# ANLP Assignment 2 Report



---

# Part 2 - Theory Questions

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

Self-attention is a core concept used in deep learning, particularly in models dealing with sequential data like text or time series. Its main purpose is to help the model focus on different parts of a sequence when predicting an output, rather than treating all elements in the sequence equally. This mechanism allows the model to capture complex relationships between different parts of the sequence, even if they are far apart from each other.

### What is Self-Attention

To understand why self-attention is important, imagine you're trying to translate a sentence from English to another language. To accurately translate a word, it might be important to look at words from earlier or later in the sentence, because they help determine the meaning. For example, in the phrase "The cat sat on the mat," the word "mat" gives context to "sat" and even "cat." A simple neural network might not easily understand this relationship if it's looking at one word at a time in isolation.

Self-attention solves this by computing attention scores for each word in relation to every other word in the sentence. These scores help determine how much importance (or attention) one word should give to other words when creating its final representation. This means that, for every word, the model learns which other words are more relevant, effectively letting the model focus on the parts that matter most to the current word.

## How It Works

Self-attention works by using three key components for each word in a sequence: **Query (Q)**, **Key (K)**, and **Value (V)**.

1. **Query** represents the word you are trying to understand.
2. **Key** is used to compare the Query with all the words, essentially measuring how related the words are.
3. **Value** represents the information contained in each word.

The Query and Key vectors are used to calculate a score for each word in the sequence. These scores indicate how relevant each word is to the current word. Once we have these scores, they are used to calculate a weighted sum of all the Value vectors. This weighted sum becomes the final representation of each word, now updated with information from the other words in the sequence.

## Capturing Dependencies

Self-attention is powerful because it allows the model to capture both short-range and long-range dependencies between words. In many real-world scenarios, words that are far apart in a sentence may have important relationships. For instance, consider the sentence, "The dog, which was barking loudly, scared the cat." The subject "dog" and the action "scared" are closely related, but they are separated by several words. Self-attention

effectively connects them by giving more weight to "dog" when calculating the representation of "scared."

### Advantages Over Previous Methods

Before self-attention, Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks were used to capture dependencies in sequences. However, these models process words one by one, which makes it difficult to learn relationships between words that are far apart, especially in long sequences. Self-attention, in contrast, allows every word to directly interact with every other word, making it much easier for the model to understand complex patterns, regardless of their distance.

---

## 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.

Transformers are incredibly powerful models that have revolutionized natural language processing (NLP) by using an architecture based on self-attention. In this architecture, all tokens in a sequence are processed simultaneously, rather than in order. While this is efficient, it presents a problem: transformers lose the ability to understand the order of words in a sentence since the model itself is permutation-invariant. To solve this, transformers use **positional encodings** to provide the model with information about the position of each word in a sequence. Essentially, these positional encodings help transformers capture the sequential relationships between words, which is crucial for understanding natural language.

### Incorporating Positional Encodings in Transformers

In a transformer, **word embeddings** are combined with **positional encodings** to make sure the model is aware of the position of each word. Specifically, once a word embedding is generated for each token, the corresponding positional encoding is **added** to it before feeding it to the model's first layer. This ensures

that each word embedding is not only informed by the meaning of the word itself but also carries information about its position in the sequence.

Traditionally, **sinusoidal positional encodings** have been used in transformers. These encodings use sine and cosine functions of different frequencies to create a unique positional value for each position in the sequence. The advantage of this approach is that it allows the model to generalize to sequence lengths it hasn't encountered during training, as the pattern of sine and cosine can extend to longer sequences.

## Recent Advances in Positional Encodings

Recent research has introduced various new types of positional encodings that aim to improve the way transformers handle positional information. Here are some examples of these advances:

1. **Learnable Positional Embeddings**: Instead of using fixed sinusoidal values, some transformers use **learnable positional embeddings**. These are vectors whose values are learned during training, similar to how word embeddings are learned. Learnable embeddings allow the model to adapt more flexibly to the data but may lose some of the generalizability provided by the sinusoidal method.
2. **Relative Positional Encodings**: In contrast to the traditional absolute encodings (which assign a specific position to each word), **relative positional encodings** focus on the relationships between tokens rather than their absolute positions. This means that the encoding represents how far apart two tokens are rather than their individual positions in the sequence. This approach is particularly useful for capturing dependencies between words that may be far apart, and has been found to be helpful in tasks like machine translation where relationships between words across the entire sequence are important.
3. **Rotary Positional Embeddings (RoPE)**: **RoPE** is a newer technique that encodes positional information by rotating the word embedding vectors based on their position. It preserves the directional relationships between tokens, and has shown good results in handling long sequences. RoPE is an efficient alternative that helps in keeping track of positional relationships effectively while maintaining computational efficiency.
4. **AliBi (Attention with Linear Biases)**: **AliBi** adds a bias term directly in the self-attention scores instead of embedding the position into the input. This

bias changes based on the distance between tokens, which gives the model a way to prioritize closer tokens and helps it deal with longer contexts. AliBi is efficient, simple, and has been shown to work well even without explicitly incorporating positional embeddings in the input layer.

### Differences from Traditional Sinusoidal Encodings

The key difference between these new approaches and the traditional sinusoidal encodings lies in how position is represented and used:

- **Flexibility**: Learnable positional embeddings adapt to the data being trained, whereas sinusoidal encodings are fixed and do not change during training.
- **Generalization**: Sinusoidal encodings are designed to generalize to sequences of different lengths because the sine and cosine functions can continue indefinitely. Learnable positional encodings, in contrast, may struggle when dealing with sequences longer than those seen during training.
- **Efficiency and Relationship Modeling**: Relative encodings, RoPE, and AliBi focus more on capturing the relationships between tokens rather than using absolute positions. This helps the transformer more effectively understand dependencies between words that might be far apart in the sequence.

---

# Part 3 - Implementation

## Hyperparameter Tuning

### Configurations Used

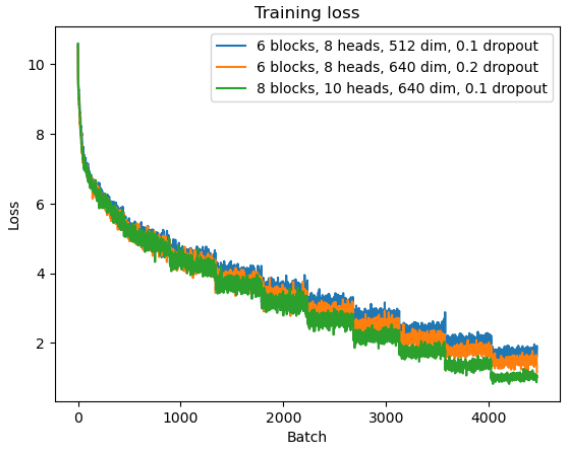
- Config 1
  - 6 layers in the encoder/decoder stack
  - 8 attention heads
  - 512-dimensional embedding vector
  - Dropout of 0.1
- Config 2
  - 6 layers in the encoder/decoder stack

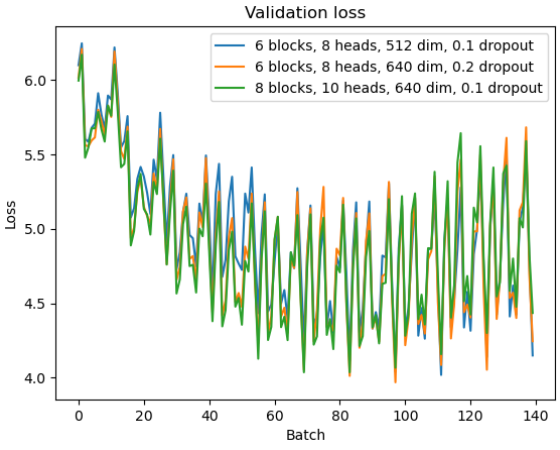
- 8 attention heads
- 640-dimensional embedding vector
- Dropout of 0.2

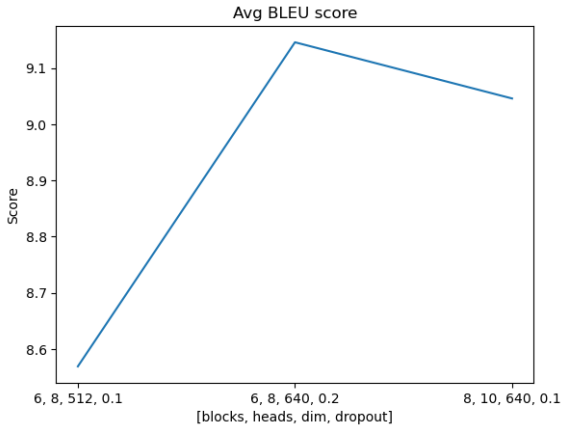
- Config 3
  - 8 layers in the encoder/decoder stack
  - 10 attention heads
  - 640-dimensional embedding vector
  - Dropout of 0.1

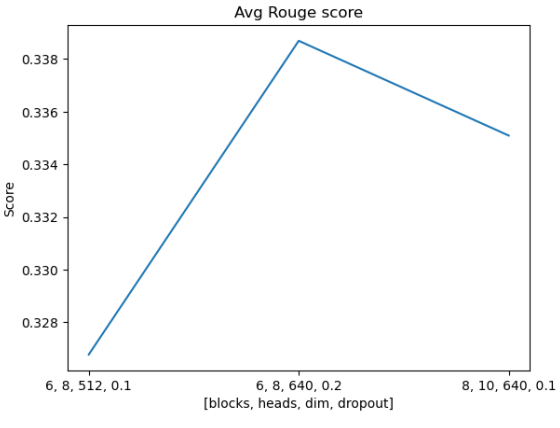
- The main motivation behind these choices was to study the impact of increasing the complexity of the model and the impact of regularization

## Plots of Losses and Metrics



## Analysis

- Initially, we try to increase the embedding dimension along with dropout
    - This leads to lower train losses but higher validation losses
- On the other hand, increasing the number of blocks in the encoder and decoder, number of attention heads as well as embedding dimension gives even better train losses, while also giving worse validation losses
- All of these show that the initial configuration of 6 blocks in the encoder and decoder models with 8 heads and a 512-dim vector with 0.1 dropout strikes a balance between enough complexity and a good enough regularization to balance the complexity
- But we observe that the quality of translation gets slightly better when we slightly increase the complexity of the model while also increasing the dropout as can be seen from the plots of the average BLEU and Rouge scores

---