Contents

1 Introduction

Nowadays natural language processing is a huge field. Not only for computerlinguistics itself, but also in the field of machine learning. It provides us many models to handle written and spoken natural language. Especially in the written language, the research made big improvements in the last few years.

Two representatives for such improvements are ELMo[Pet+18] and BERT[Dev+18]. While ELMo relies mostly on LSTMs and follows the feature-based approach, BERT follows the fine-tuning approach and relies on transformer neural networks. Transformer networks use attention blocks stacked above each other and in parallel. BERT was released by Google and is also for them a big improvement.[Gru19] Especially when it comes to parallelisation, BERT outperforms ELMo by far due to the parallelisable attention blocks.

In this report we give an overview over transformer networks and BERT in chapter 2 and 3. On top of this, a task transfer is performed, where we want to use a pretrained BERT to perform rating prediction of amazon reviews, which will be explained in chapter 4. Afterwards there will be an evaluation of the experiments on the task, where we will also take a look on what BERT focuses with its attention mechanism.

2 Transformers

2.1 Overview

Transformer models can be divided in two main components:

- Encoder
- Decoder

Each of these have a stack of six similar blocks. The concrete architecture for each can be seen in figure 1.

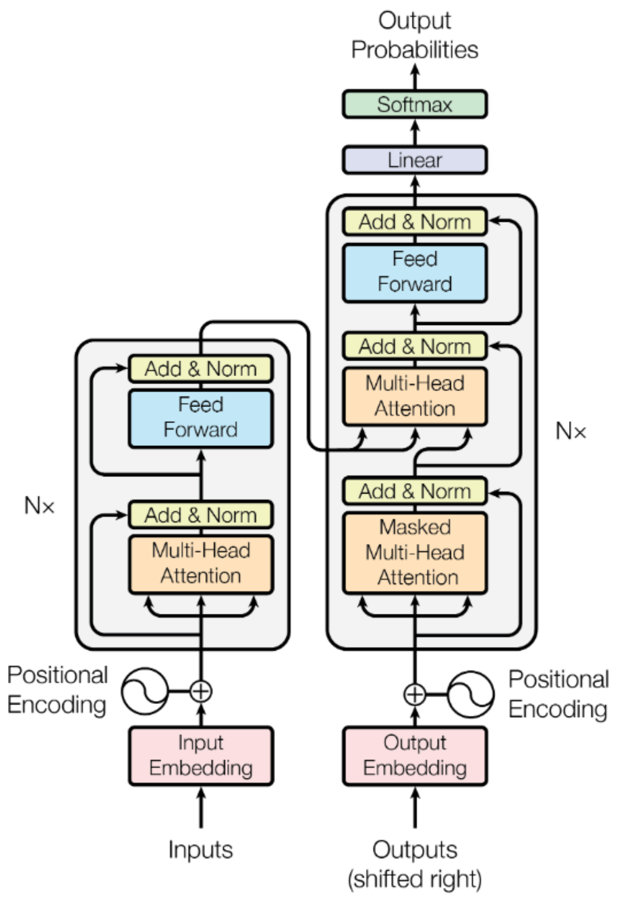


Figure 1: Architecture of a transformer, where N=6 in our example. Source: [Vas+17].

On the left side of the figure we can see the Encoder, which consists of two sub-layers. One multi-head attention layer (which will be explained lateron) and a normal feedforward network. Each of these two sub-layers have residual connection and normalisation around it.
The decoder works similar to that, but has an additional masked multi-head attention in the beginning. The multi-head attention output from the encoder is part of the input for the multi-head

attention layer of the decoder. The difference for the first input of the decoder is that the input for the first decoder block is shifted by one to the right **and** masked, so the transformer can only take positions up to j to predict position j+1.[Vas+17]

2.2 Input & Output

The input and output of the network use an additional embedding layer, which use a shared weight matrix with output of dimension d_{model}. Additionally, the output has a softmax to compute the next token probabilities. On top of that, there is also an positional embedding for the inputs. [Vas+17]

2.3 Feedforward Layer

The feedforward layer is simply a fully connected network with two layers, with an ReLU activation in between those two layers. [Vas+17]

2.4 Attention Mechanism

Due to we want to explain multi-head attention, we first have to define the initial attention mechanism. There are several attention formulas out there, but the BERT paper used the **scaled dot product attention**, which is defined as followed.

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

, where Q are the queries (=state of decoder), K are the keys(=state of encoder) and V are the values(=state of encoder).
Due to the Ashish Vaswani et al.[Vas+17] **really(!!)** do not dive into the attention mechanism, it will be explained here. We can decompose the the attention from above into

$$\begin{aligned} score(Q, K) &= QK^T \\ attention(V, K, Q) &= softmax(\frac{score(Q, K)}{\sqrt{d_k}})V \end{aligned}$$

, where the score is a function to measure the similarity of the decoder vectors to the encoder vectors. This gives us a scalar with the meaning of how similar word i is to word j. It could be replaced by any other score function. Additionally the Ashish Vaswani et al. just normalize with the square root of the dimension d_k. After we have the similarity, which is just an unscaled scalar, we scale it with the softmax function. Finally we just take the encoder word states and multiply them by the similarity we just calculated.[Vas+17]

2.5 Attention Mechanism on LSTM

To better understand the key, value, query attention mechanism, we will visualize this with help of an LSTM. **Mind, that the transformer is no LSTM. This is just for understanding, for what key, value and query stands for.**

- Calculate $score(Q, K) = QK^T$, where Q comes from the decoder and K from the encoder.

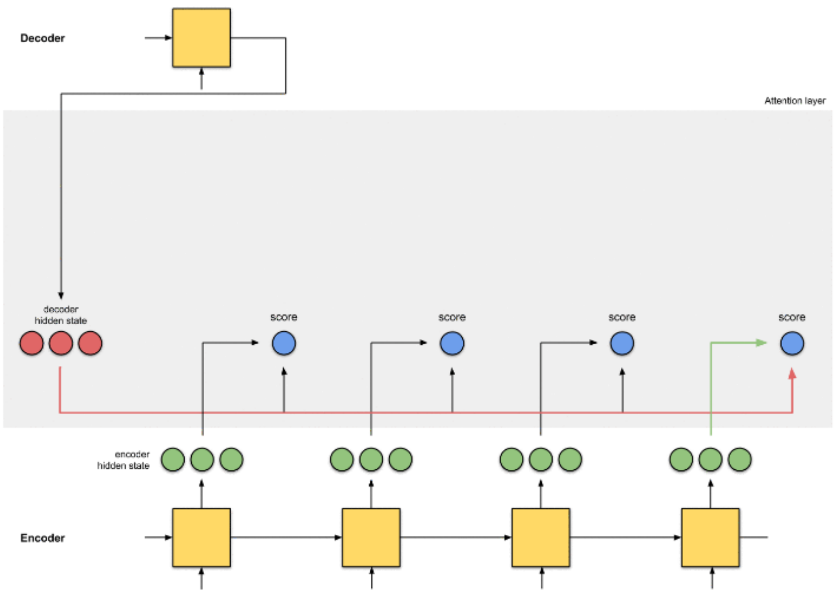


Figure 2: [Kar19]

$\longrightarrow$ Outputs a scalar for each score , which can be concluded as vector. (Also it should be scaled by d_k for the transformer case.)

- Softmax the score vector from before. $\longrightarrow$ Normalized similarity
- Multiply each encoder state by this similarity/importance.

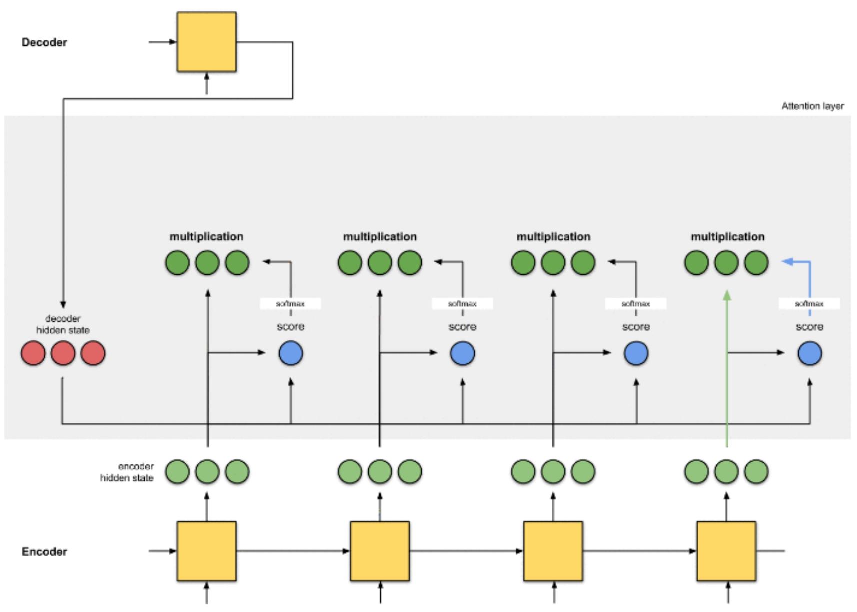


Figure 3: [Kar19]

⟶ We have the so called alignment vector now.

- Sum everything up and arrive at next decoder state.

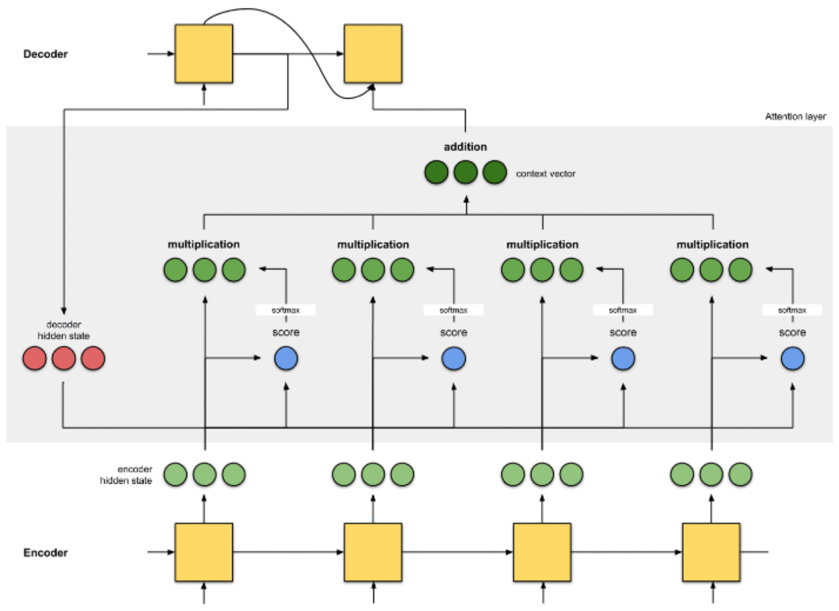


Figure 4: [Kar19]

(Overall source for this chapter: [Kar19])

2.6 Multi-Head Attention Mechanism

Analogous to the attention mechanism explained above with d_{model} dimensions as output, the multihead attention performs the attention with h different projections. In the end these h different attentions are concatenated and reduced with another weightmatrix multiplication, resulting in:

$$\begin{aligned} MultiHead(V, K, Q) &= Concat(head_1, ..., head_h)W^O \\ head_i &= attention(VW_i^V, KW_i^K, QW_i^Q) \end{aligned}$$

, where $W_i^V \in R^{d_{model} \times d_v}, W_i^K \in R^{d_{model} \times d_k}, W_i^Q \in R^{d_{model} \times d_k}, W^O \in R^{hd_v \times d_{model}}$.[Vas+17]

3 BERT

After we defined the transformers, we can continue to present BERT from Jacob Devlin et al.[Dev+18].

3.1 Architecture

In general, BERT is based on encoders of the transformer architecture. As simply as this, we can see it in figure 5.

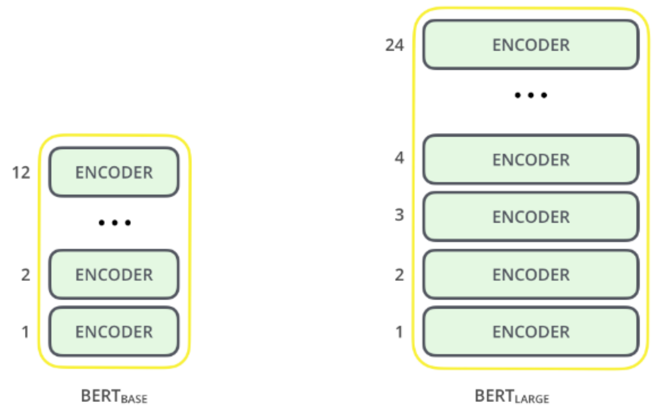


Figure 5: Architecture of BERT based on encoders of the transformer. There are two architectures: $BERT_{base}$ = small network, $BERT_{large}$ = bigger network. Source: [Riz19].

Each Encoder layer consists of encoder blocks which are connected bidirectional to each other.

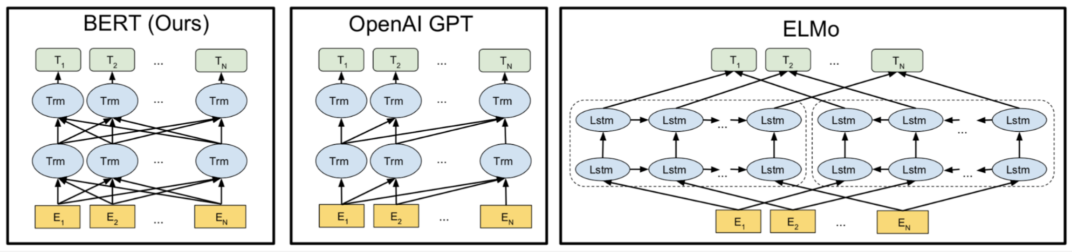


Figure 6: Comparison of BERT to ELMo (=based on bidirectional LSTM cells) and OpenAI GPT (=Based on unidirectional transformer blocks). Source: [Dev+18].

In figure 6 we can not only see the comparison of BERT to other models. We can also see the “Trm” cells connected bidirectionally to each other.[Dev+18]

3.2 Input & Output

After we defined the architecture of BERT, we will have to define the input and output. Due to we have multiple training targets lateron (Next sentence prediction, sentence classification, ...) we have different indicators for the input. There are three different embeddings, which are put together.

- Position Embedding
  A positional embedding, which is learned during training.
- Segment Embedding
  A learned embedding, which indicates if we have the first sentence or the second sentence (for example in next sentence prediction).
- Token Embedding
  Here the words are embedded and marked with different seperators.
  - $[CLS]$ = Begin of sentence
  - $[SEP]$ = Sentence seperator
  - $[MASK]$ = Masking 15%
    - 80% of 15% Use the [MASK] token
    - 10% of 15% Replace with random embedding
    - 10% of 15% Keep it unchanged

[Dev+18] All three embeddings are finally the input for the network. We can see the process in figure 7.

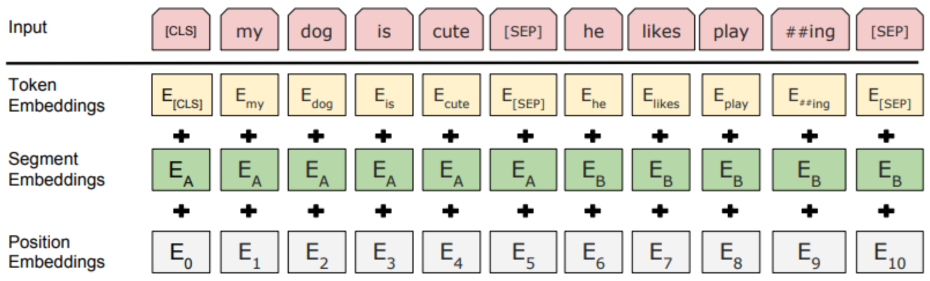


Figure 7: The 3 different embeddings summed up and representing the input for BERT. Source: [Dev+18].

The different tasks and input token embeddings or outputs can be seen in figure 8

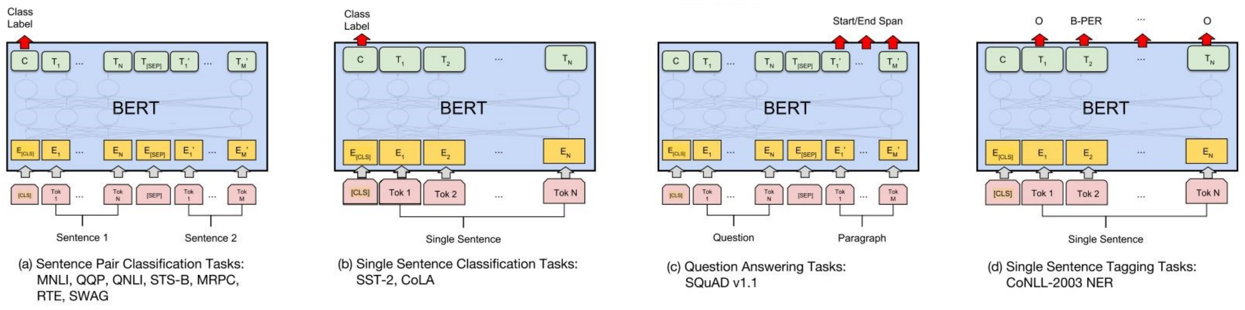

Class Label
BERT
Sentence 1
Sentence 2
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG
Single Sentence
(b) Single Sentence Classification Tasks: SST-2, CoLA
Start/End Span
Question
Paragraph
(c) Question Answering Tasks: SQuAD v1.1
O
B-PER
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Figure 8: Different tasks for BERT. a) Inputs 2 sentences with [SEP], outputs single class label b) Inputs single sentence without [SEP] for classification. c) and d) analogous, but with output for question answering or tagging. Source: [Dev+18].

For example we have two sentences seperated by the [SEP] tag for the sentence pair classification task. The class is then output to the first cell of the BERT output. We can also have just single sentence classification. For this we just leave out the [SEP] tag.
Outputs which need to generate a new sentence (for example giving an answer or POS tagging) output their results to the remaining neurons as seen in figure 8

3.3 Training

Bert is trained on different tasks with the above mentioned embeddings. There exist two tasks.

1. Masked language model (=Masked LM)
   Mask words and predict them with softmax layers.

2. Next sentence prediction (=NSP)
   Predict with first output [CLS]/C, if the second sentence is truly the next sentence of the first

input sentence

Both tasks are visualized in figure 9.

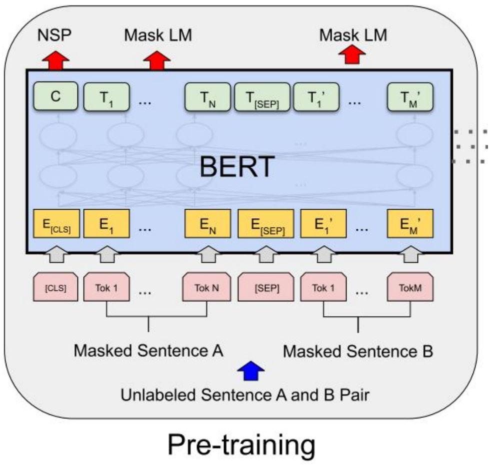


Figure 9: Masked LM and NSP visualized in one figure.Source: [Dev+18].

[Dev+18]

4 Task

After the introduction of BERT, we can continue with the task of amazon review prediction mentioned in the introduction.

4.1 Dataset

The dataset is the “Amazon Review Dataset” of Jianmo Ni[NLM19], which was first released in 2014 and updated in October 2018. It contains many metadata and our needed review text and star rating. The raw review data contain 233 million reviews. Due to this is too much for our purpose, we specialized to the Cell Phones and Accessories data, which contains 10 million reviews + ratings. [NLM19]

4.2 Goal

The goal is to predict the rating (stars from 1 to 5) of a user who wrote a review with a pretrained BERT model. This requires to add a classifier on top of a pretrained BERT model. Afterwards we want to take a look at what BERT focuses on.

4.3 Metrics

We use four different metrics.

- **Catecorical Cross Entropy Loss**
  The normal cross entropy loss with categories.
- **AUC-ROC Score**
  A standard measure for multiclass classification, which takes the area under the ROC curve into account. The ROC curve is the curve of sensitivity (also known as recall) and (1-specificity).

  $$sensitivity = \frac{TP}{TP + FN}$$
  $$specificity = \frac{FP}{FP + TN}$$

  Finally we plot sensitivity vs. (1-specificity) and get the ROC curve. For the AUC-ROC score we just calculate the area under that curve. The AUC-ROC score can be seen as confidence to distinguish between classes. It is in the interval [0,1]. A big value indicates a better seperation in prediction.[DSo18]

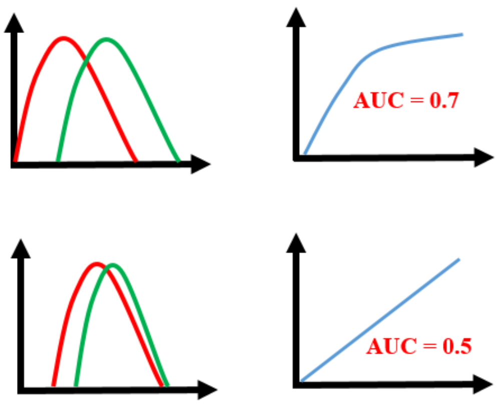


Figure 10: The AUC-ROC score with curves. Left represents two probability distributions. Right shows the AUC-ROC score with its curve. Higher=Better=Less overlapping distributions. Source: [DSo18].

- **Percentage of Correct Predictions**
  This is simply just the correctly predicted results divided through the amount of samples times 100.

$$CP = \frac{\#correct}{\#samples} \cdot 100$$

- **Pseudo Percentage of Correct Predictions**
  We relax the Percentage of correct predictions a bit, due to 1,2 or 4,5 star ratings are very close to each other and even humans often can not distinguish between them by just reading the text. BERT is still learned on predicting ratings from 1-5.
  Therefore we count a prediction as correct, if it is in range of plus or minus one of the original rating.

5 Experiments

5.1 Dataset

The datasets classes are equally distributed in each of the test, training and validation dataset ($p(c_i) = p(c_j)$ for all three splits within each split). The ratio of splitting is about:

- Training data 70%
- Testing data 20%
- Validation data 10%

Additionally we have a filtering, that each review has to be at least 100 tokens long.

5.2 Hyperparameters

Our has several components and hyperparameters:

- Batch size
  Fixed at 6 due to hardware limitations
- Number of labels = $\#labels$
  First we experimented with 5 labels (0-4) and used the "Pseudo Percentage of Correct Predictions" to get some fair conditions.
  After some experiments (more in the next section), we decided to experiment with the number of labels. We concluded the labels (0,1)=0, (2)=1, (3,4)=2 due to similarity. With this approach the "Pseudo Percentage of Correct Predictions" was **NOT** calculated for evaluation.
- Learning rate for the classifier on top of BERT = $LR_{classifier}$
- Learning rate for BERT itself for fine-tuning = LR_{BERT}
- Weight decay = Regularization (L2)
- Learning rate scheduler
  - Decay rate = LR decay
  - After how many epochs decay rate should be applied = step size (fixed to 5 after some experiments)

On the remaining parameters, we performed a grid search to find the optimal parameters in a small are.

5.3 Architectures

There are two architectures, which should be mentioned

- BERT
  For BERT we took the "bert-base-uncased" model. We made that choice due to lack of GPU memory and the not so complex task. The uncased model we took, because it performed a bit better according to the documentation of the implementation.
  Bert-base-uncased consists out of 12 layers, 12 heads (=110 million parameters) and a 768 hidden vector size. This was a fixed choice due to hardware limitations.
- Classifier on top of BERT
  The classifier was a single fully connected layer. Putting an LSTM on top would opposite the transformers basic thought and would not make any sense here.

6 Results

6.1 Parameters & Architecture

The search parameter combination can be seen below. Not all parameters were tested in large epoch counts due to long training times.
The best reults are shown in table 1

# labels	LR classifier	LR BERT	LR decay	Weight decay	Cross Entropy averaged	AUC-ROC score	% correct	relaxed % correct
5	0.001	0.00001	0.5	0.01	0.230|0.231	0.786|0.793	51.7|50.0	82.8|84.85
5	0.01	0.0001	0.5	0.01	Similar to above, but less stable			
5	0.1	0.001	0.5	0.01	Tested, but too unstable			
5	0.1	0.00001	0.5	0.01	Tested, but too unstable			
3	0.001	0.00001	0.5	0.01	0.140|0.133	0.84|0.88	69.8|74.65	
3	0.01	0.0001	0.5	0.01	Similar to above, but less stable			
3	0.1	0.001	0.5	0.01	Tested, but too unstable			
3	0.1	0.00001	0.5	0.01	Tested, but too unstable			

Table 1: Results given for (test|validation)-dataset. Relaxed percentage is missing for 3 labels. The metric would not make any sense here.

Smaller learning rates were also tested, but converge very slow, which was not practicable due to limited hardware resources. The curves for the green marked network are:

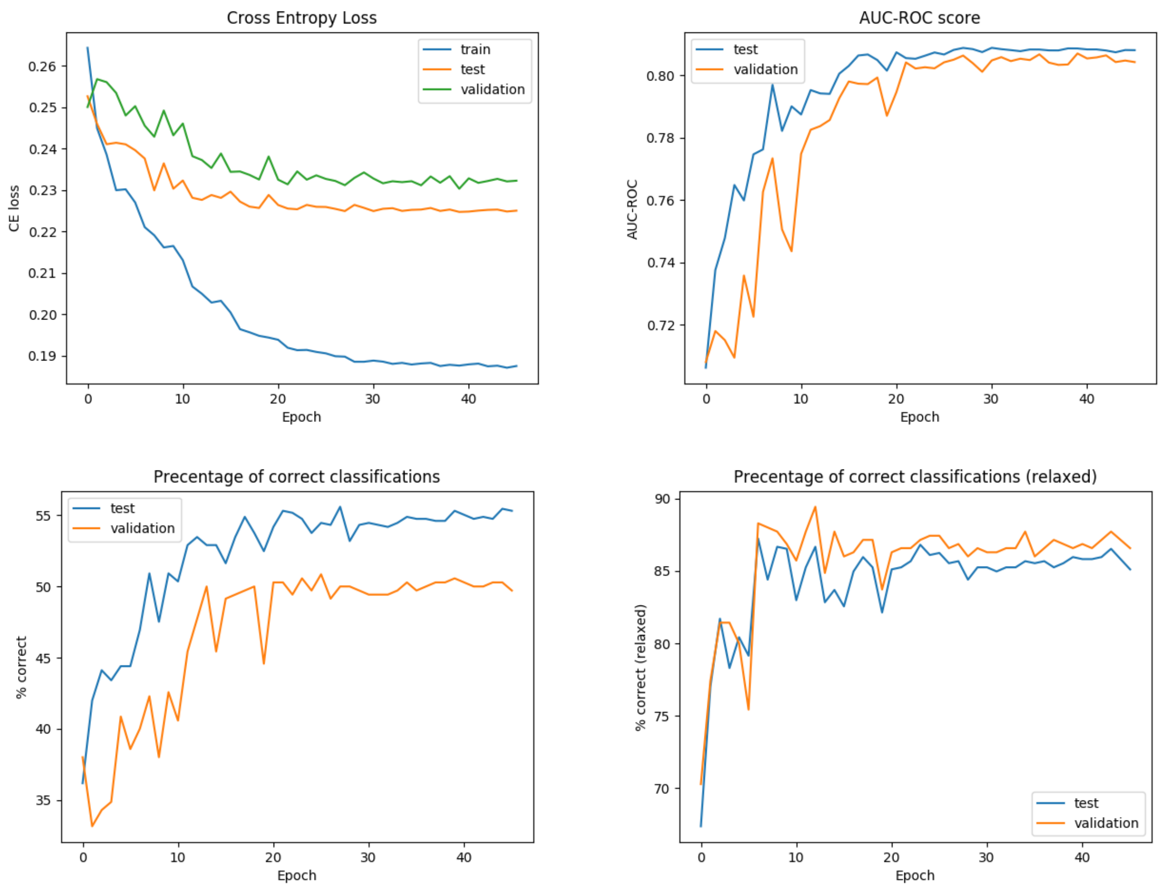


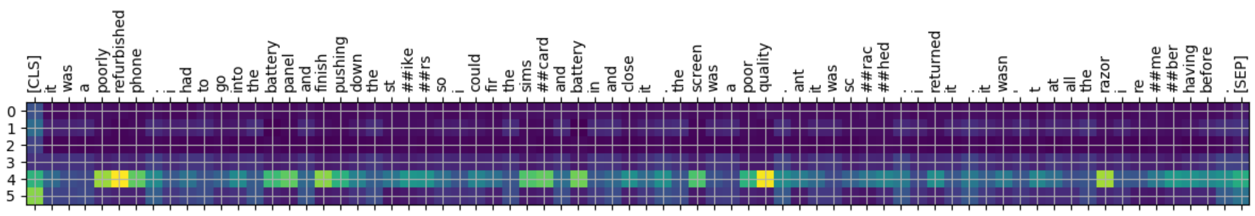
Figure 11: The four different metrics. Pay attention to the minimum y values.

We can see, that the validation, test and training curves behave similar, which indicates we did not overfit. The different learning rates for classifier and BERT come due to the finetuning approach with a pre-trained BERT. The tests which were aborted trained for round about five epochs, but did not converge in any way. Due to they have a bigger learning rate, they should converge faster. If they do not, this indicates unstability.
The AUC-ROC score is converging to a value round about 0.82 for both, test and validation dataset. This indicates, that the classes are halfway good seperable by the classification network. This makes sense, because a seperation of 3 or 4 stars is very hard to achieve in classification even for humans.
The percentage of correct classification stabilizes at 50% (85% for the relaxed). Especially the jittering around the tenth epoch of the relaxed metric indicates, that the network classifies very wrongly. For example it classifies 1 star ratings as 4 star ratings in the complete wrong corner of our labels. This gets better with training time and vanishes in the end.

6.2 Attention of the Best Model

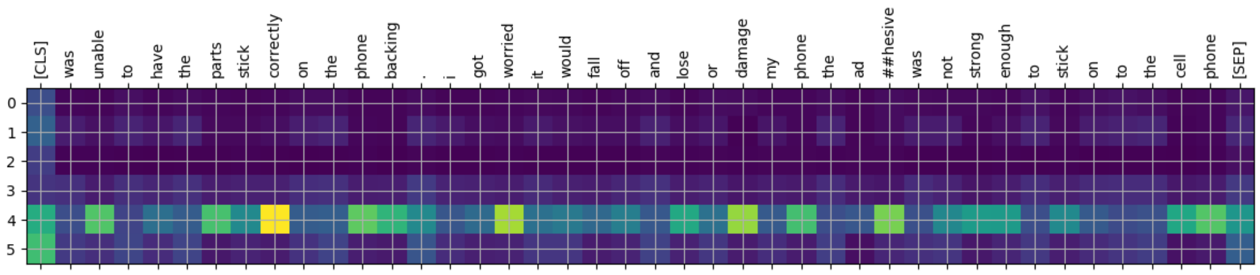
After the best result was picked, we continued with analysing the attention values. Attentions were analysed for each label seperately for the best classification result network. We plotted the attention on the first position (=[CLS]) to each other position/word for the last 6 attention layers of BERT. This was chosen, because we just use the CLS output position for the classifier anyway.
We can see, that the second last layer has most impact. The brighter the color(=yellow), the bigger the attention value.

- **1 star rating**



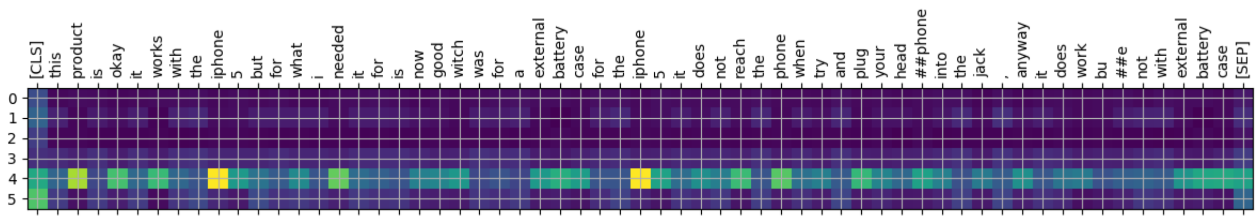
We can see the focus in "poorly refurbished phone" and “poor quality”.

- **2 star rating**



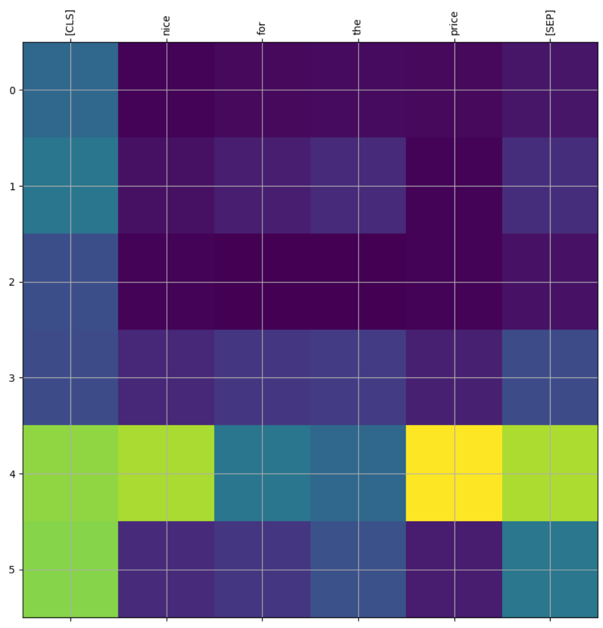
We can see the focus on words like “damage” or “unable”, but also the focus on “correctly”, which the network might mistake for a positive review.

- **3 star rating**



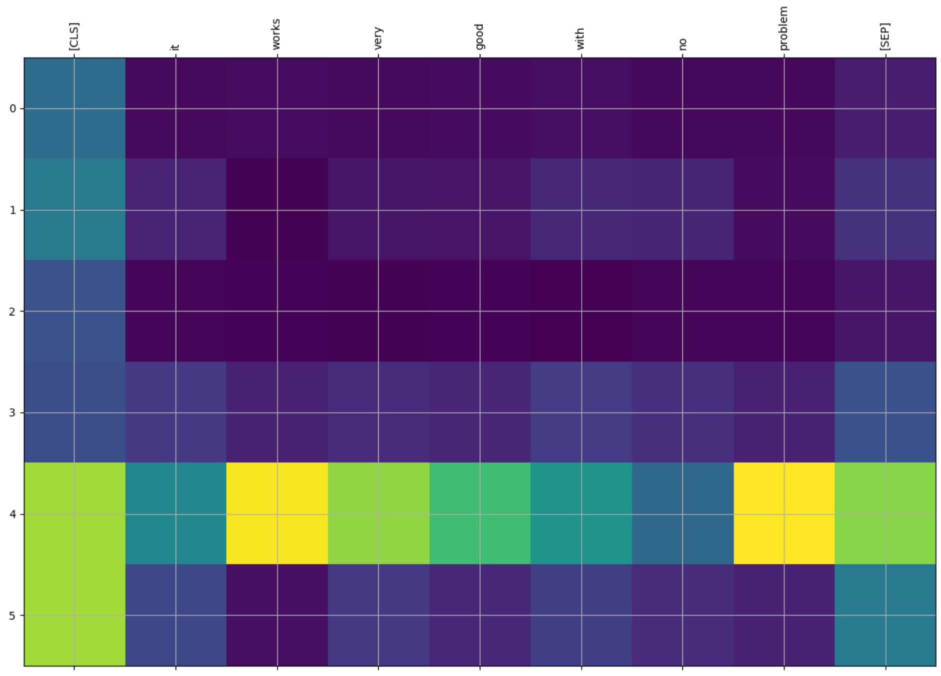
We can see the focus on “product” and “okay”. We also can see, that the word “iphone” seems to be interesting.

- **4 star rating**



Without any words, we can see that "nice price" is the most important thing in this review

- **5 star rating**



Here we can see the focus on "works very good", which is of course an indicator for a good review.

7 Problems

- Long training durations
  To make the classification converge, we needed very long training periods.
- Parameter search
  The long training duration and parameter search needed much time. Even if we shortened the parameter search training time and only trained a few epochs and looked for convergence.
- GPU memory
  This is one of the big disadvantages. While Google has TPUs with several GB of memory, we only had a GTX 1070 with 8GB of memory. Therefore we could only fit a batch size of 6 in it, even if the sequence length was shortened.
- You need to save the BERT tokenizer, which led to the fact that I needed to retrain many models, because I could not revert the conversion to IDs (=input for BERT). This costed me much time.

8 Summary

Overall, we saw that the network learned to understand the context and had the attention often on the proper words. With 52% correct classification rate the network performed ok. We can say, that the network classifies most samples in the correct region ranging from 1 to 5 stars, because the relaxed classification rate is over 80% .
The losses converged for training, testing and validation dataset. Also the AUC-ROC score showed, that the classifier is ok in seperating the classes. Still, it is unlikely that we get a score of 1, because the star ratings are discrete, but the human rating is a fluid transition.

Hauptquellen

[Dev+18] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *CoRR* abs/1810.04805 (2018). arXiv: `1810.04805`. URL: `http://arxiv.org/abs/1810.04805`.

[Kar19] Raimi Karim. *Attn: Illustrated Attention.* [Online; Dezember-2019]. 2019. URL: `https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3`.

[NLM19] Jianmo Ni, Jiacheng Li, and Julian McAuley. *Justifying recommendations using distantly-labeled reviews and fined-grained aspects| Empirical Methods in Natural Language Processing (EMNLP).* [Online; Dezember-2019]. 2019. URL: `https://nijianmo.github.io/amazon/index.html`.

[Pet+18] Matthew E. Peters et al. "Deep contextualized word representations." In: *CoRR* abs/1802.05365 (2018). arXiv: `1802.05365`. URL: `http://arxiv.org/abs/1802.05365`.

[Vas+17] Ashish Vaswani et al. "Attention Is All You Need." In: *CoRR* abs/1706.03762 (2017). arXiv: `1706.03762`. URL: `http://arxiv.org/abs/1706.03762`.

Bild und Nebenquellen

[DSo18] Jocelyn D'Souza. *Let's learn about AUC ROC Curve!* [Online; Dezember-2019]. 2018. URL: `https://medium.com/greyatom/lets-learn-about-auc-roc-curve-4a94b4d88152`.

[Gru19] Jan Grundmann. *Google BERT Update: Alle Infos Analysen.* [Online; Dezember-2019]. 2019. URL: `https://blog.searchmetrics.com/de/google-bert-update/`.

[Riz19] Mohd Sanad Zaki Rizvi. *Demystifying BERT: A Comprehensive Guide to the Groundbreaking NLP Framework.* [Online; Dezember-2019]. 2019. URL: `https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/`.