Roll No : 2023201011
ANLP
Assignment 2
Report

Questions :

Question : 1
What is the purpose of self-attention, and how does it facilitate capturing dependencies in sequences?

Answer :
The purpose of self-attention in neural networks, particularly in Transformer models, is to capture dependencies between words (or tokens) in a sequence, regardless of their distance from each other. Self-attention enables the model to consider the relationship between each word and all other words in the sequence, rather than just nearby words as in traditional recurrent or convolutional networks.

How Self-Attention Works:

1. **Input Representation**: Each word in a sequence is represented as a vector (embedding) that holds information about the word.
2. **Query, Key, and Value Vectors**: For each word (token), self-attention calculates three vectors: Query (Q), Key (K), and Value (V). These are linear projections of the input word embedding:
    - **Query (Q)**: Represents the current word and asks how much focus it should give to other words.
    - **Key (K)**: Represents other words and helps determine how relevant they are to the current word.
    - **Value (V)**: Contains the actual information of the word that is transferred if the attention mechanism decides that it is important.
3. **Attention Scores**: The Query of each word is compared with the Keys of all words in the sequence to compute attention scores. This is usually done by taking the dot product of Query and Key vectors. These scores reflect how much focus or "attention" the current word should give to the other words.
4. **Softmax and Weighting**: The attention scores are normalized using a softmax function, so they represent a probability distribution over all words. This distribution is then used to weight the Value vectors of the words.

5. **Output**: The final output for each word is a weighted sum of all Value vectors, where the weights come from the attention scores. This allows the model to incorporate information from other relevant words in the sequence.

$$Attention(Q, K, V) = softmax\left(\frac{Q K^T}{\sqrt{d_k}}\right)V$$

How Self-Attention Captures Dependencies:

- **Global Dependencies**: Self-attention allows every word in the sequence to attend to every other word. This is particularly useful for capturing long-range dependencies in the sequence, such as when the meaning of a word depends on a distant word in the sentence.
- **Parallelization**: Unlike recurrent models, which process sequences one step at a time, self-attention can process all tokens in the sequence in parallel. This makes it computationally efficient and allows the model to learn dependencies in parallel.
- **Contextual Representation**: Each word is represented in the context of the entire sequence, allowing the model to understand the importance of each word relative to the others.

By allowing a token to consider all other tokens in the sequence, self-attention enables a more nuanced understanding of relationships and dependencies, crucial for tasks like machine translation, text classification, and language modeling.

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.

Answer :

Transformers rely on **self-attention** to capture relationships between words in a sequence. However, self-attention treats words as a set without inherent order, meaning it doesn't capture the positional information of the words. Unlike recurrent neural networks (RNNs) or convolutional neural networks (CNNs), transformers do not process words sequentially or with fixed receptive fields. Thus, to make the model aware of the order of words, **positional encodings** are introduced.

Positional encodings provide the transformer with information about the position of a word in the input sequence. This is crucial for language tasks where the meaning of a sentence is highly dependent on the word order.

How Are Positional Encodings Incorporated into the Transformer Architecture?

In the transformer architecture, positional encodings are added to the word embeddings at the input layer, before feeding them into the self-attention mechanism. If X is the word embedding matrix, and PE is the positional encoding matrix, the input to the transformer is:

$$\text{Input}=X+PE$$

This combined input (word embeddings plus positional encodings) is passed through the transformer layers. This process ensures that the model can understand not only the relationships between words but also their positions within the sequence.

Traditional Sinusoidal Positional Encodings:

In the original Transformer paper, **sinusoidal positional encodings** were introduced. These encodings use sine and cosine functions to generate a unique vector for each position in the sequence. The advantage of this method is that it introduces a continuous and deterministic positional encoding scheme, which allows the model to generalize to longer sequences beyond the ones seen during training.

For a position pos and a dimension i of the embedding, the sinusoidal positional encoding is defined as:

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

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

Here, dmodel is the dimension of the word embeddings, and the encoding assigns sine and cosine values alternately to even and odd indices. The use of different frequencies allows the model to distinguish different positions in the sequence. This also makes it easy for the model to learn both absolute and relative positional information.

Recent Advances in Positional Encodings :

Several improvements and alternatives to sinusoidal positional encodings have been proposed to address some limitations, such as the inability to generalize well across tasks or sequences of varying lengths. Below are some of the recent advances:

1. **Learnable Positional Embeddings**

Instead of using a fixed sinusoidal function, **learnable positional embeddings** introduce parameters that are learned during training. Each position in the sequence has a corresponding embedding that is updated based on the task. This allows the model to learn more task-specific and flexible positional representations.

- **Advantages**: Learnable embeddings can adapt better to complex tasks.
- **Disadvantages**: These embeddings are limited to the length of sequences seen during training and may not generalize to longer sequences as well as sinusoidal encodings.

2. **Relative Positional Encodings**

Relative positional encodings provide the transformer with the ability to focus on the relative distances between words, rather than their absolute positions. This is especially useful in tasks where the relative order matters more than the absolute positions (e.g., in dependency parsing or long document analysis).

- In this scheme, the attention mechanism computes pairwise distances between tokens, allowing the model to attend to tokens based on how far apart they are.
- This encoding often involves modifications to the self-attention computation to include these relative position terms.

$$Attention(Q, K, V) = softmax\left(\frac{Q K^T + R}{\sqrt{d_k}}\right)V$$

Where R is the relative positional encoding matrix.

- **Advantages**: Improved generalization for tasks requiring attention over varying-length sequences, and better handling of long-range dependencies.
- **Examples**: Used in models like Transformer-XL and T5.

3. **Rotary Positional Embeddings (RoPE)**

Rotary Positional Embeddings (RoPE) introduce a way to rotate the query and key embeddings by an angle proportional to their position in the sequence. This encoding method is designed to be more efficient in handling both relative and absolute positions and can generalize across different sequence lengths better than fixed encodings.

- RoPE applies a position-dependent rotation matrix to both the queries and keys in the attention mechanism, allowing the dot product between queries and keys to encode both absolute and relative positional information.
- **Advantages**: Handles longer sequences more effectively, and integrates seamlessly into the self-attention mechanism without significant architectural changes.

4. **Performer and Random Feature-Based Encodings**

In more recent transformer variants like **Performer**, positional encodings are handled through random feature methods that approximate the self-attention mechanism, reducing the quadratic complexity of traditional attention. The Performer model introduces random Fourier features, which implicitly provide positional information while improving the efficiency of attention computation.

- **Advantages**: Significantly reduces memory and computational overhead, especially for long sequences.
- **Disadvantages**: This method may trade off some performance for efficiency, but it can still be effective in practical applications.

5. **Alibi (Attention with Linear Biases)**

Alibi is a recent innovation that uses linear biases to encode position information directly into the attention mechanism. Instead of adding positional encodings to the input embeddings, Alibi modifies the attention scores by applying a linear bias that grows with the distance between tokens. This approach has been shown to improve model efficiency and performance, particularly in tasks with long sequences.

- **Advantages**: Improves the scaling of transformers for very long sequences, simplifies the architecture by eliminating positional encodings as separate entities..

These advances show the evolution of positional encodings to address the challenges of capturing dependencies in sequences of various lengths and complexities while improving both accuracy and computational efficiency.

Hyperparameter Tuning:

(1).

Number of layers : 6
Number Heads : 8
Embedding Dimension : 512
Hidden Dimension : 2048
Dropout : 0.1
Batch_size : 32

Train Loss : 4.8887
Val Loss : 4.8593
Test Loss : 4.8382

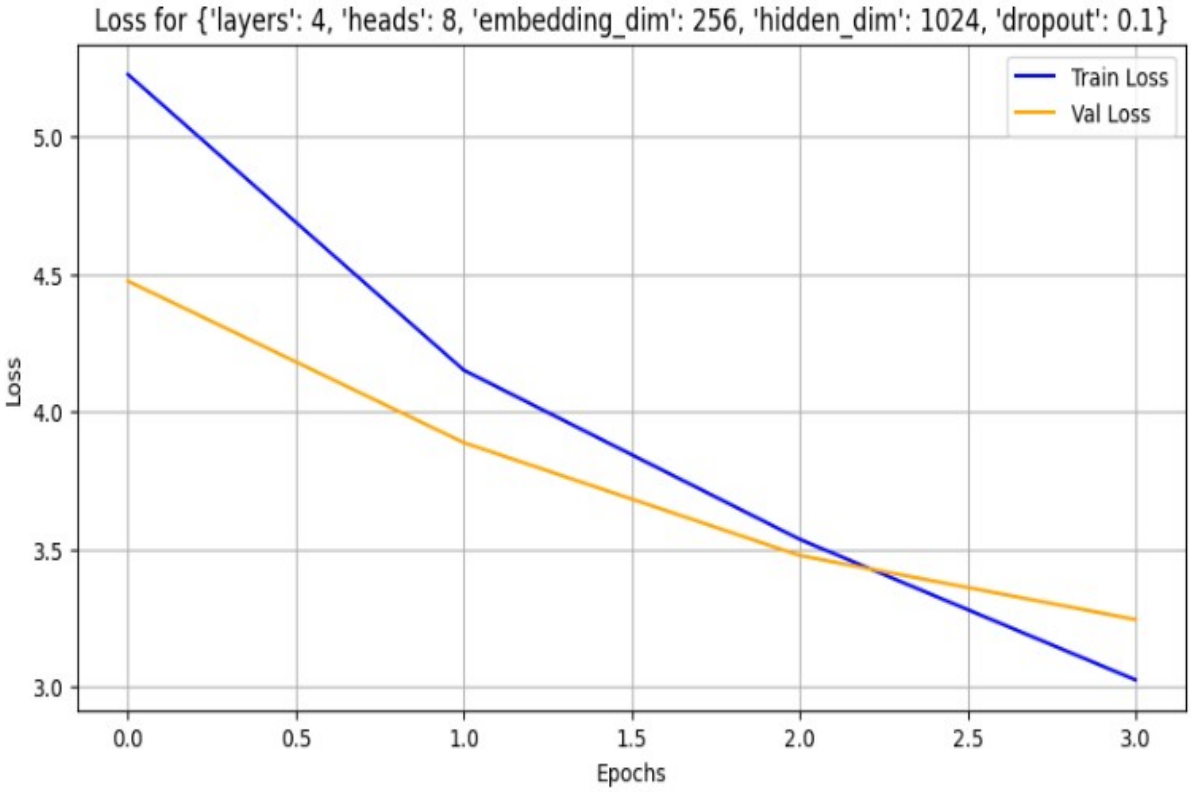
Test Bleu Score : 0.167805

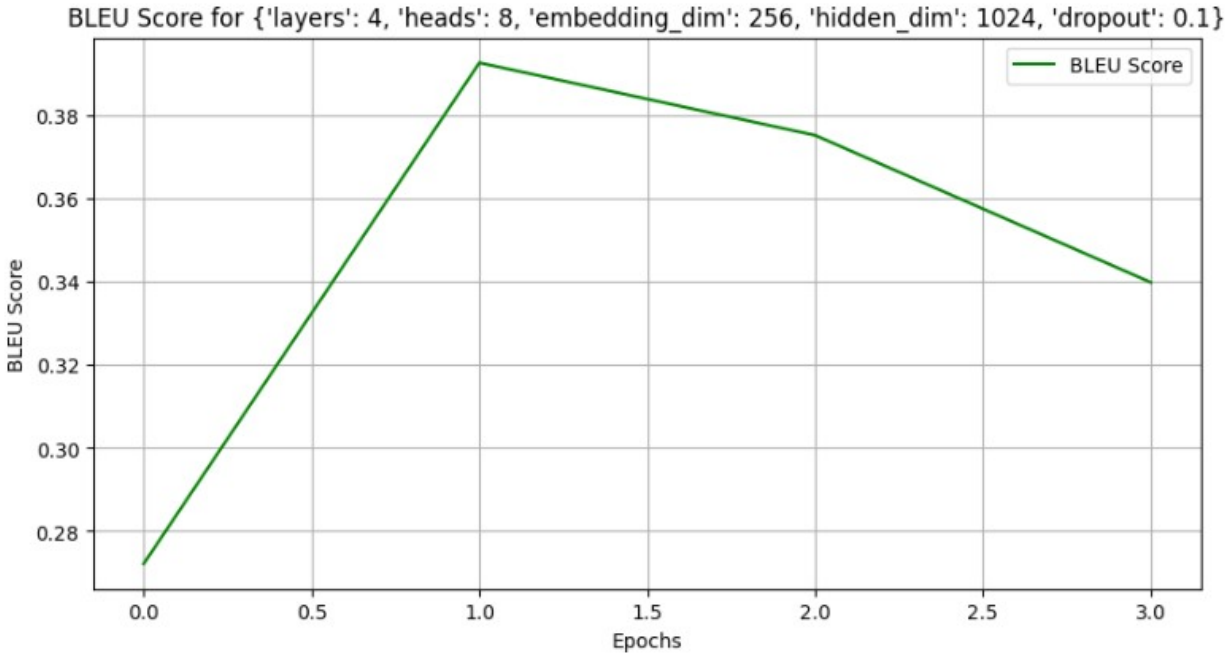
(2).

Number of layers : 4
Number Heads : 8
Embedding Dimension : 256
Hidden Dimension : 1024
Dropout : 0.1
Batch_size : 32

Train Loss : 3.0252
Val Loss : 3.2448
Test Loss : 3.1863

Test Bleu Score : 0.3407

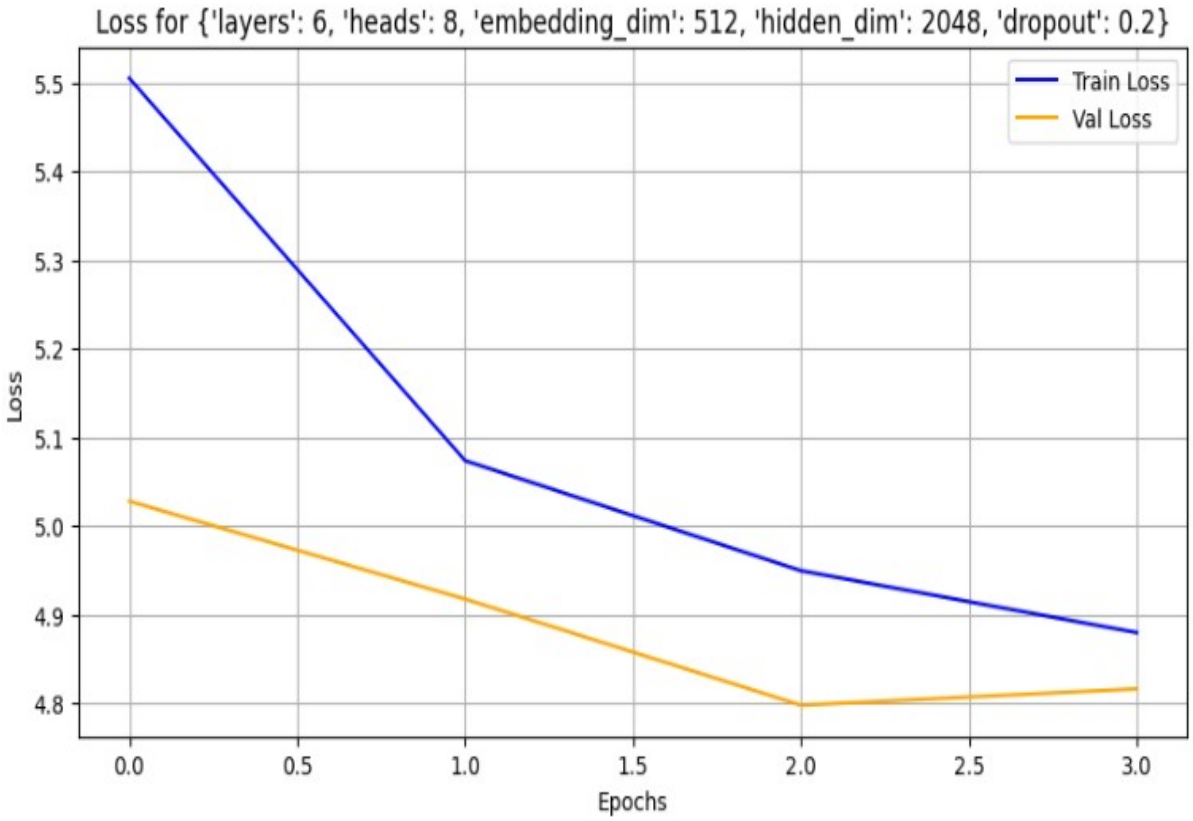

Loss for {'layers': 4, 'heads': 8, 'embedding_dim': 256, 'hidden_dim': 1024, 'dropout': 0.1}
Train Loss
Val Loss
Loss
5.0
4.5
4.0
3.5
3.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Epochs

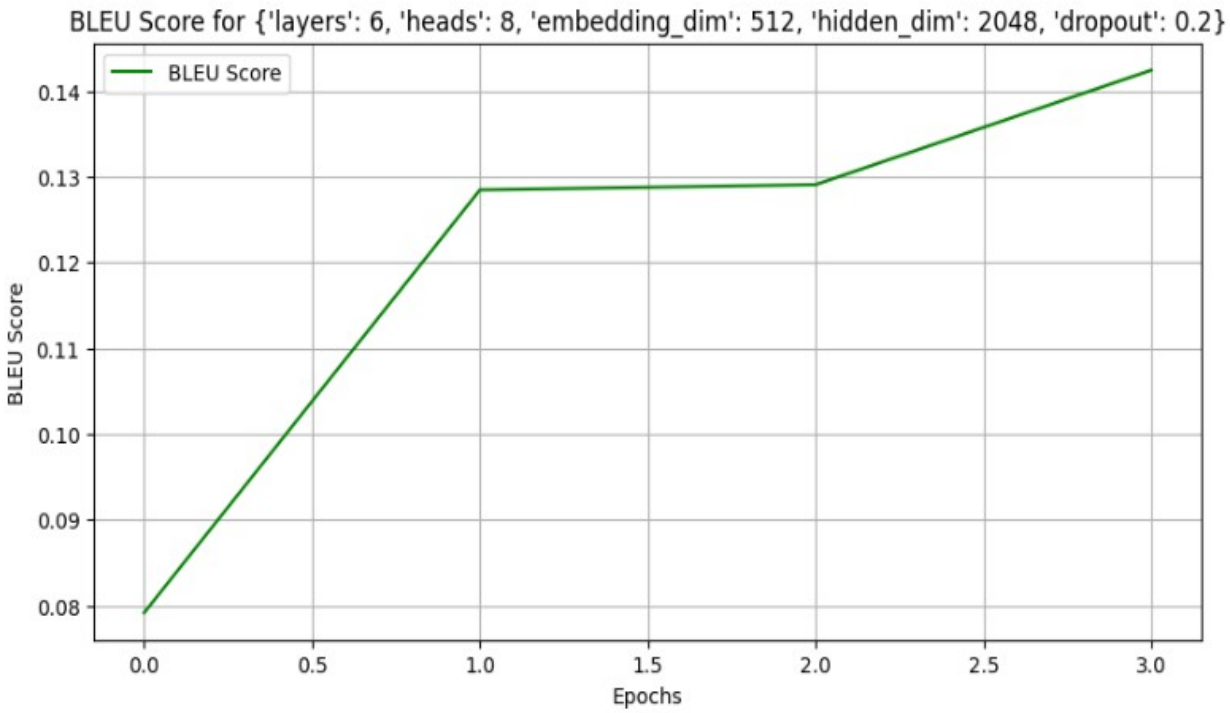

BLEU Score for {'layers': 4, 'heads': 8, 'embedding_dim': 256, 'hidden_dim': 1024, 'dropout': 0.1}
BLEU Score
BLEU Score
0.38
0.36
0.34
0.32
0.30
0.28
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Epochs

(3).
Number of layers : 6
Number Heads : 8
Embedding Dimension : 512
Hidden Dimension : 2048
Dropout : 0.2
Batch_size : 32

Train Loss : 4.8795
Val Loss : 4.8162
Test Loss : 4.7964

Test Bleu Score : 0.1576



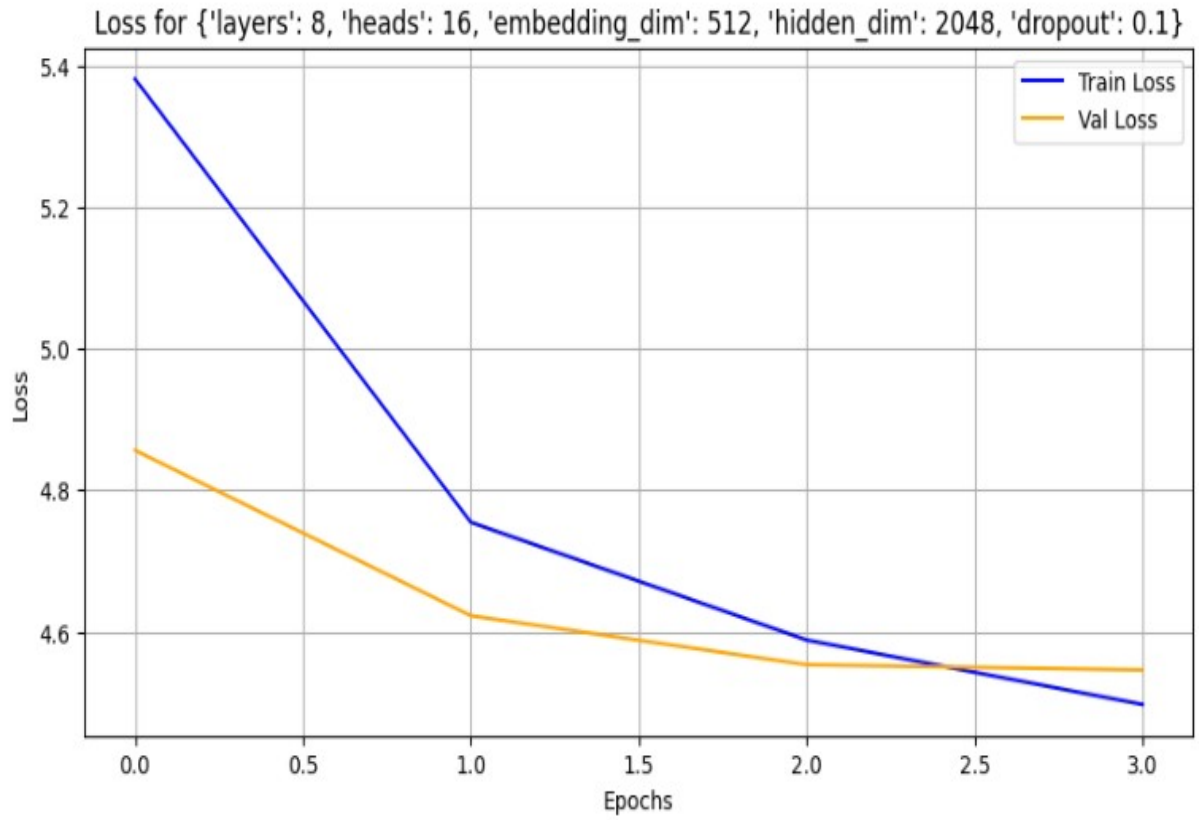


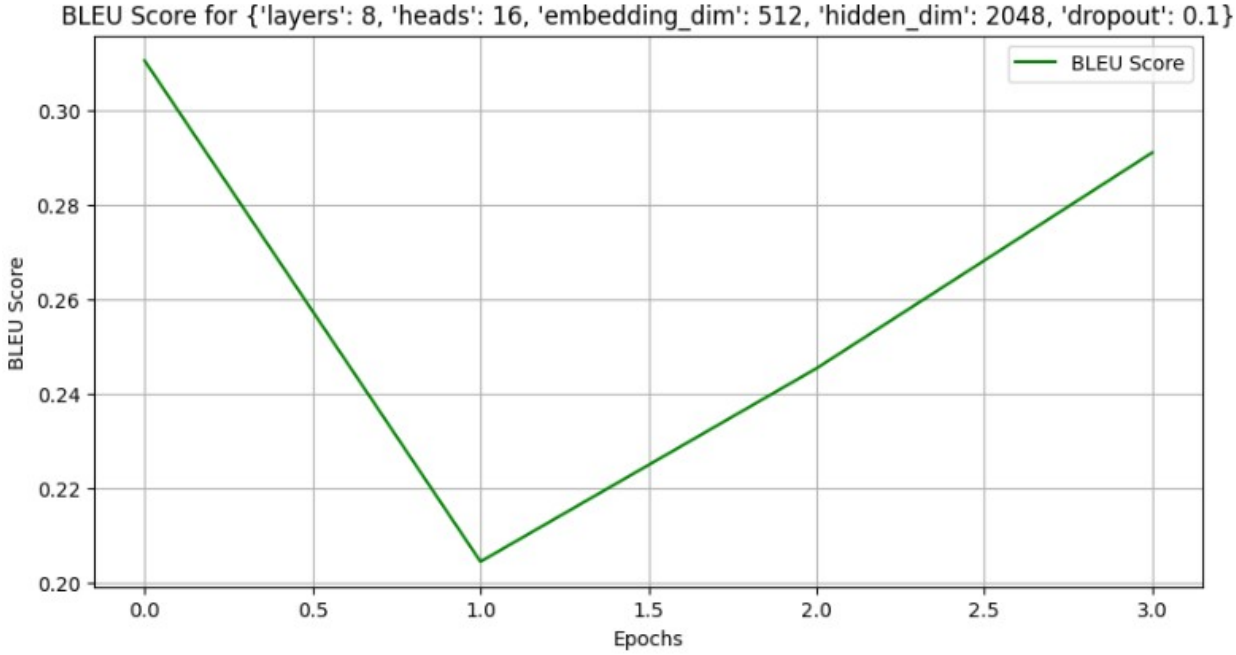
(4).

Number of layers : 8
Number Heads : 16
Embedding Dimension : 512
Hidden Dimension : 2048
Dropout : 0.1
Batch_size : 32

Train Loss : 4.4973
Val Loss : 4.5461
Test Loss : 4.5244

Test Bleu Score : 0.2911


Loss for {'layers': 8, 'heads': 16, 'embedding_dim': 512, 'hidden_dim': 2048, 'dropout': 0.1}
Train Loss
Val Loss
Loss
Epochs
5.4
5.2
5.0
4.8
4.6
0.0
0.5
1.0
1.5
2.0
2.5
3.0

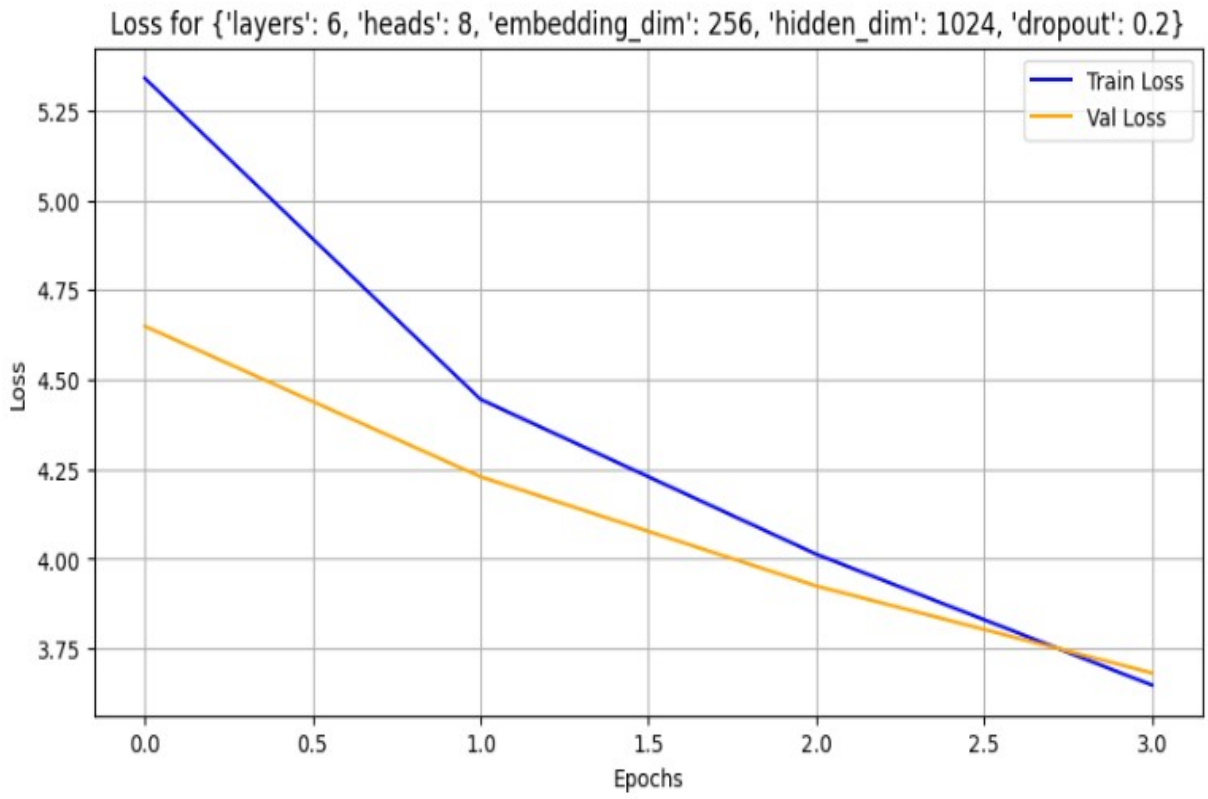

BLEU Score for {'layers': 8, 'heads': 16, 'embedding_dim': 512, 'hidden_dim': 2048, 'dropout': 0.1}
BLEU Score
BLEU Score
Epochs
0.30
0.28
0.26
0.24
0.22
0.20
0.0
0.5
1.0
1.5
2.0
2.5
3.0

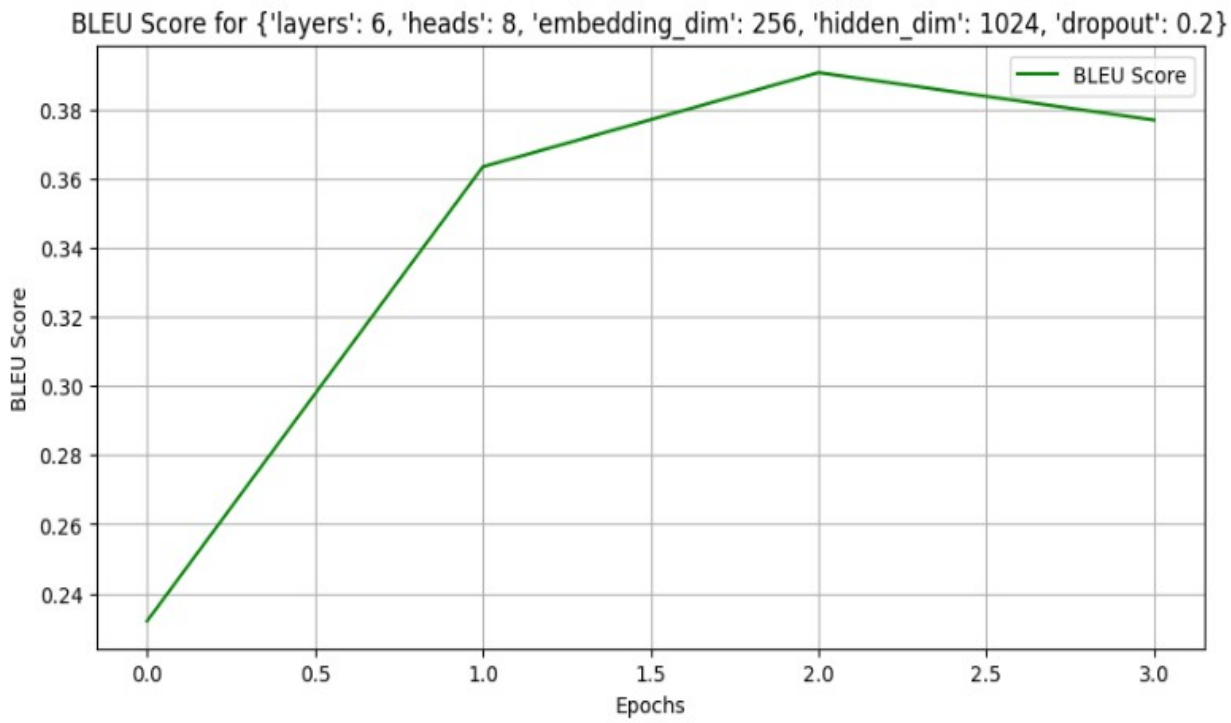
(5).

Number of layers : 6
Number Heads : 8
Embedding Dimension : 256
Hidden Dimension : 1024
Dropout : 0.2
Batch_size : 32

Train Loss : 4.4973
Val Loss : 4.5461
Test Loss : 4.5244

Test Bleu Score : 0.2911





Analysis :

- Reducing the number of layers and the embedding/hidden dimensions dramatically improved the BLEU score. The lower BLEU score with 6 layers and a higher hidden dimension suggests overparameterization, which may lead to overfitting and poorer generalization on unseen data.

- Losses also showed slight improvement with higher dropout, but the BLEU score dropped, indicating that increasing regularization through dropout reduces model's expressiveness.

- A **0.1 dropout** provides better performance, while a **0.2 dropout** may add too much regularization, reducing model accuracy in producing correct translations.

- More layers do not always lead to better performance, as adding layers may make the model harder to train and prone to overfitting. A model with fewer layers (like 4) provides a better trade-off between capacity and generalization.

- For this machine translation task, a smaller model (embedding = 256, hidden = 1024) is more effective, likely because it learns more generalized patterns without overfitting.

- More attention heads (beyond 8) do not significantly impact performance in this case, indicating that adding more heads may add computational complexity without clear performance benefits.

- The configuration with 4 layers, 8 heads, 256 embedding, and 1024 hidden dimensions outperformed the more complex setups. Simpler models showed better BLEU scores and lower losses, indicating better generalization.