CSE 589 Paper

Yash Shah, Tanner Thornton

February 27, 2020

Abstract

From the paper "Attention is All You Need," the concept of the transformer architecture was introduced. The transformer architecture is solely based upon the attention mechanisms, dispensing with recurrence and convolutions entirely. The success of transformers in various NLP tasks has led to the development of many variations and improvements on the original design. We seek to construct a review of the high-level implementation and effects of the basic transformer, BERT, and transformer-XL as well as their application in certain problems. Specifically, we will examine sentiment classification and machine translation. Additionally, background information will be included, as some architectures are heavily dependent on previous models, including RNN and LSTM.

Contents

1 Introduction

Transformers are a relatively new architecture based on neural networks that seek to build context in a richer way than previous models. They are based on an attention mechanism, which we will discuss later. First, we will provide some optional background reading on other sequential models that were historically used for natural language processing before discussing the basic transformer in-depth. The goal of this section is to combine information from their introduction with some recent minor developments. Afterward, we will discuss some major developments that have occurred, which produced models distinct enough to be considered their own class. Finally, we discuss some applications of transformers, specifically how the high level functionality of this attention mechanism applies and improves on previous designs.

2 Background

2.1 Recurrent Neural Networks

At a very high level, RNNs differ from other ANNs in that they have a feedback process in the update step. This modifies a "hidden state" dependent on time, in the sense that it depends on the previous execution of the network. There were two types, output feedback based on the output of the activation function $\sigma(v)$ and activation feedback based on the the v passed into σ, but output feedback is much more common because it helps constrain the gradient. The hidden state and the input have independent weights, in that the weights stored are different between them. These weights follow a similar formula, so they will be generalized as w. The application of gradient descent to minimize a cost function $E(t) = \frac{1}{2}e^2(t)$ with instantaneous error $e(t)$ and some learning rate η produces

$$\Delta w_i(t) = \eta \frac{\partial E(t)}{\partial w_i(t)} = \eta e(t) \frac{\partial e(t)}{\partial w_i(t)}.$$

With some simplification, this produces

$$w(t+1) = w(t) + \eta e(t) \frac{\partial e(t)}{\partial w_i(t)},$$

which should be a familiar formula. Essentially, it executes propagation similar to other neural networks with respect to it's previous input.

A common problem with RNNs is the exploding or vanishing gradient problem, where the derivatives as propagation continue become incredibly high and overpower the rest of the model or become incredibly small and the network ceases to learn. Both of these result from the chain rule and the exponential nature of constant multiplication. There are some methods to counteract this such as gradient clipping, a much better solution can be found in LSTM. Additionally, the hidden state is passed

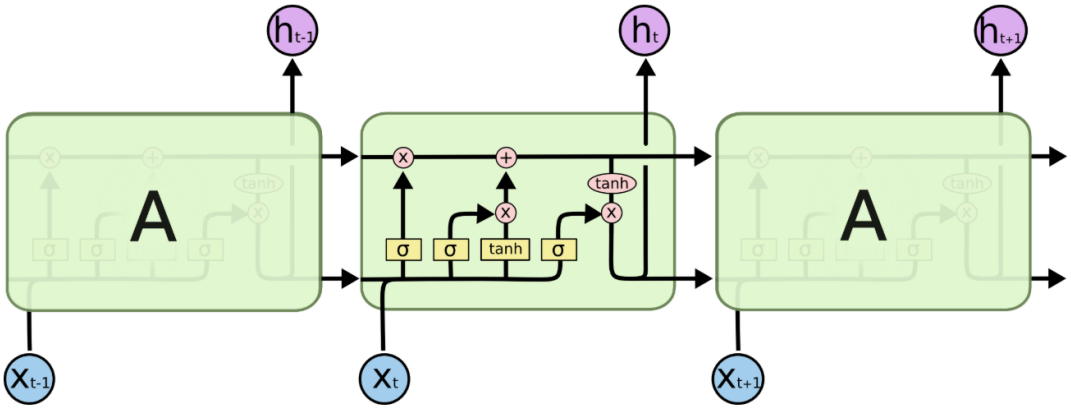

Figure 1: Several LSTM cells. [1]

every time, and can be easily overwritten with information that may not be as relevant as the previous hidden state but is much more recent.

2.2 LSTM

In the course of speaking or writing, one might mention a specific location and an implied language. Depending on what was said in between the location and the language, the distance between the explicit and implicit may be to great for the hidden state to meaningfully track. While the gradient problem is not solved completely, this is exactly the problem LSTM seeks to solve.

The key to LSTMs is the cell state C, the horizontal line running through the previous state to the current state and then to the next state, as seen in Figure 1. The LSTM uses gates comprised of sigmoid based network layers with inputs h_{t-1} of the previous hidden state and the current data input x_t. Each sigmoid gate has individual weights, and the output vectors are pointwise multiplied with another vector to achieve an effect. or the hyperbolic tangent regularized output of C into the next hidden state. The first gate is multiplied with the previous cell state and simply determines how much of the previous cell state should be forgotten. The second gate rescales the hyperbolic tangent transformation of h_{t-1} and x, determining how much of the current input is useful and should be transferred to the cell state. The third gate takes the previous hidden state and current input and applies it to the hyperbolic tangent transformation of the cell state to determine the final form of the hidden state; the cell state does not change at this time. The same process occurs for each cell in an LSTM, but there are some modifications including models where the cell state is passed to each gate, or the Gated Recurrent Unit (GRU), where the input gate is simply the complement of the forget gate. In any case, the fundamental principle is the same: the cell state carries information from previous states a farther distance than what previously possible by separating it from the hidden state, giving it

a longer short-term memory.

3 Basic Transformer

Transformers architecture is introduced in the paper "Attention is All You Need" which makes use of attention mechanisms, dispensing with recurrence and convolutions. We describe this model with five parts: Encoder-Decoder structure, Attention Mechanism, Multi-head Attention, and Positional Embedding.

3.1 Encoder-Decoder Structure:

Most competitive neural sequence transduction models have an encoder-decoder structure. Here transduction means the conversion of input sequences into output sequences. On a higher level, encoder transforms input sequences of symbol representations $(x_1, x_2, ..., x_n)$ to a sequence of continuous representations $z = (z_1, z_2, ..., z_n)$. Given z, the decoder generates output sequence $(y_1, y_2, ..., y_m)$ of symbols one element at a time.
At each time step, the model is auto-regressive - consuming the previously generated symbols as additional input when generating the next - which is the core component in mostly all sequential models.
The transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 2, respectively. As each layer has a residual connections around it of layer normalization, it avoids the vanishing gradient problem in deep networks.
Basically, the transformer model follows the same general pattern as a standard sequence to sequence with attention model as shown in Figure 3.

The figure above shows that input language being encoded word by word and after end of sentence $< s >$, the decoder LSTM will start to produce the translated sentence, one word at a time. It is important to notice that for each output of the decoder, the output of encoder is being used as "context" to an attention module - this means that the decoder output is gated by attention based on the relationship between output word and all the words of input. Notice that the decoder is also auto regressive here i.e. it uses its previous output words to generate the next word.

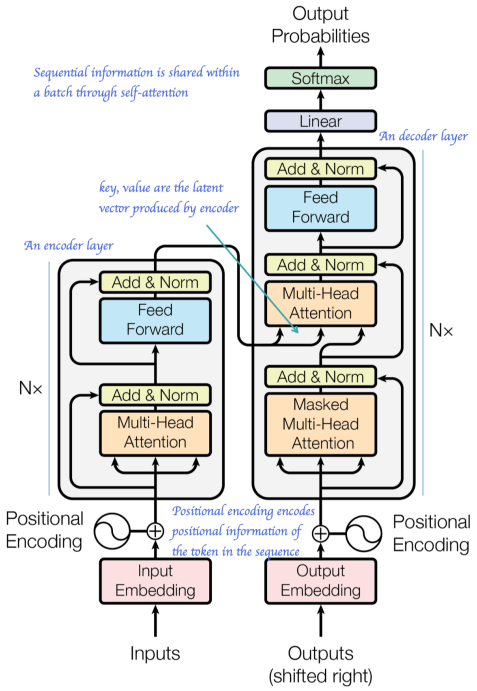

Figure 2: Figure Basic Transformer model architecture

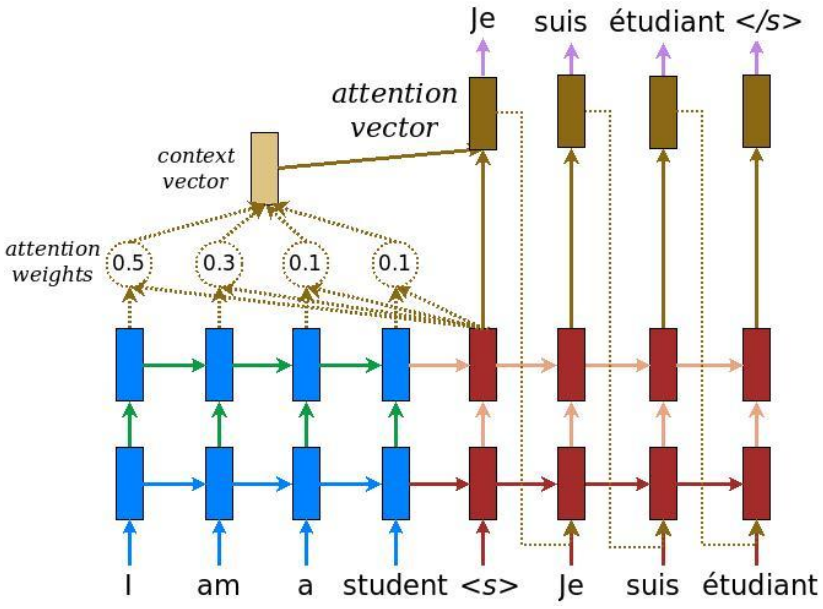

Figure 3: Sequence-to-sequence with attention.

In short, the following formulas are used to compute attention weights (α_{ts}), context vector (c_t) and attention vector (a_t):

$$\alpha_{ts} = \frac{exp(score(h_t, \bar{h}_s))}{\sum_{s'=1}^{S} exp(score(h_t, \bar{h}_{s'}))} \text{[Attention weights]}$$

$$c_t = \sum_s \alpha_{ts} \bar{h}_s \text{ [Context vector]}$$

$$a_t = f(c_t, h_t) = tanh(W_c[c_t; h_t]) \text{ [Attention vector]}$$

$$score(h_t, \bar{h}_s) = h_t^T W \bar{h}_s \text{ [Luong's multiplicative attention]}$$

There are two most commonly used attention functions: additive attention (Bahadanau Attention [2]) and multiplicative attention (Luong Attention [3]). In addition to following multiplicative attention, transformer architecture has an additional factor of $\sqrt{d_k}$. This architecture makes use of multiplicative attention as it is "much faster and more space efficient in practice, since it can be implemented using highly optimized matrix multiplication code" [4].

3.2 Attention Mechanism:

Similar to how neural networks are inspired by human neuron, the attention mechanism is inspired by human attention, in that we attend to particular details and adjust to attend to everything that is relevant over time.
A self-attention mechanism relates positions within a single sequence in order to compute a representation of the sequence. In practice, we parallelize to compute the attention function on a set of queries combined together into matrix Q. The keys and values are also converted together into matrices K and V. The attention is computed in a following manner

- First, take the query and each key to compute the matrix multiplication i.e $Q \cdot K^T$
- We scale them by dividing by $\sqrt{d_k}$. where d_k is the depth.
- We take the softmax() activation function to find the attention weight, which is defined as : $softmax(z_i) = \frac{exp(z_i)}{\sum_i exp(z_i)}$
- then we multiply it with value matrix.

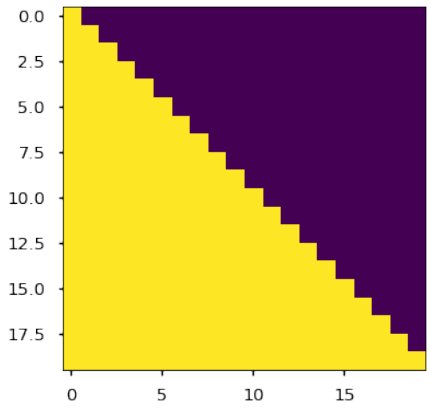

Figure 4: The future leaks are prevented with masking. Yellow region here indicates future information which are zeroed out and only purple region is visible to self-attention layers. This is a matrix plot of mask. It indicates that predictions for position i can only be dependent of the positions i and i above it

Key thing to notice here about the keys, queries and values are:

- For the self-attention layers in the encoder, all of the keys, values, and queries come from the output of the previous layer.
- For the ”encoder-decoder attention” layers, the memory keys and values come from the output of the encoder whereas, queries come from the previous of the decoder layer.
- In the encoder, encoder-decoder, and decoder layers, self-attention allow each position to attend over all positions up to and including the current position.

Since decoder is auto-regressive, we have to stop leftward information flow i.e. masking out the illegal connections i.e. future source in formations are set to $-\infty$. This is done quite cleverly. As the transformers do not support direct mechanism that records time dependence, $Q \cdot K^T$ computes similarities between each words in Q and K, without taking account for ”future leaks”. Recall that for decoder's self attention layer, Q comes from output document and K, V comes from output document. [5]

Now we see why dividing by $\sqrt{d_k}$ is necessary. To understand this, consider that Q and K have a mean of 0 and variance of 1. Their matrix multiplication will yield us a mean of 0 and variance of d_k. Dividing by this produces another variance of 1 for the next level.

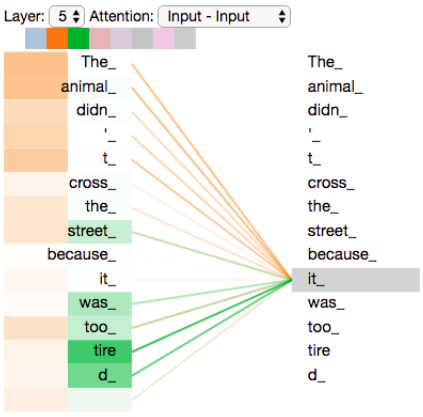

Figure 5: Figure shows how the current word ”it” is attending some of the other positions via two of the eight different attention heads

3.3 Multi-head Attention:

The multi-head attention is used to exploit the features of parallel processing and to improve the performance of the attention layers in the following ways:

- The multi-head attention expands model's ability to focus on different positions, simultaneously.
- One more advantage of this is that the attention layer gets multiple ”representation subspaces.” For this paper, we consider 8 different subspaces in which Q, V, and K will be different - as the $W^Q/W^K/W^V$ will be different - and self-attention will be applied to all these subspaces in a parallel manner. Then, all those 8 different matrices will be concatenated to one matrix as the feed-forward layer is expecting one matrix.
- The concatenation is done by multiplying large concatenated tensor with a weight vector W^O, which was trained jointly with the model.

Let's take an example, showing the results of how two of eight different attention heads are attending. We consider ”The animal didn't cross the street because it was too tired” [6]

3.4 Positional Embedding:

As the order of the input sequence is very important in interpreting the language for translation, sentiment etc. The words may change their meaning if the order is different and they definitely change their sentiment when having different arrival order. Encoder-decoder rely on the order of the

word as the order holds important sentence or language structure, grammar and semantics. Ideally, when encoding positions of a sentence, the time steps should be unique and deterministic. The model should also handle encoding dynamically, and without inductive bias.
The encoding that was proposed is very simple, yet genius and this encoding incorporates and satisfies all of the criteria above. The d-dimensional vector containing all the information about a specific position in a sentence which is not incorporated in a model.
Let pos be desired position in a sentence, then positional encoding is defined as

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

where $i \in [0, 255]$ if $d_{model} = 512$
The frequency of sinusoidal waves are defined as ([4], [7])

$$\omega_k = \frac{1}{10000^{\frac{2i}{d_{model}}}},$$

It is obvious to see that the frequencies are decreasing along the vector dimension. Thus it forms a geometric progression from 2π to $10000 \cdot \pi$ on the wavelengths. ([4], [7])
This method of encoding allows the model to easily learn to attend by relative positions, since for any offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

3.4.1 Position-wise Feed-Forward Networks:

In addition to attention sub-layers, the linear transformation with ReLU activation is performed.

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

While the linear transformations are the same across different positions, they use different parameters from layer to layer.

$$\text{learningRate} = \frac{\min(\text{stepNum}^{-\frac{1}{2}}, \text{stepNum} * \text{warmupSteps}^{-\frac{3}{2}})}{\sqrt{d_{model}}}$$

This corresponds to increasing learning rate linearly for initial training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. The proposed model used 4000 as warmup steps.

3.5 Transformer's Fallbacks

Although it is a novel approach for language understanding, and it was state-of-the-art technique for some time, this model has some limitations as shown in [8]

- For time-series, the inefficiency occurs when the output for a time-step is calculated from entire history of the sequence whereas considering the inputs and current hidden-state would make it more efficient. Transformers perform on entire history of sequence and that's one of the fallback.
- Transformers assume that there is a relationship in input i.e. temporal or spatial relationships.The lack of these relationship will cause it to see text bag of words.

4 Transformer Developments

4.1 BERT

BERT stands for Bidirectional Encoder Representations from Transformers, and is a conceptually simple model for natural language processing problems. It was introduced and detailed in late 2018, but the seminal paper has been updated recently [9]. BERT introduces a bidirectional transformer, which uses the same base units as a normal transformer encoder, but produces a fully connected network instead of a left-to-right model. The bidirectionality is important because context is often provided throughout the sentence, not just the information written prior. It does this by pre-training on two tasks, masked language modeling and next sentence prediction. Masked language modeling replaces some word in a sentence with a token that BERT then operates on to predict the same encoded result as the original sentence. The mask is introduced to make sure that in the fully connected network the word is not seen from another unit, thereby mapping some identity function. Next sentence prediction is done to give the model relations between sentences. The pre-training is considered unsupervised, is done over a large corpus to create a robust model. Once the pre-trained BERT model is constructed, with some transformations, the model undergoes more "relatively inexpensive training" and is then prepared to complete a specific task with a high degree of accuracy [9]. With pre-training, the time and data required to train a new model decreases dramatically because most of the basics are offloaded. It is of note that in an associated blog post by some of the authors, that, in addition to the introduction of transformers a year prior, part of the success of BERT is due to the increase in computing power since the last attempt at bidirectional networks. [10].

4.2 Transformer-XL

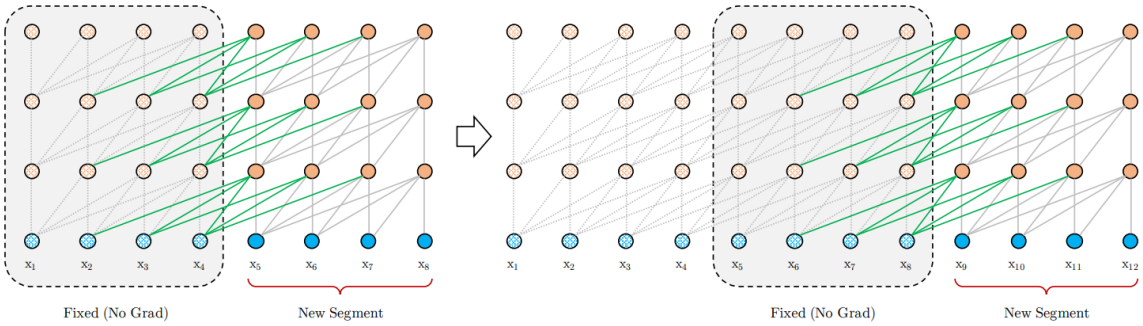

Figure 6: Training phase for Transformer-XL. Note the maximum diagonal length.

At a simple level, Transformer-XL, is a literal extension of the transformer architecture that provides richer connections between the execution of preset length segments. The sequence hidden states are cached for later use when continuing training on the next segment in the data. These hidden states carry the information from the previous training segment while the new segment trains as seen in Figure 6. This speeds up training by preventing recalculation and extends the context available to the transformer in a very direct way, which is visualized in Figure 7. In addition to providing better results, the model is claimed to execute orders of magnitude faster than some previous transformer models on large attention lengths. It is of note that these claims were based on computing with only one GPU and the extended context reduces the parallelizability of the model, so these results may not be representative of the practical efficiency gain. [11].

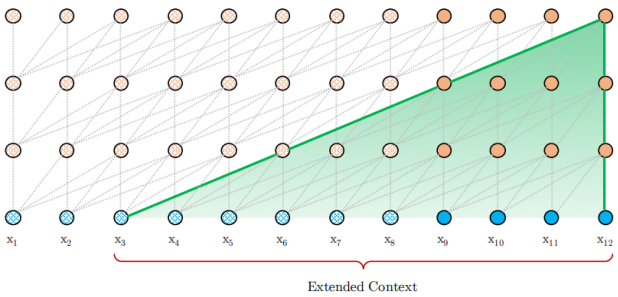

Figure 7: Evaluation for Transformer-XL. The extended context is generated by the forward pass of the hidden state.

5 Applications

5.1 Machine Translation

Evaluating machine translation is a difficult task. While evaluating, we look for the correctness of the translation and grammatical structure of the sentence. We define two criteria for evaluating translation with Adequacy and Fluency. Adequacy is similar to word-to-word translation, but it is not fluent, i.e. not a usual grammatical structure in native language. Whereas, fluency is when grammatical structure in native language is correct, but the correctness of translation is lost. When the translation is both adequate and fluent, the translation will be both grammatically correctly structured and correctly translated.

Transformers were mainly introduced for understanding the context of the sentence. As evaluating translation includes the understanding of the context. As described by the resource, the transformers solve the ”coreference resolution” problem. Below are two english sentences with their french translations. [12]

The animal didn't cross the street because it was too tired.
L'animal n'a pas traversé la rue parce qu'il était trop fatigué.

The animal didn't cross the street because it was too wide.
L'animal n'a pas traversé la rue parce qu'elle était trop large.

Figure 8: The context describes that ”it” refers to two different objects in both English and French

It is obvious to us that the first ”it” refers to the ”animal” and second ”it” refers to the ”street.” The transformers solve this problem using Attention mechanism as the attention mechanism. The self-attention mechanism correlates ”it” word accordingly and when translating the sentence to French, it solves it correctly. In the figure, we can see how ”it” word is being referenced in two different English sentences, and how the attention affects the machine interpretation. [12]

5.2 Sentiment Classification

The problem of sentiment classification, as defined in [13], is,given a document d, to ”discover all opinion quintuples $(e_i, a_{ij}, s_{ijkl}, h_k, t_l)$ in d.” These variables were previously defined as the entity, the relevant aspect of the entity, the sentiment on that aspect, the holder of the opinion, and the time it was stated. This quintuple problem statement was designed many years ago, and the author of the text acknowledges that, while the problem formulation has been investigated, feasible and accurate solutions did not exist at the time, despite the existence of several techniques that are somewhat related

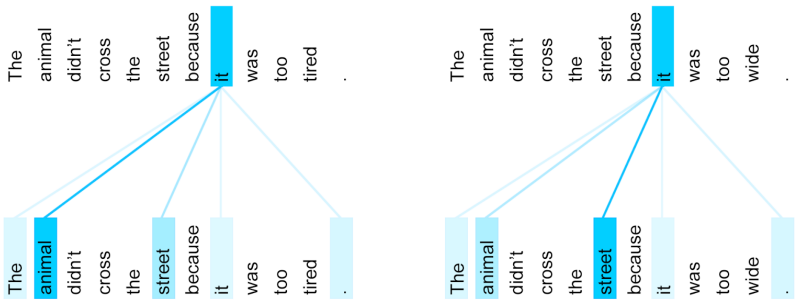

Figure 9: How the word ”it” is attended to ”street” and ”animal” in two different sentences

to transformers. In the case of LSTM, the context required to extract the desired information could not be provided by the cell state, and the quintuple could not be easily constructed. The distance between sentiments and objects tends to be small, so sentiment labels and the corresponding object, identified in the quintuple problem statement as an aspect of some entity, are easily found. However, the task of automatically attaching aspect to entities can be difficult. This relationship can be thought of as a type of disambiguation, which, thanks to the attention mechanism, transformers excel at. Certain tasks identified in the book, such as implicit aspect sentiment analysis, may still be difficult for transformers to accomplish because the attention mechanism operates on a more generalized reference scale without specifically adhering to a rigid quintuple structure, as identified in the problem, but the end goal of classifying the sentiment is still very much possible.

6 Conclusion

The transformer architecture is a significant improvement over previous methods of maintaining context, and has many more applications than those discussed. The architecture is designed with natural language processing in mind, and achieved state-of-the-art results when it was produced. In several NLP tasks, the basic transformer was superseded primarily by transformer derivatives and improvements. It remains a relevant and important aspect of NLP, and new developments are still occurring. As they improve and adapt, the basic theories of self-attention and positional embedding as we discussed will not change significantly.

References

[1] Christopher Olah. Understanding LSTM networks, August 2015.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

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

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

[5] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*, 2017.

[6] Jay Alammar. The illustrated transformer, 2018.

[7] Amir Hossein Kazemnejad. Transformer architecture: The positional encoding, 2019.

[8] Transformer model for language understanding, 2019.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[10] Jacob Devlin and Ming-Wei Chang. Open sourcing BERT: state-of-the-art pre-training for natural language processing, November 2018.

[11] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.

[12] Jakob Uszkoreit. Transformer: A novel neural network architecture for language understanding, 2017.

[13] Bing Liu. *Sentiment Analysis and Opinion Mining*. Morgan & Claypool Publishers, May 2012.