DD2424 Project: Amazon Product Reviews Summarization – Comparing LSTM-based and Transformer-based Approaches
Group 190

Adam Jacobs	**Cuong Duc Dao**	**Fynn van Westen**	**Donggyun Park**
adamjac@kth.se	cuongdd@kth.se	fynn@kth.se	donggyun@kth.se

Abstract

In this project, we are interested in comparing the usages of two competing state of the art approaches in sequence modeling, i.e., transformer and long short-term memory, in abstractive text summarization. Specifically, we experiment with transformer-based and LSTM-based encoder-decoder models for summarizing Amazon product reviews by implementing the architectures in Pytorch. A qualitative comparison between the generated results follows thereafter, which shows similar results. Due to a lack of time and resources a larger quantitative comparison was not made.

DD2424 Deep Learning in Data Science. (May 2021), Stockholm, Sweden.

1 Introduction

There are several natural language processing (NLP) models for various types of tasks. The main interest of our group is to compare the complexity and the performance of LSTM and a transformer architecture on text summarization and possibly fine tune them to increase their performance on Amazon product reviews dataset.
It was planned that both of those models will use text embeddings from BERT [1] since it is a pre-trained model. The thought here was to fine tune it along with our own purpose to observe and demonstrate its performance on the given dataset. But as seen in methodology, we found it more feasible with a simpler Glove embedding [2].

2 Related Work

2.1 Long Short-Term Memory

Long Short-Term Memory [3] is a specific type of recurrent neural networks used to deal with dependencies of inputs in sequence. In large RNN, the network starts to forget the first inputs after many updates, this is caused because of the information lost at each step going through the RNN. In order to solve this issue, long term memory is needed and that is when LSTM appears. Its powerful capacity to store long term dependencies comes from the abilities of its "gates". These are multiplication operations, and it is decided what information is going forward and forgot.

2.2 Transformers

The most famously known Transformer architecture was proposed in the paper "Attention is all you need" [4] in 2017. On a simple level a transformer is a machine learning architecture consisting of an encoder and decoder component. The great advantage of transformers compared to various types RNNs is that they allow for parallelization computation of sequences during processing time (instead of re-occurences like RNNs), increasing learning speed and computational efficiency by a manifold. To compensate for the loss of positional information in the sequence, the content embeddings are extended with a positional embedding. Combined with a multi-head attention mechanism, this allows Transformers to keep sequential information non the less their parallel processing approach.

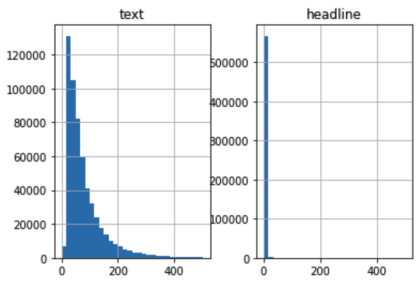
2.3 Self-Attention

Simplified speaking, what self-attention does can be described as, weighting individual words in the input sequence according to the impact they should have on the generational process of the target sequence.
Transformer use multi-head attention, which is an attention module that runs multiple self-attention calculations in parallel and concatenates the results to a final output, that is linearly mapped to the output dimensionality defined by the model parameters. "Intuitively, multiple attention heads allows for attending to parts of the sequence differently" [Lilian Weng]. A closer description can be found in the appendix 6.2. Finally the model utilizes simple feed-forward layers with dropout for regularization purposes. Putting all these components together as in figure 7 gives us the iconic transformer architecture described in [4].

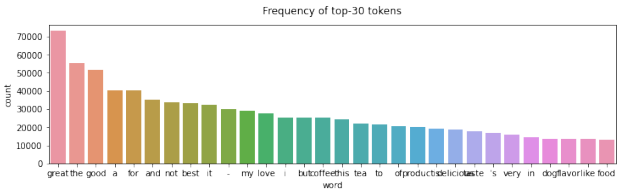
3 Methodology

3.1 Data pre-processing

From the Amazon fine food reviews dataset, the text reviews and the summaries is split ($568,427$ samples) to training (80 %), validation (80 %), and test (10 %). The characters believed to not have any effect on training for text summarization are removed with spaCy [5], and converted to lowercase. Also, the sentences need to have a fixed max length. The input reviews are longer than the summary labels so two different sequence lengths is chosen. In figure 1: on the left shows number of samples by length, right shows most frequent words. Neutral words, e.g 'the', 'a' or 'for', should not degrade



(a) The number of samples per length



(b) Distribution of top-30 most frequent tokens in the dataset

Figure 1: Statistical overview of the dataset

the learning, but the surplus of positive 'great' and 'good' probably unbalances the dataset, which we did not account for initially.

3.2 Sentence embedding

During our planning phase it became evident, that the initially planned use of BERT [1] for the embedding would inflict much more complexity, since the pretrained model is a fully functional transformer by itself. We felt like this would be simply utilizing the work of others, and go against our own set learning goal for this project. Instead we utilized pre-trained 50dim 6B GLOVE-embeddings [2] from Stanford University [6]. The reasoning behind this decision is to ensure a fair comparison with the from scratch implementation of the LSTM architecture.

After the initial tests we decided to limit the length of input tokens to a size of 128 and the summaries to a length of initially 10, which was extended to 17. Including the <SOS> and <EOS> tokens. The choice of these parameters was based on analysis of the underlying dataset, where the inputs had max length 3432 and average 80.3 - and max 42 and average 4.3 for the summaries.

3.3 LSTM-based model

3.3.1 LSTM-base encoder-decoder model

We employ an LSTM-based encoder-decoder architecture for this task. The encoder of this architecture is a single-layer unidirectional LSTM [7] which takes as an input a batch of embedded sentences with size (n_batch, 128, 50) where 128 is the sequence length (i.e., length of each embedded sentence) and 50 is the size of the dimension from GloVe embedding. The encoder produces a hidden context vector of that sentence of the dimension (n_batch, 128, 128). The decoder of our model is a unidirectional LSTM that decodes the last hidden state of the encoder into a sequence of output tokens.

3.3.2 Training

The general training scheme is to leverage mini-batch stochastic gradient descent. First, we initialized the weights of the embedding with the pre-trained 6B 50-dimensional GloVe [6] and freeze these weights during training. Other model's weights were initialized with Xavier uniform and the biases were set to 0.0. The input sentence is forwarded through the embedding layer to get an embedded representation. Then each token of the input was fed into the encoder together with the hidden state from the last time step, we keep the last hidden state of the encoder as the context vector. Next, we feed the context vector and a <SOS> token to the decoder, which would generate the next likely word given the previous word and the hidden state of the decoder.
We used cross-entropy as our loss and monitored its values on both training and validation passes. We use Adam optimizer [8] with learning rate $\eta = 0.001$, batch size of 64. We measured the performance of the model on the validation set after every 1000 update steps and plot the loss in figure 4. Additionally, we applied dropout to the output of the embedding layers for both the encoder and decoder with the probability $p = 0.5$ during training.

3.3.3 Generation

The generation phase is similar to the forward pass described in the training. The input test sentence is first forwarded through the embedding layer. Then each input token is fed to the encoder. The last hidden state of the encoder is used as the context vector. The decoder then takes this context vector together with the <SOS> token as its input and starts generating one token at a time. The generated token and the hidden state of the the decoder at time step $t-1$ are used as its input in time step t. The generation stops when we encounter the <EOS> tokens or when the length of the generated output goes beyond 10.

3.3.4 Evaluation

During training, we monitor both the train batch loss and the validation loss as indicators of our model. We also frequently print out the predictions of the model after every couple hundreds update steps. A proper evaluation method for summarization quality would be the ROUGE [9] score. However, due to the limited time we had, we did not compute that score. Instead, we manually inspect the quality of our generated summary.

3.4 Transformer-based model

3.4.1 Transformer Architectures

We tested two architectures: (1) 3 layers, 5 heads, 50d embedding, 512d feedforward; (2) 6 layers, 10 heads, 100d embed, 512d feedforward. We made the architecure smaller than the baseline [4] to train quicker. In (1) we have a frozen pre-trained 6B 50 dim glove [6]. In (2) we concat the glove with a 50d trainable embedding so that each head gets more data points.

3.4.2 Training

When it comes to training the transformer we use Teacher Forcing by giving the decoder the ground truth labels, i.e the text that it is supposed to generate, as text input. See an example visualization in figure 5 in the Appendix. Batch sizes over 100 was tested first but made the GPU run out of memory. In the end a batch size of 60 seemed to suffice. Xavier initialisation was used. The optimizer Adam [8] was used. When it comes to the learning rate we tested a fixed learning rate, linearly annealed, cyclical [10], and also together with a warmup [11]. See the results section for more on this.

3.4.3 Generation

For the generation process we generate summaries from random samples in the test set, and then qualitatively assess them. The results are displayed in figure 2a. The process of generation is done autoregressively. See figure 6 in the appendix for a visualisation.

3.4.4 Evaluation

For training and evaluation (on the validation set) cross entropy loss was utilized. See a plot of this in 3a. The loss was further used for experiments with extensions of the model like learning rate scheduling or warm-up. We also computed an accuracy that was the average of correctly guessed words per batch, which is just the number of correct guesses divided by the non-padded lengths of the sentences. See appendix section 6.3 for how we implemented it. The loss and accuracy is used to track the training, while the test generations were assessed qualitatively due to time constraints.

4 Experiments

For both the LSTM and the Transformer model the goal was to generate text and do a qualitative comparison. We did not have time to look into any more advanced quantitative comparisons. See Figure 2 for examples of generated text from both architectures.

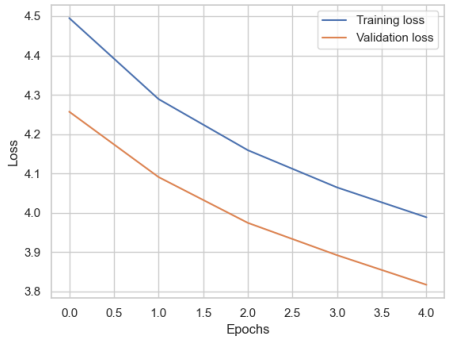
```
GENERATED:
 the best
REAL:
<SOS> matters of taste <EOS> <PAD> <PAD> <PAD> <PAD> <PAD>

GENERATED:
 not bad , but not the best
REAL:
<SOS> waste of money <EOS> <PAD> <PAD> <PAD> <PAD> <PAD>

GENERATED:
 the best tea ever
REAL:
<SOS> great green tea <EOS> <PAD> <PAD> <PAD> <PAD> <PAD>

GENERATED:
 a great product
REAL:
<SOS> they are powdered and they are eggs <EOS> <PAD>

GENERATED:
 not as good as the original
REAL:
<SOS> old and stale <EOS> <PAD> <PAD> <PAD> <PAD> <PAD>

GENERATED:
 the best ! ! ! ! ! !
REAL:
<SOS> hates it ! <EOS> <PAD> <PAD> <PAD> <PAD> <PAD>

GENERATED:
 great coffee
REAL:
<SOS> makes a good cup of coffee <EOS> <PAD> <PAD>
```

(a) Examples of Generated sentences and their respective Real labels in the test data set with Transformer-based model

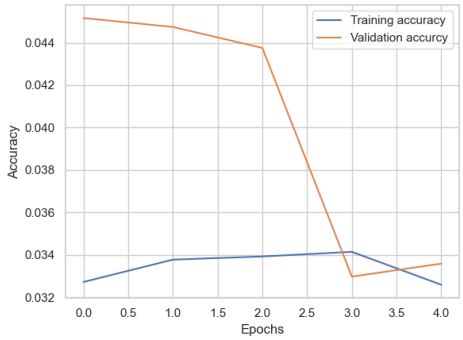
```
GENERATED
<SOS> best <EOS>
REAL
<SOS> delicious ! <EOS>

GENERATED
<SOS> greate <EOS> <EOS>
REAL
<SOS> great flavor no bite ! <EOS>

GENERATED
<SOS> my best <EOS>
REAL
<SOS> excellent choice ! <EOS>

GENERATED
<SOS> greate <EOS>
REAL
<SOS> my favorite morning cup <EOS>

GENERATED
<SOS> greate <EOS>
REAL
<SOS> oh my ! beyond awesome ! ! <EOS>
```

(b) Examples of Generated sentences and their respective Real labels in the test data set with LSTM-based model

Figure 2: Examples of generated summarization



(a) Cross entropy loss for transformer with the bigger architecture



(b) Accuracy for transformer with the bigger architecture

Figure 3: Transformer Training plots

4.1 Transformer

The transformer was trained and tested on an Nvidia 2080 TI GPU, which we had access to for only a few days. We lowered the max length of the labels to 10 in order to not run out of memory.

Some of the results after training look meaningful whereas. the majority of the generated texts say positive words such as 'great' or 'best'. Often both the training and validation loss and accuracy oscillated or even if the loss went down the accuracy still kept at around 2-3 %. And the generation only generated <EOS> tokens. The accuracy should be taken with a grain of salt since it only compares matches word by word, but not the whole sentence.

The bigger architecture (2) gave some reasonable text in the generation, as seen in figure 2a, even though the quantitative results were poor, as seen in figure 3. However, when we trained for longer than 5 epochs we went back to generating just <EOS>. So there was still instability in the training.

For this the learning rate was 0.0001 after doing a lr search. Training with annealed and cyclical lr and warmup did not give any different results.

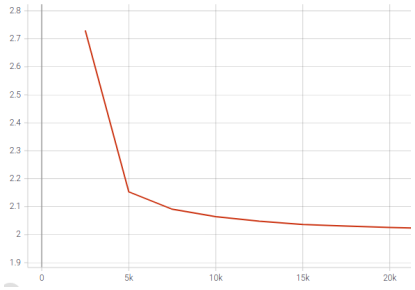
The generated output (figure 2a) overfitted as predicted to a more positive sentiment. For example, "matter of taste" becomes "the best", and "waste of money" becomes "not bad, but not the best". Secondly, it seems to identify the product as a key element n some occasions, as observed in "great green tea" becoming "the best tea ever", and "makes a good cup of coffee" becoming "great coffee". It also seems the model has learned the point of a summarization being short since it does not generate words until the maximum.

4.2 LSTM model

The LSTM-based model was trained and evaluated on a nVidia Tesla T4 with 16GB of RAM. We also had access to it for three days and costed us $\$50$. We had to repeat the experiments several times to figure out a good set of hyperparameters. We trained our models for 10 epochs, each epoch took approximately 1.4 hours.

Figure 4 shows the training and validation losses for our LSTM-based model. Though the losses look good, we found that they are not good indicator of the actual performance of the model. Our LSTM variants were able to generate some meaningful summarizations but the majority of the generated texts were related to the positive words which are dominant in the training dataset. Figure 2b shows some reasonable cases (from different test batches) where our model generates summaries close to the real labels.

During the training, we could see that the model started by learning to generate the <SOS> followed by <EOS> for the first couples of batches. Then it became able to generate sentences of length 2 to 3 words. However, from the half of the first epoch onward, the model seems to just stuck at generating `<SOS> great <EOS>`. We conjecture that again, since "great" is the most dominant words in the dataset, our model seems to have been stuck with it as the model's representation capability was not good enough to catch unique features from the other words.



(a) Train loss measure after each epoch

(b) Validation losses over 20k update steps. During training, we measure the performance of the model after every 1000 update steps and record the validation loss.

Figure 4: Training and validation losses for LSTM model

5 Conclusion

Through this project, we have observed multiple challenges working with LSTM and Transformer since they take a longer time to train than other networks we have previously worked with. Because of the time and resource issues a more advanced comparison could not be made. But we think that the qualitative comparison was sufficient as a baseline, which gives similar results for the transformer and LSTM. An overall observation made was that the transformer model was more efficient in the utilization of computing resources, assumable due to its parallelization capabilities. The biggest improvement to be made here is to balance the dataset instead. The chosen topic also turned out to be more difficult than expected, and our team should have probably just focused on one architecture in particular. Many hours were put at intensive code analysis and reconfiguration, with behaviour hard to interpret. Nevertheless, it was a highly valuable research experience, from which everyone involved learned a lot about NLP research and implementation aspects. After encountering all the problems described above, we are even prouder that we were able to generate some non trivial results.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[2] J. Pennington, R. Socher, and C. Manning, "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, Oct. 2014, pp. 1532–1543. [Online]. Available: https://www.aclweb.org/anthology/D14-1162

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[5] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spaCy: Industrial-strength Natural Language Processing in Python," 2020. [Online]. Available: https://doi.org/10.5281/zenodo.1212303

[6] "Stanford pre-trained glove."

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[9] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://www.aclweb.org/anthology/W04-1013

[10] C. M. Lee, J. Liu, and W. Peng, "Applying cyclical learning rate to neural machine translation," 2020.

[11] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, H. Zhang, Y. Lan, L. Wang, and T.-Y. Liu, "On layer normalization in the transformer architecture," 2020. [Online]. Available: https://openreview.net/forum?id=B1x8anVFPr

6 Appendix

6.1 Transformer Training and Generation

6.2 Transformer attention

The mathematical process of scaled dot-product attention is described by formula 1 and visualized in 8b where masking methods, like padding and look-ahead masks, can optionally be used to deal with the ambiguous input length.

$$ScaledDot - productattention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

Variable h in figure 8a refers to the number of heads, which has to correspond to the input dimensionality, also referred to as depth. Each head gets a set of matrices Q, K, V where Q and K are used to calculate the scaling of the values in V. The concept behind Q(query) K(key) and V(value) originates from retrieval systems, where a query, what a user send in the system, is compared against keys which link to the resources stored in the system, called values. As the key word self-attention indicates the three matrices root from the same source. The resulting h-vectors then can be processed in parallel and finally concatenated again to the original dimensionality.

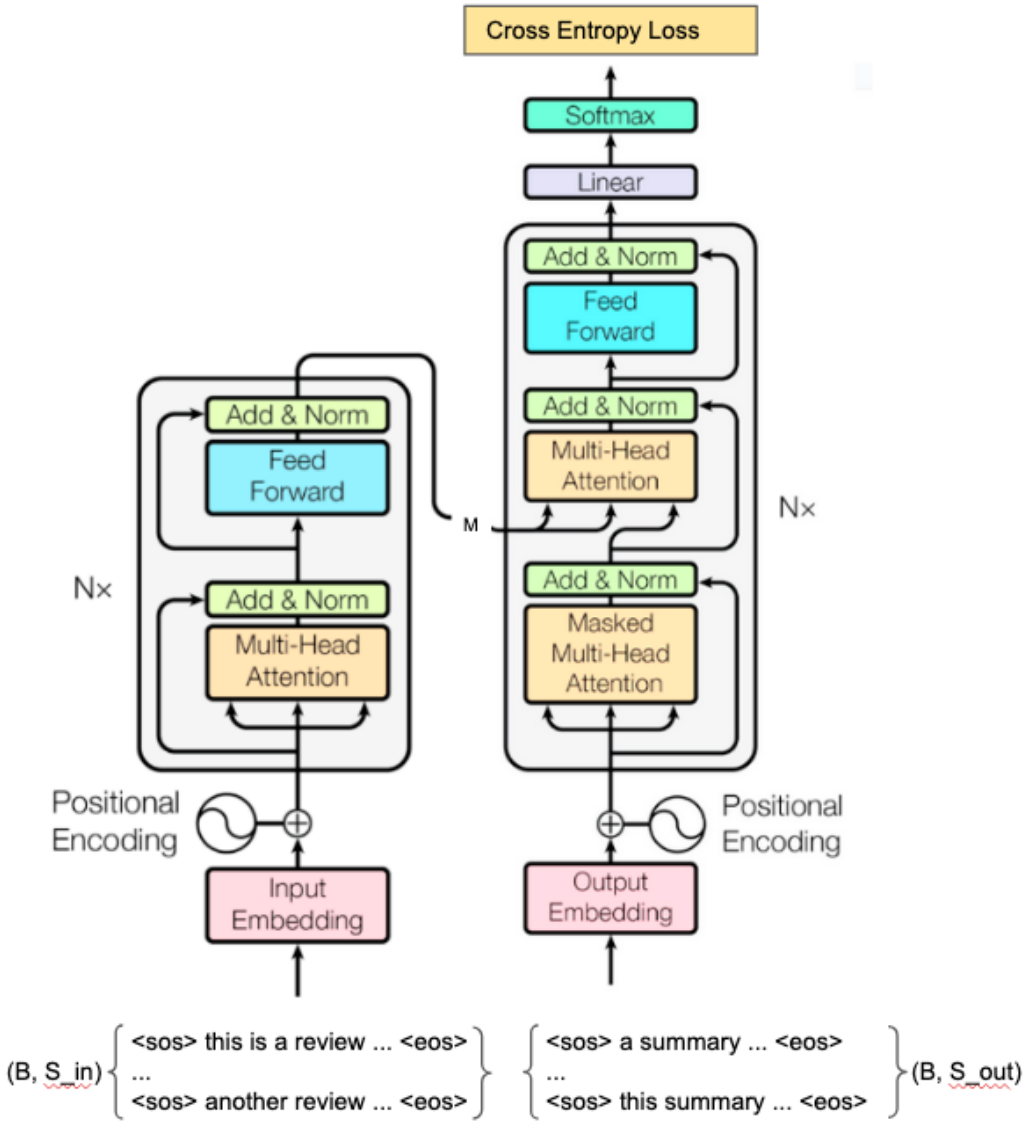


Figure 5: Example of training iteration in the Transformer. S_{in} is the max length of input sequences, and S_{out} the max length of label/target sequences. B is the batch size

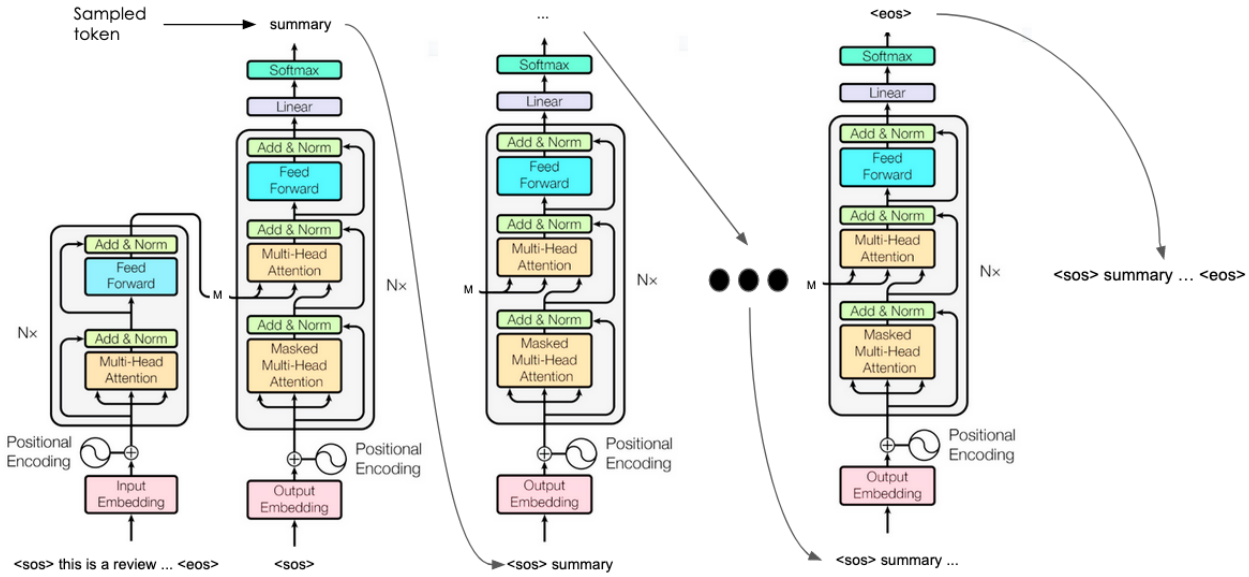


Figure 6: Example of generating text auto-regressively in the transformer.

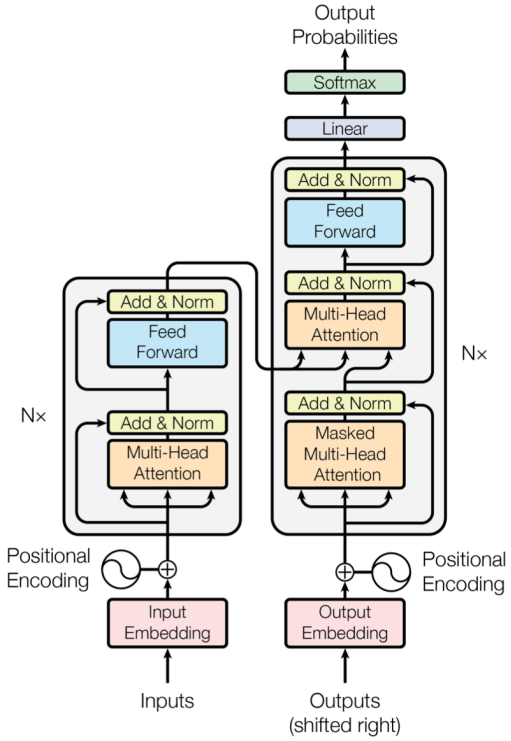
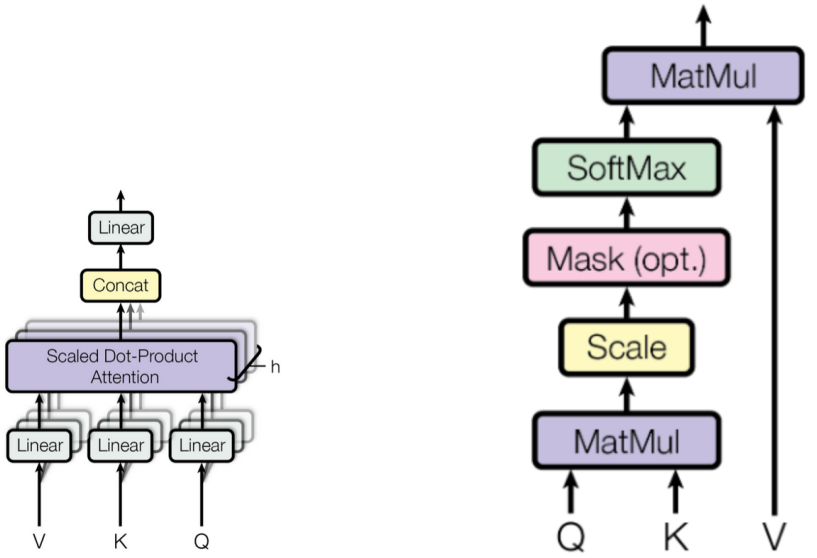


Figure 7: Transformer architecture



(a) Multi-head attention (b) Scaled Dot-product attention

Figure 8: Transformer attention mechanism

6.2.1 Self-Attention in transformer models

The paper "Attention is all you need"**??** describes the use of attention mechanisms in transformers in three different ways, specifically multi-head attention. The queries, keys and values in the self-attention layer of the encoder component all come from the same source, the previous layer of the encoder. This way each position of an encoder layer can attend to each position in the predeceasing layer.
In the decoder component self-attention works similarly, allowing to attention between all decoder positions, up to that specific position. This restriction is implemented to prevent leftward information flow, meaning preventing the model from peeking in the "future". If this would not be done the models auto-regressive property would be compromised. Implemented is this by a masking mechanism which sets the all illegal connections, for example padding tokens that should not be considered, to $-inf$ in the softmax input, effectively scaling them away.
Finally the "encoder-decoder attention" layers, which sits on top of the previous self attention layers in the decoder component, and combining the two components, receive the queries from the previous decoder layer. The keys and values are supplied by the encoder as an abstract memory. This way every position in the decoder is able to attend over all positions in the input sequence.
Self-attention is the mechanism which allows the model to focus on specific aspects of the inputs, which are relevant for the problem domain. In the summarization example at hand, it might identify the central components of context like a product and the notion of the described associations, if it was positive or negative seen.

6.3 Transformer Accuracy

The accuracy measurement ACC we used was an average of the number of matching words between generated output and labels for a batch. This accuracy was then also averaged out per epoch. For one batch we did the following:

1. Compute a $(batch, sentence\_length)$ shaped matrix E that contains for each word in each sentence a 1 if the words match, otherwise 0. Here you compare the words by the target output, which will be a sentence built by choosing the most probable words, and the target label, i.e the ground truth label.
2. With the padding mask P, with shape $(batch, sentence\_length)$, mask away the comparisons between <PAD> tokens by computing $E_{pad} = E * P$

3. Sum each row in E_{pad} to get the number of matches M. Also compute the sum of each non-padded length of every sentence in the batch L. These lengths can be different for every sentence in the batch.
4. Compute the accuracy $ACC = M/L$