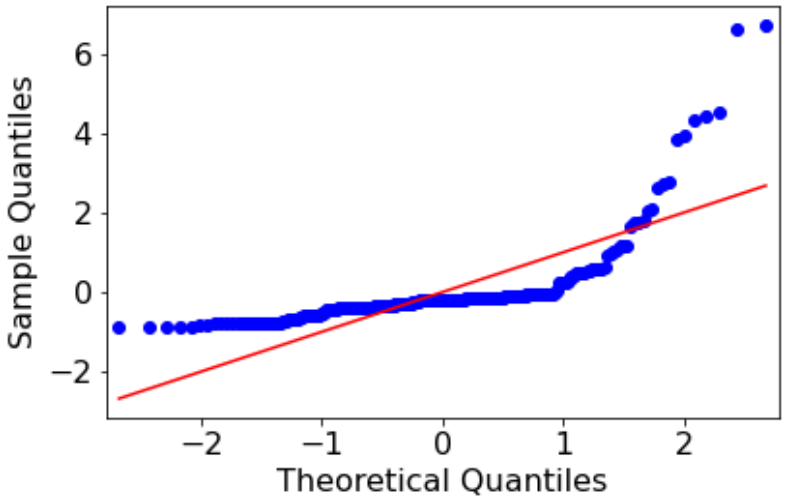

Figure 32: Q-Q plot for the inference time grouped by batch size

All Q-Q plots indicate a non-normal distribution of the respective residuals, which leads to the conclusion that the assumption about the normal distribution cannot be met.

In order to check if the assumption of the homogeneity of variances can be met, we carry out Levene's test (just as in section 8 in the appendix). The null-hypothesis states that samples from populations have equal variances. The test results in very small p-values for the maximum sequence length, the batch size and the model (4.389e-9, 0.0008 and 2.790e-16). Hence, we we reject the null hypothesis to a significance level of 0.05 for both variables. Consequently, the homogeneity of variances cannot be assumed. The test results in a p-value of 0.540 for the task, which is why the null hypothesis cannot be rejected to a significance level of 0.05, indicating that homogeneity of variances can be assumed.

All in all, the tests should be regarded with caution since not only the between group independence questioned, but the respective groups do not always incorporate the same number of examples and are therefore not balanced. Since some assumptions could not be met, the values resulting from the conducted ANOVAs should be regarded with caution.

Check assumptions for linear regression

The same assumptions that we mention in section 8 in the appendix have to be fulfilled in order to conduct a linear regression for the inference time.

For the linearity assumption, the same argument as in section 8 in the appendix applies.

To further check the normality assumptions, we display a Q-Q plot of the residuals.

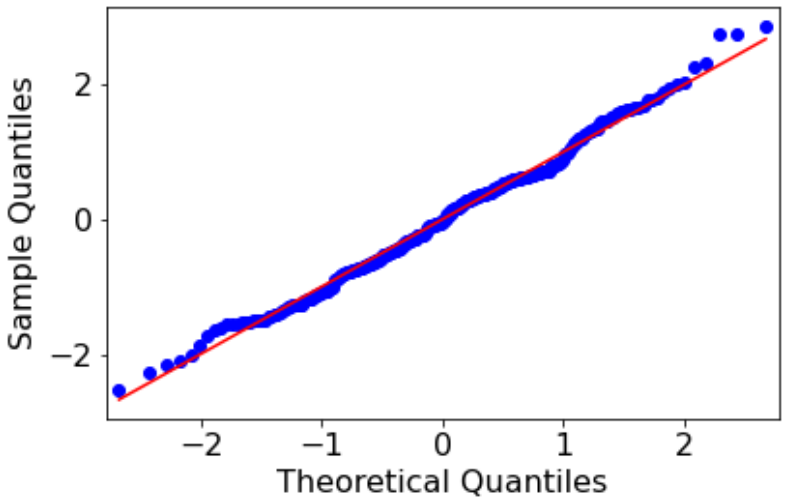

Figure 33: Q-Q plot for the residuals of the log-linear regression for log. inference time

Having a look at figure 33, we can see that the normality assumption can be met. This is probably caused by taking the logarithm of the fine-tuning time.
To detect possible multicollinearity, we regard the condition number. Since this number takes on a high value (2.51e+03), multicollinearity might be present and hence, the assumption cannot be met.
In order to ascertain if auto-correlation of lag 1 of the error terms is present, we conduct a Durbin-Watson test. The null hypothesis is that no autocorrelation is present in the data. If the test statistic takes on a value of 2 evidence of autocorrelation is present. The test statistic which takes on a value of 1.143 in our case and is therefore higher than 1.0. Nonetheless, since it surpasses the value

1.0 only by a small amount, it could be an indicator for positive serial correlation (Wikipedia, the free encyclopedia, 2020c). Note that this statement only follows a rule of thumb.
To test if homoscedasticity is present, we conduct a Breusch-Pagan test. The null hypothesis is that homoskedasticity is present (Wikipedia, the free encyclopedia, 2020b). In our case the p-value is 2.828e-6. Hence, we can reject the null hypothesis and therefore homocedasticity of the residuals. Hence, the fifth assumption can be fulfilled in our case.

Due to the fact that not all assumptions for conducting a linear regression could be fulfilled, the conducted log-linear regression should be regarded with a grain of salt.

Random Forest Regression - Variable importance

In order to get better insights about the variable importance of variables which could influence the logarithmic inference time, we perform a Random Forest Regression, in which the logarithmic inference time is the dependent variable and the remaining variables are the independent ones. The following table displays the respective variable importances. Note that the categorical variables task and model were encoded in order to perform this analysis.

Variable	**Variable Importance**
max. sequence length	0.505
model	0.248
batch size	0.231
task	0.017

Table 36: Random Forest Regression for log. inference time: Variable Importance

Electronic Annex

The electronic appendix contains all datasets, Python scripts, sweep configuration files and plots that were used in the scope of this master's thesis.

To run the analyses, the following requirements must be met:

- Python, version 3.8 (Van Rossum et al., 2009)
- Visual Studio Code, version 1.48
- Jupyter Notebook, version 6.0.3 (Kluyver et al., 2016)

The enclosed USB flash drive contains the following folders and files which are located in the folder *Master's Thesis Korotkova*[52]:

- *Master's Thesis_Korotkova.pdf*: the present scientific work in .pdf format
- *GLUE.ipynb*
  - download GLUE tasks data following the instructions on https://github.com/nyu-mll/GLUE-baselines
  - some superficial analyses of the GLUE tasks
- https://github.com/annakorotkova/transformers: modified `transformers` module[53]
  - the module `transformers` by *huggingface* (version 3.0.0) was forked from https://github.com/huggingface/transformers
  - added argument *finetuning_iters* in *training_args.py*[54] to pass number of iterations (default: 3)
  - measurement of the fine-tuning time was integrated in the script *trainer.py*[54] inside the *train()* function

[52]The electronic annex can also be retrieved at the GitHub repository https://github.com/annakorotkova/Exploration-of-fine-tuning-and-inferencetime-of-large-pre-trained-languagemodels-in-NLP/settings

[53]for more details about the changes, see https://github.com/annakorotkova/transformers/blob/master/README_Anna's_Notes.md

[54] can be found in the folder *src/transformers*

  - measurement of inference time was integrated to the script *trainer.py* inside the *_prediction_loop()* function, which was later used by the *evaluate()* function[55]

- *sweep files*: contains two sub-folders[56]
  - *finetuning time* contains the sweep configuration files used in the analysis to perform the measuring of fine-tuning time
  - *inference time* contains the sweep configuration files used in the analysis to perform the measuring of fine-tuning time. Those files are additionally marked with "inf" in their file name
  - files with "unc" in their name incorporate the configuration for time measurement for all regarded models, except for distilbert-base-cased and bert-base-cased
  - files with "cased" in their name incorporate the time measurement for distilbert-base-cased and bert-base-cased This distinction was performed since fine-tuning distilbert-base-cased and bert-base-cased failed on several GLUE tasks
  - files with "all" in their name incorporate the configuration for time measurement for all regarded models
  - files with "xlnet" in their name incorporate the configuration for time measurement for xlnet-base-cased. This distinction was performed since xlnet-base-cased takes long fine-tuning times and could therefore not be fine-tuned for all GLUE tasks
  - Note! For some configurations there only exists the configuration file for fine-tuning time measurement for distilbert-base-cased and bert-base-cased. This is the case for tasks where the fine-tuning of these models already failed. Hence, the inference time could not be measured.
  - files with "crashed" in their name are configuration files created for runs that crashed before and should be rerun

[55] all modifications to the codes are marked with a comment "# added by Anna"

[56] Important to note: this file also contains sweep files of runs that crashed in order to rerun them. Hence, the content of some files might overlap. They were added to match the runs that are logged in wandb

  - files with "reduced" in their name contain a limited sweep since it is run for try out reasons (if prior runs failed for example)

Hereby, the fine-tuning and the measurement of the fine-tuning as well as inference time was performed on a virtual machine. To execute the measurement of the fine-tuning and inference time, respectively, the following steps were executed:

- Cloning of the forked and modified repository https://github.com/annakorotkova/transformers
- Adding downloaded GLUE files to workspace folder in Visual Studio Code
- Creating sweep configurations
- Run sweeps with the following commands in the terminal:
  - To enable an execution of the measurements without crashing if the computer turn to standby mode, the command **`screen -S mysession`** should be used[57]
  - **`wandb sweep {}`** with {} being the placeholder for the respective sweep name, e.g. sweep_wnli_cased.yaml.
    → **`wandb sweep sweep_wnli_cased.yaml`**
  - The terminal outputs some sweep characteristics (such as sweep ID or the link where the sweep can be viewed). It also puts out a line of the following form: "Run sweep agent with: wandb agent anna_korotkova/transformers-examples_text-classification/o7ifco7k"[58]
    This line should be copied and executed in the terminal

Figure 34 illustrates the last key point.

```
ubuntu@anna-gpu:~/transformers$ wandb sweep sweep_wnli_cased.yaml
wandb: Creating sweep from: sweep_wnli_cased.yaml
wandb: Created sweep with ID: o7ifco7k
wandb: View sweep at: https://app.wandb.ai/anna_korotkova/transformers-examples_text-classification/sweeps/o7ifco7k
wandb: Run sweep agent with: wandb agent anna_korotkova/transformers-examples_text-classification/o7ifco7k
```

Figure 34: Run a sweep, which is logged at wandb

[57] the session can be checked with the command `screen -r -d mysession`
[58] o7ifco7k is the sweep ID (in wandb the runs are logged under this name)

After following these steps, the fine-tuning for the respective sweep should be performed. Hereby, it should be noted that a wandb account is required in order to enable this process[59]. The logged runs from our analysis can be retrieved at https://wandb.ai/anna_korotkova/transformers-examples_text-classification/sweeps?workspace=user-anna_korotkova. The names of the respective sweeps include their respective sweep ID and some information about the sweep itself. The naming of these follows a similar to the one of the *sweep files*.

Furthermore, the electronic annex incorporates the following files and sub-folders:

- *wandb time data* contains two sub-folders
  - *finetuning time*: contains contains all Excel datasets which incorporate the fine-tuning time data that was used in the analysis (within the script *Analysis finetuning time.ipynb*)
  - *inference time*: contains all Excel datasets which incorporate the inference time data which was used in the analysis (within the Python script *Analysis inference time.ipynb*).
  - All Excel files were downloaded from wandb (the respective sweeps can be viewed at https://wandb.ai/anna_korotkova/transformers-examples_text-classification/sweeps?workspace=user-anna_korotkova. The data can be retrieved when clicking on the respective sweep, followed by clicking on the sweep table). It should be noted that the files do not match exactly the wandb sweeps since some files were revised. This was done, so that no runs are included two times since some sweeps were rerun because they crashed the first time.
- *Analysis finetuning time.ipynb*
  - Descriptive analysis of the minimum fine-tuning time of finished runs for different hyperparameter combinations
  - ANOVAs of fine-tuning time grouped by task, model, maximum sequence length and batch size, respectively

[59] A helpful website for enabeling a quick start is https://docs.wandb.com/quickstart

  - Log-linear regression with log. fine-tuning time as the dependent variable
  - Regression Random Forest with log. fine-tuning time as the dependent variable
  - Exploration of the relation between fine-tuning time and accuracy

- *Analysis inference time.ipynb*
  - Descriptive analysis of the minimum inference time of finished runs for different hyperparameter combinations
  - ANOVAs of fine-tuning time grouped by task, model, maximum sequence length and batch size, respectively
  - Log-linear regression with log. inference time as the dependent variable
  - Regression Random Forest with log. inference time as the dependent variable
  - Exploration of the relation between inference time and accuracy

- *plot* contains two sub-folders
  - *fine-tuning time*: This sub-folder contains all plots that were generated during the analysis of the fine-tuning time (run script *Analysis finetuning time.ipynb*)
  - *inference time*: This sub-folder contains all plots that were generated during the analysis of the inference time (run script *Analysis inference time.ipynb*)

- *finetuning_time_table.xlsx*
  - Table of fine-tuning times grouped by model, number of examples (task size), maximum sequence length and batch size (run script *Analysis finetuning time.ipynb*))
  - The fine-tuning time is colored with a color palette (from Excel) that ranges from green to red. Green stands for small fine-tuning times and red for high ones

- *inference_time_table.xlsx*
  - Table of inference times grouped by model, number of examples (task size), maximum sequence length and batch size (run script *Analysis inference time.ipynb*))
  - The inference time is colored with a color palette (from Excel) that ranges from green to red. Green stands for small fine-tuning times and red for high ones