Report

Penumalla Aditya Pavani , 2021101133

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

Self-attention in transformers is essential for capturing relationships between words in a sequence, regardless of how far apart they are. Unlike traditional models like RNNs or LSTMs that process sequences step by step, self-attention allows the model to consider the entire sequence at once for each word. This means that when predicting the next word or translating between languages, the model can "attend" to all other words in the sequence and assign importance based on their relevance to the current word. This is particularly useful in tasks like translation, where word dependencies are not always close together.

Self-attention captures dependencies in a sequence using three components: Key (K), Query (Q), and Value (V).

1. **Query (Q)**:For each word, the model asks, "What other words are important to me?"
2. **Key (K)**: The Key helps answer the question. Each word in the sentence has a Key that tells the model how relevant it is to other words when they ask about it.
3. **Value (V)**: The Value is the actual information that gets passed along. Once the Query finds the important words using their Keys, it looks at their Values to decide how much attention to give them.

In practice, for each word in the sequence, the model compares its Query with all other Keys to find out which words are related or important. Then, based on this comparison, it gathers information (Values) from the most relevant words to better understand the current word. This process allows the model to capture dependencies, even between distant words.

For example, in the sentence "The cat chased the mouse," when focusing on "chased" (the Query), the model might find that "mouse" (Key) is important to understand the action. It then gathers the Value of "mouse" to inform how to process "chased." This helps the model understand relationships and context better. Mutli-head attention helps to capture different kinds of relationships among the 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** because, unlike models like RNNs or LSTMs that process words sequentially, transformers process all words in a sentence simultaneously. Without positional information, the model would not know the order of words, which is crucial for understanding the meaning of a sentence. Positional encodings provide information about the position of each word in the sequence, ensuring the model can differentiate between words based on their order.

How positional encodings are incorporated:

- Positional encodings are special vectors that represent the position of each word in the sequence. These encodings are added to the word embeddings to introduce positional information. The common formula used for positional encodings involves sinusoidal functions:

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

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

- pos is the position of the word in the sequence.
- i is the dimension of the encoding vector.
- $dmodel$ is the dimensionality of the word embedding.

This formula ensures that each word gets a unique position-related encoding, where,sine values are applied to even positions and the cosine values are applied to odd positions.

- Each word embedding is combined with its corresponding positional encoding so that the model knows not only the meaning of the word but also its position in the sequence.
- This addition happens before the self-attention mechanism, allowing the transformer to consider both the word and its position when processing the sentence.

For example, in the sentence "She reads books," positional encodings help the model recognize that "She" comes first and "books" comes last, which is important for understanding the subject and object of the sentence.

Recent Advances in Positional encodings:

- **Learned Positional Encodings**:
  Unlike the fixed sinusoidal encodings, where position is determined by a mathematical formula, learned positional encodings treat positions as trainable parameters. These embeddings are learned alongside the model weights, allowing the transformer to adapt the positional information to the specific task. This approach often performs better in certain domains where positional patterns may not follow simple regularities.
- **Relative Positional Encodings**:
  Instead of encoding the absolute position of words, relative positional encodings focus on the distance between words. This method allows the model to capture how far apart words are from each other, which is particularly useful in cases where understanding word-to-word relationships matters more than their absolute positions in the sequence. This approach helps models generalize better to longer or unseen sequences.
- **Rotary Positional Embeddings (RoPE)**:
  RoPE is a more recent innovation that uses rotational matrices to encode positional information. This method injects positional information into the word embeddings in a way that allows the transformer to compute relative positions between tokens more effectively. RoPE can handle much longer sequences than traditional encodings while maintaining efficiency.

How they differ from traditional Encodings:

These newer positional encoding methods surpass traditional sinusoidal encodings by providing greater flexibility and adaptability. While sinusoidal encodings are fixed and cannot adapt to specific tasks, learned and relative encodings allow transformers to dynamically adjust positional information based on the data. This results in improved performance, especially for longer sequences or tasks where positional relationships are crucial. By better capturing word-to-word interactions and contextual dependencies, these advanced encodings enable transformers to more effectively model complex linguistic patterns and handle a wider variety of tasks.

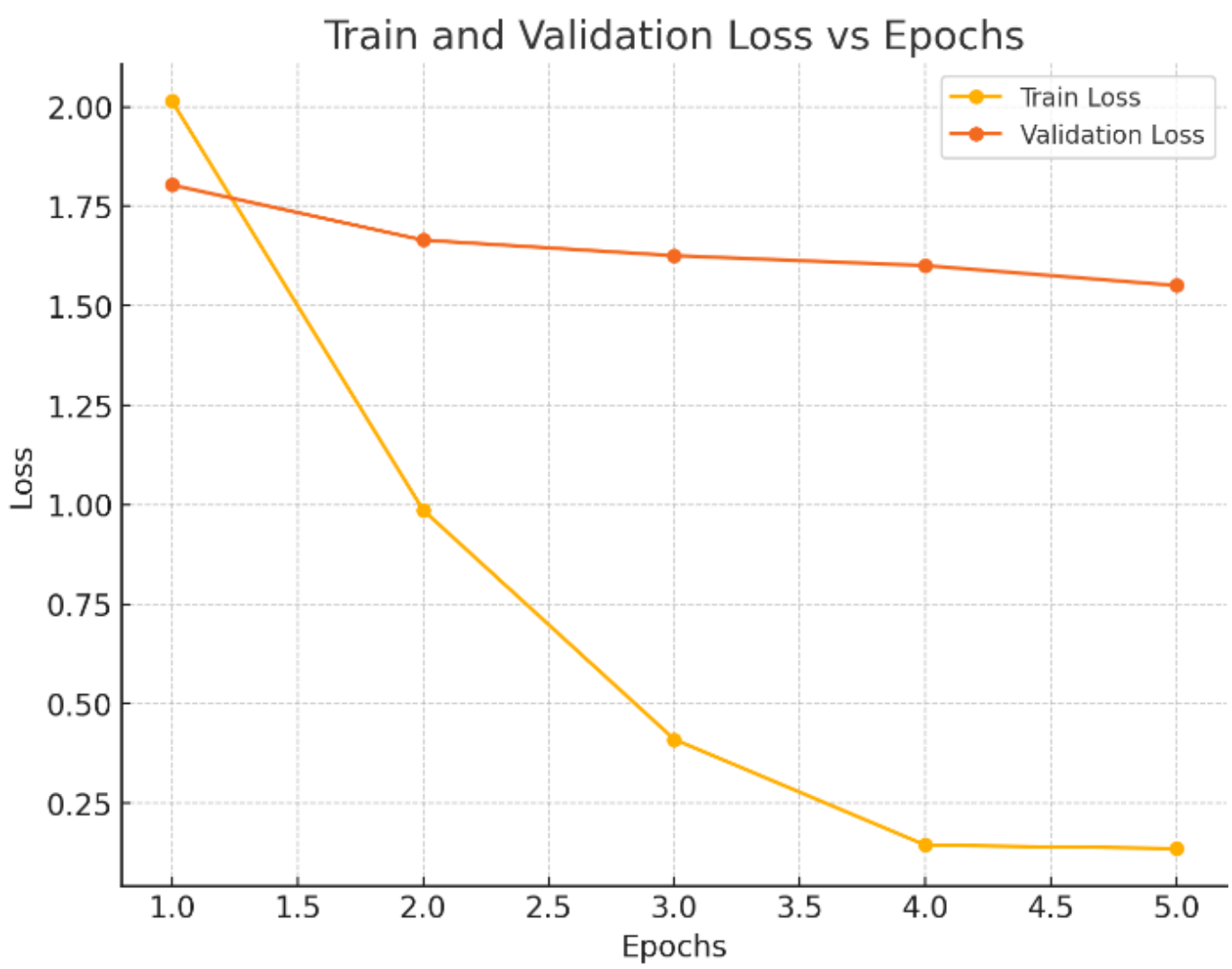
---

3) Training

Hyperparameters

```
epochs = 5
batch_size = 64
d_model = 256
heads = 8
num_encoder_layers = 1
num_decoder_layers = 1
dropout = 0.1
d_ff = 1024
max_len = 1000
lr = 0.001
```

```
train size: 30000
val size: 887
test size: 1305
training started
Training..
Epoch 1/5 - Step 100 - Train Loss: 2.0145
Epoch 1/5 - Step 200 - Train Loss: 1.2396
Epoch 1/5 - Step 300 - Train Loss: 2.1892
Epoch 1/5 - Step 400 - Train Loss: 1.5347
Epoch 1/5 - Train Loss: 1.9061 - Val Loss: 1.8036
Epoch 2/5 - Step 100 - Train Loss: 0.9868
Epoch 2/5 - Step 200 - Train Loss: 0.2701
Epoch 2/5 - Step 300 - Train Loss: 0.2374
Epoch 2/5 - Step 400 - Train Loss: 0.2943
Epoch 2/5 - Train Loss: 0.4104 - Val Loss: 1.6650
Epoch 3/5 - Step 100 - Train Loss: 0.1375
Epoch 3/5 - Step 200 - Train Loss: 0.1628
Epoch 3/5 - Step 300 - Train Loss: 0.1265
Epoch 3/5 - Step 400 - Train Loss: 0.1711
Epoch 3/5 - Train Loss: 0.1446 - Val Loss: 1.6259
Epoch 4/5 - Step 100 - Train Loss: 0.1350
Epoch 4/5 - Step 200 - Train Loss: 0.1198
Epoch 4/5 - Step 300 - Train Loss: 0.1294
Epoch 4/5 - Step 400 - Train Loss: 0.1387
Epoch 4/5 - Train Loss: 0.1361 - Val Loss: 1.6009
Epoch 5/5 - Step 100 - Train Loss: 0.1284
Epoch 5/5 - Step 200 - Train Loss: 0.1284
Epoch 5/5 - Step 300 - Train Loss: 0.1168
Epoch 5/5 - Step 400 - Train Loss: 0.1171
Epoch 5/5 - Train Loss: 0.1256 - Val Loss: 1.5509
Test Loss: 1.5573
```



Average Blue Score : 0.107

3.3) Hyperparameter Tuning

Parameters Tuned:

```
d_model = [256,512]
n_heads = [4,8]
n_en_layers = [1,2]
```

```
n_de_layers = [1,2]
dropout = [0.1,0.2]
```

The encoder and decoder layers are kept same while tuning.
The parameter which are kept constant are:

```
batch_size = 64
max_len = 1000
lr = 0.001
d_ff = 2048
epochs = 5
```

Best Hyperparameters :

```
d_model = 512
n_heads = 8
n_en_layers = 1
n_de_layers = 1
dropout = 0.1
```

Best Blue Score:

```
Average BLEU score: 0.11659722286517978
```

The larger model dimension of 512 and eight attention heads enhance the model's ability to learn intricate patterns and relationships in the data, allowing for richer representations. Meanwhile, using a single encoder and decoder layer simplifies the architecture, reducing training time and mitigating the risk of overfitting. A dropout rate of 0.1 effectively provides regularization, promoting generalization without significantly hindering the model's performance.