AI CLUB

TASK 3

Learnings from 'Attention is All You Need'

Introduction

Attention is a mechanism utilized in the transformer architecture to enable different vectors in the input space to retain significant information related to one another. It facilitates a focus on relevant information and captures long-range dependencies. For instance, when creating embeddings to predict a word, attention allows us to leverage all preceding words while forecasting the next word in a sequence.

Working of Self-Attention

In the context of predicting the next word in a sequence, embeddings are initially generated using an encoder layer that encodes various data about a specific token into an N-dimensional column matrix representing a vector. We then execute the attention function, defined as:

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

Here, Q represents the query, K denotes the key, and V signifies the value. Q is computed by multiplying a specific token embedding by WQ, and similarly, K and V are derived through the same matrix multiplication. Q can be viewed as a particular token posing a question to its surrounding tokens, K serves as the answer to that question, and V represents the vector that must be added to the original embedding to convey the required meaning.

The aforementioned formula operates as follows: QK^T is the dot product of the query matrix and the K matrix. A high value indicates that a specific embedding is highly correlated with the query, making it relevant for modifying the initial embedding. We divide by $\sqrt{d_k}$ to optimize computation time. Before applying the softmax operator, we perform a process called masking, wherein we set the values of the dot product of all embeddings of the words following a particular word to negative infinity (due to softmax). This ensures that future words do not influence the current word's embedding. Essentially, we aim to predict the next word in a given sequence sequentially, preventing future words from revealing the answer prematurely. We then apply the softmax function to normalize the values of the dot product, resulting in a probability distribution of the various dependencies of the embeddings. The V vector is subsequently multiplied by this distribution to obtain a weighted sum of the changes in the embeddings, thereby encoding the meanings of the surrounding words. This is then added to our original embeddings to produce a new embedding.

In the description above, we have outlined what is known as a Single Head of Attention. In practice, multiple heads of attention operate in parallel, each with its own Q, K, and V matrices, proposing their own modifications to the original embedding for each token. Consequently, the final new embedding after processing through an attention block is obtained by summing all these proposed changes with the original embedding:

$$MultiHead(Q,K,V\ ) \ = \ Concat(head1,...,headh)WO \ where \ headi \\ = \ Attention(QW_{Qi}\ ,KW_{Ki}\ ,VW_{Vi}\ )$$

Multi-head attention enables the model to jointly attend to information from different representation subspaces at various positions, overcoming the averaging limitation present in a single attention head.

Multilayer Perceptron

This layer operates similarly to traditional neural networks, where the various embeddings in the matrix do not interact as they do in the attention layer; instead, we perform matrix multiplication. In this layer, we store data related to specific embeddings within the model. It begins by multiplying the original vector with a weight matrix W to generate a new higher-dimensional vector. This can be conceptualized as a dot product between the rows of the weight matrix and the embedding, thereby encoding relevant data. We then obtain certain positive and negative values. Negative values indicate that a particular row of the weight matrix is unrelated to the embedding, while positive values suggest a relationship. Consequently, we apply the ReLU function to retain only the positive values, setting the negative values to zero. We then multiply another weight matrix with this high-dimensional vector, resulting in a vector of the same dimensions as our initial vector, which is added to the original embedding. This process enables the model to learn and store specific data concerning the various embeddings.

Advantages of Self-Attention

The total complexity per layer is significantly reduced compared to Recurrent and Convolutional networks, allowing for parallelization during training, which greatly decreases the model's training time. Additionally, it facilitates the learning of long-range dependencies due to shorter path lengths compared to RNNs and CNNs.