UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Faculty for Computer Science, Electrical Engineering and Mathematics
Department of Computer Science

Seminar Report

Submitted in Partial Fullfilment of the Requirements for the Degree of

Master of Computer Science

Attention is all you need

by
Samrat Dutta

Thesis Supervisor:
Dr. Mohamed Sherif

Paderborn, February 5, 2021

Abstract

Traditionally sequence to sequence problems like neural machine translations have been solved using recurrent neural networks (RNN) and more recently with a combination of RNN with attention mechanisms. But these methods are bottlenecked by the need for sequential computations. This paper proposes a new model (Transformer) for sequential transduction which is parallelizable and solely relies on attention mechanisms to achieve better results and requiring significantly less time to train.

Contents

1 Introduction

Sequence transduction problems like language modelling or machine translation has long been achieved using implementations of Recurrent Neural Networks (RNN) like Long Short Term Memory (LSTM) [11]. These are encoder-decoder based sequential systems which have recently been improved by the use of attention mechanisms in conjunction [3]. Computational methodology has significantly improved as well, improving the efficiency of such models. But these systems are bottlenecked by their inherent sequential nature.
The paper Attention Is All You Need by Vaswani et al. [12] introduces a new model called a transformer which is completely based on attention mechanisms and thus can achieve more parallelization, improving efficiency as well as improving the quality of translations.

1.1 Background

In natural language processing, after tokenization of a sentence we usually proceed with converting each token into a vector. This is in general done using an embedding algorithm. These vectors act as inputs to our sequence transduction models.

RNN Recurrent Neural Networks implementations consist of an encoder and a decoder. The encoder accepts each input vector at each timestep and generates a hidden state. The hidden state h_t is computed at timestep t using the input vector at timestep t and the hidden state from the previous timestep h_{t-1}. When the end of the sentence is encountered, the final hidden step, is called the context vector and is consumed by the decoder which then produces output translations iteratively and creates new hidden steps for the next round. This architecture is described in detail, in a paper by Sutskever et al. [11]

Attention The dependencies between words are extremely important for translations and is preserved in the context vector. This means that all the necessary information needs to be compressed into the fixed length context vector by the encoder. This proved to be a bottleneck when dealing with long sentences where the performance of these systems deteriorated rapidly. To counteract this, Bahdanau et al. (2014) [3] and Loung et al. (2015) [8] proposed a technique called Attention. This new mechanism passes all the intermediate hidden steps to the decoder instead of only the final vector. And the decoder scores these hidden states based on how related they are to certain words in the sentence and based on these scores, calculates weighted sum after multiplying the hidden states with their softmaxed scores (Amplifies high scoring states while heavily penalizing low scoring states). [2]

Transformer Attention has enabled us to model dependencies efficiently, especially for long sentences disregarding the distances. But this is generally used

in conjunction with RNNs, which are constrained by their sequential computations. Extended Neural GPU [16], ByteNet [6] and ConvS2S [5] are some models which compute the hidden steps in parallel but relating the signals from arbitrary input/output positions require computations that grow in number with increasing distance. The Transformer model introduced in this paper solves both these problems by reducing the number of operations needed to relate signals to a constant number (number of heads in Multi-Head Attention) and by completely removing the sequential aspect of RNNs by exclusively using attention.
The transformer uses a method called self-attention to relate different positions of the same sequence in order to work out dependencies between each word and different parts of the input sentence. We will take a look into this mechanism in more detail in the following chapter.

2 Methodology

In this section, we briefly talk about attention, particularly about Q, K, V attention - the implementation used by the authors of the paper. Then we talk about the processing of the inputs which would give us enough tools to discuss the architecture of the proposed Transformer model. We finish the discussion by talking about the output and training.

Attention Attention can be thought of as a weighted average pooling where a set of data points absorb information from other data points. This is often achieved as a weighted sum where the weights come from a softmax function, i.e they sum up to 1. Thus the weights form a probability distribution and in some way imply how much a particular data point is "attending" to other data points.[13] These weights are also known as attention scores.

$$value = \sum_{i=1}^{n} w_i k_i, \quad \text{where} \quad \sum w = 1 \quad \text{(softmax)}$$

When the weights are a function of/derived from the same set of inputs, this type of attention is known as self attention.

Q,K,V Attention Attention can be implemented in several ways. The form of attention implemented by the authors is Q, K, V attention which is a scaled dot product attention, where the letters stand for Query, Key, Value. This concept can be easily understood with the analogy of a query in a search engine. Search engines index millions of pages and transform them into metadata that contain keywords and metrics about that page. When a query is received, the query is matched with millions of such metadata instances to form a relevance score for each related page. [1]
In the same way, in Q, K, V attention, each data point is transformed into a query, a key, and a value. The metadata in the case of the search engine can

be thought of as the key. The query for one datapoint is then related to the key of all the other data-points to produce a weight score. This weight score is then scaled down by dividing it by the square root of the dimension of the key and then a softmax function is applied to derive a probability distribution. The scaling is necessary to ensure that the inputs to the softmax don't become too large. The sum of the products of the derived weights with the corresponding values gives us the attention mapping for that item. This is then done for each of the other data points. Assuming the data points are x_i, then attention z_i is calculated as [1]

$$\begin{aligned} q_i &= W_q x_i \\ k_i &= W_k x_i \\ v_i &= W_v x_i \\ w'_{ij} &= \frac{q_i^\intercal k_j}{\sqrt{d_k}} \\ w_{ij} &= softmax(w'_{ij}) \\ z_{ij} &= \sum_j w_{ij} v_j \end{aligned}$$

This is usually done using Matrix multiplication as those algorithms are highly efficient these days. The query, key, and value vectors are combined into the Q, K, and V matrices, and then the attention is calculated together as :

$$Attention(Q, K, V) = softmax(\frac{Q^\intercal K}{\sqrt{d_k}})$$

Why Self Attention ? Long distance dependencies have been difficult to learn for language models in the transduction tasks because of the length of the path that the forward and backward signals have to travel.
A recurrent layer requires $O(n)$ sequential operations to connect all positions but a self attention layer can do it in a constant number of operations. Often the length of the sentence is smaller than the dimension of the model. In all those cases, self attention works faster.

2.1 Model Architecture

Like most sequence transduction models, the Transformer is also based on an encoder-decoder architecture. In the model described in the paper, the encoding component is composed of a stack of N=6 encoders and the decoding component is also composed of a stack of 6 decoders. In this section, we discuss in depth the architecture of both the encoder and the decoder and describe its inner workings. But first, we briefly discuss the input and output representations for this model.

Input and Output representations The transformer is a sequence to sequence transduction model. This means that it takes in any sequence of data and outputs a sequence. In the case of machine translation, data can be in the form of a sequence of words (sentence). These words are input in the form of word embeddings, i.e vectors of floats of dimension d_{model}. For this paper, the authors have chosen $d_{model} = 512$ These embeddings can be trained during the training process of the Transformer or can be pre-trained and fixed, in which case they act as a lookup table.

Positional Encoding The biggest advantage of the transformer model is that it is not sequential in nature. Hence we can train the model in parallel by feeding it multiple words in the sentence all at once. But this also means that the model produces the same outputs regardless of the positions of the words in the sentence. Syntax and positioning are important in language-related tasks hence it is important to preserve/embed this information. This is achieved by using positional encoding. Each position of the input statement can be represented by a vector of length d_{model} which is unique for every position, can be calculated deterministically and the distances should be consistent across sentences with different lengths.
The authors use an interleaving sine and cosine function to produce positional encodings in this model.

$$\vec{p}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(w_k t) & \text{if } i = 2k \\ \cos(w_k t) & \text{if } i = 2k+1 \end{cases}$$

where,

$$w_k = \frac{1}{10000^{2k/d}}$$

[7]

Since these position vectors have the same dimension as the input word embeddings (d_{model}), they can simply be added together. This new encoded vector acts as the input to the encoder and decoder stacks.

Encoders The encoders can be broken down into 2 layers, a self-attention layer and a simple, point-wise fully connected feed-forward network. All of the encoders are identical but are weighted differently. The self-attention layer encodes the input values based on their relationship with other words in the input sentence and then passes it on to the feed-forward network which is independently applied to each of these outputs. [12]

Decoders The decoder also consists of the same two layers as the encoder but with an additional multi-head attention layer in between that processes the output of the encoder stack. This helps the decoder to relate to the different parts of the input sentence as well. The attention layer in the decoders differs from that of the encoder layer, as it is blocked from attending to positions that succeed the word at the current position. This masking ensures that the predictions only depend on the outputs that have already been seen. [12]

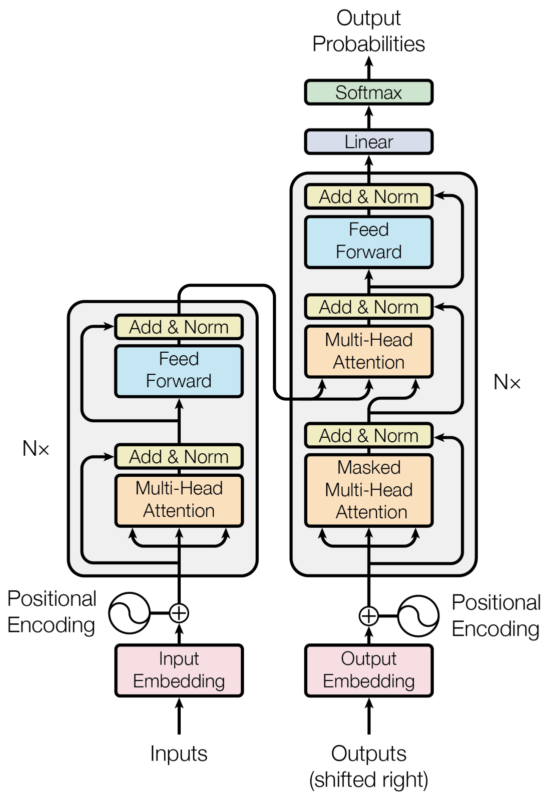
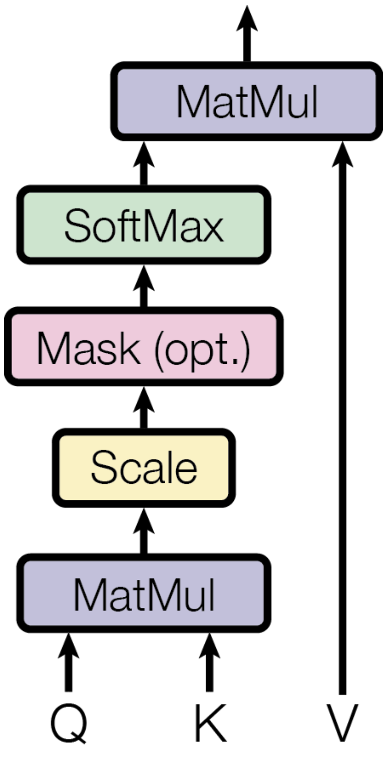


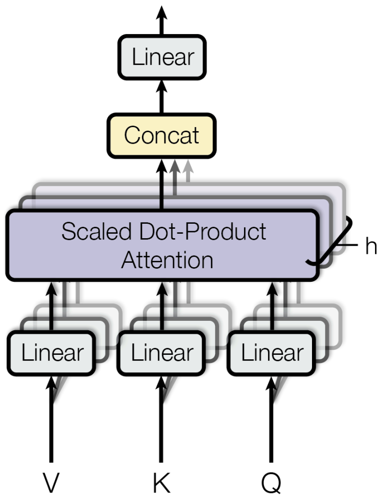
Figure 1: Architecture of a Transformer. [12]

2.1.1 Encoder

Multi Head Attention The position encoded input vectors are first transformed into the query, key, and value vectors and then passed through the first layer of the encoder which is the attention layer. The implementation of attention has been discussed in a previous section. The resulting attention vectors capture relationships between each of the words with other words. Though every other word contributes to the attention vector, any one of the weights might be dominant and drown out other relationships.[1] But there might exist multiple different types of relationships between words. We can think of this as how in any sentence there are different types of relations like verb-noun, noun-adjective, etc. And hence the same word may be related to different words by different relationships.
Performing one attention function is not enough to capture all these different relationships. The authors have calculated attention h times with different learned query, key, and value vectors each of the dimensions d_q, d_k, d_v



(a) Scaled dot product attention



(b) Multihead attention

Each application of the function is called a head and in the paper, the authors have suggested using $h = 8$ heads. The idea is that each of the heads captures a different aspect of the relationships between the words and because of having multiple heads learning multiple different weight matrices independently, the model learns to attend to different representational subspaces at different positions.[12]

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_8)W^O$$
$$\text{where,} \quad head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$
$$\text{where,} \quad W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v} \text{and} W^O \in \mathbb{R}^{hd_v \times d_{model}}$$

As mentioned, in the paper the authors have used the number of heads $h = 8$ as parallel attention heads (hence multi head attention), and $d_k = d_v = d_{model}/h = 64$ Each head produces a d_v dimensional vector value which is concatenated together. It is then projected using the W^O matrix.
The described architecture constitutes one of the multi-head attention layers in one encoder. In the model proposed, there are $n_x = 6$ such encoders stacked one after the other. Hence in all, there are 48 such attention heads that capture a wide variety of relationships between the words.

Feed Forward Network and Residuals The output of the multi-head attention layer is fed into a fully connected point-wise feed-forward neural network applied identically to each position. This layer performs a ReLU non-linearity sandwiched between two layers of liner transformations.[12]

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

The linear layers use different parameters in the different layers. The input and output dimensions are $d_{model} = 512$ while the inner layer has a dimension $d_{ff} = 2048$. Each of these sub-layers, that is, the multi-head attention layer and the feed-forward network is skipped with a residual connection and a layer normalization. After the application of the sublayer function, the gradients in the matrix may need to be stabilized. Hence the inputs to the sublayer is added to the output to distribute the gradient. This has proved to improve the speed and accuracy of training.[9]

$$LayerNorm(x + SubLayer(x))$$

where, $Sublayer(x)$is the function implemented by the sublayer.

The final output of the last encoder is then transformed into a set of Keys and Values and fed into the Encoder-Decoder multi-head attention layer of the decoder.

2.1.2 Decoder

The decoder also consists of the same number of layers as the encoder $N_x = 6$. In contrast to the encoder, the decoder has 3 sublayers. A masked multi-head attention layer at the bottom, followed by the encoder-decoder multi-head attention layer and finally the fully connected point-wise feed-forward network. Each of these sublayers is also skipped using a residual connection followed by a layer normalization because of the same reasons mentioned in the encoder subsection.

Masked Multi-Head Attention The decoder produces output iteratively in each time step using the output from the previous step. This sequential action precludes parallelization. Thus the whole benefit of using a transformer model is to take advantage of parallel processing would be lost. During training, since we already have labeled training data, it is possible to pass the whole translated sentence to the decoder. However, while processing each position, the position can see ahead. This is an undesired behavior. For each increasing timestep, the decoder should only be able to see the outputs of the previous step. This can be solved by using a mask before the softmax layer which turns all the future values to negative infinity.[4]
This method ensures that each position produces the output without seeing the calculated values from the future positions. The mask modification to the multi-head attention layer is thus a crucial first layer in the decoder.

Encoder-Decoder Multi-Head Attention The second layer in the decoder is the Encoder-Decoder attention layer. This layer facilitates relating the output sequences with the input sequences and in training the model in connecting the words from the input language with the words in the output language. The Query inputs in this layer come from the previous decoder layer (masked multi-head attention) while the set of Keys and Values are received from the encoder.[12]
This layer also works like the other multi-head attention layers we have seen with the exception that this is not self attention anymore. The queries from the output attend to the keys and values of the input sequence. After receiving up to the last decoder output, it is used as the query and can now attend to all the keys and values from the encoder layer. This happens iteratively for every output. Eventually, every position in the decoder attends over.all positions of the input sequence.[12]

2.2 Output

The final layer of the decoder produces an output in the form of a vector at each time step. This vector needs to be converted to a word prediction. This is done by first applying a linear layer and converting the vector into a logit of the dimension of the total length of the vocabulary. A softmax function is then used to convert the logit vector into a probability distribution. Each index of this vector corresponds to one unique word in the vocabulary. Hence, the word corresponding to the index with the highest softmax value is then chosen as the output of the model.[1]
A trained model will generate the highest softmax probabilities corresponding to the correct word translation. But an untrained model on the other hand will most probably show a different, distribution. Various error measures might be used to quantify this error. One such method might be to subtract the vector from the one-hot vector for the correct labeled word. This may then be used to train the weights used throughout the model using backpropagation in order to train them.[1]

3 Training

The authors trained the model on the WMT 2014 English-German dataset which contains over 4.5 million translated sentences, with 37000 tokens and for Enlish-French they used WMT 2014 English-French dataset containing 36 million sentences, 32000 tokens. They trained the model in batches containing sentences consisting from 25000 source and target tokens.[12]
The models were trained in one machine consisting of 8 NVIDIA P100 GPUs. The base model was trained for 12 hours for a total of 100000 steps.
Another model, called the big model having bigger hyperparameters described below was trained for 300000 which took 3.5 days.[12]

Optimization The authors used the Adam's optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. And they changed the learning rate by increasing it during the first steps ($warmup\_steps = 4000$) and decreasing it proportinal to the inverse square root of the step [12] according to the formula :

$$lrate = d_{model}^{-0.5}.min(step\_num^{-0.5}, step_num.warmup\_steps^{-1.5})$$

Regularization The authors achieved regularization by applying Residual Dropouts to the output of each sublayer and also during the encoding of the word embeddings with the positional encodings. This is done to avoid overfitting of the date. The rate of dropout used by the authors is 0.1. [12]
Label smoothing is also employed by the authors in order to improve accuracy and the BLEU scores.

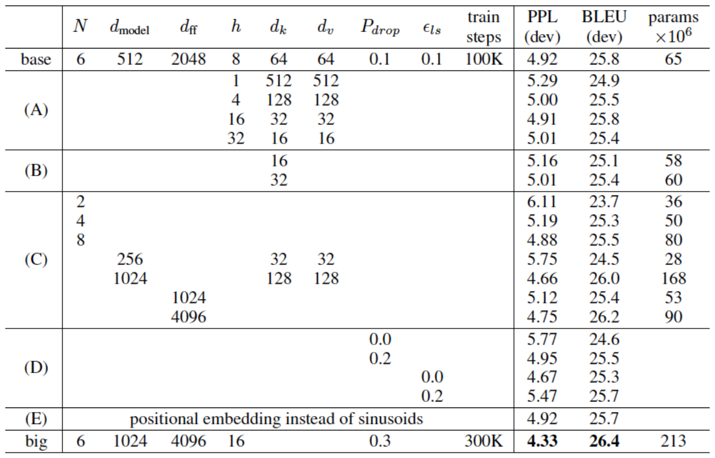
4 Comparison

The big transformer model achieved a state-of-the-art BLEU score of 28.4 in the WMT 2014 English-to-German translation task. This score is over 2.0 points better than the previous best. And in the English-French translation task, the same model achieved a significant BLEU score of 41.0 which is better than every other single model previously tested.[12]
But the significant improvement and benefit can be seen in the training cost. Both the models are far cheaper to train than the other competitive models. This is because of the parallel trainability of the model which reduces the training times substantially.[12]

Model Name	BLEU		Training Cost	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [6]	23.75			
Deep-Att + PosUnk [15]		39.2		1.0×10^{20}
GNMT + RL [14]	24.6	39.92	2.3×10^{19}	1.4×10^{20}
ConvS2S [5]	25.16	40.46	9.6×10^{18}	1.5×10^{20}
MoE [10]	26.03	40.56	2.0×10^{19}	1.2×10^{20}
Deep-Att + PosUnk Ensemble [15]		40.4		8.0×10^{20}
GNMT + RL Ensemble [14]	26.30	41.16	1.8×10^{20}	1.1×10^{21}
ConvS2S Ensemble [5]	26.36	41.29	7.7×10^{19}	1.2×10^{21}
Transformer (base model)	27.3	38.1	3.3×10^{18}	
Transformer (big)	28.4	41.0	2.3×10^{19}	

Table 1: Comparison between Transformers and other competitive models in EN-DE and EN-FR Machine Translation tasks. [12]

Variation of hyperparameters The hyperparameters of the model were varied and the performance was observed on the development set newstest2013 for English-German translations, to understand how the parameters affect the model. (Figure 2)
Row A : On varying the number of attention heads while keeping the total computations constant the authors noticed that increasing the number of heads leads to a decrease in the quality of translations.
Row B : Using a smaller dimension for the key d_k the authors observed a worse performance.
Row C and D : Increasing the number of layers used and the dimensions of the model produces better results thus confirming that bigger models perform better. They also observe that over can be avoided by using residual dropouts.
Row E : Instead of using the sine-cosine position encoding described earlier, the authors used an encoding that is learned in conjunction with the model. This produced similar results to the base model. [12]



	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Figure 2: Comparison of results for various configurations of the hyperparameters. [12]

5 Discussion

This paper presented a revolutionary idea at that time in the field of sequence to sequence transductions which changed the landscape of Natural Language Processing quite a bit. Before the publication of this paper, almost every competitive sequence to sequence model was based on Recurrent Neural Networks or Convoluted Neural Networks. This was the first proposed model that used only Attention mechanisms and achieved similar or better results in machine translation tasks.

Works based on this paper Several important papers were published after the publication of this paper based on the Transformer model. Some of the modern Transformer based architectures like BERT, GPT-2, Transformer-XL are state-of-the-art right now. This itself speaks volumes about the importance of this paper.

Strengths

- The biggest strength of this paper is that it introduces a model that can be trained in parallel. This reduces costs and training time dramatically. This can be seen in Table 1. The training cost reduction is significant when compared to the other models.
- The current performance is limited only by the hardware. The models can be as large as we want, it only increases the size of the matrices for the dot products. As long as the hardware can calculate them in a reasonable time, the model will work well.

- This is a generic model that can work on any set of data. This has the potential to be influential in fields other than Natural Language Processing.
- The transformer models long-distance relationships between words very well and removes the restrictions that even LSTMs have.

Weaknesses

- Though the biggest strength of the model is the ability to train parallel, it is not a priority in many cases. We can let models train for a long time because once it is trained, it performs translations in comparative time.
- The improvements in the quality of translation is not very significant for the base model.

References

[1] Jay Alammar. The illustrated transformer. http://jalammar.github.io/illustrated-transformer/, 2018.

[2] Jay Alammar. Visualizing a neural machine translation model (mechanics of seq2seq models with attention). https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/, 2018.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[4] Rachel Draelos. The transformer: Attention is all you need. https://glassboxmedicine.com/2019/08/15/the-transformer-attention-is-all-you-need/, 2019.

[5] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017.

[6] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aäron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *CoRR*, abs/1610.10099, 2016.

[7] Amirhossein Kazemnejad. Transformer architecture: The positional encoding. https://kazemnejad.com/blog/transformer$_architecture_positional_encoding/$, 2019.

[8] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.

[9] Sabyasachi Sahoo. Residual blocks — building blocks of resnet. https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec, 2018.

[10] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR*, abs/1701.06538, 2017.

[11] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[13] Lilian Weng. Attention? attention! *lilianweng.github.io/lil-log*, 2018.

[14] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[15] Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. Deep recurrent models with fast-forward connections for neural machine translation. *CoRR*, abs/1606.04199, 2016.

[16] Łukasz Kaiser and Samy Bengio. Can active memory replace attention?, 2017.