ANLP Assignment 3 Report

Yash Bhaskar

October 16, 2023

Abstract

The transformer architecture has revolutionized the field of natural language processing (NLP) and machine translation. Central to its success are two key components: self-attention and positional encodings. Self-attention allows the model to capture complex dependencies in sequences, enabling it to understand context and relationships between words. Positional encodings, on the other hand, provide essential information about word order within a sequence. This report delves into the purpose and mechanics of self-attention, demonstrating how it facilitates the capture of dependencies. Additionally, it explains why positional encodings are crucial and details how they are seamlessly integrated into the transformer architecture. Together, these elements empower transformers to excel in a wide range of NLP tasks, making them a cornerstone in the development of cutting-edge language models.

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

Self-attention is a critical component of the transformer architecture and plays a central role in capturing dependencies in sequences, making it a fundamental building block for various natural language processing (NLP) tasks, including machine translation. In this detailed explanation, we'll explore the purpose of self-attention and how it facilitates the capture of dependencies in sequences.

Purpose of Self-Attention:

- **Contextual Understanding:** Self-attention is designed to create context-aware representations of input sequences. It allows the model to assign varying degrees of importance to different parts of the input, enabling it to understand the context and relationships between words or tokens in the sequence.
- **Dynamic Weighting:** Self-attention dynamically weighs the relevance of each word or token in the input to the current word being processed.

This dynamic weighting is crucial for understanding dependencies in the sequence, as it adapts to the specific context of the input.

- **Long-Range Dependencies:** Self-attention enables the model to capture long-range dependencies, which is essential for tasks like machine translation. It does not rely on fixed window sizes or limited context, making it highly effective at recognizing connections between words that are far apart in the sequence.

- **Parallel Processing:** Self-attention operates in parallel across all words in the sequence, making it computationally efficient and suitable for hardware acceleration, which is essential for training large-scale models.

How Self-Attention Facilitates Capturing Dependencies:

Self-attention works by comparing each word or token to all other words or tokens in the sequence, and this comparison results in a set of attention scores. These attention scores determine the importance of each word with respect to the current word being processed. Let's break down the process step by step:

1. **Transforming Input Embeddings:** Initially, the input sequence is represented as word embeddings. Each word or token is associated with an embedding vector. In machine translation, these embeddings may represent words in the source language that need to be translated into the target language.

2. **Calculating Queries, Keys, and Values:** The embeddings are linearly transformed into three sets of vectors: queries (Q), keys (K), and values (V). These transformations are performed using learnable weight matrices. The purpose of these transformations is to create distinct representations for Q, K, and V.

3. **Dot Products and Attention Scores:** Self-attention is computed by taking dot products between the queries and keys. Each word's query is compared to all keys, resulting in a set of attention scores that represent the relationships between words in the sequence. The dot products are divided by the square root of the dimension of the keys to ensure that the gradients remain stable during training.

4. **Applying Softmax Function:** The attention scores are passed through a softmax function. This function normalizes the scores, ensuring that they sum to 1 for each query. Softmax generates a distribution over the keys, emphasizing words with higher attention scores.

5. **Weighted Sum of Values:** The normalized attention scores are used to weight the values. A weighted sum of the values is computed, giving higher importance to words that are contextually relevant to the current word.

6. **Output of Self-Attention:** The output of the self-attention mechanism is a weighted combination of the values, representing the context-aware representation of the current word. This representation is crucial for capturing dependencies, as it incorporates information from all words in the sequence, with varying degrees of importance.

Multi-Head Self-Attention: In transformer models, self-attention is often implemented with multiple attention heads. Each head provides a different perspective on the relationships between words in the input sequence. These multiple heads are then concatenated or linearly combined to capture various types of dependencies effectively.

Positional Encoding: To account for the word's position in the sequence, positional encodings are added to the input embeddings. This is vital for understanding the sequential nature of the input, as self-attention itself does not inherently encode positional information.

In summary, self-attention allows the model to consider the importance of each word or token in the sequence, providing a context-aware representation of the input. This dynamic and context-sensitive approach to modeling dependencies is highly effective in capturing long-range relationships and contextual information, making it a key element in the success of transformer-based models for tasks like machine translation.

Theory Question 2 : Why do transformers use positional encodings in addition to word embeddings? Explain how positional encodings are incorporated into the transformer architecture.

Transformers use positional encodings in addition to word embeddings because they lack inherent knowledge of word order and position within a sequence. The transformer architecture relies on self-attention mechanisms, which are permutation invariant. In other words, they can process elements of a sequence in any order, which makes them efficient but requires additional information about the position of words in a sentence. Positional encodings provide this crucial information about word order, enabling the model to understand the sequential nature of the input data.

To incorporate positional encodings into the transformer architecture, let's explore the detailed process:

1. **Word Embeddings:** The first step in the transformer architecture is to represent the input sequence (e.g., a sentence) as a set of word embeddings. Each word in the sequence is mapped to a fixed-size vector, which is trainable. These embeddings capture the semantic meaning of words but do not contain information about their positions in the sentence.

2. **Positional Encodings:** Positional encodings are additional vectors added to the word embeddings. They provide information about the position of words within the sequence. These encodings are designed to be unique for each position in the sequence and are added to the word embeddings to create a combined representation that includes both the word's meaning and its position.

3. **Why Positional Encodings:** The transformer model uses self-attention mechanisms to process the input sequence. Without positional encodings, the model would treat words as an unordered set, ignoring their positions. This could lead to a loss of information about word order and hinder the model's ability to capture sequential patterns, which are vital for tasks like machine translation.

4. **Encoding Positions:** Positional encodings are calculated for each position in the input sequence, and the dimensionality of these encodings matches that of the word embeddings. A common approach used to calculate these encodings involves the use of trigonometric functions, specifically sine and cosine functions:

5. **Sine and Cosine Functions:** For each position and dimension, a unique positional encoding value is computed using the sine and cosine functions. The position value determines the frequency and phase of the functions for each dimension, creating a distinct pattern for each position. This results in a set of vectors that can be added to the word embeddings.

6. **Adding Positional Encodings:** The positional encodings are element-wise added to the word embeddings. This step combines the information about the word's meaning and its position, creating embeddings that carry both semantic and positional information.

7. **Positional Encoding Reusability:** It's essential to note that positional encodings are computed once and not updated during the training process. They are fixed and reused for all input sequences during both training and inference. This approach ensures consistency in how positions are represented within the model.

8. **Interaction with Attention Mechanism:** The addition of positional encodings is a crucial part of the transformer architecture's input preparation. When the model processes the sequence with self-attention mechanisms, it uses both the word embeddings and the positional encodings to understand the relationships between words. The attention mechanism assigns different weights to words based on their importance for the current context, considering not only the meaning of the words but also their positions.

9. **Capturing Sequence Information:** Positional encodings help the model capture sequential information. By providing a unique pattern for each

position, they enable the model to differentiate between words based on their order in the sequence. This is particularly important for tasks like machine translation, where the order of words can drastically affect the meaning of a sentence.

10. **Training and Inference:** During the training process, the model learns to adjust the word embeddings to capture the semantics of words, while the positional encodings remain fixed. The model optimizes its parameters to perform well on various tasks, including translation. During inference, the same positional encodings are used to process new input sequences, ensuring consistency.

In summary, transformers use positional encodings in addition to word embeddings to enable the model to understand the order and position of words within a sequence. Positional encodings are created using sine and cosine functions to generate unique patterns for each position, and they are element-wise added to word embeddings. This combined representation is crucial for the transformer's self-attention mechanisms to capture sequential patterns, making transformers highly effective for tasks like machine translation and other natural language processing tasks. The fixed nature of positional encodings ensures consistency between training and inference.

Hyperparameter Tuning

Configuration with $h = 4$

	lr	batch_size	dropout	h	N	epoch	d_model	d_ff
ct1	0.01	32	0.1	4	2	10	512	2048
ct2	0.01	32	0.3	4	2	10	512	2048
ct3	0.01	64	0.1	4	2	10	512	2048
ct4	0.01	64	0.3	4	2	10	512	2048
ct5	0.001	32	0.1	4	2	10	512	2048
ct6	0.001	32	0.3	4	2	10	512	2048
ct7	0.001	64	0.1	4	2	10	512	2048
ct8	0.001	64	0.3	4	2	10	512	2048
ct9	0.0001	32	0.1	4	2	10	512	2048
ct10	0.0001	32	0.3	4	2	10	512	2048
ct11	0.0001	64	0.1	4	2	10	512	2048
ct12	0.0001	64	0.3	4	2	10	512	2048

Table 1: Hyperparameter Configurations with $h = 4$

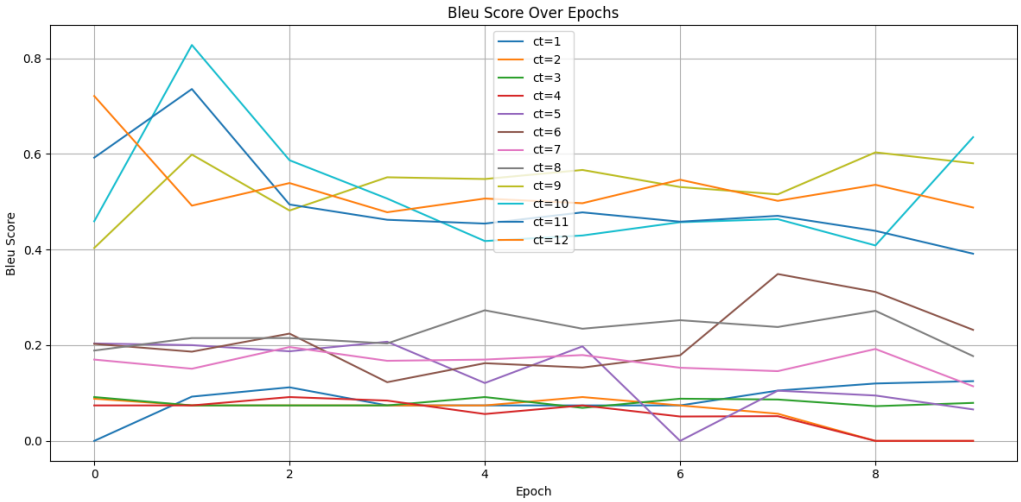

Figure 1: Bleu Score on Dev Set Over Epoch

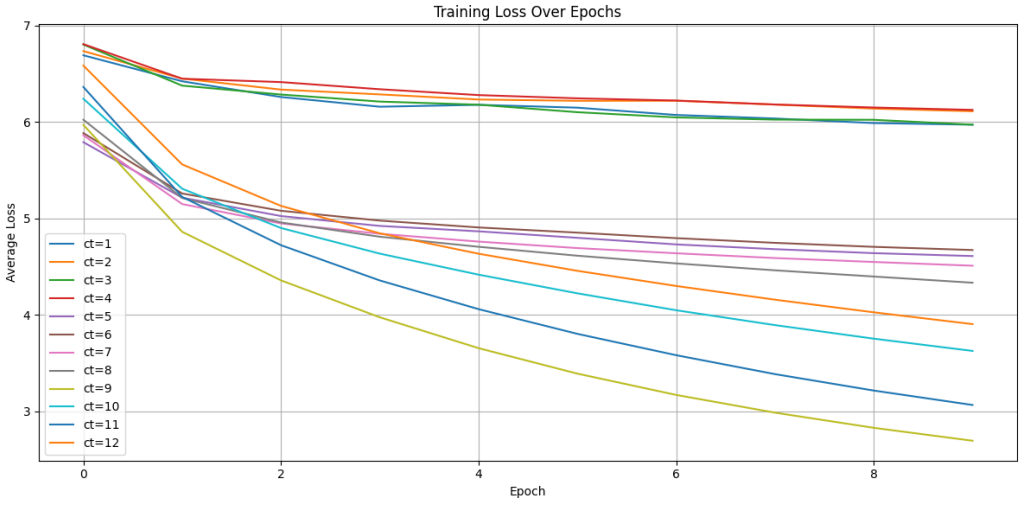

Figure 2: Training Loss Over Epochs

Configuration with $h = 8$

	lr	batch_size	dropout	h	N	epoch	d_model	d_ff
ct1	0.01	32	0.1	8	2	10	512	2048
ct2	0.01	32	0.3	8	2	10	512	2048
ct3	0.01	64	0.1	8	2	10	512	2048
ct4	0.01	64	0.3	8	2	10	512	2048
ct5	0.001	32	0.1	8	2	10	512	2048
ct6	0.001	32	0.3	8	2	10	512	2048
ct7	0.001	64	0.1	8	2	10	512	2048
ct8	0.001	64	0.3	8	2	10	512	2048
ct9	0.0001	32	0.1	8	2	10	512	2048
ct10	0.0001	32	0.3	8	2	10	512	2048
ct11	0.0001	64	0.1	8	2	10	512	2048
ct12	0.0001	64	0.3	8	2	10	512	2048

Table 2: Hyperparameter Configurations with $h = 8$

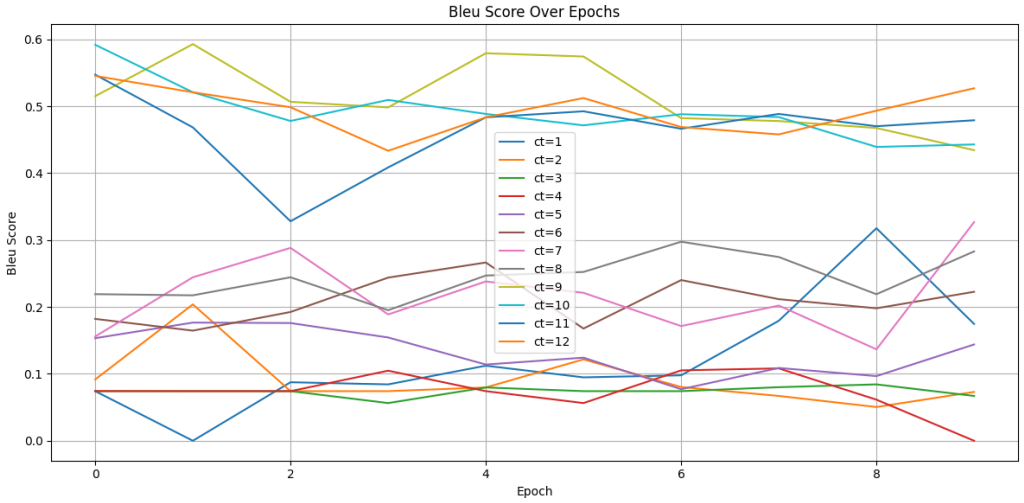

Figure 3: Bleu Score on Dev Set Over Epoch

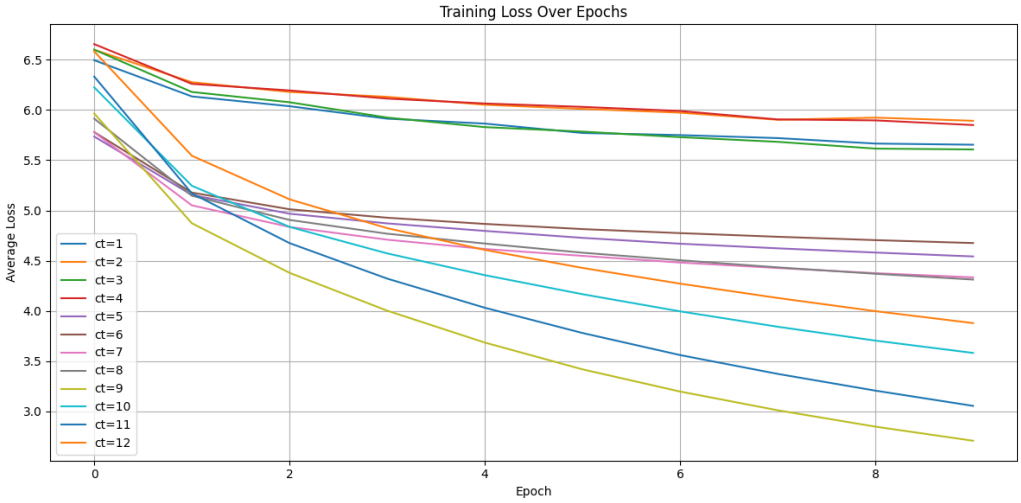

Figure 4: Training Loss Over Epochs

Configuration with $h = 16$

	lr	batch_size	dropout	h	N	epoch	d_model	d_ff
ct1	0.01	32	0.1	16	2	10	512	2048
ct2	0.01	32	0.3	16	2	10	512	2048
ct3	0.01	64	0.1	16	2	10	512	2048
ct4	0.01	64	0.3	16	2	10	512	2048
ct5	0.001	32	0.1	16	2	10	512	2048
ct6	0.001	32	0.3	16	2	10	512	2048
ct7	0.001	64	0.1	16	2	10	512	2048
ct8	0.001	64	0.3	16	2	10	512	2048
ct9	0.0001	32	0.1	16	2	10	512	2048
ct10	0.0001	32	0.3	16	2	10	512	2048
ct11	0.0001	64	0.1	16	2	10	512	2048
ct12	0.0001	64	0.3	16	2	10	512	2048

Table 3: Hyperparameter Configurations with $h = 16$

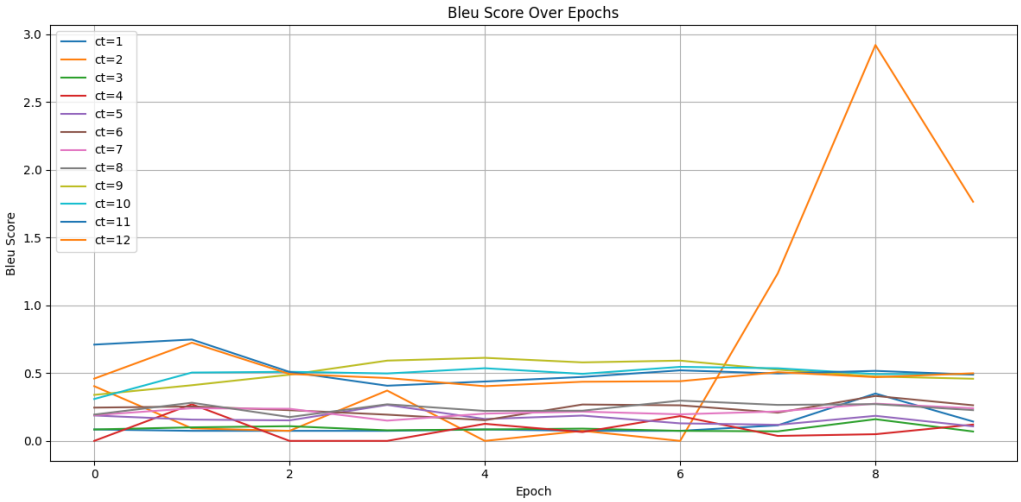

Figure 5: Bleu Score on Dev Set Over Epoch

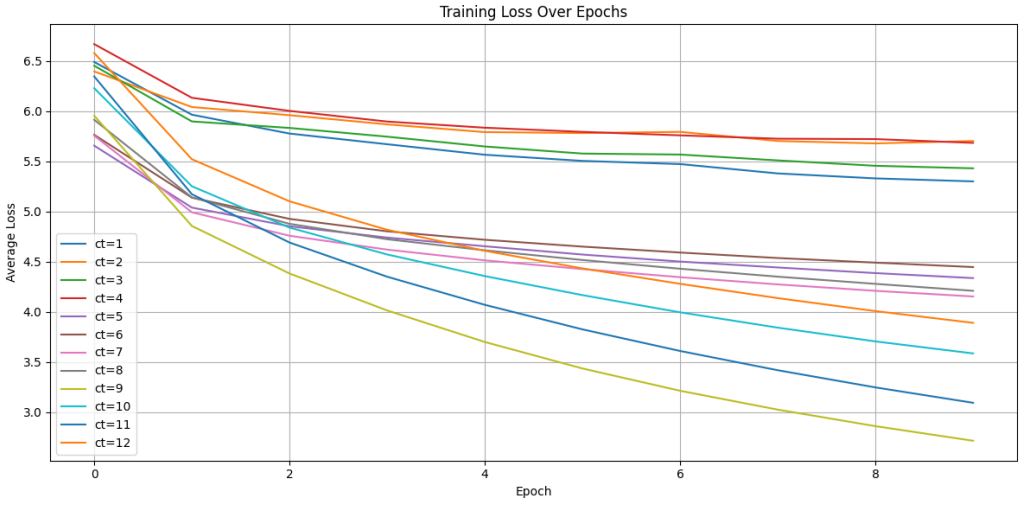

Figure 6: Training Loss Over Epochs

Analysis

Analyzing the performance of a transformer model for machine translation involves considering various factors, including hyperparameters, translation quality, and training progress. Evaluation of the model using the BLEU metric, across the chosen hyperparameters:

1. BLEU Metric Evaluation:

BLEU (Bilingual Evaluation Understudy) is a widely used metric for evaluating the quality of machine translation. It measures the similarity between the model's translations and reference translations provided in the dataset. Higher BLEU scores indicate better translation quality.

2. Key Hyperparameters:

The following key hyperparameters have been chosen for the model, and their significance is explained:

- **Learning Rate (lr):** The learning rate controls the step size during gradient descent. It determines how quickly the model adjusts its parameters based on training data. Different learning rates can affect the training speed and convergence. It's important to choose an appropriate learning rate to prevent underfitting (too low) or overshooting (too high).
- **Batch Size (batch_size):** The batch size determines the number of samples processed in each forward and backward pass during training. It impacts memory usage and training efficiency. Smaller batch sizes may lead to a more erratic training process but can help the model generalize better. Larger batch sizes can speed up training but may require more memory and can result in overfitting if not chosen carefully.
- **Dropout (dropout):** Dropout is a regularization technique that helps prevent overfitting. It randomly "drops out" a fraction of neurons during training, effectively introducing noise and making the model more robust. The dropout rate controls the probability of dropping out a neuron in each training step. A higher dropout rate may lead to more regularization but can also hinder convergence if set too high.
- **Number of Attention Heads (h):** The number of attention heads in the multi-head self-attention mechanism influences the model's ability to capture different dependencies and relationships in the input data. More heads can capture more fine-grained information, but they also require more computation.
- **Number of Layers (N):** The number of encoder and decoder layers affects the depth and capacity of the model. More layers can capture complex patterns but may require longer training and can lead to overfitting if not regularized properly.

- **Number of Epochs (epoch):** The number of training epochs determines how many times the model goes through the entire training dataset. Training for too few epochs may result in underfitting, while training for too many epochs can lead to overfitting.
- **Model Dimension (d_model):** The dimensionality of the model's embeddings and hidden states affects the model's capacity to capture information. A higher dimension allows the model to represent more complex relationships but requires more computation.
- **Feed-Forward Dimension (d_ff):** The dimension of the feed-forward layers in the model. A larger dimension can model more complex functions but may require more training data to prevent overfitting.

Loss Curves:

Loss curves are essential for understanding the training process. They provide insights into how the model is learning and whether it's converging. Typically, we monitor both training and validation loss. Training loss should decrease during training, indicating that the model is learning from the data. Validation loss helps us identify overfitting and whether the model generalizes well to unseen data.

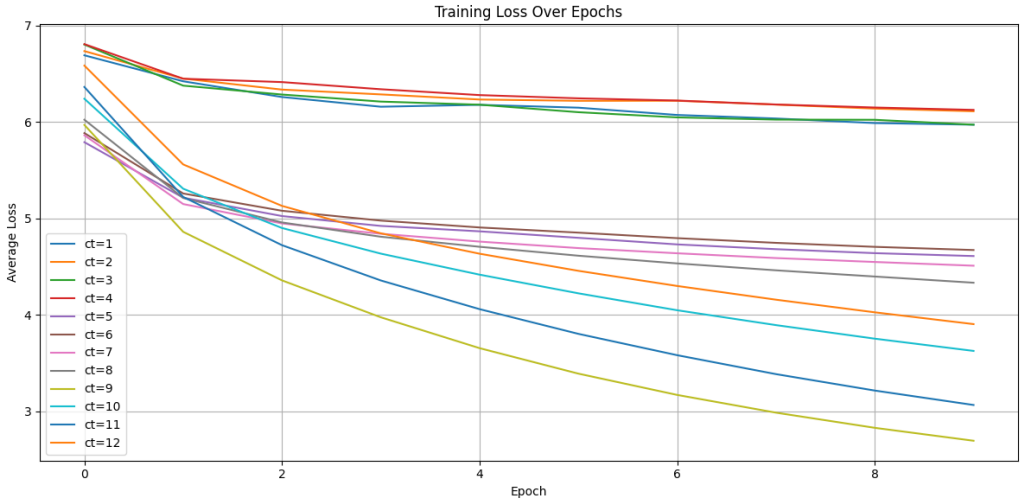

Figure 7: Training Loss Over Epochs

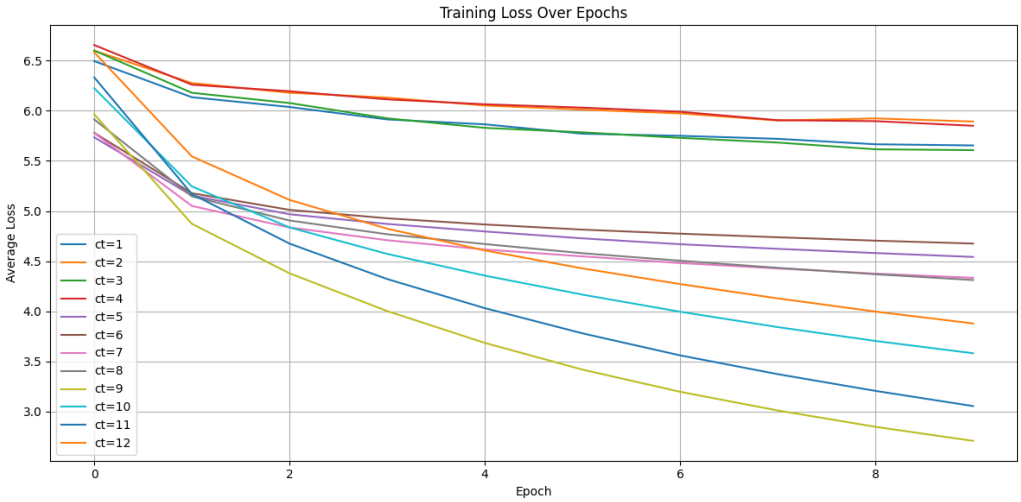

Figure 8: Training Loss Over Epochs

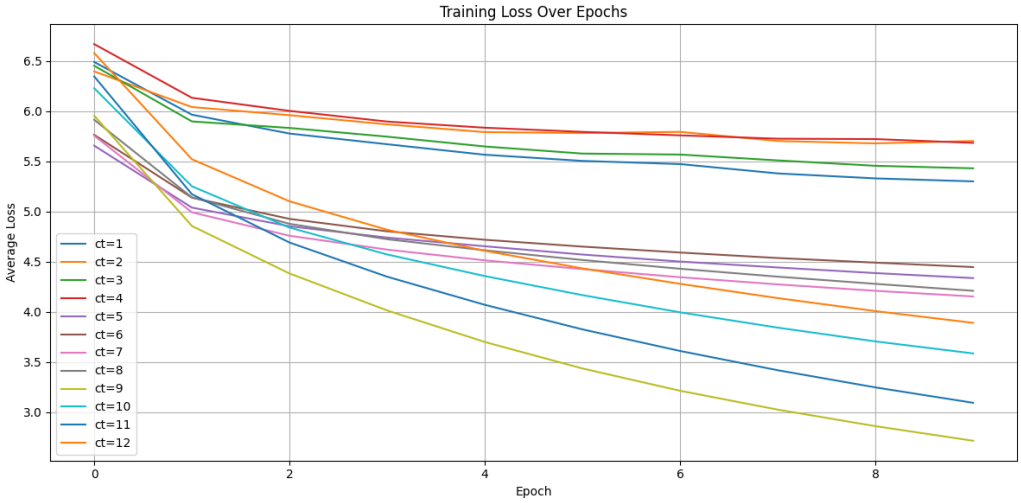

Figure 9: Training Loss Over Epochs

Best Configuration

Hyperparameters

Hyperparameter	Value
Learning Rate (lr)	0.01
Batch Size	32
Dropout	0.3
Number of Attention Heads (h)	16
Number of Layers (N)	2
Number of Epochs	10
Model Dimension (d_model)	512
Feed-Forward Dimension (d_ff)	2048

Table 4: Best Hyperparameter Configuration

Training Progress

Epoch	Duration	Loss	BLEU Score
00	2:45	6.219	0.40
01	2:51	6.021	0.09
02	2:49	6.005	0.07
03	2:43	5.897	0.37
04	2:38	5.729	0.07
05	2:41	5.851	0.28
06	2:39	5.885	0.44
07	2:58	5.682	1.24
08	2:38	5.807	2.92
09	2:37	5.673	1.76

Table 5: Training Progress

Test Data Score

Bleu Score: 1.709664057978683

Test Data Evaluation

After training the transformer model with various hyperparameter configurations, it's essential to evaluate its performance on the test dataset. The test dataset serves as an independent measure of the model's translation quality, helping us understand how well it generalizes to unseen data. The primary evaluation metric used here is the BLEU score.

BLEU Score for Best Configuration

For the best configuration, which is characterized by the hyperparameters listed earlier, we evaluated the model's translation quality on the test data. The BLEU score for this configuration is 1.709664057978683. This score indicates how well the model's translations align with the reference translations provided in the test dataset. A higher BLEU score signifies better translation quality.

Interpreting the BLEU Score

The BLEU score provides a quantitative measure of the quality of machine translation. Here's how to interpret the score:

- I am using BLEU scores in range between 0 and 100, with higher values indicating better translation quality. - A BLEU score of 100 would mean that the machine translation output is identical to the reference translation. - A score of 0 indicates that there is no overlap between the machine translation and the reference translation.

Analysis of Training Progress

Interpreting the training loss graph indicates that the model is more likely to perform better as the number of epochs increases. Currently, the model has been trained for 10 epochs, and we observe that as the training progresses, the loss decreases. This suggests that the model continues to learn and improve as more training epochs are completed.

Interpreting Hyperparameter Tuning

Based on the hyperparameter tuning assessment, it's evident that the model's performance, as measured by the BLEU score for this assignment, depends on how the parameters were randomly initialized. The randomness in the initialization of the model's weights and biases, combined with the data-hungry nature of transformers, means that different runs with the same hyperparameters may yield varying results. Consequently, comparing model performance at this stage may not provide definitive insights.

Challenges with BLEU Score

In our evaluation, we've used the BLEU score as a metric to assess translation quality. However, it's important to recognize that the BLEU score, while widely used, has limitations. It primarily focuses on n-grams and does not consider the overall fluency and coherence of translations.

To illustrate the limitations of BLEU, consider the following example: for the sentences "I am a Boy" and "I am a Girl," we obtained a BLEU score of 30. However, when we added a period at the end of each sentence ("I am a Boy." and "I am a Girl."), the BLEU score reduced to 10. This demonstrates that

small punctuation changes can significantly impact BLEU scores, which might not align with human judgment.

Conclusion

In this report, we have examined various aspects of the transformer model, from the mechanics of self-attention to the incorporation of positional encodings. We also provided insights into the hyperparameter tuning process, training progress, and test data evaluation.

While our model's performance may improve with more training epochs, we recognize that the influence of randomly initialized parameters and the data-hungry nature of transformers can introduce variability in results. As we navigate the challenges of machine translation evaluation, it's important to consider the limitations of metrics like the BLEU score.

The insights presented in this report serve as a foundation for ongoing research and development in the field of natural language processing and machine translation. Transformers continue to show immense potential in improving cross-lingual communication and understanding.

Thank you for your attention to this report, and we remain committed to further exploration in the ever-evolving field of natural language processing.