Self-Attention

Saahith Janapati

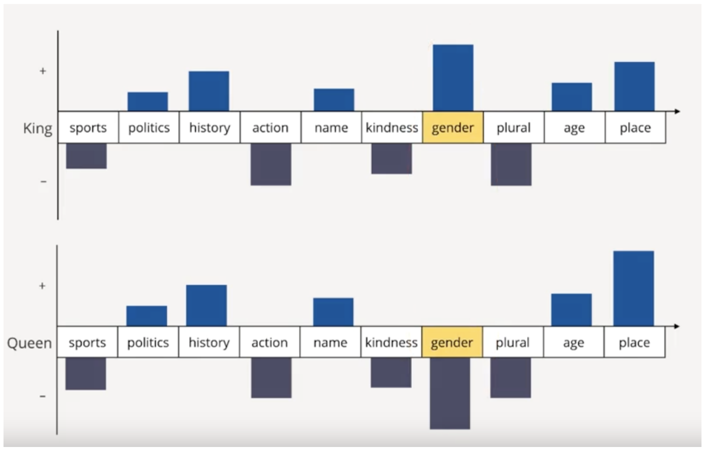
December 2020

1 Introduction

In this lecture, we will discuss the Self-Attention Mechanism. The different types of attention mechanisms have taken the deep learning world by storm; Attention has been found to improve neural network performance on a variety of tasks. Self-attention is at the heart of some of the latest NLP breakthroughs like OpenAI's GPT-3.

2 Word Embeddings

We will first briefly discuss the concept of word embeddings. As you know, neural networks operate on vectors, not words. So, if we are doing some NLP task, we need to first convert the text of interest to a vector representation. With word embeddings, we can replace every word in our text with some n-dimensional vector. This n-dimensional vector, our embedding for the word, captures the semantic meaning of the word in vector space. You could imagine that each dimension of the vector relates to some aspect of meaning like in the picture below.



It's unlikely that the every dimension will correlate exactly to some concept easily understood by humans, but thinking about embeddings like this is helpful for building intuition.

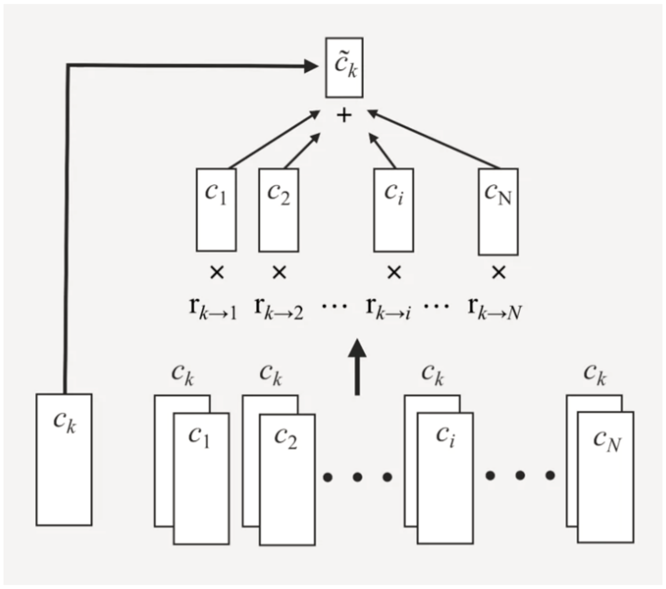
If you're curious about methods to generate embeddings like these, check out this past lecture on Word2Vec, a popular algorithm for generating embeddings.

Using these embeddings, it becomes easy to calculate the similarity between two words. We can simply take the dot product between two words, which is a measure of similarity of two vectors. (Think, when does the dot product get maximized? It's when the angle between the two vectors is 0 degrees.)

3 Self-Attention Mechanism

In this lecture, we will be discussing a specific type of attention mechanism called the self-attention mechanism. This mechanism is used to refine our word embeddings to also capture any contextual information present in our sequence.

This diagram outlines the basic operations of the self-attention mechanism, which will go through step-by-step.



We start off with some sequence of words and we generate the word embeddings for those words. The embedding for the first word of our sequence is c_1, the embedding for the second word is c_2, and so on for all N words of our sequence. As mentioned before, the goal of the self-attention network is to update the embeddings for all the words to also take into account the contextual information present in our sequence (how each word relates to the other words in the sentence).

To create the updated vector for word k, we first compute the dot product between c_k and the embedding for every word (including word k). Then for every position i in our sequence, using the dot product values, we compute $r_{k\to i}$, as shown below using the softmax function to ensure the values are positive.

$$r_{k\to i} = \frac{\exp(c_k \cdot c_i)}{\exp(c_k \cdot c_1) + \exp(c_k \cdot c_2) + \exp(c_k \cdot c_3) + \cdots + \exp(c_k \cdot c_N)}$$

$$\Big\downarrow \ \text{simplified representation}$$

$$= \frac{\exp(c_k \cdot c_i)}{\sum_{n=1}^{N} \exp(c_k \cdot c_n)} \geq 0$$

$r_{k\to i}$ quantifies the relative similarity between word k and word i. The higher the value is, the more similar the two words are.

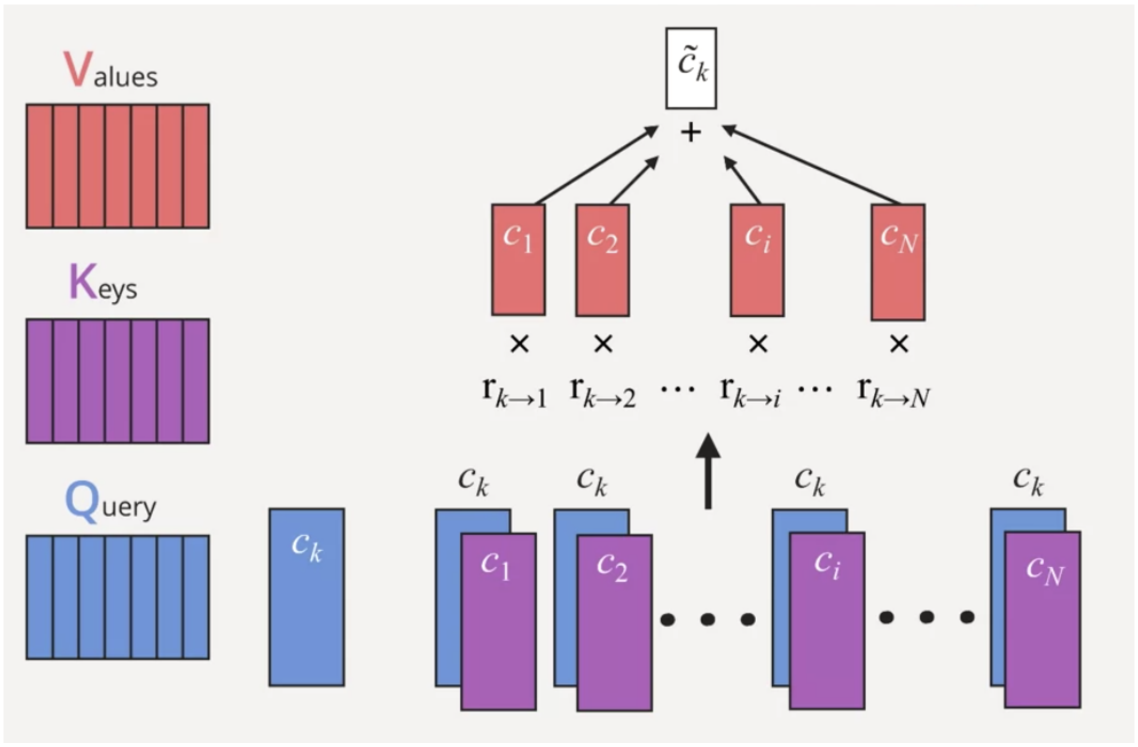
As shown in the diagram above, we then use these r values to calculate a weighted sum of all the original word emebddings to create a vector representation for word k, denoted by the symbol $\tilde{c_k}$. This is formalized in the equation below.

$$\tilde{c}_k = (r_{k\to1} \times c_1) + (r_{k\to2} \times c_2) + \cdots + (r_{k\to i} \times c_i) + \cdots + (r_{k\to N} \times c_N) = \sum_{n=1}^{N} r_{k\to n} \times c_n$$

This entire process is repeated for every position k.

3.1 Terminology: Query, Value, Key

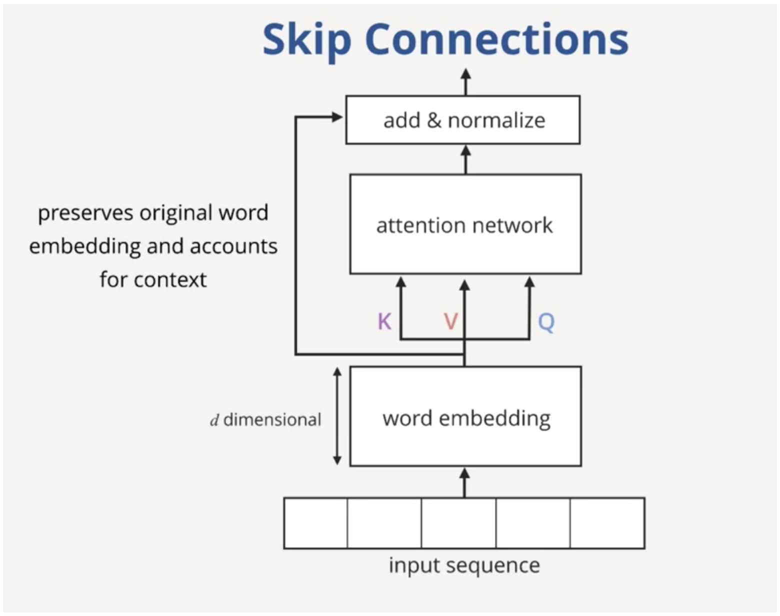
The paper that introduced self-attention (Attention is All You Need) used the terminology query, value, and key to describe the various vectors involved in the self-attention network. They borrowed this terminology from database retrieval systems. The query is the vector to be modified, the keys are the word embeddings of all the words in the sequence, and the values are vectors that are weighted using the $r_{k\to i}$ values. Understanding this terminology is important for understanding later concepts we may discuss, such as transformer networks.



3.2 Modifications

3.2.1 Skip Connections

One modification that has been empirically found to improve performance of self-attention networks on a variety of tasks is to add skip connections between the word embeddings and the outputs of our attention network (the process described above) to preserve some of the original word meaning. This is shown in the image below.



3.2.2 Positional Encodings

You may have noted that output vectors we generate using the self-attention mechanism are order-invariant relative to the sequence. That is, a different order of the input sequence would lead to exact same output vectors. But we know from experience that different ordering of words can lead to completely different semantic meaning. To ensure that positional information is also captured in our output vectors, we introduce the concept of positional encodings.

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

where

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

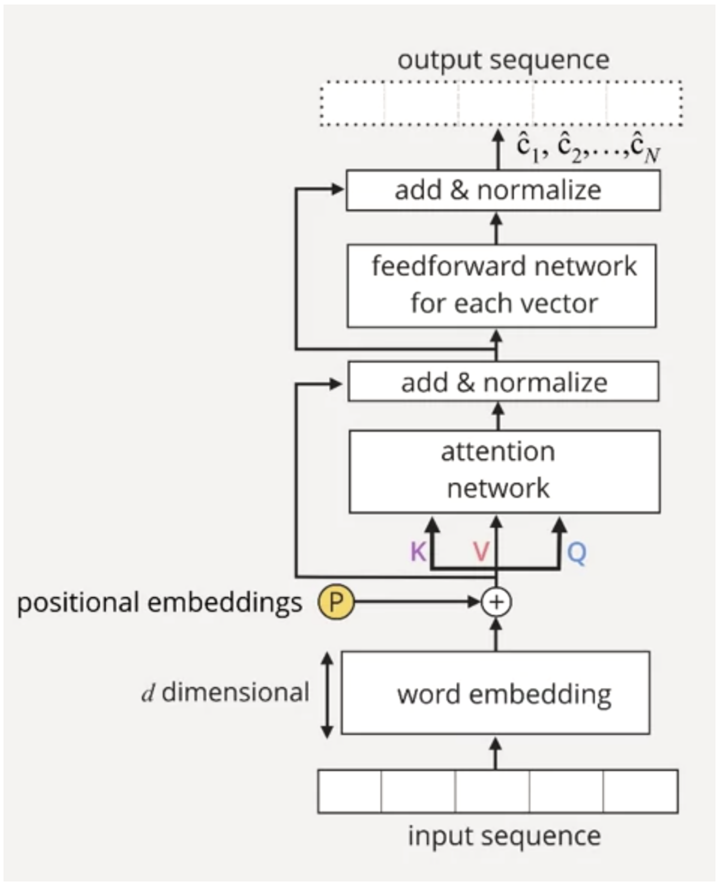
$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

The vector we end up generating using this formula captures the positional information for each word. We then simply add this signal to the word embeddings before passing them on to our attention network. The attention network can then also make use of positions of words to generate final output vectors that more accurately represent the sequence.

Check out this blog post that goes further into the theory about positional encoders.

4 The Attention-Based Sequence Encoder

We can put these two modifications together to create the Attention-Based Sequence Encoder. In addition to the two modifications we have already discussed, we also add a feedforward network (the weights of which are trainable) that each output vector is pushed through, along with some normalization layers. The end product is depicted in the following diagram.

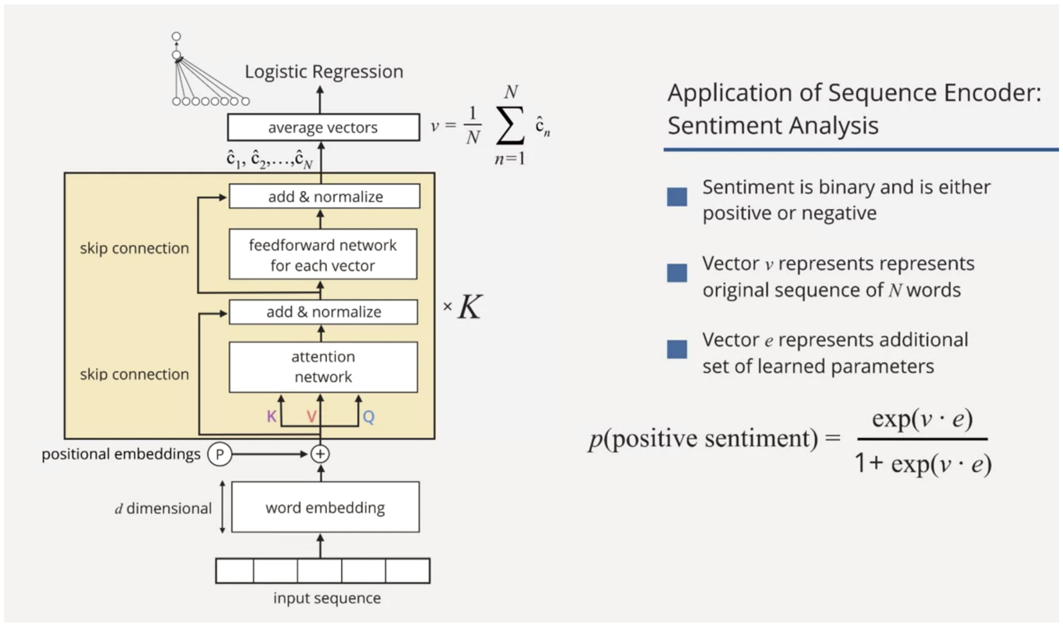


5 Applications of Self-Attention

Although self-attention is perhaps most popular for being used in transformer networks, they are also widely used in sentiment analysis and sequence modeling tasks. The specifics of these tasks are described below.

5.1 Sentiment Analysis

We can easily adapt the Attention-Based Sequence Encoder to create a sentiment analysis model as shown below. Here, we simply average the output vectors generated by our Attention-Based Sequence Encoder and perform a dot product with vector e, which is trainable. The probability of a positive sentiment can then expressed using the equation in the following image.



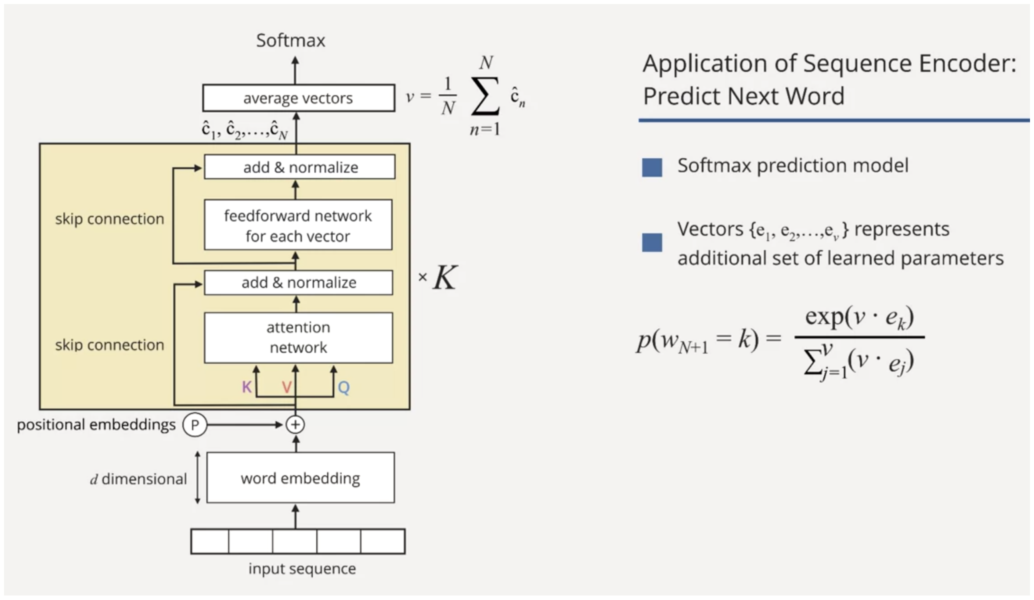
Application of Sequence Encoder: Sentiment Analysis

- Sentiment is binary and is either positive or negative
- Vector v represents represents original sequence of N words
- Vector e represents additional set of learned parameters

$$p(\text{positive sentiment}) = \frac{\exp(v \cdot e)}{1 + \exp(v \cdot e)}$$

5.2 Predicting Next Word

We can also adapt the Attention-Based Sequence Encoder to do next-word prediction. Here, we use the softmax function along with the vectors $e_1, e_2 \ldots e_N$ for each possible word we could predict.



Application of Sequence Encoder: Predict Next Word

- Softmax prediction model
- Vectors $\{e_1, e_2, \ldots, e_v\}$ represents additional set of learned parameters

$$p(w_{N+1} = k) = \frac{\exp(v \cdot e_k)}{\sum_{j=1}^{v}(v \cdot e_j)}$$

5.3 Transformers

Self-Attention blocks are also used extensively in transformer networks that have been at the heart of huge advancements in NLP, like BERT and OpenAI GPT-3. Transformers can be used for language modeling and translation tasks. We may have a lecture in the future on this topic.

6 Advantages of Using Self-Attention

One of the main advantages of using self-attention (rather than simply using word embeddings) for some task is that the resulting output vectors make use of contextual information present in the sequence. This can improve performance on tasks like sentiment analysis and predicting next words.

Another advantage of using self-attention for such tasks (as opposed to recurrent neural networks) is that self-attention networks can make more efficient use of parallel computing power in the form of GPUs. In a recurrent network, as you know, the inputs on one step are the outputs of another. On the other hand, with self-attention networks, there is no inherent order in which the output vectors must be computed. Thus, self-attention networks lend themselves well to GPU optimization.

7 Image Credits

All images were taken from lectures from the Duke Introduction to Machine Learning Course on Coursera.