Literature Notes of

Language Model

XUE Bo-yang

2020.7.2 - 2020.7.8

Attention Is All You Need

I. Introduction

This paper proposed a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with RNNs and CNNs entirely, including an encoder and a decoder. Recurrent models preclude parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples, and how to align the input sequences still remains challenging. This study achieves state-of-the-art on two popular benchmarks

II. Methodology

2.1 Attention

Attention is used to calculate "relevance" of sequences. Attention can also be described as follows, mapping inputs through query(Q), key(K) value(V) to outputs, where query(Q), key(K) value(V) are vectors related to shape of inputs. Outputs are the weights of all values, which are calculated by queries and each key. The calculation method is shown as follows:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The first step in calculating self-attention is to generate three vectors from each input vector of encoder. In other words, for each word vector, we create a query vector, a key vector, and a value vector. These three vectors are created by word embedding and multiplication by three weight matrices. It can be found that these new vectors are lower in dimension than word embedding vectors. Actually, it is a map of word to word model. Each word has its own weight corresponding to other words and masked Self-Attention is used here to mask future information for current sequence when processing language model.

Multi-Head Attention is used to improve performance by dividing the query(Q), key(K) value(V) into 8 encoders/decoders (in this paper), and concatenate them in final layer.

2.2 Transformer

Position Embedding

Sequence information is important which represents the global structure. In order to capture position information, we create position embedding for each word which has

the same size with word embedding in last dimension, and then the original word embedding and position embedding are added together to form the final embedding as the input of Encoder/Decoder. The calculation formula of Position Embedding is as follows:

$$PE(pos, 2i) = \sin\left(pos/10000^{2i/d_{model}}\right)$$

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

Where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to 10000 · 2π. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, $PE(pos+k)$ can be represented as a linear function of $PE(pos)$.

Encoder

Encoder has N=6 layers, each layer includes two Sub-Layers: the first Sub-Layer is Multi-Headed Self-Attention mechanism, used to calculate the input Self-Attention; the second Sub-Layer is a full connection layer. In each Sub-Layer we have introduced the residual network, and the output of each Sub-Layer is as follows:

$$LayerNorm\ (x + Sunlayer(x))$$

Decoder

Decoder also has N=6 layers, each layer includes 3 Sub-Layers: (1) The first is Masked Multi-head Self-Attention, which is also the input Self-Attention, but as it is a generation process, at time i, there is no result at times after i, only at times less than i. As a result, Mask is necessary for future information. (2) The second Sub-Layer is a fully connected network, similar as Encoder. (3) The third Sub-Layer is to perform Attention on inputs of Encoder. At the same time, the Self-Attention layer in the Decoder needs to be modified, because only the input before current time can be obtained, so Attention is only performed for the input before the time i, which is also called the Mask operation.

My Transformer Test Model: https://github.com/AmourWaltz/Transformer_testing

Character-Level Language Modeling with Deeper Self-Attention

I. Introduction

This paper proposed a deep transformer model (64-layer) with a fixed context length, which outperforms RNN variants, and achieving state-of-the-art on two popular benchmarks. In experiments, the study finds it important to add auxiliary losses at intermediate network layers and intermediate sequence positions.

II. Methodology

2.1 Character Transformer Model

Language models assign a probability distribution over token sequences $\boldsymbol{t_{0:L}}$ by factoring out the joint probability as follows, where $\boldsymbol{L}$ is the sequence length:

$$\Pr(\boldsymbol{t_{0:L}}) = P(\boldsymbol{t_0}) \prod_{i=1}^{L} P(\boldsymbol{t_i}|\boldsymbol{t_{0:i-1}})$$

A transformer network is trained to predict the conditional probability $P(\boldsymbol{t_i}|\boldsymbol{t_{0:i-1}})$ by processing the character sequence $\boldsymbol{t_{0:i-1}}$.To calculate $P(\boldsymbol{t_i}|\boldsymbol{t_{0:i-1}})$, initial model with the causal attention mask is referred to limit information flow from left to right, which is similar as decoder part of Transformer. As information isn't allowed to pass between batches, contexts is relatively short-term when predicting.

2.2 Auxiliary Losses

It is found that training a network deeper than ten layers will be challenging with slow convergence and poor accuracy, so several types of auxiliary losses, corresponding to intermediate positions, intermediate layers, and nonadjacent targets, are added to speed up convergence of the training significantly.

Multiple Positions

Each position in the final layer is added prediction tasks, extending predictions from one per example to sequence length, which is standard practice in RNN based approaches. It is found that adding this auxiliary loss speeds up training and gives better results.

Intermediate Layer Losses

Lower layers are weighted to contribute less and less to the loss as training progresses. If there are n layers total, then the l_{th} intermediate layer stops

Multiple Targets

At each position in the sequence, the model makes two (or more) predictions of future characters. For each new target we introduce a separate classifier, so that we will produce two or more loss functions for each position, and choose the total loss as the main loss. The others are called character auxiliary losses by a multiplier of 0.5 before being added to their corresponding layer loss.

2.3 Positional Embeddings

Since the model in this paper is deeper, this time information may be lost during the propagation through the layers. In order to solve this problem, a 512-dimensional Positional Embeddings matrix is added to each layer, which is learnable.

III. Conclusion

Character-level VS Word-level

Character-level language models are usually built on the basis of character sequences, which can avoid the OOV (out of vocabulary) problem while word-level language models often encounter, but optimization is more difficult on character-level language model because character sequences are generally longer than word sequences. For this problem, some literatures proposed to train over mini-batches in sequential order of texts to capture longer context information, but the back propagation is truncated because the gradient computation doesn't proceed further than a single batch. This paper utilizes Transformer for character-level language modeling and has achieved the state-of-the-art results.

Auxiliary Losses

This paper is based on Transformer to model character sequences, by introducing auxiliary losses to speed up convergence of the model, effectively reducing the training time. Multiple positions loss is main loss as most language models. Intermediate layer losses and target positions multiple steps serve as additional regulations.