Understanding Transformers

Article · April 2023

CITATIONS
0

READS
110

1 author:

Ankit Hanwate
Indian Institute of Technology Bombay
2 PUBLICATIONS **0** CITATIONS

SEE PROFILE

Understanding Transformers

Ankit Hanwate
Chemical Engineering
IIT Bombay
Mumbai, India
190020045@iitb.ac.in

***Abstract*—This article explains the state of art deep learning architecture called Transformers.**

Keywords—Self attention, multi-head attention, encoders and decoders, positional encoding, RNNs

INTRODUCTION

Transformers have taken the machine learning world by storm, and are now one of the most widely used models for a wide range of natural language processing (NLP) tasks, including language translation, question answering, and text generation. The original transformer architecture was introduced in the paper "Attention is all you need" by Vaswani et al. (2017), and since then, it has been shown to outperform recurrent neural networks (RNNs) on many NLP tasks. In this article, we will explore how transformers work and how they have replaced RNNs as the go-to model for NLP tasks.

A. Recurrent Neural Networks

Before we start with our description of transformers, it's important to understand RNNs, which were the dominant model for NLP tasks prior to the introduction of transformers. RNNs are neural networks that are designed to handle sequential data, such as text. They work by maintaining a hidden state that is updated as new input is fed into the network. The hidden state contains the information of the previous state inputs which allows the network to maintain a pattern and context and the temporal dependencies.

There's another type of recurrent neural networks called Bi-direction RNNs. Sometimes we want an output prediction that depends on the whole sequence input. For example, in speech recognition, the correct interpretation of the current sound as a phoneme may depend on the next few phonemes because of co-articulation and potentially may even depend on the next few words because of the linguistic dependencies between nearby words: if there are two interpretations of the current word that are both acoustically plausible, we may have to look far into the future (and the past) to disambiguate them. This is also true of handwriting recognition and many other sequence-to-sequence learning tasks. For such cases we need Bi-directional recurrent neural networks. This article will not deep dive into these architecture as our major focus is on Transformers.

B. Limitations of RNN

RNNs were very popular when they were the only architecture present to deal with the sequential data. However, RNNs have some limitations. One major problem is the issue of vanishing gradients, which occurs when the gradients used to update the parameters of the network become so small that they effectively vanish, preventing the network from learning long-term dependencies. The model stops learning because of small gradients and this is because over different time stamps the weights corresponding to features remain the same. There's a specific reason for why this happens as per Ian Goodfellow[1]. The problem arises because the architecture uses the same weights and biases for features in each iteration over a time stamp without changing them. And since these weights get multiplied by the same number again and again for each iteration the gradient become small and the model does not converge and gives rise to slow learning. Additionally, RNNs are computationally expensive to train and require significant computational resources, making them difficult to scale.

II. TRANSFORMER ARCHITECTURE MODEL

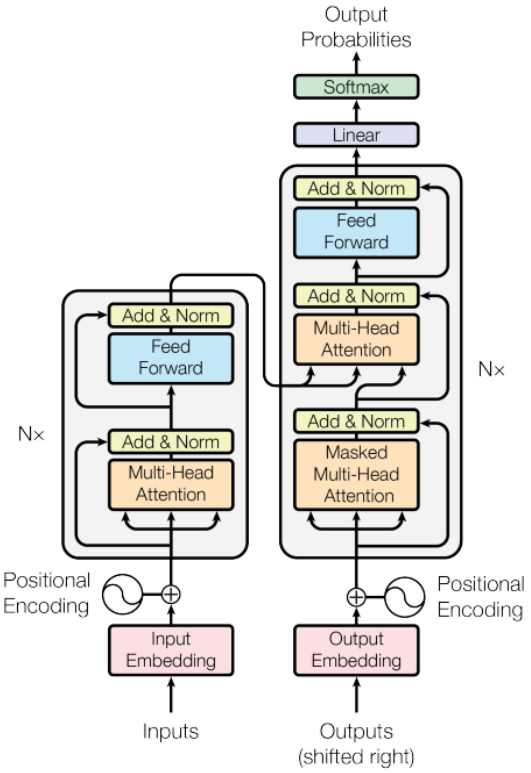

Fig 1: Transformer Architecture[2]

Vaswani et al.(2017) introduced the transformer model, which was designed to overcome the limitations of RNNs. The transformers are attention-based models that are capable of processing sequential data without the need for recurrence. Instead, it uses a self-attention mechanism that allows it to consider all of the input tokens simultaneously, rather than processing them one at a time.

The architecture consists of an encoder and a decoder, with each component consisting of multiple layers. The encoder takes the input sequence and produces a set of hidden representations, while the decoder takes these representations and generates the output sequence.

The transformer architecture consists of a stack of identical layers, with each layer consisting of two sub-layers - a multi-head self-attention layer, and a feedforward network. Each sub-layer is followed by a layer normalization step. We can see the layers in Fig 1.

The multi-head self-attention layer allows the model to attend to different parts of the input sequence, while the feedforward network applies a nonlinear transformation to the outputs of the attention layer. The layer normalization step is used to normalize the output of each sub-layer to have zero mean and unit variance, improving the stability and speed of training.

Layer normalization[2] is defined as follows:

$$X_{\text{normalized}} = \alpha\frac{(X - \mu)}{\sigma} + \beta$$

where α and β are learned scaling and shifting parameters, μ and σ are the mean and variance of the input X, respectively.

The encoder and decoder architectures differ slightly, with the decoder also including an additional masked multi-head self-attention layer. The masked self-attention layer ensures that the model only attends to previous tokens when generating the output, preventing it from peeking ahead. Let's look into each of the parts individually.

A. *Self-Attention*

Transformers do not use the concept of recurrence which is currently used in RNNs. They use a specific notion of attention also called self-attention which was a key innovation in NLP. Self-attention allows the model to focus on different parts of the input sequence to generate the output. Each input token is transformed into three vectors - the query vector, the key vector, and the value vector. These vectors are then used to calculate a score for each token in the sequence, indicating how much attention should be paid to that token when generating the output.

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

Here, Q = Query vector, K = Key vector, V = attention weights, d_k = dimensionality of key vector

The attention scores are calculated using the dot product between the query vector and the key vector, which is then scaled by the square root of the dimensionality of the key vector. The resulting scores are then normalized using the softmax function to produce a set of weights that indicate how much attention should be paid to each input token. Finally, the value vectors are multiplied by the attention weights and summed to produce the output vector.

B. *Multi-Head Attention*

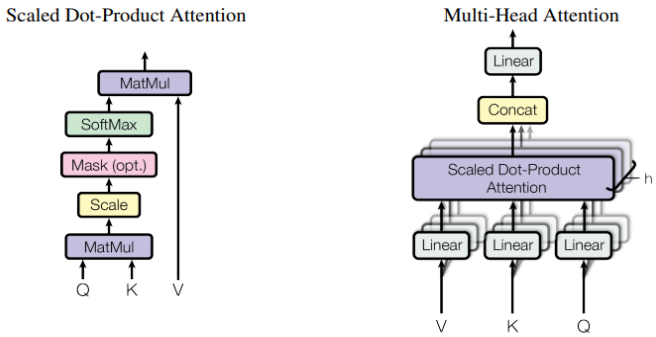

Fig 2: Multi Head Attention model

To improve the effectiveness of self-attention, the transformer model uses multi-head attention. Multi-head attention allows the model to attend to different parts of the input sequence simultaneously, allowing it to capture different types of information. In multi-head attention, the input tokens are transformed into multiple sets of query, key, and value vectors, each of which is used to calculate a set of attention scores. The resulting attention vectors are concatenated and transformed using a linear layer to produce the final output.

Transformers use multi head attention in three different ways:

a) In layer "encoder decoder" attention, the memory keys are derived from the encoder output while on the other side the queries are derived from the decoder layer. As a result decoder's all position can now pay attention to every position in the input sequence. Similar to the encoder decoder mechanism in sequence to sequence models.

b) Self-attention layers are present in the encoder. All of the queries, keys and values in a self-attention layer originate from the same source.

c) Similar to this, the decoder's self-attention layers enable each position to pay attention to all positions up to and including it. To maintain the auto-regressive characteristic of the decoder, we must prohibit leftward information flow. By masking away (setting to infinity) any input values for the softmax that correspond to illicit connections, we implement this inside scaled dot-product attention.

C. *Positional Encoding*

Only challenge with transformers is that they do not have any inherent notion of sequencing as the model

processes all input tokens simultaneously basically no recurrence and no convolution. In order to make use of the sequence we must inject some information about the relative and absolute position of the tokens in sequence. To address this issue, the transformer model uses positional encoding. Positional encoding is a method of adding information about the position of each token to the input embeddings. We add positional encodings to the input embeddings at the bottom of the encoder and decoder stacks. The positional encoding vector is added to the input embeddings, allowing the model to distinguish between different positions in the input sequence. The positional encoding vectors are calculated using a fixed function that depends only on the position of the token and the dimensionality of the embedding.

D. Advantages of Transformers over RNNs

Transformers have a really good edge over the traditional Recurrent neural network for NLP tasks. First, they are highly parallelizable, allowing them to process input sequences more quickly and efficiently. This makes them ideal for large-scale NLP tasks, such as language translation.

Second, transformers are able to capture long-term dependencies more effectively than RNNs, thanks to the self-attention mechanism. The self-attention mechanism allows the model to focus on different parts of the input sequence when generating the output, allowing it to maintain a sense of context and capture long-term dependencies.

Another thing is that transformers are more stable during the training period than RNNs. RNNs can suffer from the vanishing gradients problem as discussed earlier, which can prevent them from learning long-term dependencies. Transformers do not have this problem, as they use self-attention instead of recurrence as explained earlier in the article.

E. Applications of Transformers

Transformers have been applied to a wide range of NLP tasks. With the increase in the computation power of working devices in today's world use of transformers has increased mainly because of the accuracy it provided in the tasks performed. They have been particularly effective at language translation, achieving state-of-the-art performance on many benchmark datasets.

Some of the applications of transformers are mentioned below:

Language translation: Transformers models have been used in translating the languages around the world with great accuracy. Most renowed of these is the Google Neural Machine Translation system.

Image Captioning: These models can generate captions from the content of the image. This feature is mainly useful for image search engines and for visually impaired users.

Sentimental Analysis: Models are capable to clasiffy text as positive, negative and neutral. This has been used for social media monitoring, product reviews and undersranding costomers feedbacks.

F. Conclusion

Transformers have revolutionized the field of NLP, replacing RNNs as the go-to model for many tasks. The self-attention mechanism allows transformers to capture long-term dependencies and attend to different parts of the input sequence, while the parallelizable architecture makes them highly efficient for large-scale NLP tasks. The transformer architecture has been applied to a wide range of NLP tasks, achieving state-of-the-art performance on many benchmark datasets. As the field of NLP continues to grow, it is likely that transformers will continue to play a central role in the development of new models and applications.

ACKNOWLEDGMENT

I'm grateful to Prof. Amit Sethi for his constant guidance and support throughout the duration of the course. I'm thankful to the professor for giving me a opportunity to work on this project. His methods of teaching and explaining the concepts have helped me gain more knowledge on Machine Learning in general.

REFERENCES

[1] Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville

[2] Attention is All you Need (neurips.cc) Ashish Vaswani, 2017, p. 15)

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016..

Special Mention to Optimus Prime as well. How can we forget Optimus when there's a talk of Transformers 😊