CS7.501: Advanced Natural Language Processing
Assignment Report

Chetan Mahipal

Course Instructor - Dr. Manish Shrivastava

IIIT Hyderabad - October 2, 2024

1 Theory Question

1.1 Question-1

Attention is a mechanism that allows neural networks to focus on relevant parts of input data. It dynamically weights the importance of different elements, enabling the model to capture long-range dependencies and context-sensitive information. It is calculated in the following way:

- **Q (Query):** Q = Represents the current word/token for which we're calculating attention.
- **K (Key):** K = Represents all words/tokens used to match against the query.
- **V (Value):** V = Represents the actual content used to compute the attention output.

The formula for attention is:

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

Self Attention, as the name suggests it is attention being performed between tokens of the same sequence. This forms the essential part of any Language Modeling, Se2qSeq etc. It helps to capture dependencies of one word on other words of the sequence. For example, the word **bank** has more dependence on word **river** or **Transaction**, irrespective of their appearance in the sequence.
The purpose of self-attention can be broadly listed as below:

1. **Contextual Understanding**: Consider the sentence:*"The apple fell from the tree and hit the ground."*

   The word *"apple"* and *"ground"* are important to understanding what hit the ground. Without self-attention, a model might focus only on local context, like the verb *"hit"* and the noun *"ground"*.

   The model can assign higher attention to other selective tokens when processing the token in discussion, which helps in forming the understanding between the two and, improving contextual comprehension.

2. **Long Range Dependencies**: Consider the sentence: *"The seminar that was held last Friday on quantum computing was extremely insightful."*

The noun *"seminar"* and the adjective *"insightful"* are far apart in this sentence. Traditional models, such as RNNs, might struggle to capture the dependency between *"seminar"* and *"insightful"* due to the long distance between them.

When processing the tokens, self-attention can assign a higher attention weight to the tokens far from them, allowing the model to understand the connection. This solves the problem of **Vanishing Gradient**, which is that the gradient of start tokens become zero as we approach the end in long sequences, thus losing the context.

The above purpose is achieved by employing following techniques while devising State of the Art Architecture models:

1. **Creating direct connections:** Self-attention allows each element in a sequence to directly interact with every other element, regardless of their distance in the sequence. This bypasses the limitations of recurrent or convolutional architectures that struggle with long-range dependencies.

2. **Adaptive weighting:** Self-attention dynamically computes attention weights for each pair of elements. These weights determine how much each element should focus on other elements when updating its representation. This allows the model to adaptively emphasize relevant connections and de-emphasize irrelevant ones.

3. **Parallel processing:** Unlike recurrent models, self-attention can process all elements of a sequence in parallel, making it more efficient for capturing both short-term and long-term dependencies simultaneously.

4. **Position-aware representations:** Through positional encodings, self-attention can incorporate sequence order information, allowing it to capture both content-based and position-based dependencies.

5. **Multi-head mechanism:** By using multiple attention heads, self-attention can capture different types of dependencies or relationships within the same sequence, providing a richer representation of the data.

1.2 Question-2

1.2.1 Introduction

Transformers have revolutionized natural language processing and other sequence-based tasks. A key component of their architecture is the use of positional encodings alongside word embeddings. Below headings explores the necessity of positional encodings, their incorporation into the transformer architecture, and recent advances in this area.

1.2.2 Necessity of Positional Encodings

Transformers use self-attention mechanisms to process input sequences in parallel, which provides significant computational advantages. However, this parallelization comes at a cost: the model loses information about the relative or absolute positions of tokens in the sequence. Positional encodings address this limitation by allocating position information into the input embeddings.

The reasons for using positional encodings include:

1. **Preserving sequence order**: Unlike recurrent neural networks (RNNs), transformers do not process tokens sequentially. Positional encodings allow the model to differentiate between tokens based on their position in the sequence.

2. **Enabling attention to consider position**: The self-attention mechanism can utilize position information when computing attention weights, allowing it to capture position-dependent patterns.

3. **Maintaining translation invariance**: Properly designed positional encodings allow the model to generalize to sequences of different lengths and maintain translation invariance in the learned representations.

1.2.3 Incorporation into Transformer Architecture

Positional encodings are incorporated into the transformer architecture as follows:

1. Generation: Positional encodings are generated for each position in the input sequence. The encoding is typically a vector of the same dimensionality as the word embeddings.

2. Addition: The positional encoding vector is element-wise added to the corresponding word embedding vector:

$$\text{Input} = \text{WordEmbedding} + \text{PositionalEncoding} \tag{1}$$

3. Propagation: The combined input (word embedding + positional encoding) is then fed into the transformer's encoder and decoder stacks. The position information is preserved and utilized throughout the network's layers.

1.2.4 Traditional Sinusoidal Positional Encodings

The original transformer paper proposed sinusoidal positional encodings, defined as:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \tag{2}$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}) \tag{3}$$

Where pos is the position, i is the dimension, and d_{model} is the embedding dimensionality. These encodings have several advantageous properties:

- They can handle variable-length sequences.
- They allow the model to easily attend to relative positions.
- They can extrapolate to sequence lengths not seen during training.

1.2.5 Recent Advances in Positional Encodings

Several alternative approaches to positional encodings have been proposed in recent years:

1. Relative Positional Encodings: Introduced by Shaw et al. (2018), these encodings represent relative distances between tokens rather than absolute positions. They can be particularly useful for tasks where relative positions are more important than absolute ones.

2. Rotary Position Embeddings (RoPE): Proposed by Su et al. (2021), RoPE applies a rotation matrix to the token embeddings based on their position. This approach unifies absolute and relative position encoding and has shown improved performance in various tasks.

3. ALiBi (Attention with Linear Biases): Press et al. (2021) introduced ALiBi, which adds a

linear bias term to the attention scores based on the distance between tokens. This method has shown strong performance, especially in extrapolating to longer sequences.

1.2.6 Comparison with Traditional Sinusoidal Encodings

Recent advances in positional encodings differ from traditional sinusoidal encodings in several ways:

- **Relativity**: Several recent methods focus on encoding relative positions rather than absolute positions, which can be beneficial for certain tasks and architectures.
- **Integration**: Some newer methods (e.g., RoPE, ALiBi) integrate position information more deeply into the attention mechanism, rather than simply adding it to input embeddings.
- **Extrapolation**: While sinusoidal encodings are known for their ability to extrapolate to unseen sequence lengths, some newer methods (e.g., ALiBi) have shown even better extrapolation capabilities.
- **Computational Efficiency**: Some recent approaches (e.g., ALiBi) can be more computationally efficient than traditional sinusoidal encodings, especially for long sequences.

2 HyperParameter Tuning

We test the performance of are model and try to optimize it by finding the better combination of Hyperparameters. The hyperparameters we varied are:

- Number of Layers
- Number of attention heads
- Dimensions of word Embeddings
- Dropout
- Hidden Dimension of Feed Forward Network

The following combinations have bee used after many experiments.

2.1 Combination-1

HyperParameters used are:

- **Number of Layers of Encoder/Decoder**: 4
- **Number of Attention Head**: 4
- **Embedding Dimension**: 100
- **Hidden Dimension**: 300
- **Dropout**: 0.1

The **Bleu Score** for above combination of Parameters is: 0.222. The training loss graph obtained on running 15 epochs is displayed in Fig 1

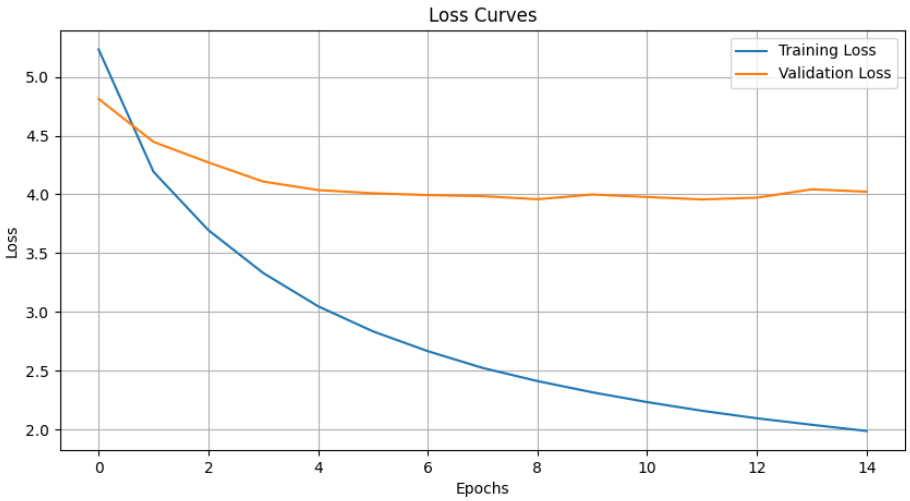

Figure 1: *Training Loss Graph*

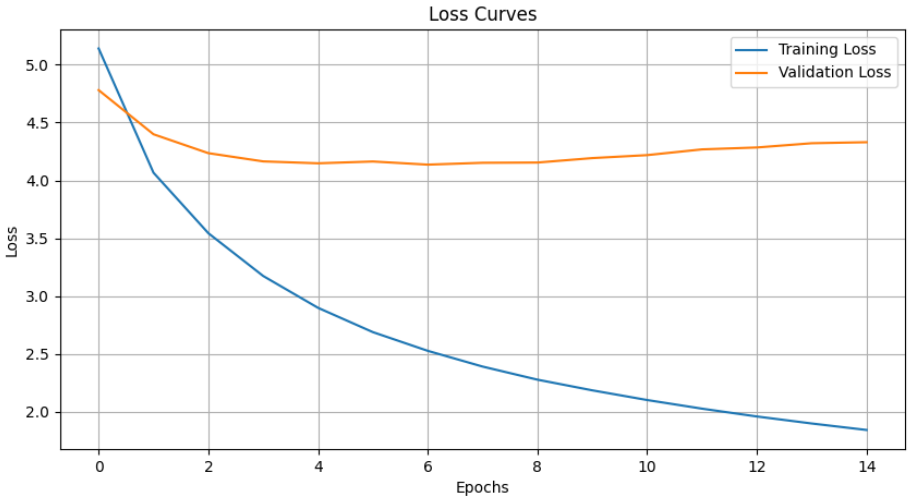

Figure 2: *Training Loss Graph*

2.2 Combination-2

HyperParameters used are:

- **Number of Layers of Encoder/Decoder**: 2
- **Number of Attention Head**: 4
- **Embedding Dimension**: 128
- **Hidden Dimension**: 256
- **Dropout**: 0.1

The **Bleu Score** for above combination of Parameters is: 0.188. The training loss graph obtained on running 15 epochs is displayed in Fig 2

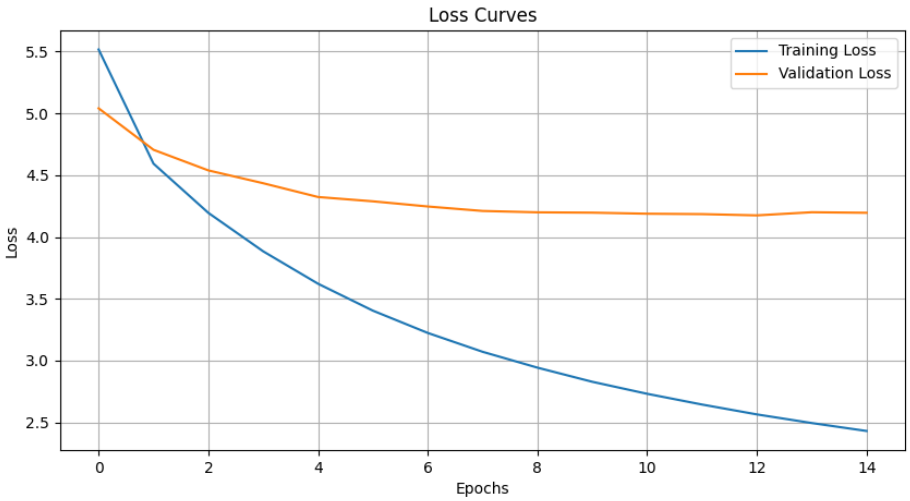

Figure 3: *Training Loss Graph*

2.3 Combination-3

HyperParameters used are:

- **Number of Layers of Encoder/Decoder**: 4
- **Number of Attention Head**: 5
- **Embedding Dimension**: 100
- **Hidden Dimension**: 300
- **Dropout**: 0.3

The **Bleu Score** for above combination of Parameters is: 0.213. The training loss graph obtained on running 15 epochs is displayed in Fig 3

2.4 Combination-4

HyperParameters used are:

- **Number of Layers of Encoder/Decoder**: 4
- **Number of Attention Head**: 5
- **Embedding Dimension**: 50
- **Hidden Dimension**: 300
- **Dropout**: 0.1

The **Bleu Score** for above combination of Parameters is: 0.175. The training loss graph obtained on running 15 epochs is displayed in Fig 4

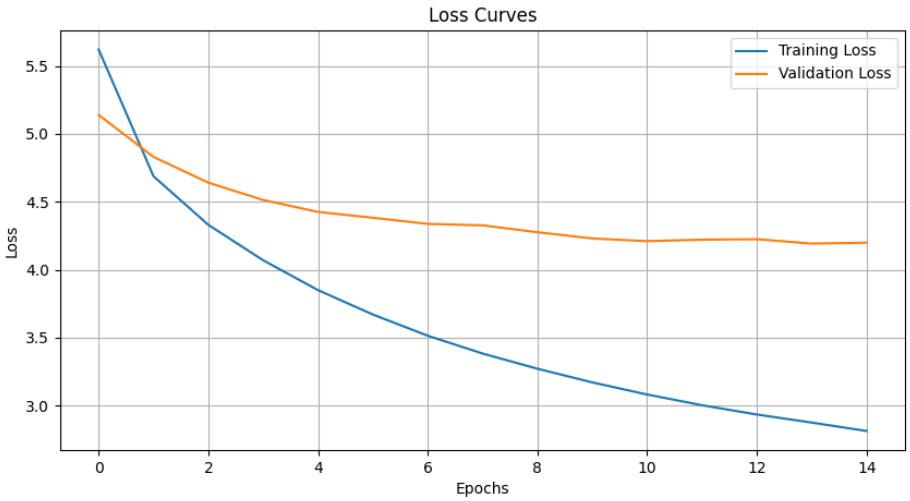

Figure 4: *Training Loss Graph*

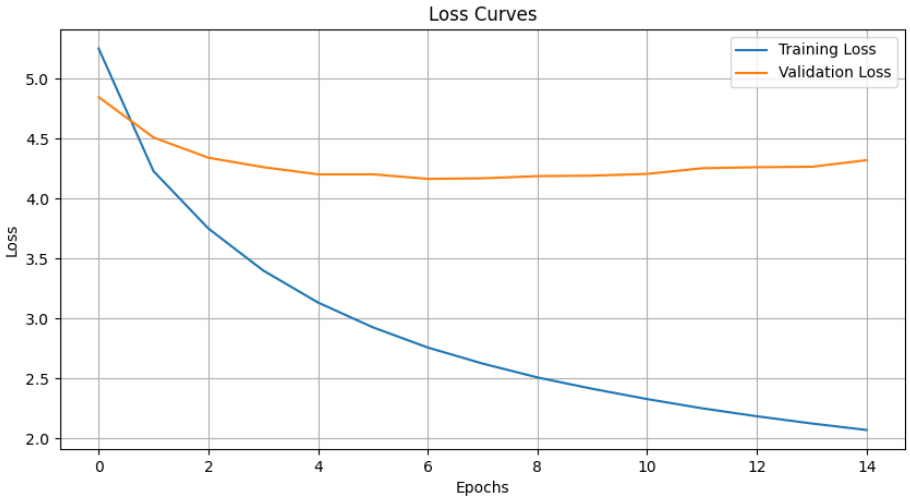

Figure 5: *Training Loss Graph*

2.5 Combination-5

HyperParameters used are:

- **Number of Layers of Encoder/Decoder**: 2
- **Number of Attention Head**: 4
- **Embedding Dimension**: 100
- **Hidden Dimension**: 300
- **Dropout**: 0.1

The **Bleu Score** for above combination of Parameters is: 0.0.226. The training loss graph obtained on running 15 epochs is displayed in Fig 5

Note : For evaluation purpose all the pre-trained models with python notebooks have been saved under HyperParamter_Tuning folder for reference and re-creation of results.

2.6 Analysis

On analyzing the above combinations and some other combinations in experiment phase following are the conclusions drawn from it:

1. As the training dataset is less, on increasing the hidden dimension of Feed Forward Layer, the model doesn't not learn anything significant and starts returning Special tokens.

2. As the validation set is much less compared to training, after certain epochs the validation set loss converges while training loss still reduces.

3. Increasing or decreasing the embedding dimension after a certain value leads to less Bleu score, demonstrated with size 50 and 300 in above combinations.

4. The dropout rate might effect when we have huge dimensions for our model, but in this case as we kept dimension to near about 100 and other layers not that much, the ideal dropout is 0.1. We can increase dropout for combination having 300 embedding dimension, 6 head and 4 layers to 0.3.

5. Increasing number of heads helps in gaining more context and relation between words.

6. Increasing number of layers with less dropout leads to over-fitting and model performing bad on translation. This problem can solved by using Beam search etc., for translation.

7. Given the dataset was small for translation task the best achieved Bleu score while experimenting was 0.25 and above results have 0.225. This may be increased by changing techniques of predicting translation using Top p% prob or beam search.

Note: All the losses having been calculated by ignoring $< pad >$ token for better training and understanding.