Attention Is All You Need Implementation

Mark Van Genderen*
Department of Electrical
and Computer Engineering
The University of British Columbia
Vancouver, BC V6T 1Z4
markavan@student.ubc.ca

Abstract

Transformers are a type of deep-learning model that has gained notoriety for often being the best model for language processing and computer vision tasks. This paper presents an implementation of the original transformer model from the 2017 paper Attention Is All You Need [1] with an emphasis on creating easily digestible code and explanations. With readability being prioritized over efficiency, the corresponding Jupyter Notebook of my model provides a new perspective on how transformer model code can be written and contributes to the expanding transformer learning community. On the Multi30k [7] English-German dataset the model received a BLEU score of 36.56 after being trained on Google Colab with Nvidia's T4 GPU for under 2 hours–making my test results easy to reproduce for readers. A transformer model diagram is provided with more detail than any diagram readily obtainable on the web, and attention graphs are employed to show the final results of self-attention on different sentences.

1 Introduction

At the time of writing, Attention Is All You Need [1] is listed on Google Scholar as being cited by 71465 sources, and is largely recognized as one of, if not the most influential deep learning paper of the decade. Many transformers have been developed since with better performance (such as GTP-2, GTP-3, and BERT), with some transformers becoming household names as ChatGPT grows in popularity. Given this paper's relevance and the large learning community surrounding transformers, I considered it an ideal subject for independent study.

The objective of this project is to improve my personal understanding of transformers and make my own contribution to this learning community. I am presenting an implementation of the transformer model described in Attention Is All You Need [1] with the exception of using learned positional encoding instead of proposed static sin/cos positional encoding.

The introduction of the transformer was significant due to its superior performance and stark difference form previous models designed for the same tasks. Unlike previous language processing and computer vision models which relied on recurrent or convolutional neural networks [1], the transformer learns mostly from self-attention.

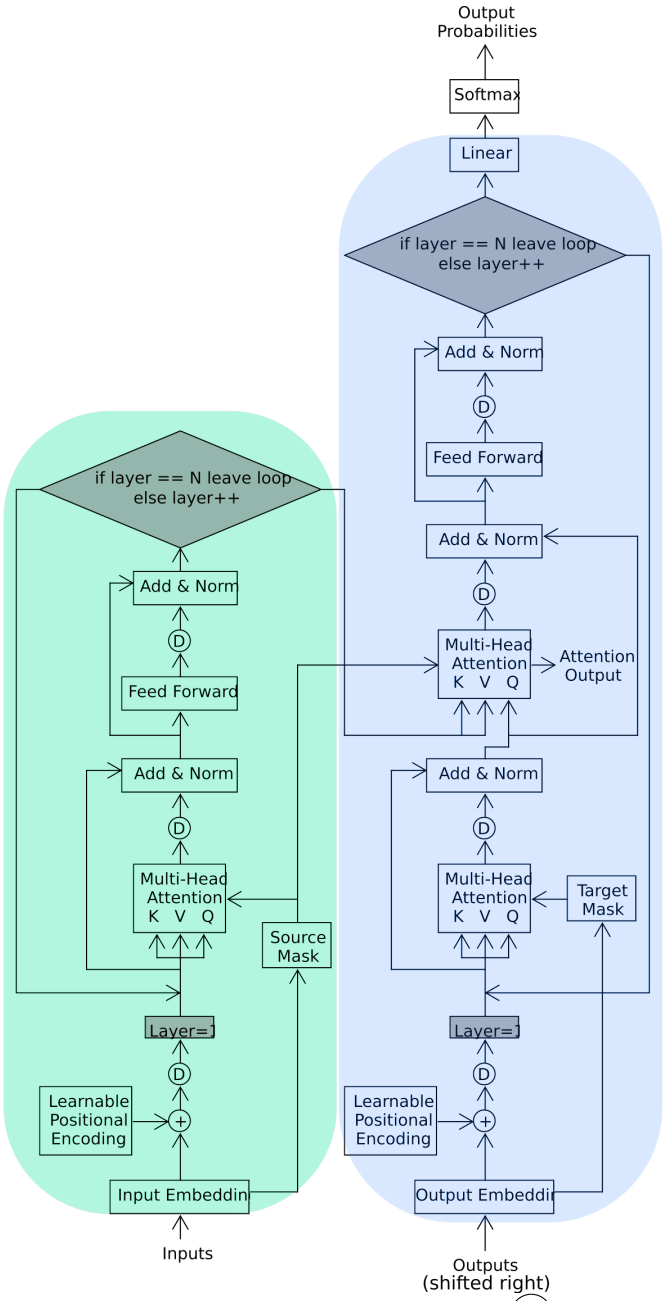
A naive explanation of self-attention is that it allows inputs to 'pay attention' to each other, learn how much they should pay attention to who, then provide an output. In our use case of translating from German to English, self attention 'pays attention' to the know German words and already translated English words to predict the next English word. In essence, attention is used to learn patterns between words and determine the next most probable word, given how important or how much attention is being paid to the other words.

*connect at https://www.linkedin.com/in/mark-v/

Preprint. Under review.

2 Model

Throughout this section a description will be provided for each of the elements in the model overview illustrate in figure 1. It is worth noting that I initially implemented the positional encoding and model configuration python classes of my codebase in a homework assignment [10], and that while little of this section's content discusses tensor dimensions, they are thoroughly commented in the relevant code.



green -> encoder, blue -> decoder, grey -> logic, (D) -> Dropout:

Figure 1: Tranfromer Model

2.1 Layer Normalization

In 2016 layer normalization [2] was introduced as an alternative to batch normalization [3] with superior performance at small batch sizes. The reasoning behind this is fairly intuitive, and can be recognized at a glance from the comparison of the batch normalization (1) and layer normalization (2) formulas below:

$$\mu_b = \frac{1}{B}\sum_{i=1}^{B} x_i \quad \sigma_b^2 = \frac{1}{B}\sum_{i=1}^{B}(x_i - \mu_b)^2. \quad y_i = \gamma\frac{x_i - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} + \beta, \tag{1}$$

$$\mu_l = \frac{1}{H}\sum_{i=1}^{H} x_i \quad \sigma_l^2 = \frac{1}{H}\sum_{i=1}^{H}(x_i - \mu_l)^2. \quad y_i = \gamma\frac{x_i - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}} + \beta, \tag{2}$$

Note: B denotes the number of batches, and H denotes the number of hidden units/features in a layer

In layer normalization, the input is normalized across the hidden units [2] of a batch, independent of other batches. Since, unlike batch normalization, layer normalization has independent calculations for different batches, it easy to see that its calculations will not be as effected by batch size.

2.2 Embedding

For our implementation we used 'nn.Embedding' [4] to create a learned embedding which is much more efficient and takes up less memory space than other methods like one-hot vector embedding. The embedded values are then scaled by $\sqrt{d_{model}}$ [1] which makes the model easier to train.

2.3 Positional Encoding

Positional encoding does exactly as the name suggests, it encodes the position of the input words. This is necessary, since unlikely with many recurrence and convolution models, ours does not provide any other indication of the order of the embedded words. The paper recommends adding a fixed sin and cos function, but in this re-implementation, a learnable normal weighted matrix is used with the PyTorch function 'n.init.normal'. For either method the purpose remains the same, identical words in different positions should have different values, and word order should be recognized.

2.4 Source and Target Mask

The source mask is added to prevent Multi-head Attention from wasting its time paying attention to 'pad' tokens [9]. Adding a source mask does not affect the result of the transformer but it will save computation time. Using the source mask, all padded values are set to -inf which prevents the attention mechanism from being affected by their values during softmax. There is an example of the use of a source mask in the provided Jupiter Notebook.

The target mask is used so that the decoder does not see the tokens for future targeted words. This is important since we are training the decoder to predict the next word, and if we always provided the decoder with what it needs to predict, then it will not learn. A good analogy for this is how students may not learn much from a practice quiz if they look over the solution before and during the quiz. This mask removes the values from the upper left-hand corner of the $(QK^T)/(\sqrt{d_k})$ multiplication from consideration by setting their value to -inf directly before entering a softmax function which will likely output these values as zero.

2.5 Dropout

Dropout is added to prevent overfitting to our training data. The use of dropout in our model is shown in figure 1, equation 3, and equation 6. The placement of dropout is as recommended in section 5.4 of Attention Is All You Need [1].

2.6 Multi-Head Attention

As suggested by the title of this paper, multi-Head attention and self-attention are the most substantial innovations introduced in Attention Is All You Need [1].

$$Attention(Q, K, V) = Dropout(Softmax\big(Mask(\frac{QK^T}{\sqrt{d_k}})\big))V \tag{3}$$

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{4}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{5}$$

Q, K, and V represent query, key, and value respectively. A major advantage of this approach is that each head can be computed in parallel and attention can be paid to all past words, in lieu of the short-term memory employed in recurrent language models.

2.7 Feed Forwards

$$FFN(x) = max(0, Dropout(xW_1 + b_1))W_2 + b_2 \tag{6}$$

The feed-forward layer can be described by the equation above and adds learnable parameters to our model.

2.8 Encoder

At a high level, the encoder's function is to make the input sequence to the decoder with some learnable parameters. A diagram of this section of the model can be found in green on figure 1.

2.9 Decoder

At a high level, the decoder's function is to receive an encoded input sequence, compare it to the target sequence, and output the likelihoods of any next word. A diagram of this section of the model can be found in blue on figure 1.

2.10 Transformer

The transformer in its entirety is composed of the encoder, decoder, and some loss function to make sense of the probabilities and train our model. The 'Inputs' in figure 1 refers to a tokenized version of the input string. Tokenization refers to the process of converting text into an array of words and punctuations, where the elements of the array maintain the original ordering of the text. A '<sos>' token is appended to the start, and a '<eos>' token appears at the end of the array to indicate the start and end of the text. The 'Outputs (shifted right)' in figure 1 is the tokenized version of the output string shifted right once–meaning that the '<sos>' token is removed.

3 Experiments

Due to time constraints, I did not write my own training code. I wrote the model section of the codebase and incrementally compared my result to those of other tested transformer examples [5,6] to help with debugging. The training and setup section of my codebase was written by Ben Trevett [5] and attribution has been provided to him on the codebase as his MIT license requires. The training uses an Adam optimizer with a static learning rate and PyTorch CrossEntropyLoss [4] function.

3.1 Descriptions of datasets

My model was trained on the Multi30k dataset [7]. The original paper used the WMT 2014 English-German dataset [8]. The Multi30k provided the following:

train
(en) 29000 sentences, 377534 words, 13.0 words/sent
(de) 29000 sentences, 360706 words, 12.4 words/sent
val
(en) 1014 sentences, 13308 words, 13.1 words/sent
(de) 1014 sentences, 12828 words, 12.7 words/sent
test_2016_flickr
(en) 1000 sentences, 12968 words, 13.0 words/sent
(de) 1000 sentences, 12103 words, 12.1 words/sent

Multi30k used a combination of factors to come up with their data. They had professional English-German translators convert ~30,000 English descriptions to German and ~150,000 crowd-sourced individuals describe images in their native language. The WMT 2014 dataset is many magnitudes larger and is composed of approximately 4 million sentences, whose English-to-German translations were found from a large variety of sources.

Given my limited computer hardware resources and time constraints, using the WMT 2014 to train my model was not feasible.

3.2 Comparison

My model received a BLEU score of 36.56, a model utilising the same dataset, initialisation and training [5] received BLEU score of 36.52, and the Attention I All You Need paper [1] received a BLEU score of 28.4.

The BLUE scores indicate that my model performed better than the initial transformer model [1] I am trying to reproduce, however, since my model was trained and tested on a far smaller dataset, it is likely that when validated against the same set of data, the original paper's transformer has superior performance.

3.3 Analysis of experimental results

The model seems to work well as indicated from the BLUE score and the following attention graphs which were provided inputs from the test data.

source tokens = ['eine', 'mutter', 'und', 'ihr', 'kleiner', 'sohn', 'genießen', 'einen', 'schönen', 'tag', 'im', 'freien', '.']
target tokens = ['a', 'mother', 'and', 'her', 'young', 'song', 'enjoying', 'a', 'beautiful', 'day', 'outside', '.']
predicted target tokens = ['a', 'mother', 'and', 'son', 'enjoying', 'a', 'beautiful', 'day', 'outside', '.', '<eos>']

Lighter colors correlate to higher attention:

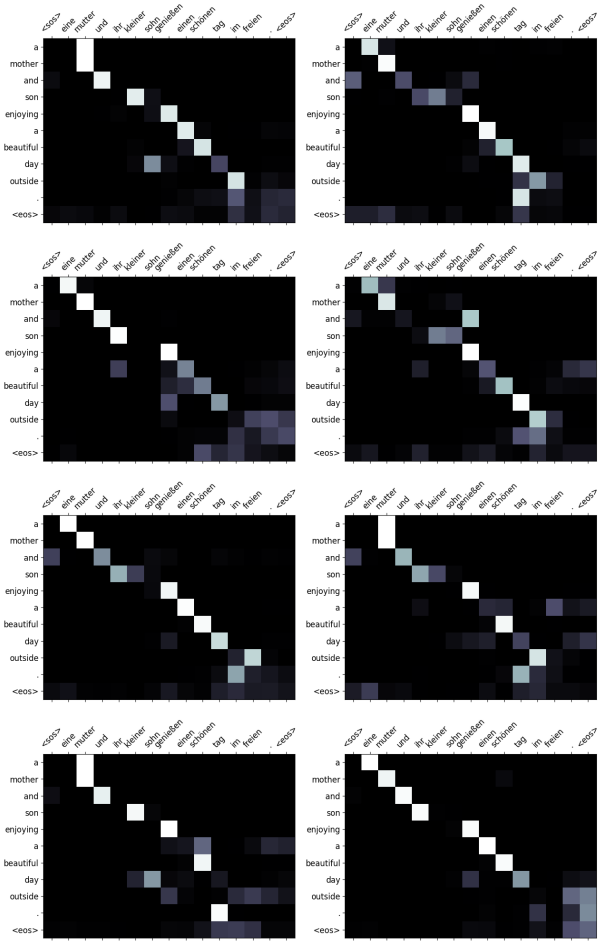


Figure 2: Attention at each head from second Multi-Head Attentions block in decoder

We can clearly see a negatively sloped line connecting the input German words across the x-axis to the outputted English translation of the German words indicated across the y-axis.

4 Conclusion & Future Work

My implementation of the transformer model from Attention Is All You Need [1] was shown to function similarly to a model trained on the same dataset but would likely perform worse than the original transformer [1] when provided the same tests since they used a dataset which is several magnitudes larger.

In the future, I would like to write my own training code and experiment with adjusting transformer parameters and the optimizer. I would also like to train a transformer model on a dataset with millions instead of thousands of sentences and find a way to overcome the corresponding hardware challenges of such an undertaking.

5 References

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[2] Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv preprint arXiv:1607.06450.

[3] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). pmlr.

[4] PYTORCH documentation¶. PyTorch documentation - PyTorch 2.0 documentation. (n.d.). Retrieved April 20, 2023, from https://pytorch.org/docs/stable/index.html

[5] Trevett, B. (2021, March 12). Pytorch-seq2seq/6 - attention is all you need.ipynb at master · Bentrevett/Pytorch-seq2seq. GitHub. Retrieved April 20, 2023, from https://github.com/bentrevett/pytorch-seq2seq/blob/master/6

[6] Rush, S. (n.d.). The annotated Transformer - Harvard University. Retrieved April 20, 2023, from http://nlp.seas.harvard.edu/annotated-transformer/

[7] Elliott, D., Frank, S., Sima'an, K., & Specia, L. (2016). Multi30k: Multilingual english-german image descriptions. arXiv preprint arXiv:1605.00459.

[8] Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Leveling, J., ... & Tamchyna, A. (2014, June). Findings of the 2014 workshop on statistical machine translation. In Proceedings of the ninth workshop on statistical machine translation (pp. 12-58).

[9] Jake Tae. (2021, January 20). Attention is all you need. Jake Tae. Retrieved April 20, 2023, from https://jaketae.github.io/study/transformer/#self-attention

[10] Mahdavi, S., Yan, Q., & Liao, R (2023). CPEN400D: Deep Learning Programming Assignment 2[Class Assignment]. The University of British Columbia, CPEN400D.

A Appendix

A.1 Training Example

source tokens = [’eine’, ’frau’, ’mit’, ’einer’, ’großen’, ’geldbörse’, ’geht’, ’an’, ’einem’, ’tor’, ’vorbei’, ’.’]
target tokens = [’a’, ’woman’, ’with’, ’a’, ’large’, ’purse’, ’is’, ’walking’, ’by’, ’a’, ’gate’, ’.’]
predicted target tokens = [’a’, ’woman’, ’with’, ’a’, ’large’, ’purse’, ’walks’, ’past’, ’a’, ’gate’, ’.’, ’<eos>’]

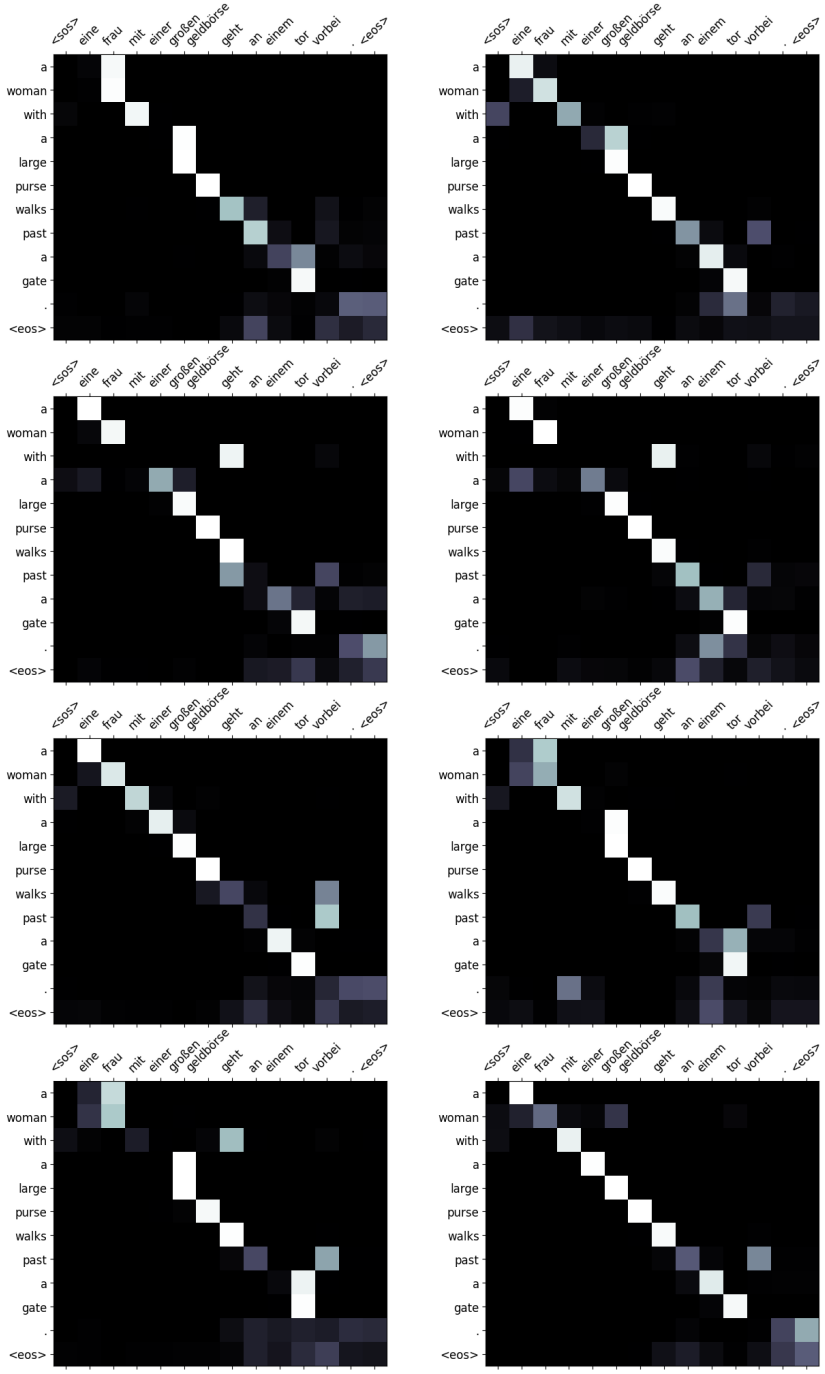


Figure 3: Attention at each head from second Multi-Head Attentions block in decoder

A.2 Validation Example

source tokens = ['ein', 'brauner', 'hund', 'rennt', 'dem', 'schwarzen', 'hund', 'hinterher', '.']
target tokens = ['a', 'brown', 'dog', 'is', 'running', 'after', 'the', 'black', 'dog', '.']
predicted target tokens = ['a', 'brown', 'dog', 'running', 'after', 'the', 'black', 'dog', '.', '<eos>']

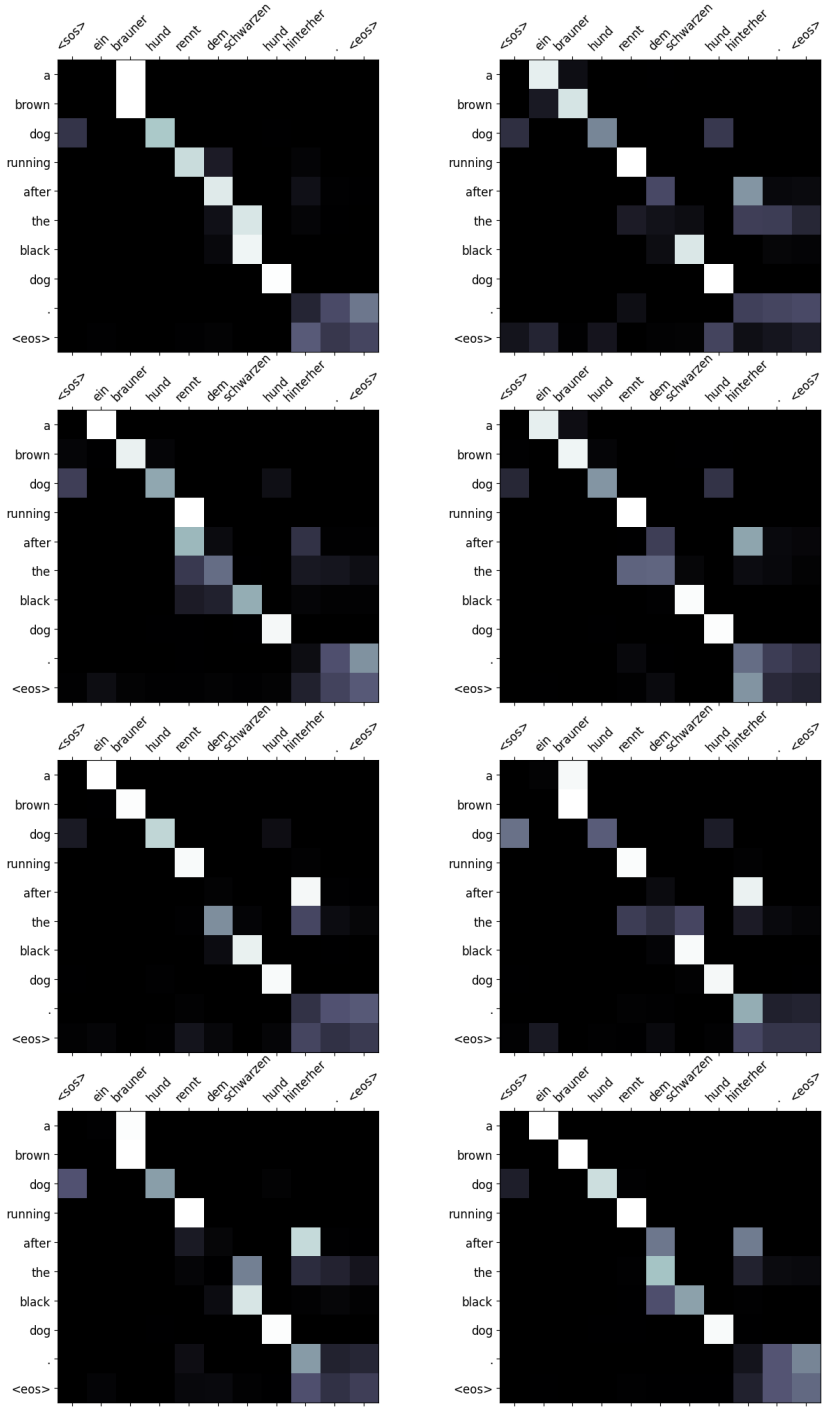


Figure 4: Attention at each head from second Multi-Head Attentions block in decoder

A.3 Testing Example

source tokens = [’eine’, ’mutter’, ’und’, ’ihr’, ’kleiner’, ’sohn’, ’genießen’, ’einen’, ’schönen’, ’tag’, ’im’, ’freien’, ’.’]
target tokens = [’a’, ’mother’, ’and’, ’her’, ’young’, ’song’, ’enjoying’, ’a’, ’beautiful’, ’day’, ’outside’, ’.’]

predicted target tokens = [’a’, ’mother’, ’and’, ’son’, ’enjoying’, ’a’, ’beautiful’, ’day’, ’outside’, ’.’, ’<eos>’]

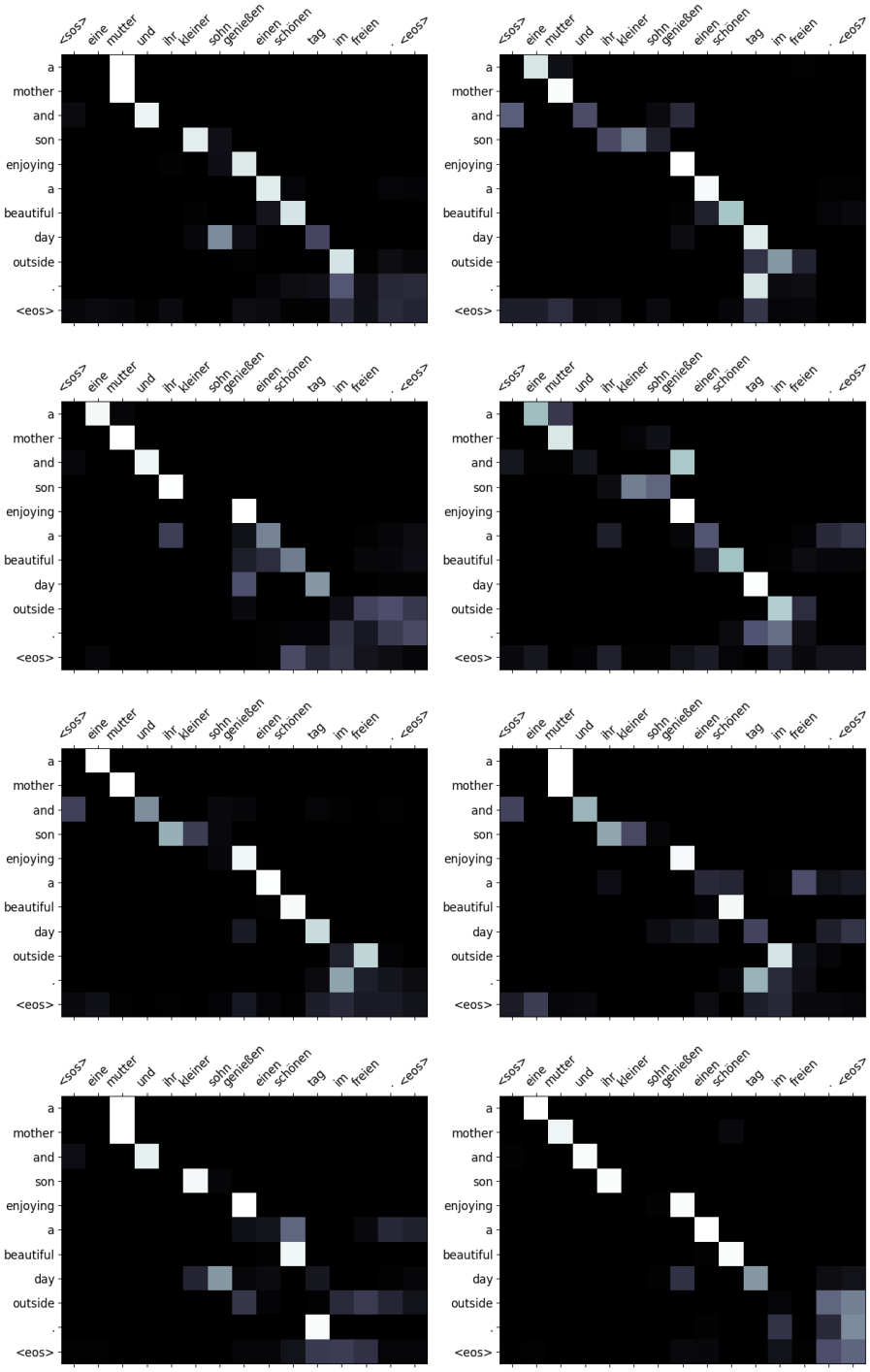


Figure 5: Attention at each head from second Multi-Head Attentions block in decoder