ANLP Assignment 2

Theory Questions

Question 1

What is the purpose of self-attention, and how does it facilitate capturing dependencies in sequences?

Self-attention is a pivotal mechanism in modern neural networks, particularly within transformer models, which has transformed the way sequences, especially in natural language processing (NLP), are handled. Its primary purpose is to allow models to dynamically weigh the importance of different elements in a sequence, enabling them to capture dependencies regardless of their distance from one another.

Purpose of Self-Attention

The self-attention mechanism serves several key purposes:

- **Dynamic Contextualization**: It allows the model to assess the relevance of each element in the input sequence relative to others. This is crucial for understanding context, as the meaning of words can significantly change based on their surrounding words.
- **Capturing Long-Range Dependencies**: Traditional models like recurrent neural networks (RNNs) often struggle with long-range dependencies due to their sequential processing nature and issues like vanishing gradients. Self-attention addresses this by enabling direct interaction between all elements in a sequence, facilitating the learning of relationships over long distances.
- **Parallelization**: Unlike RNNs that process tokens one at a time, self-attention allows for simultaneous computation of attention scores for all tokens. This leads to significant improvements in efficiency and speed during training and inference.

Mechanism of Self-Attention

Self-attention operates through several steps involving three main components: **Queries**, **Keys**, and **Values**. Here's how it works:

1. **Transformation into Vectors**: The input sequence is transformed into three vectors:
    - **Query (Q)**: Represents what the model is focusing on.
    - **Key (K)**: Acts as a reference for each element in the sequence.
    - **Value (V)**: Holds the actual information that will be aggregated based on attention scores.
2. **Calculating Attention Scores**: The model computes attention scores by measuring the similarity between the Query vector and all Key vectors using a dot product. These scores are then scaled and passed through a softmax function to produce attention weights. This process ensures that higher weights are assigned to more relevant elements.
3. **Weighted Sum**: The attention weights are applied to the Value vectors, producing a weighted sum that represents the output of the self-attention layer. This output reflects a contextualized representation of the input sequence, emphasizing important elements based on their relevance.
4. **Multi-Head Attention**: To enhance its ability to capture diverse relationships within the data, self-attention is often implemented as multi-head attention. This involves running multiple sets of Queries, Keys, and Values in parallel, allowing the model to focus on different aspects of the input simultaneously before concatenating the results for further processing.

Facilitating Capturing Dependencies

Self-attention excels at capturing dependencies in sequences due to its unique architecture:

- **Direct Connections**: Every element in a sequence can interact with every other element directly, eliminating the need for sequential processing that limits traditional models. For example, in understanding sentences like "The cat, which was chased by the dog, ran up the tree," self-attention can directly relate "cat" with "ran," effectively capturing long-range dependencies that RNNs might miss due to intervening words.

- **Contextual Representations**: By attending to all tokens in a sequence simultaneously, self-attention produces rich contextual representations that encapsulate nuanced meanings based on surrounding words. This allows models to perform better on various NLP tasks such as translation and sentiment analysis.
- **Flexibility and Scalability**: Self-attention's non-sequential nature allows it to adapt easily to varying input lengths and complexities. The transformer architecture built on self-attention scales well with increased computational resources, leading to advancements in large-scale models like BERT and GPT-3.

In summary, self-attention is crucial for effectively capturing dependencies within sequences by providing a mechanism that evaluates all parts of an input simultaneously, allowing for dynamic contextualization and improved performance across various tasks in NLP. Its introduction has fundamentally changed how models process language, leading to significant advancements in understanding and generating human-like text.

Question 2

Why do transformers use positional encodings in addition to word embeddings? Explain how positional encodings are incorporated into the transformer architecture. Briefly describe recent advances in various types of positional encodings used for transformers and how they differ from traditional sinusoidal positional encodings.

Transformers, a popular architecture in natural language processing (NLP), utilize **positional encodings** alongside **word embeddings** to capture the order of words in a sequence. This is essential because, unlike recurrent neural networks (RNNs), transformers process all input tokens in parallel, which means they lack inherent information about the sequential relationships between words.

Why Positional Encodings are Necessary

Word embeddings represent the semantic meaning of words but do not convey any information about their positions in a sentence. In tasks where the order of words significantly affects meaning—such as language translation or text generation—this absence of positional information can hinder the model's performance. Positional encodings serve as a mechanism to inject this necessary sequential context into the model.

- **Parallel Processing**: Transformers operate on sequences by processing all tokens simultaneously, treating them as a set rather than a sequence. Consequently, without positional encodings, the model would interpret the input as a bag-of-words, losing track of the order and relationships among words.
- **Sequential Structure Understanding**: By adding positional encodings to word embeddings, transformers can differentiate between tokens based on their positions, enabling them to understand the sequential structure of the data and generate coherent outputs[1][2].

Incorporation of Positional Encodings in Transformer Architecture

In the transformer architecture, positional encodings are added to input embeddings before they are fed into the encoder and decoder stacks. This addition combines both semantic and positional information into a single representation.

Mathematical Representation

The traditional sinusoidal positional encoding is defined mathematically as follows:

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

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

Where:

- **pos** is the position of the word in the sequence.
- **i** is the dimension index.
- **d_model** is the dimensionality of the input embeddings.

This method generates unique sinusoidal values for each position, ensuring that each position has a distinct representation while also allowing for relative positioning through the periodic nature of sine and cosine functions. The resulting positional encoding vectors are then added element-wise to the input embeddings[2][3][4].

Recent Advances in Positional Encodings

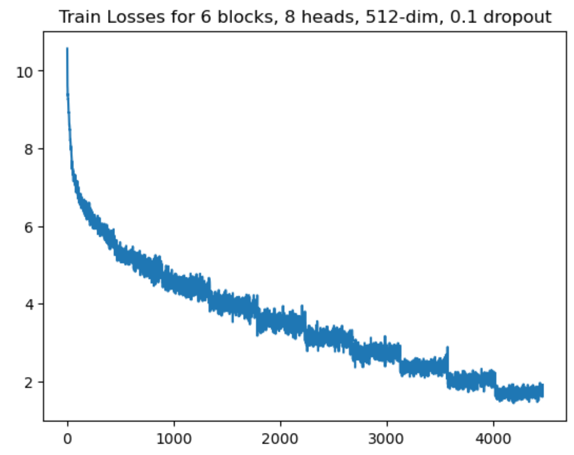
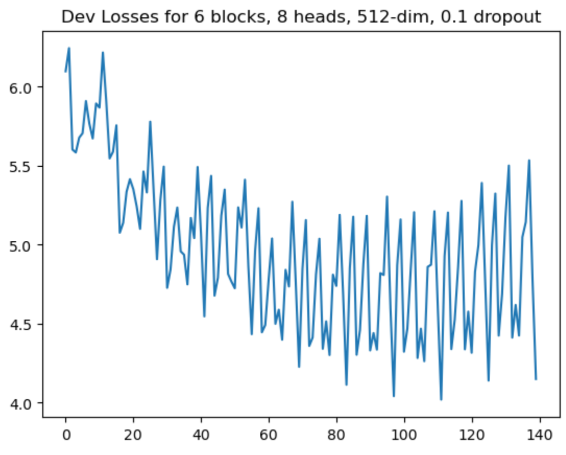
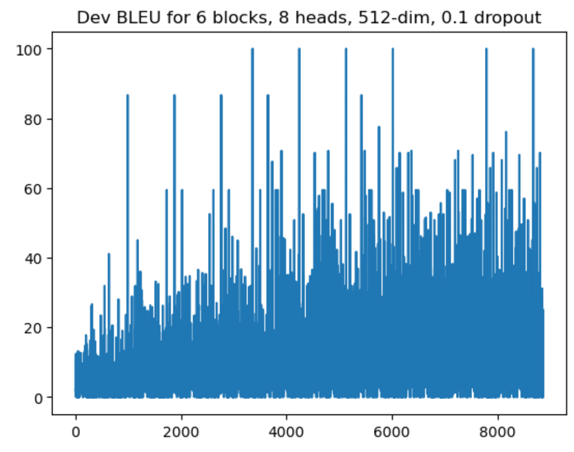
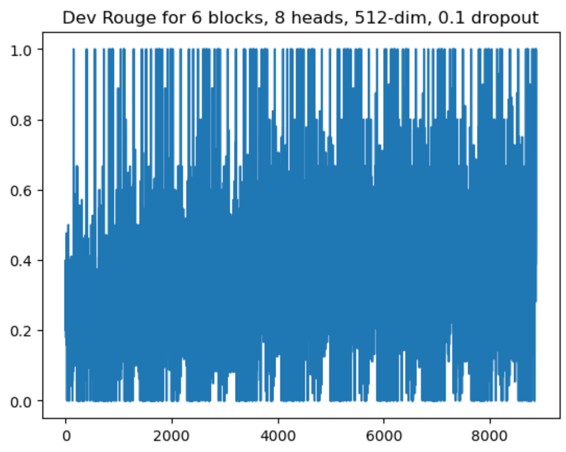
Recent research has explored various alternatives to traditional sinusoidal positional encodings. Some notable advancements include:

- **Absolute Position Embedding (APE)**: This method assigns a fixed vector to each position in a sequence. While straightforward, it may not generalize well across different sequence lengths.
- **Relative Positional Encoding (RPE)**: Introduced in models like T5, this approach focuses on encoding positions relative to one another rather than absolute positions. This can improve performance on tasks requiring understanding of relative distances between tokens.
- **ALiBi (Attention with Linear Biases)**: This method modifies attention scores based on token positions without requiring explicit positional encoding vectors. It allows for more flexible handling of varying sequence lengths and has shown promising results in length generalization tasks.
- **Rotary Positional Embeddings**: This approach incorporates rotation into the embedding space to encode positional information dynamically. It has been found effective in certain transformer architectures by enhancing their ability to capture relationships between tokens over varying distances.

Research indicates that some newer methods like No Positional Encoding (NoPE) can outperform traditional methods by effectively learning both absolute and relative positioning without explicit encoding terms[5].

In summary, while traditional sinusoidal positional encodings laid the groundwork for incorporating order information into transformers, ongoing innovations are refining how these models understand and leverage positional information for improved performance across various NLP tasks.

Tuning


Train Losses for 6 blocks, 8 heads, 512-dim, 0.1 dropout
10
8
6
4
2
0
1000
2000
3000
4000

Dev Losses for 6 blocks, 8 heads, 512-dim, 0.1 dropout
6.0
5.5
5.0
4.5
4.0
0
20
40
60
80
100
120
140

Dev BLEU for 6 blocks, 8 heads, 512-dim, 0.1 dropout
100
80
60
40
20
0
0
2000
4000
6000
8000

Dev Rouge for 6 blocks, 8 heads, 512-dim, 0.1 dropout
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000