Assignment 2

Siya Puttagunta

2021101062

Theory Questions

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

The main purpose of self-attention in models like transformers is to capture dependencies among various parts of sequences whereby the model can focus on the relevant parts of the input while generating an output for each position in the sequence.
Self-attention lets the model understand the relation among words (or tokens) in the sequence, even when they are far apart. So, in the sentence, "The dog that ran across the street is fast," the word "dog" is related to "fast," although they occur far apart in the sentence. Self-attention captures this dependency.

It assigns different attention weights for the different parts of the sequence during the processing of each token, enabling it to pay attention to parts of the input that are important for understanding context.

How Self-Attention Captures Dependencies:
For every word in a sequence, self-attention computes a weighted sum of all other words (even itself), which enables it to "
pay attention" to relevant words according to their relationships. In this paper, the scores of attention are computed based on dot products of the "query," "key," and "value" vectors of each word.

Query: It is the word for which we are computing attention.

Key: It represents the words we compare the query to.

Value: It essentially corresponds to the real information that the words it is focused on carry.

The self-attention mechanism is different from standard recurrent models such as LSTMs since it processes a sequence in
parallel and can attend directly to any word with another word in the sequence. Thus, it is much more efficient at capturing the long-range dependencies.

It is also independent of the words' position, but the **positional encodings** enable it to keep track of the word order in the sequence. This lets self-attention capture more of the sequence in language and is not skewed towards only neighbouring words.

2.2 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 use positional encodings along with the word embeddings because they do not have any natively word-over-word ordering like conventional recursive or convolutional networks. Since transformers process all tokens in a sequence in parallel (not sequentially) one needs to use the positional encodings, which give the model information about the position of each token within the input sequence.

Why Positional Encodings Are Used ?

Word embeddings contain meaning in terms of semantics but do not contain information about their position in the sentence. Nonetheless, position information does matter for language understanding **Ex:** "The cat chased the mouse" is different from "The mouse chased the cat"

Positional encodings are essentially a technique that would inject sequence position information into the model, allowing transformers to distinguish words that stand at different positions.

How Positional Encodings Are Added

Positional encodings are added directly to the word embeddings. Each word within the input sequence is turned into an embedding vector and a positional encoding is added to it. The positional encoding has the same dimensionality as the word embedding and therefore can be added element-wise. This combined representation (word embedding + position encoding) is then fed into the transformer layers.

Mathematically:
Input to the model = Word Embedding + Positional Encoding

This inclusion enables the model to preserve both semantic meaning- from word embeddings; and positional information- from positional encodings.

Traditional Sinusoidal Positional Encodings:
Employing sinusoidal functions in encoding positions lets the model generalize longer sequences of input than the ones it has seen during training time.

The encoding is determined on the sine and cosine functions of different frequencies. That is, for each dimension in the positional encoding vector, there is a corresponding frequency.

The position encoding for a position pos and dimension i is defined as:

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

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

This makes it possible for the transformer to understand the relationship of tokens that are close and far from each other.

Recent Development of Positional Encodings

There have been many variants and alternatives proposed to sinusoidal positional encodings that tackle efficiency and flexibility in the design of the transformer models:

1. Learnable Positional Embeddings :

Instead of using fixed sinusoidal encodings, learnable positional embeddings allow the model to learn the best way to represent positions during training. Each position is associated with a trainable embedding vector, similar to word embeddings which are optimized along with other model parameters. Learnable embeddings can be more expressive and better adapt to the dataset. The model may not generalize as well to sequences longer than those encountered during training.

2. Relative Positional Encodings:

Relative positional encodings view the relationships as a relative distance between tokens rather than an absolute position within the sequence.
This will enable the transformer to learn positional relationships from how far apart tokens are rather than requiring a fixed position number
It was first introduced in models like Transformer-XL and T5 and it has proven to extend the generalisation capabilities for sequences of variable lengths.

3. Rotary Positional Encodings (RoPE):

RoPE is the newest method that includes positional information directly in the self-attention mechanism. It alters the query and key vectors in the self-attention mechanism through rotation-based transformations, so that encoding of the position in the attention mechanism is more efficient.
Useful in GPT-4 and ChatGPT models, which perform better in long sequences

Difference from Traditional Sinusoidal Encodings

Traditional sinusoidal positional encoding is a fixed and deterministic function for the separation of word positions in sequences for a transformer. They are uniform and nonadaptive to a task.

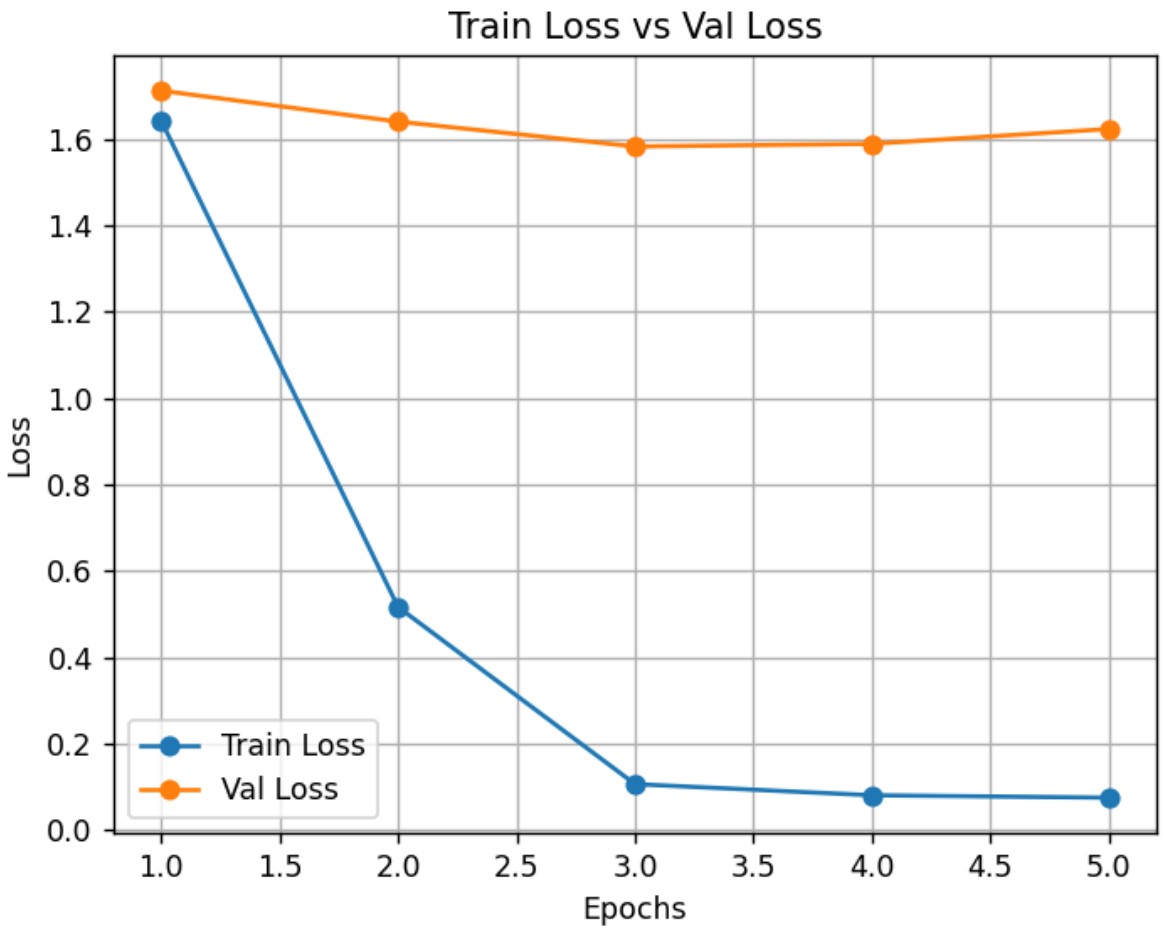
In contrast, recent work introduces learnable positional encodings that the model is allowed to learn representation conditioned on the dataset. With this flexibility, the performance could potentially improve because of capturing task-specific positional dependencies. Although sinusoidal encodings are intuitive and easy to understand, learnable methods have more expressiveness but are less intuitive.

3

Training

```
Epoch 1/5, Train Loss: 1.6423, Val Loss: 1.7111
Epoch 2/5, Train Loss: 0.5168, Val Loss: 1.6395
Epoch 3/5, Train Loss: 0.1070, Val Loss: 1.5820
Epoch 4/5, Train Loss: 0.0812, Val Loss: 1.5881
Epoch 5/5, Train Loss: 0.0757, Val Loss: 1.6225
Test Loss: 1.4675519088419473
```

Loss Curve



Testing

Average BLEU score: **0.124**

3.3 Hyperparameter Tuning

The following hyperparameters were changed:

- number of encoder layers: [1, 2]
- number of decoder layers: [1, 2]
- dimension of word embeddings: [256, 512]
- number of heads: [4, 8]
- dropout: [0.1, 0.2]

The other hyperparameters which were kept constant are the following:

- dimension of feed forward: 1024
- batch size = 32
- number of epochs: 5
- learning rate: 0.001

Best Hyperparameters:

- number of encoder layers: 1
- number of decoder layers: 2
- dimension of word embeddings: 256
- number of heads: 8
- dropout: 0.1

Best BLEU Score: 0.1424

Analysis

Increasing the number of encoder layers decreases the performance of the model whereas increasing the number of decoder layers increases the performance of model. Deeper models can tend to overfit due to increase in number of layers. Increasing the number of heads leads to learning different features and capture diverse dependencies. Increasing dropout may lead to regularisation which results in low performance.