ADVANCED NLP - ASSIGNMENT 2

-

PRISHA,2021101075

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

Self-attention was introduced to address a fundamental limitation in traditional sequence processing models like RNNs and LSTMs, which struggle to capture long-range dependencies in a sequence effectively. In these models, the ability to remember information from earlier in the sequence fades as the model processes more elements, especially when dealing with lengthy sequences. This **hampers the model's ability to capture distant relationships between tokens**. **Self-attention** solves this by allowing the **model to focus on different parts of the sequence** at each step, regardless of how far apart the tokens are. This mechanism enables **efficient learning of both short- and long-distance dependencies**.

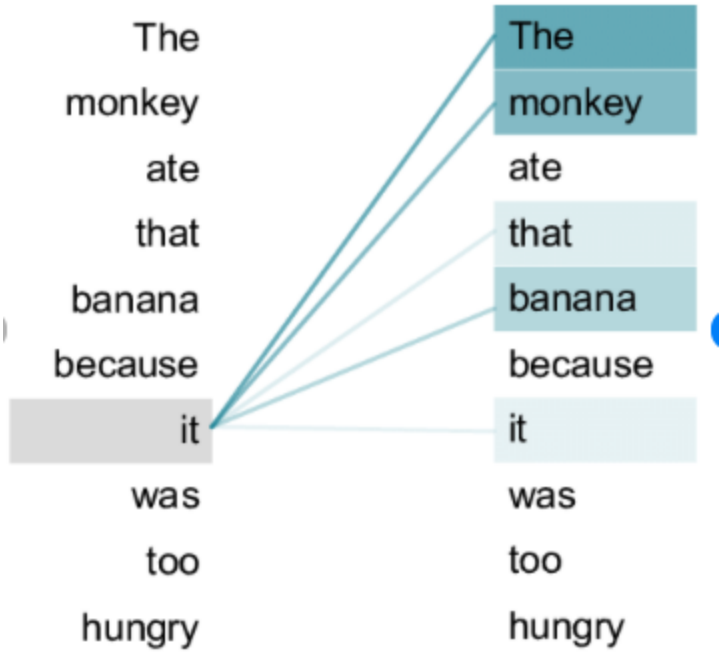
What is Self-Attention?

Self-attention is a mechanism that enables a model to weigh the importance of different tokens in a sequence when making a prediction about a particular token. In essence, each token is able to "attend" to every other token in the sequence, calculating its relevance using a system of learned weights. This way, tokens are no longer constrained by the linearity of traditional sequence models.

For example, in the sentence *"The monkey ate that banana because it was too hungry"*, as shown in the image, the word "it" can attend to "monkey" because the self-attention mechanism captures the relationship between them, even though they are separated by several tokens. This is done by computing attention scores for each pair of tokens.

Self-attention operates by transforming the input sequence into **three vectors: query, key, and value**. These vectors are obtained through linear transformations

of the input. The attention mechanism calculates a weighted sum of the values based on the similarity between the query and key vectors. The resulting weighted sum, along with the original input, is then passed through a feed-forward neural network to produce the final output. This process allows the model to focus on relevant information and **capture long-range dependencies.**



An example of the self-attention mechanism following long-distance dependency in the Transformer encoder.

Mathematics Behind Self-Attention

Let's denote:

- Q (Query): The query vector for each word (which represents what the word is querying about other words).
- K (Key): The key vector for each word (which helps assess whether a word is relevant to the query).
- V (Value): The value vector for each word (the actual content information).

The attention score between two tokens is computed using the dot product between their query and key vectors. The scores are then normalized (typically using a softmax function) to produce the attention weights. These weights are finally applied to the value vectors to create a weighted sum that represents the attended output for each token.

The self-attention mechanism is calculated as follows:

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

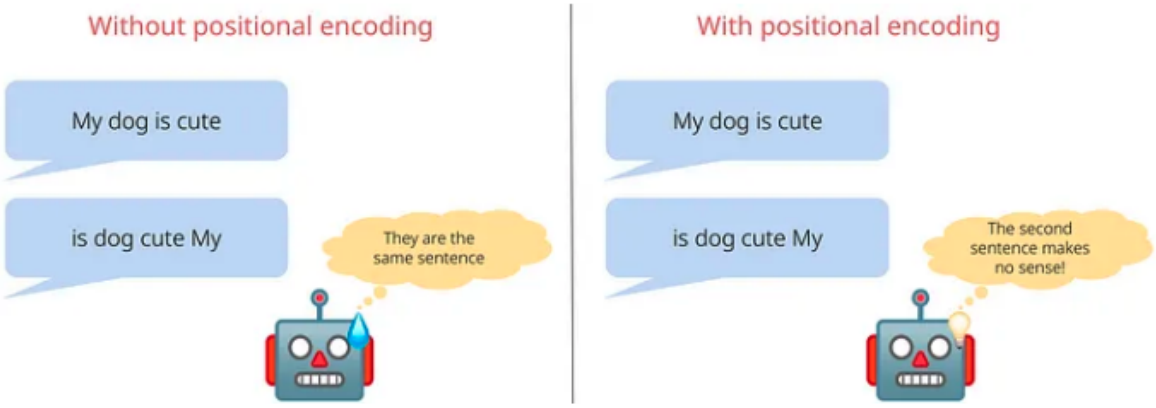
Where:

- d_k is the dimensionality of the key vectors.
- The dot product Q * K^T gives a matrix of attention scores between tokens.
- The softmax ensures that the attention scores sum to 1, making them interpretable as probabilities.

Q- 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 built around the self-attention mechanism, which allows each token in a sequence to attend to every other token, irrespective of its position in the sequence. While this is powerful for capturing global dependencies, it

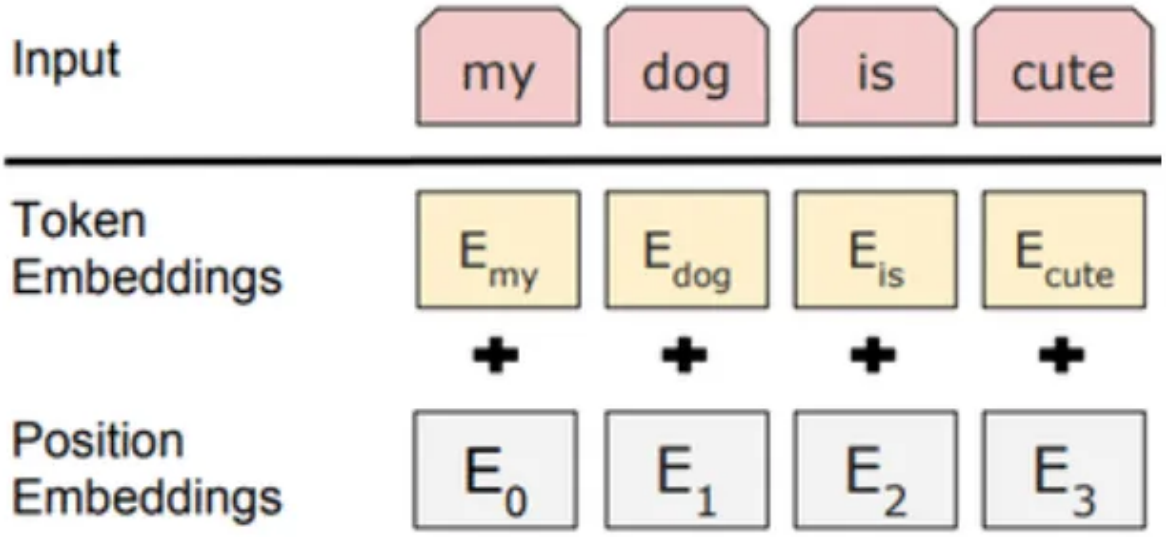
introduces a challenge: **the model does not inherently know the order of the words in the sequence.**As seen in the left side of the image, without positional encoding, the model cannot distinguish between “My dog is cute” and “is dog cute My” because both sequences have the same set of tokens, leading to nonsensical outputs.



Positional encodings provide the model with information about the relative or absolute positions of tokens in a sequence. This allows the transformer to know where each word is located, thus preserving the structure and meaning of the sentence, as shown in the right side of the image. With positional encodings, the model recognizes that rearranging the words changes the meaning.

Incorporation of Positional Encodings

Positional encodings are added to the word embeddings at the input layer of the transformer architecture. The resulting input is a sum of the token embedding and its positional encoding. This sum is then passed through the attention mechanism, enabling the model to make sense of both the word and its position in the sequence.



In the original transformer model, **sinusoidal positional encodings** were used, which encode positional information as sine and cosine functions of different frequencies. These encodings have the property of being continuous and allowing the model to generalize to sequence lengths it hasn't seen before.

The formula for sinusoidal positional encoding is as follows:

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

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

Where:

- `pos is the position of the token in the sequence.`
- `i is the dimension index of the positional embedding.`
- `d_model is the dimensionality of the model's embeddings.`

This encoding gives the transformer a consistent way to represent the positions of words, while also allowing smooth extrapolation to sequences longer than those

seen during training.

Recent Advances in Positional Encodings

Since the introduction of sinusoidal positional encodings, various advanced positional encoding methods have been developed to address specific challenges and improve transformer performance:

1. **Learned Positional Embeddings:**

   Rather than using a fixed, mathematical function, these embeddings are learned during training, similar to word embeddings. This approach offers more flexibility, allowing the model to adapt to the task-specific positional patterns in data.

2. **Rotary Positional Embeddings (RoPE):**

   Introduced in recent transformer architectures, RoPE encodes relative positions using rotational transformations in the embedding space. This method provides a smooth way to capture relative distances between tokens, which is important for tasks like language modeling, where relative order often carries more significance than absolute position.

3. **Relative Positional Encodings:**

   Instead of encoding absolute positions, some models focus on the relative positions between tokens. This type of encoding is beneficial in tasks where relationships between nearby words are more important than their specific locations in the sequence.

4. **Fourier Feature Positional Encodings:**

   These encodings map the positional information to higher-dimensional spaces using Fourier transformations. This allows transformers to better capture long-range dependencies while maintaining the flexibility to encode positional information in a more expressive manner.

Hyper Parameter Tuning Results

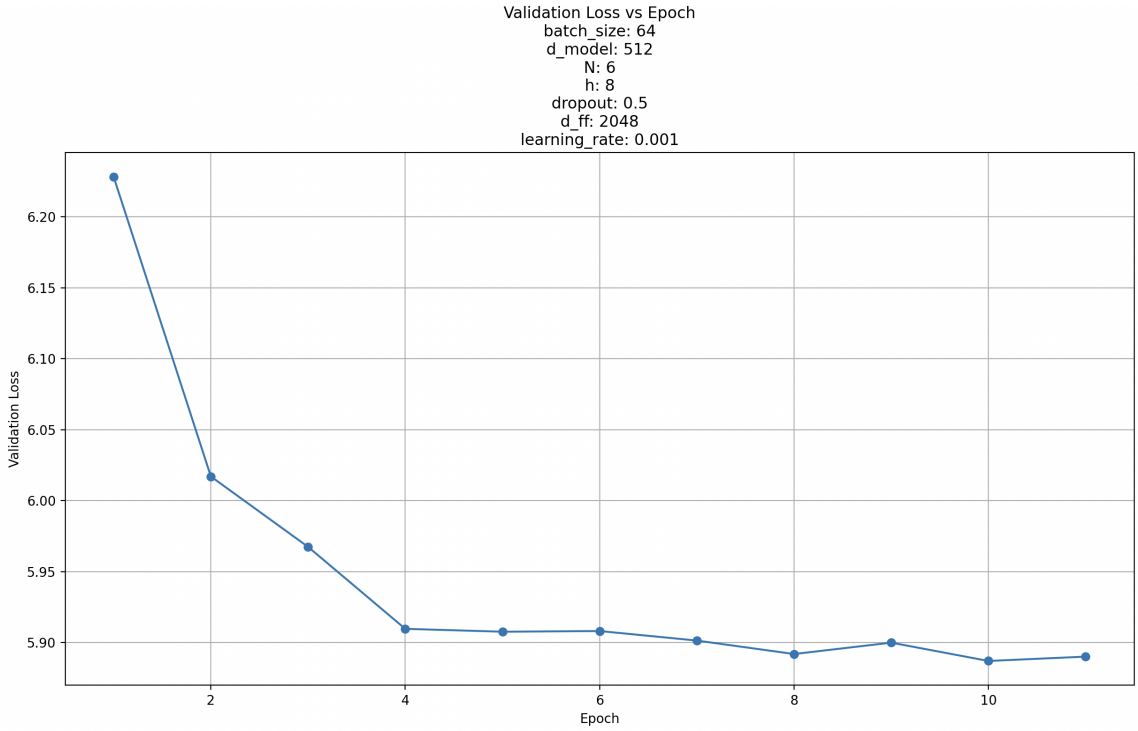
Configuration 1

PARAMETERS	VALUES
BATCH SIZE	64
DIMENSIONALITY	512
LAYERS	6
HEADS	8
DROPOUT	0.5
HIDDEN DIMENSION	2048
LEARNING RATE	0.001

SCORE	VALUES
BLEU	0.0054
ROUGE1	
ROUGE2	
ROUGE- L	

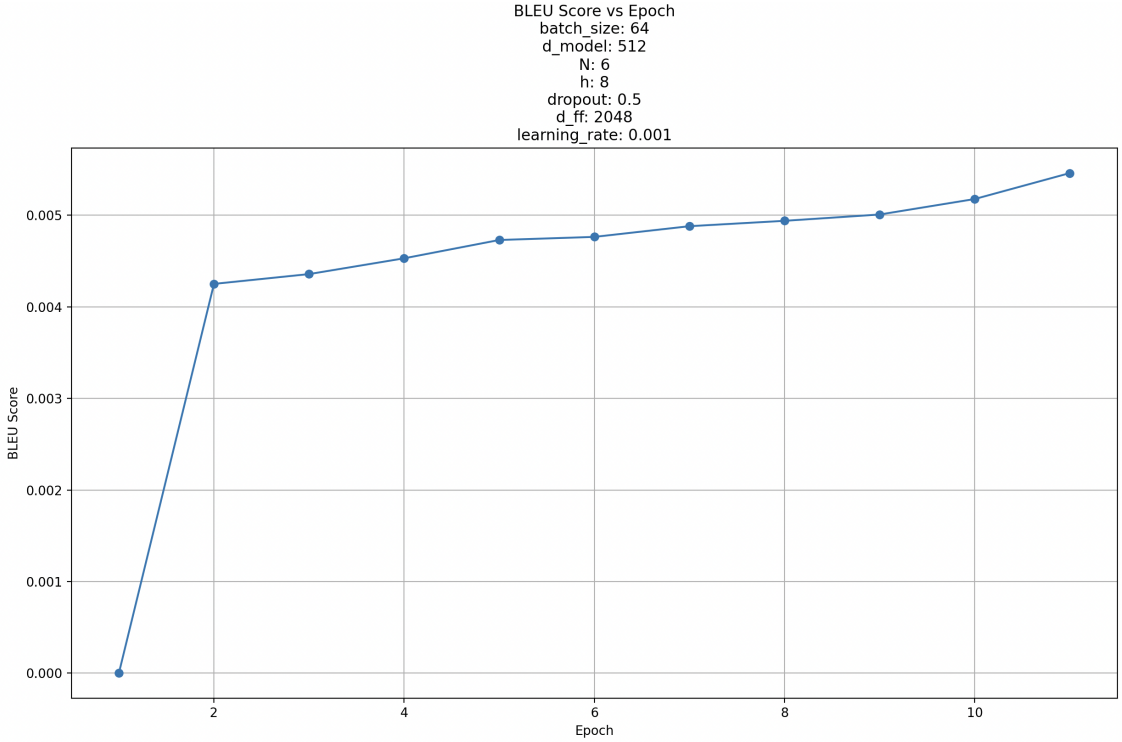
Given are the graphs :

- VALIDATION LOSS WITH EPOCHS



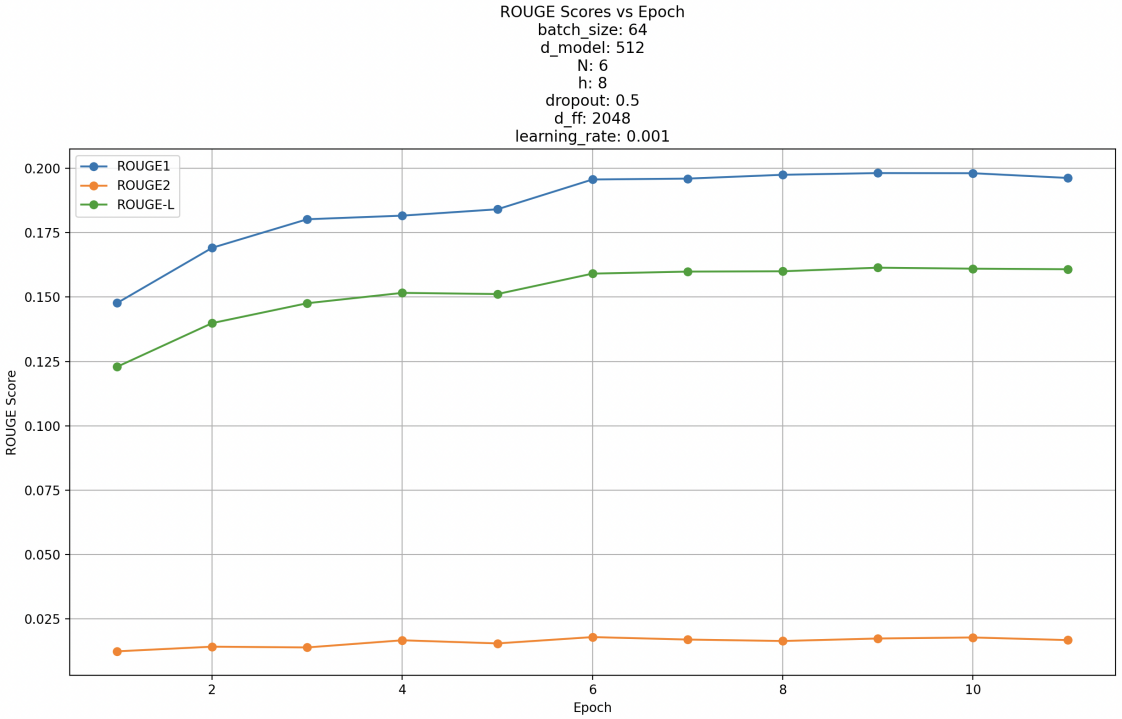
Validation Loss vs Epochs

- BLEU SCORE WITH EPOCHS



Bleu Score with Epochs

- ROUGE SCORE WITH EPOCHS



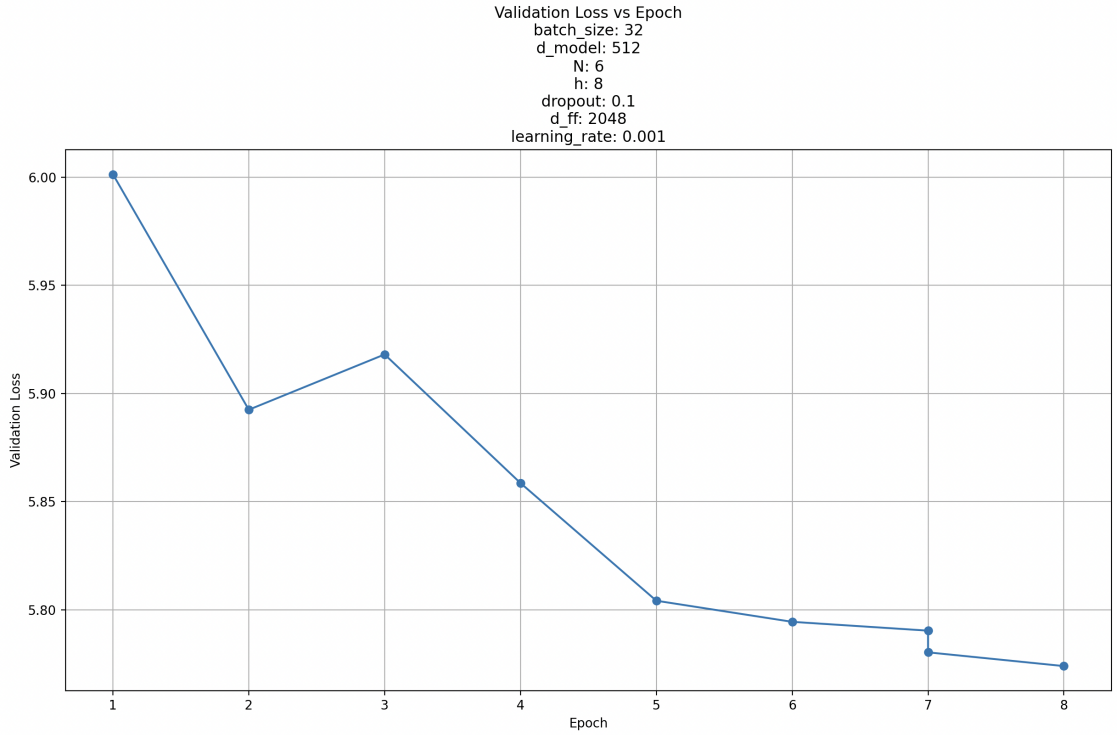
Configuration 2

PARAMETERS	VALUES
BATCH SIZE	64
DIMENSIONALITY	512
LAYERS	6
HEADS	8
DROPOUT	0.5
HIDDEN DIMENSION	2048
LEARNING RATE	0.001

SCORE	VALUES
BLEU	
ROUGE1	
ROUGE2	
ROUGE- L	

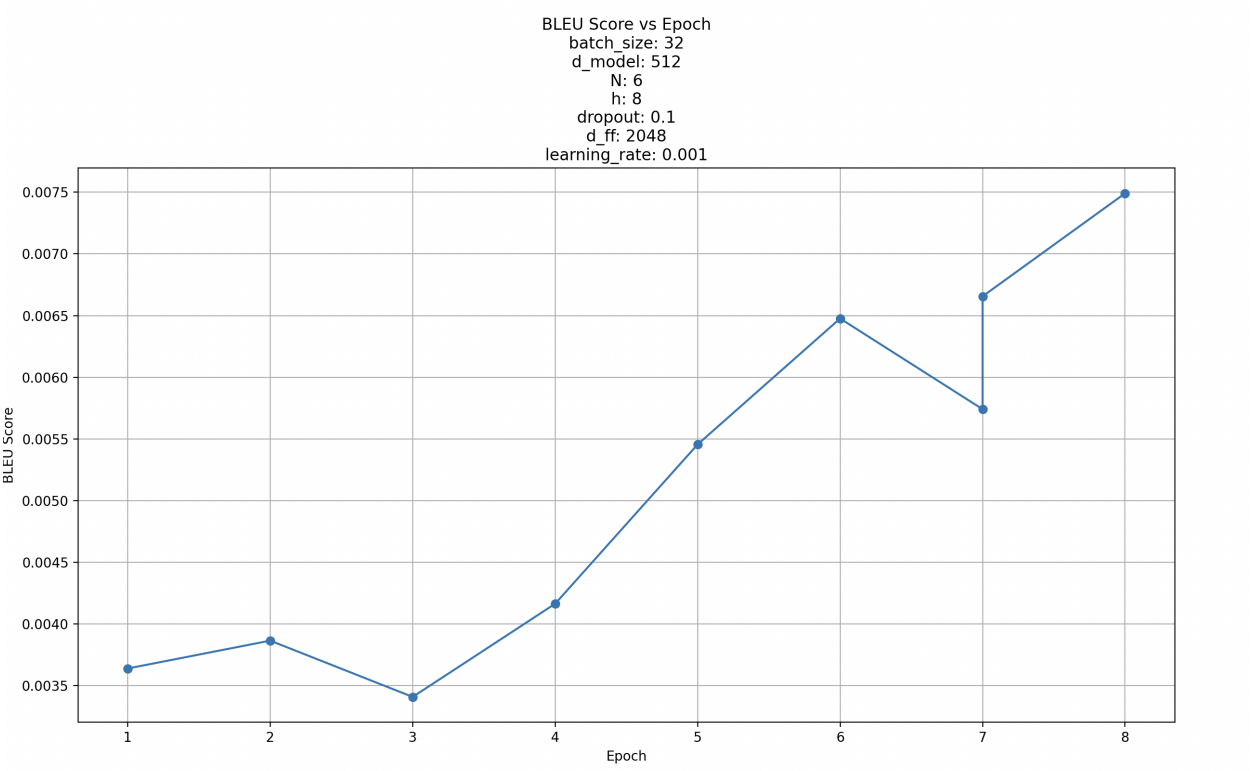
Graph are given below :

- VALIDATION LOSS WITH EPOCHS



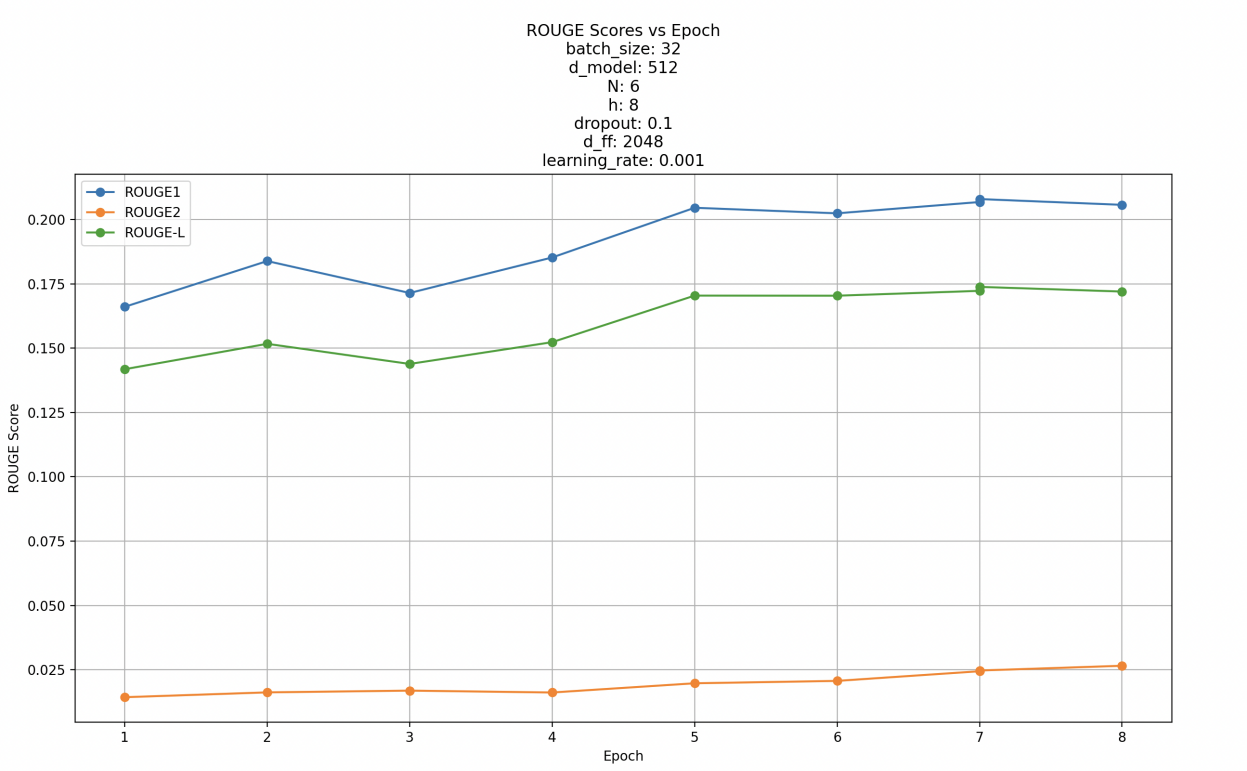
Validation Loss with Epoch

- BLEU SCORE WITH EPOCHS



Bleu Score with Epochs

- ROUGE SCORE WITH EPOCHS



Rouge Score with Epochs

OBSERVATION : The transition from configuration 1 to configuration 2 demonstrates that decreasing the dropout rate from 0.5 to 0.1 leads to improved results.

Configuration 3

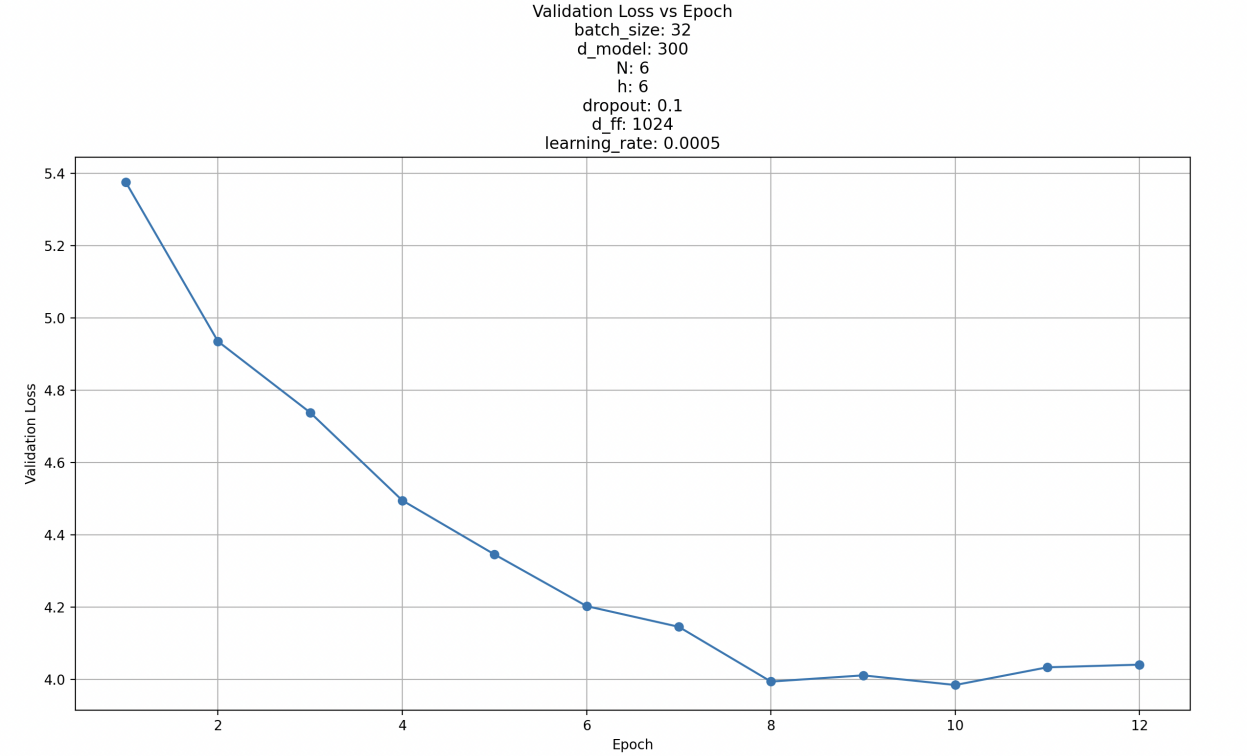
PARAMETERS	VALUES
BATCH SIZE	32
DIMENSIONALITY	300
LAYERS	6
HEADS	6
DROPOUT	0.1
HIDDEN DIMENSION	1024
LEARNING RATE	0.0005

SCORE	VALUES
BLEU	0.049
ROUGE1	0.291
ROUGE2	0.1215
ROUGE- L	0.262
Validation loss	4.040

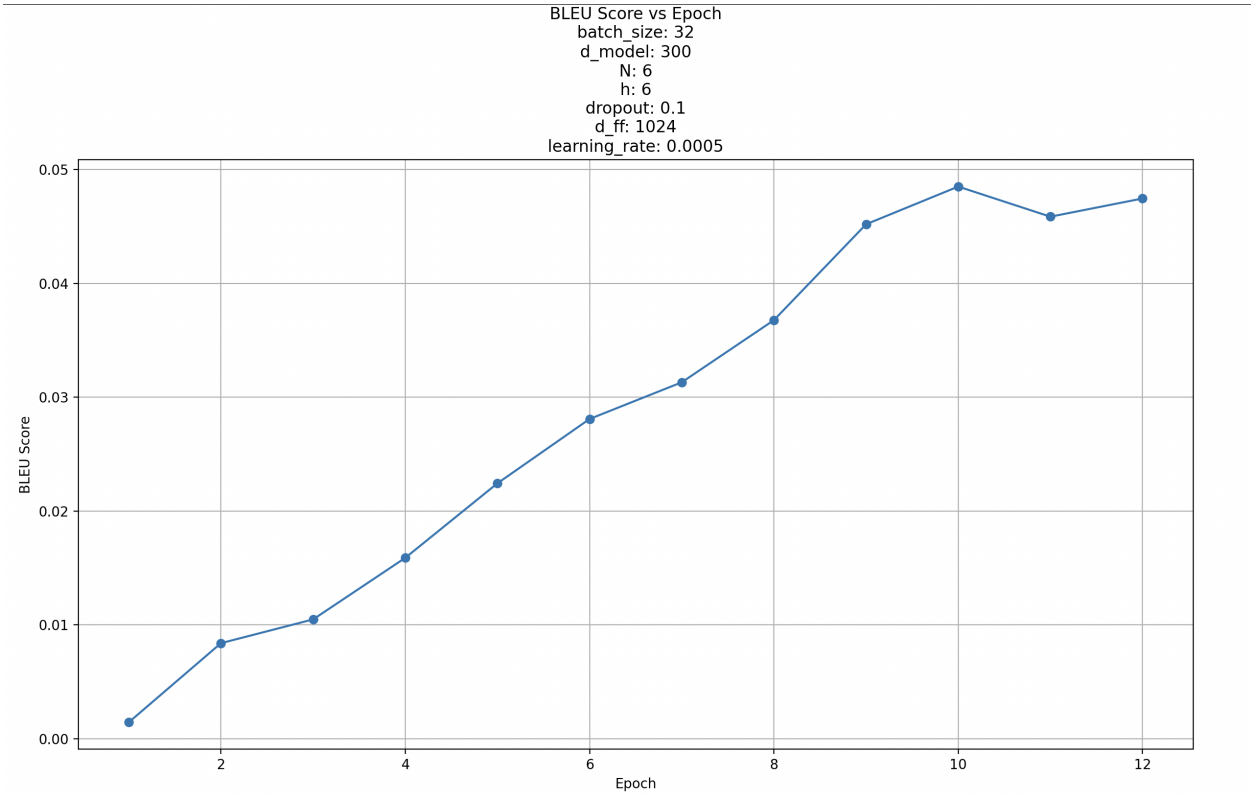
Above scores are wrt to test set.

Graph are given below :

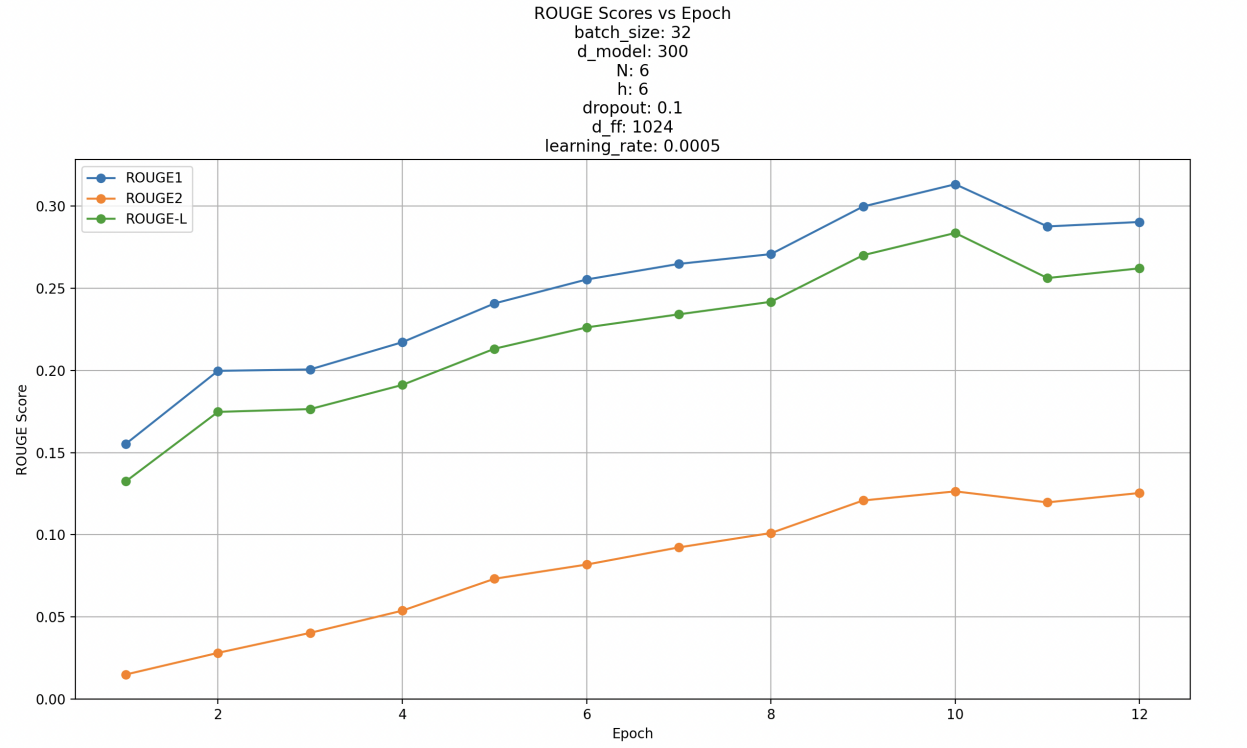
- VALIDATION LOSS WITH EPOCHS



- BLEU SCORE WITH EPOCHS



- ROUGE SCORE WITH EPOCHS



Configuration 4

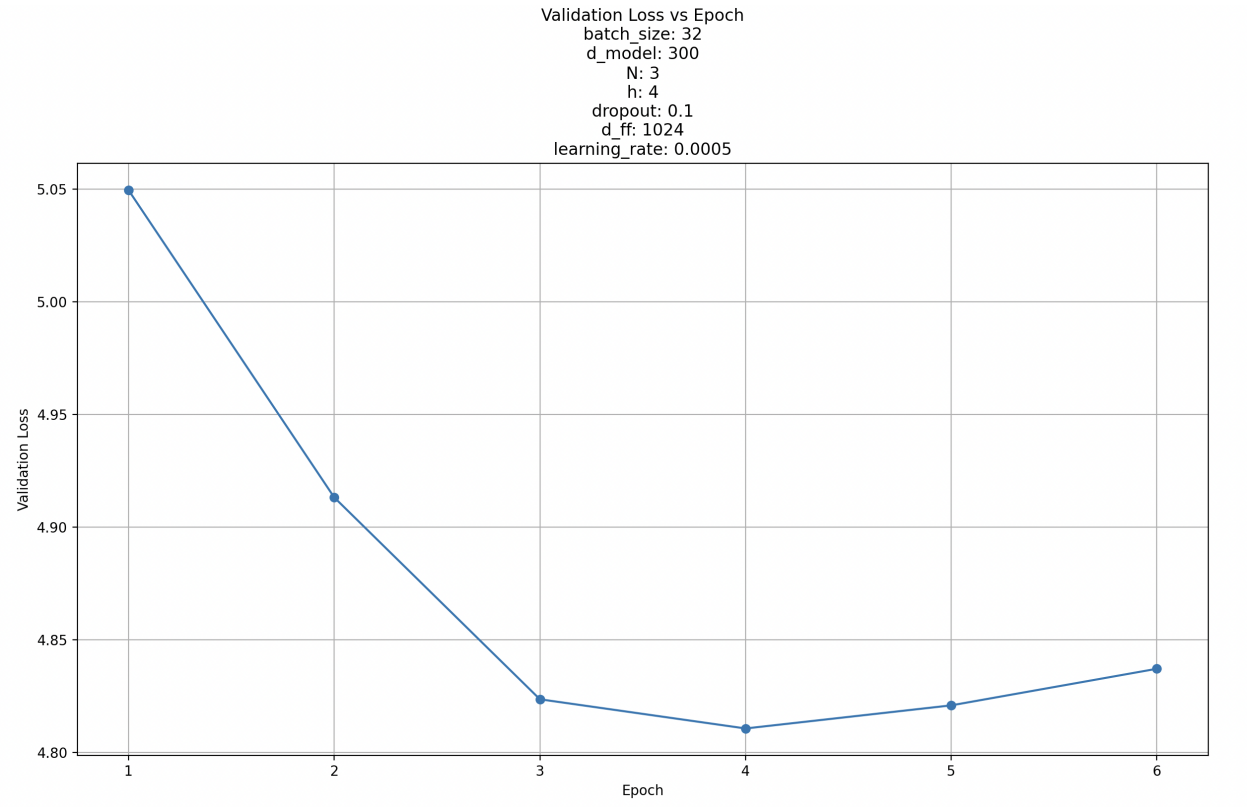
PARAMETERS	VALUES
BATCH SIZE	32
DIMENSIONALITY	300
LAYERS	3
HEADS	4
DROPOUT	0.1
HIDDEN DIMENSION	1024
LEARNING RATE	0.0005

SCORE	VALUES
BLEU	0.056
ROUGE1	0.3551
ROUGE2	0.1658
ROUGE- L	0.3254
Validation loss	4.8305

Above scores are wrt to test set.

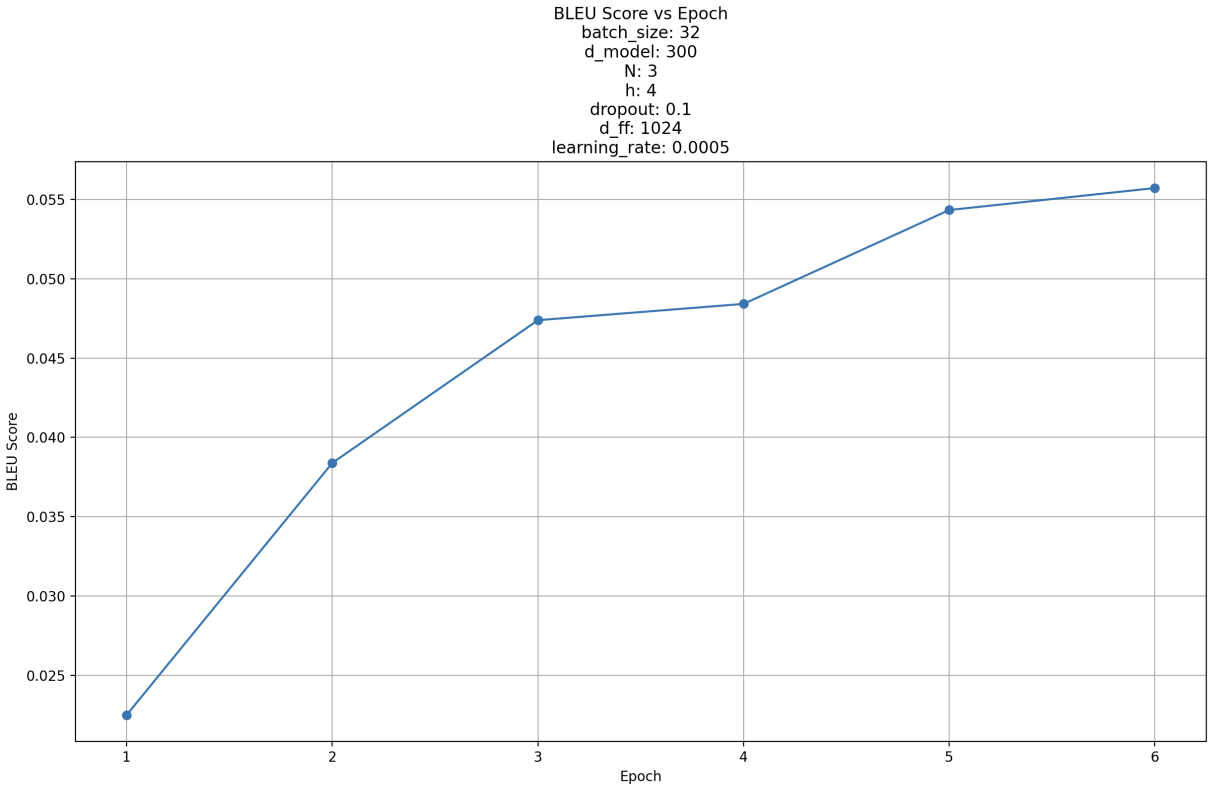
Graph are given below :

- VALIDATION LOSS WITH EPOCHS



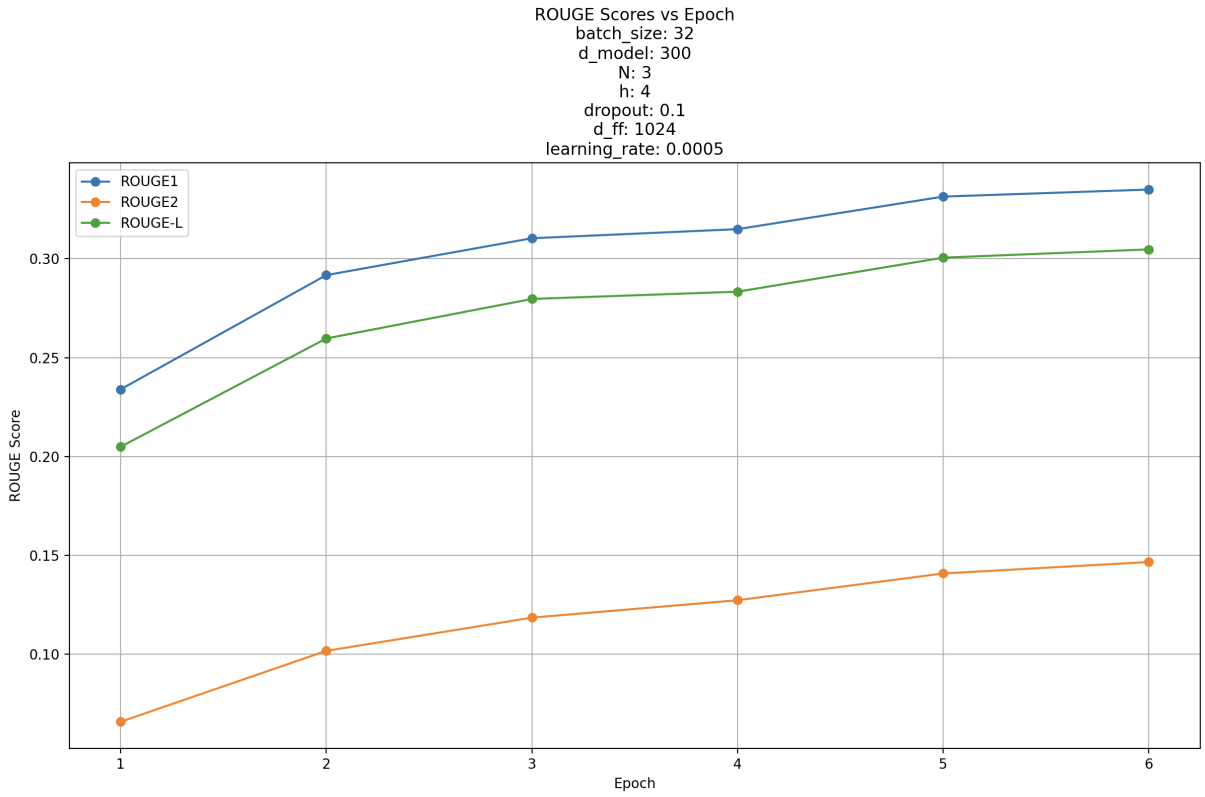
Validation Loss with Epochs

- BLEU SCORE WITH EPOCHS



Bleu Score with Epochs

- ROUGE SCORE WITH EPOCHS



Rouge Score with Epochs

OBSERVATION : We can see that decreasing the number of heads , layers and reducing the learning rate leads to significant improvements in the score; all scores have improved.

Configuration 5

PARAMETERS	VALUES
BATCH SIZE	32
DIMENSIONALITY	300
LAYERS	2
HEADS	3
DROPOUT	0.1
HIDDEN DIMENSION	2048

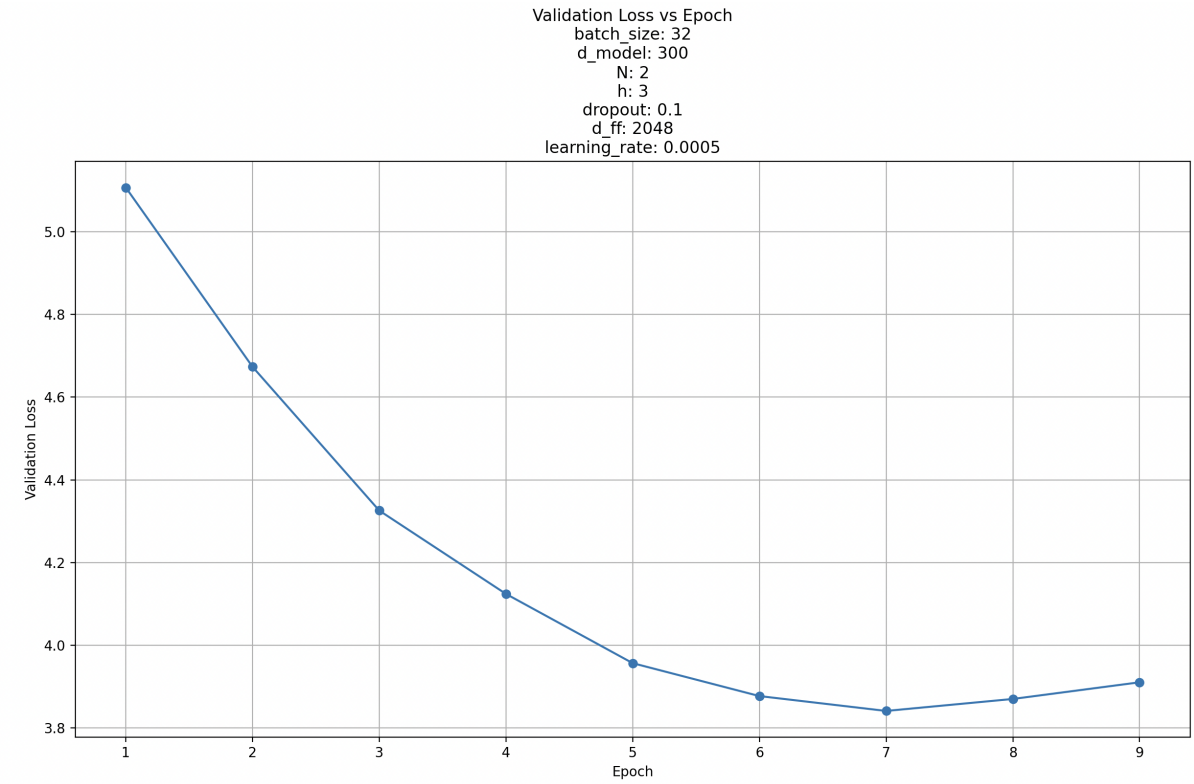
SCORE	VALUES
BLEU	0.076
ROUGE1	0.3609
ROUGE2	0.1702
ROUGE- L	0.3322
Validation loss	3.810

LEARNING RATE	0.0005

Above scores are wrt to test set.

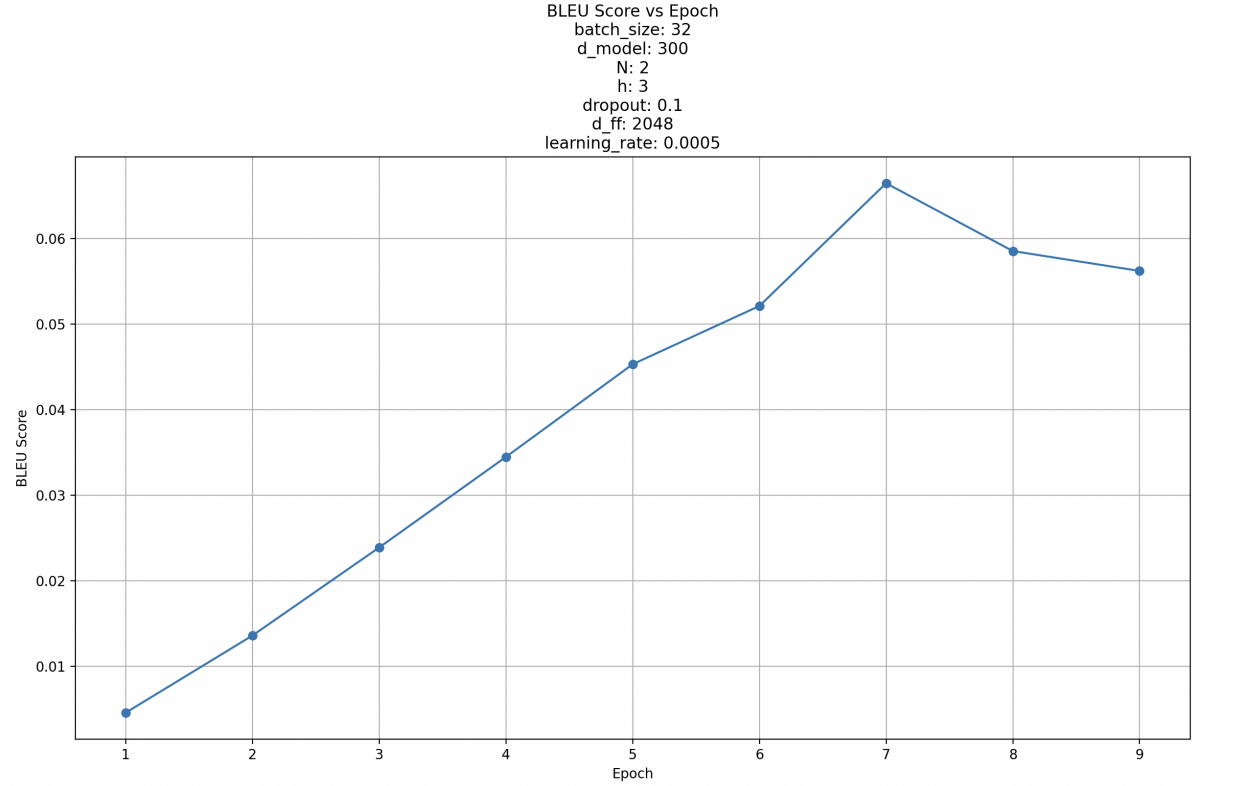
Graph are given below :

- VALIDATION LOSS WITH EPOCHS



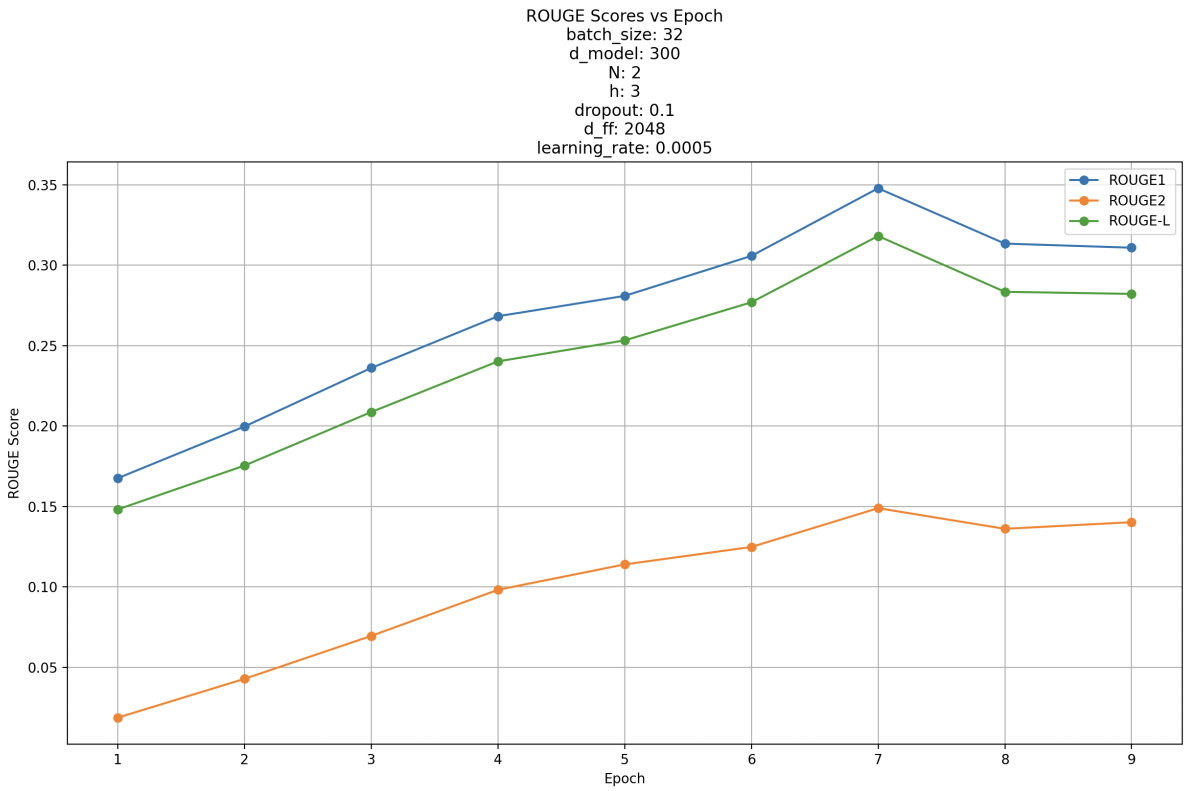
Validation Loss with Epochs

- BLEU SCORE WITH EPOCHS



Bleu Score with Epochs

- ROUGE SCORE WITH EPOCHS



Rouge Score with Epochs

OBSERVATION :We observe that further decreasing the number of heads results in a slight improvement in the score.

Configuration 6 ***(BEST CONFIGURATION)***

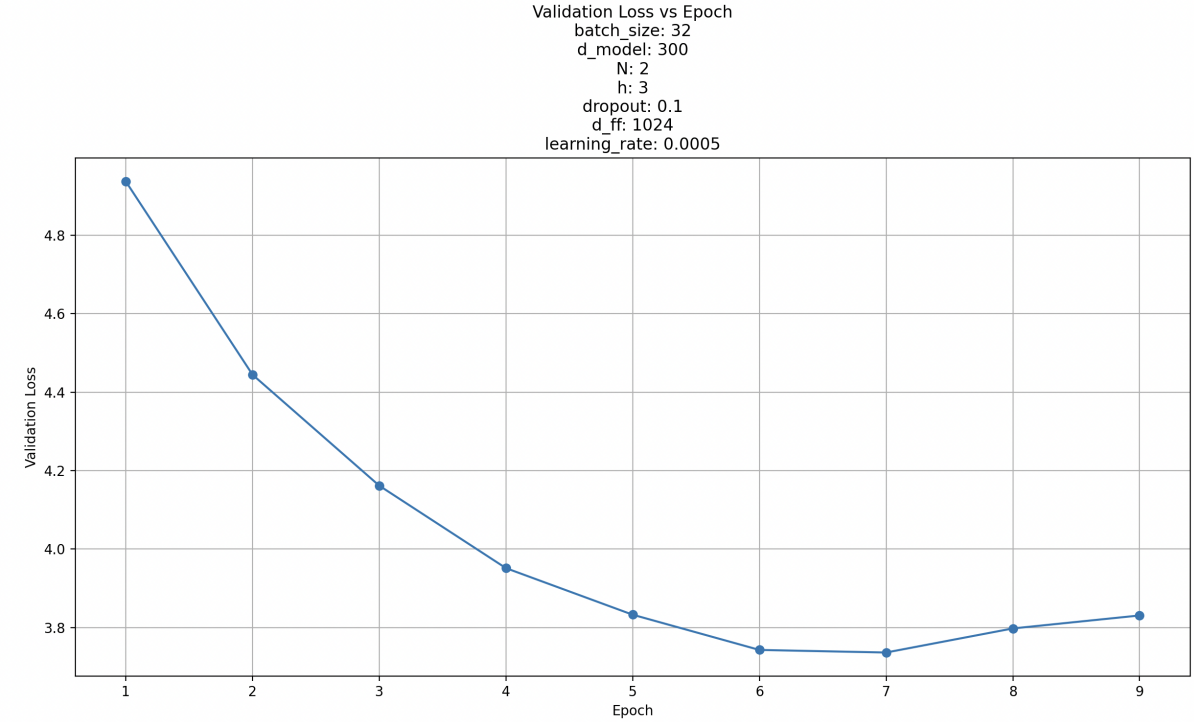
PARAMETERS	VALUES
BATCH SIZE	32
DIMENSIONALITY	300
LAYERS	2
HEADS	3
DROPOUT	0.1
HIDDEN DIMENSION	1024
LEARNING RATE	0.0005

SCORE	VALUES
BLEU	0.1639
ROUGE1	0.3542
ROUGE2	0.1744
ROUGE- L	0.3307
Validation loss	3.1022

Above scores are wrt to test set.

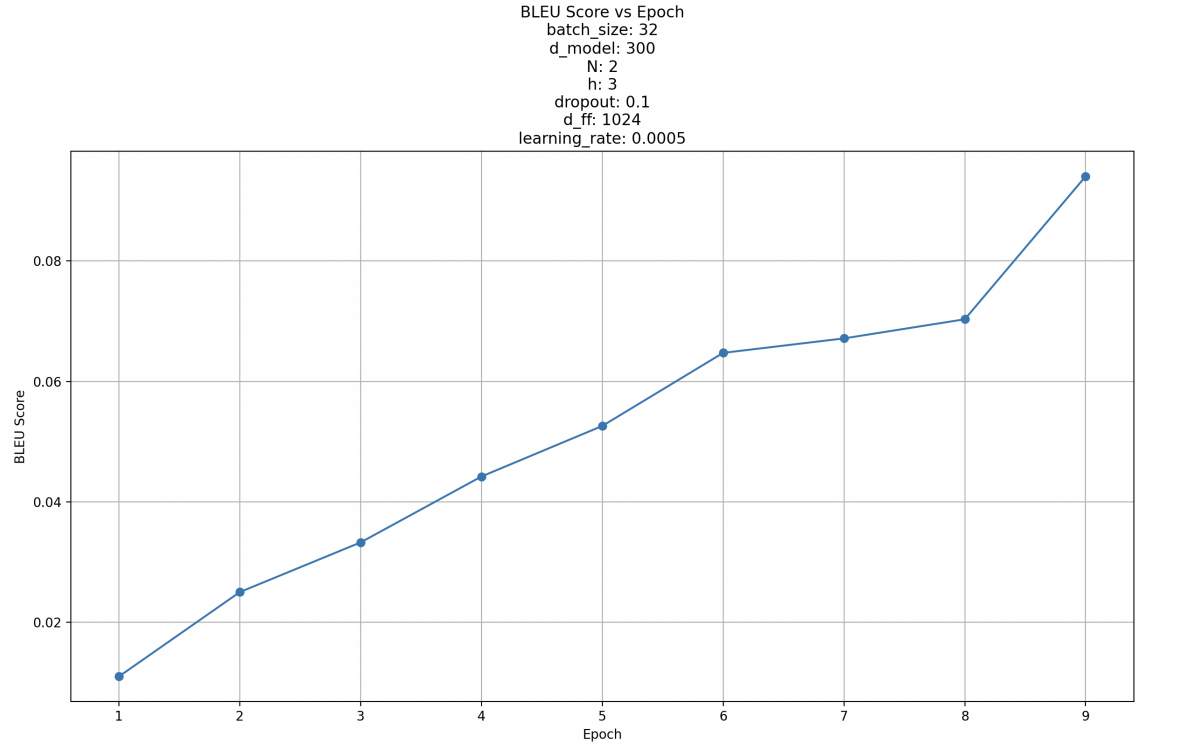
Graph are given below :

- VALIDATION LOSS WITH EPOCHS



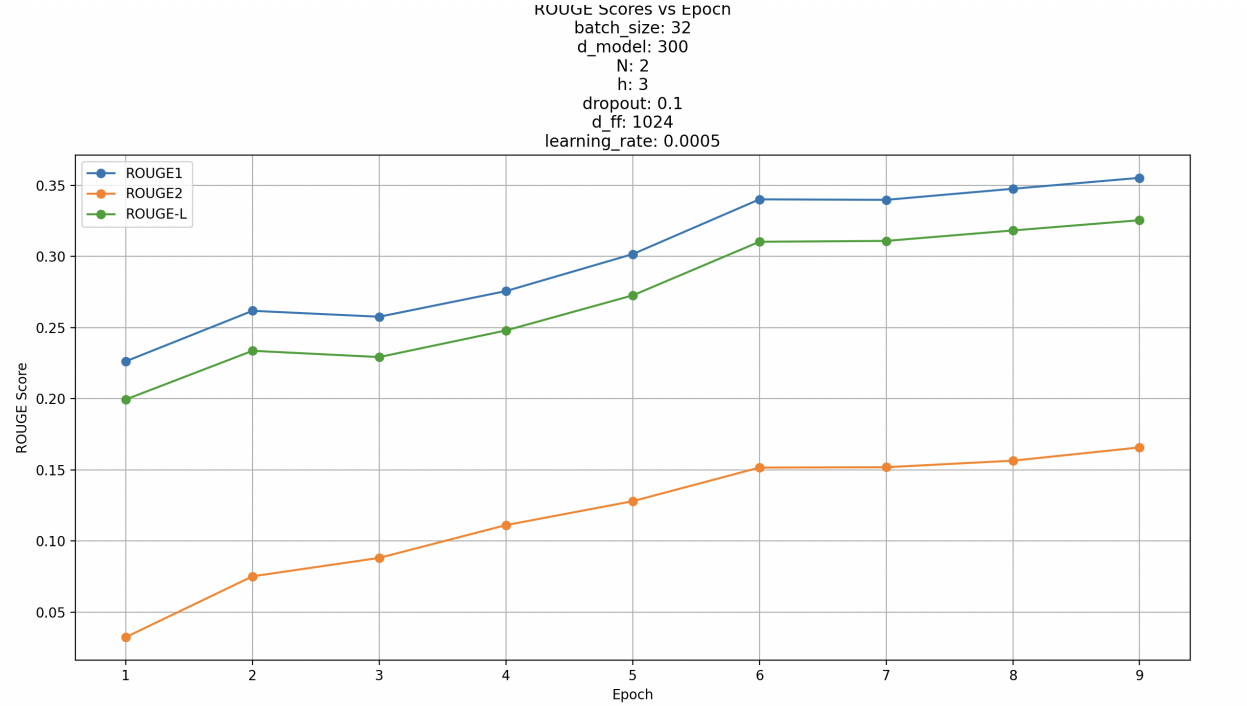
Validation loss with Epochs

- BLEU SCORE WITH EPOCHS



Bleu Score with Epochs

- ROUGE SCORE WITH EPOCHS



Rouge Score with Epoch

OBSERVATION: Decreasing the size of hidden dimensionality, reducing dropout, decreasing learning rate, and adjusting the number of decoder heads lead to the best results for the given dataset.