Appendix A: An Introduction of Transformer-based language models for Political Scientists

Jianjun Yu[1]

[1]*Political science department, University of Iowa, Iowa City, IA, USA. Email: Jianjyu@uiowa.edu*

Transformative advancements in NLP have been witnessed through the emergence of transformer-based language models. These models, such as OpenAI's GPT and Google's BERT, have gained significant attention in recent years for their superior performance across various language-related tasks. Transformer-based language models are deep-learning language models that use architecture proposed by the transformer, a Google-designed machine translator. At the core of the transformer architecture lies the self-attention mechanism, which enables the model to weigh the importance of different words within a sentence by considering their interdependencies. These weights are then utilized to determine how each word's meaning is influenced by the other words in the context, resulting in enhanced word representations (embeddings) and paving the way for various text analysis applications.

Figure 1 provides an illustration of how the weight is calculated for a word in the self-attention mechanism. Consider a given text represented as $(x_1, x_2, x_3, ..., x_n)$, where x_i denotes the tokens (words) in the text. The input to a self-attention block is a sequence of vectors $(a_1, a_2, a_3, ..., a_n)$, with each vector a_i corresponding to the word embedding of x_i, also incorporating its positional information within the text (positional encoding). It's worth mentioning that various methods exist for aggregating positional information, and NLP scholars debate the optimal approach. One common method is to append a vector of order information to the word embedding. These word embeddings are generated by the embedding layers of a transformer-based model using techniques such as Word2Vec or GloVe, where each word is represented by a vector that captures its meaning, ensuring that words with similar meanings have similar vectors (the cosine similarity between two vectors is closer to one).

Now, let's delve into the process of generating the weight for x_1. First, the vector a_1 is multiplied by a matrix W_q, resulting in a vector q_1 of the same length. Subsequently, each vector in the sequence $(a_1, a_2, a_3, ..., a_n)$ is multiplied by a matrix W_k to yield a corresponding vector k_i of the same length. Finally, q_1 is multiplied with each k_i to produce a new sequence of vectors $(\alpha_{1,1}, \alpha_{1,2}, \alpha_{1,3}, ..., \alpha_{1,n})$. This sequence represents the raw weight for x_1 and indicates the importance of token x_i to x_1. These weights also referred to as attention scores. Then, they will undergo normalization using a softmax function to obtain a new sequence of vectors $(\alpha'_{1,1}, \alpha'_{1,2}, \alpha'_{1,3}, ..., \alpha'_{1,n})$, ensuring that $\sum_{i=1}^{n} \alpha'_{1,i} = 1$. The softmax function normalizes the attention scores as follows:

$$\alpha'_{1,i} = \frac{\exp(\alpha_{1,i})}{\sum_{j=1}^{n} \exp(\alpha_{1,j})} \tag{1}$$

This normalized sequence of vectors $(\alpha'_{1,1}, \alpha'_{1,2}, \alpha'_{1,3}, ..., \alpha'_{1,n})$ serves as the weight for x_1. The self-attention block applies the same procedure to calculate weights for each token x and generate a weight matrix $\alpha'_{i,j}$. The elements within the weight matrix $\alpha'_{i,j}$ indicate the significance of token x_j in relation to a token x_i.

Figure 2 visually demonstrates how the influence of other words aggregates into the meaning of a token x_i. Once the weights are computed, a_i is multiplied by another matrix W_v, resulting in a new sequence of vectors $(v_1, v_2, v_3, ..., v_n)$. Subsequently, the new embedding b_i, which encompasses

Political Analysis (2023)

DOI: 10.1017/pan.xxxx.xx

Corresponding author
Jianjun Yu

Edited by
John Doe

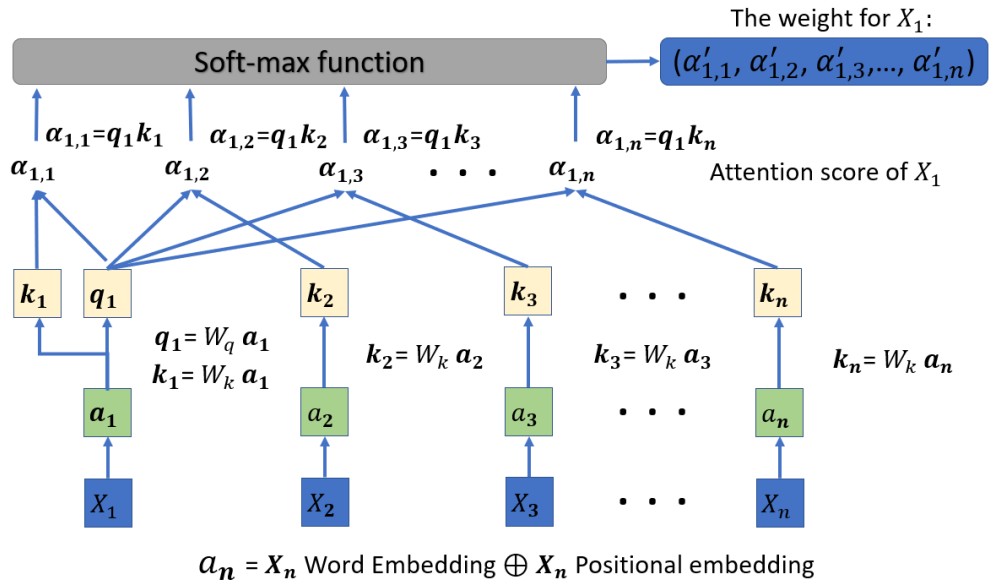

Figure 1. How to calculate the weight for a token X_i

the meaning of token x_i and the influence of other tokens on it, is calculated as:

$$b_i = \sum_{j=1}^{n} \alpha'_{i,j} \cdot v_j \tag{2}$$

The matrices W_q, W_k, and W_v in the self-attention mechanism are commonly referred to as Query, Key, and Value matrices, respectively. These matrices are learned during the training process of the model and are shared across all text inputs processed by the same self-attention block. To gain a better understanding of these matrices, let's consider how search engines rank information for users. When a user enters a query into a search engine, the search engine aims to find websites that are relevant to the query. However, relevance alone is not sufficient for determining the ranking of the search results. A good website should not only be relevant but also contain valuable and useful information. In this context, we can view the query as the user's search query, the key as the representation of a website's topic or theme, and the value as the measure of the usefulness or informational content of the website. To generate a rank of answers, a search engine evaluates the closeness or similarity between the query and the topic of each website and then uses the closeness to weigh the value of the information present on the website.

This concept parallels how humans process information. When we view an image or read a text, our attention is directed to certain aspects depending on our interests (similar to a query). The degree of alignment between our interests and the content of a specific region (represented by the key) influences our focus. Then our focus and the value or significance of the information conveyed by different regions (reflected in the value) determine the overall information we get from an image or a text.

In the self-attention mechanism of TLMs, the Query, Key, and Value matrices serve a similar purpose. They capture the relationships between tokens in a sequence by comparing the query representation of a token with the key representations of other tokens. The resulting weights or attention scores quantify the importance of each token in relation to the others. These weights are then used to combine the corresponding value representations, generating enriched embeddings that reflect the mutual influences and context-dependent relationships among the tokens.

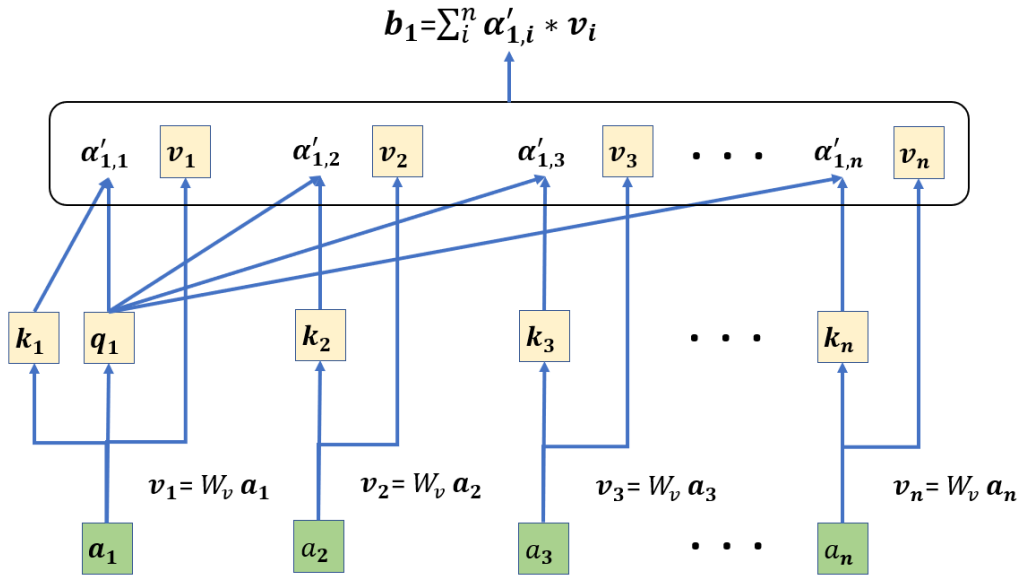

Figure 2. How the influence of other words aggregates into the meaning of a token x_i

In a transformer-based model, multiple self-attention blocks are combined in two different ways. The first approach is the multi-heads approach, where the same word embedding sequence $(a_1, a_2, a_3, ..., a_n)$ is simultaneously inputted into several self-attention blocks. Each self-attention block generates a new embedding sequence $(b_1, b_2, b_3, ..., b_n)$, and these sequences are then aggregated to produce a single embedding sequence $(b'_1, b'_2, b'_3, ..., b'_n)$.

To aggregate the embedding sequences, different methods can be employed, but a common approach is to concatenate the sequences in order. This results in a sequence that is j times longer than the original sequence, where j represents the number of heads (the number of self-attention blocks). Subsequently, a linear function is applied to reduce the dimensionality back to the original length of n tokens. The parameters of this linear function are learned during the training process.

The multi-head self-attention structure allows each attention head to capture different dependencies and relationships within the input sequence. By attending to different parts of the sequence simultaneously, the model can capture a broader range of contextual information and capture more nuanced patterns.

The second approach to combining self-attention blocks is by stacking them in a sequential manner. In this case, a self-attention block is added at the end of another self-attention block, making the structure deeper. The output word embedding sequence from the last self-attention block becomes the input for the next self-attention block. This depth in the structure enables the model to capture increasingly complex relationships and dependencies among the tokens in the input sequence.

Most TLMs use both approaches. Figure 3 illustrates the structure of the base BERT model as an example. The input text is first converted into a word embedding sequence, and then positional encoding is added to each embedding to incorporate position information. The resulting sequence is then fed into a multi-head self-attention block with 12 heads. After the self-attention block, the output embedding sequence is added to the input embedding sequence using a residual connection. This technique helps address the vanishing gradient problem by allowing the model to retain information from earlier layers. The sequence is then layer normalized to normalize the values across the sequence. Following the normalization step, the sequence is passed through a

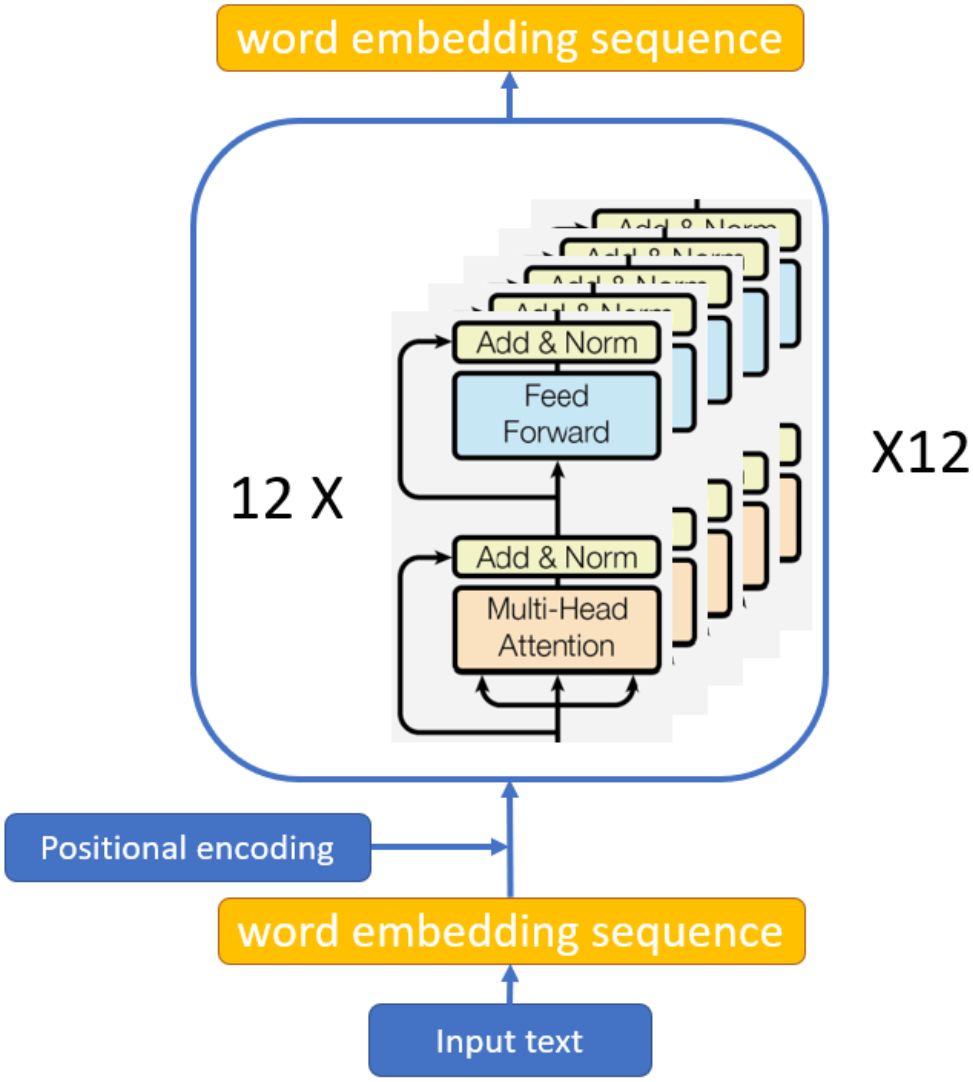

Figure 3. The structure of the base BERT model

fully connected layer, added with its self again, and normalized, which transforms the embeddings into a new word embedding sequence. This sequence then undergoes the same process again. The entire process is repeated 12 times, resulting in the final word embedding sequence of the input text.

It's worth noting that different TLMs may have variations in the number of self-attention blocks, the inclusion of additional layers between self-attention blocks, and the specific types of additional layers used. These variations allow different models to specialize in different tasks or have different capacities for capturing complex patterns and dependencies in the input data.

TLMs are trained using a self-supervised learning approach, which differs from supervised learning, where labeled data is required. In self-supervised learning, the models generate labels from the training data itself, eliminating the need for human-labeled data. In the case of TLMs for NLP tasks, the training sets consist of a vast amount of text data. During training, a portion of a input text is masked, and the model is trained to predict the masked part based on the surrounding context. This process is known as masked language modeling.

By training the model to predict the missing parts of the text, it learns to understand the context and relationships between words, which enables it to generate meaningful representations for the

input text. These learned representations can then be used for various downstream tasks, such as text classification, named entity recognition, or machine translation.

TLMs have emerged as highly efficient and accurate models, surpassing traditional models like CNN and RNN in various machine-learning tasks. Their effectiveness has been demonstrated across a wide range of applications, including chatbots, supervised text classification, text generation, and translation. Furthermore, they have gained popularity in fields like radio and video analysis, as well as attracting interest from researchers working in image analysis.

However, the adoption of transformer-based models in political science research has been limited. Currently, their application in political studies has primarily focused on tasks such as supervised text classification and emotion detection, utilizing models like BERT and GPT. This paper aims to explore the potential of transformer-based models in text clustering and replace commonly used topic models. By leveraging the power of transformer-based models, it is hoped that other scholars will be inspired to explore their applications in various other tasks within the field of political science.