REPORT ASSIGNMENT – 2

Theory:-

1) A fundamental idea in natural language processing (NLP), self-attention is a crucial part of many contemporary NLP models, including the Transformer architecture. Its objective is to document the interdependencies and connections among various sequence elements, such as the words in a sentence or the tokens in a document. Self-attention makes this easier by enabling the model to evaluate the significance of each component of the sequence before making predictions or recording data.

This is how self-attention functions and how it makes capturing dependencies easier:

- . Input Sequence Representation: A self-attentional process starts with an input sequence, which is a collection of vectors. In natural language processing (NLP), these vectors frequently correspond to word embeddings, in which each word in the sequence is represented as a vector

* "Query, Key, and Value": introduces three sets of vectors for each sequence element:

  - Query: This represents the element for which we want to compute the attention weights.

  - Key: These are the components of the query that we will compare in order to determine the attention scores.

  - Value: This stands for the components from which we will derive the result by allocating weights based on the attention ratings.

- . Weighted Sum: The values are weighted based on the attention scores. Higher attention scores and more output contributions are given to elements that are more pertinent to the inquiry.

- . Dot Product Attention: To calculate self-attention, the query and key vectors are often dot products that are then multiplied by a softmax operation to produce

attention scores. The relevance of each element in the sequence to the current question is indicated by the attention score for that element.

- . Multiple Heads: In practice, self-attention is typically implemented with multiple sets of query, key, and value vectors, which are called "attention heads." The model may learn multiple sorts of dependencies since each head captures a distinct feature of the relationships between the parts.

- . Concatenation and Linear Transformation: To create the final self-attention output for a specific element in the sequence, the outputs from several attention heads are frequently concatenated and linearly modified.

The method used to calculate the attention scores is crucial for capturing interdependence in sequences. Based on the content of the query and key vectors, a model computes attention by taking into account how each element in the sequence relates to the current element. Higher attention scores are assigned to elements that are dependent upon or semantically related to one another, whereas lower attention scores are assigned to less important elements. This enables the model to successfully capture long-range dependencies and interactions between elements by focusing on the most crucial portions of the sequence while making predictions.

Self-attention enables the model to consider the relative importance of each piece in the sequence as well as their relationships with one another, which makes it easier to identify dependencies in sequences. For many NLP activities, including text generation, translation, and language understanding, this process is essential.

2) Transformers provide the model information about word positions in a sequence by using positional encodings in addition to word embeddings. In contrast to convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which are designed to automatically capture spatial or sequential information, respectively, the original Transformer architecture was not designed to automatically infer word positions within an input sequence. For the model to comprehend and capture spatial or sequential dependencies in the data, positional encodings are essential.

The Transformer architecture incorporates positional encodings in the following ways:

• Positional Encoding Function: The positions of the words in the input sequence are represented by a collection of additional learned embeddings, or positional encodings. When these embeddings are coupled with the word embeddings, a combined embedding is produced that contains data regarding the word's identity as well as its placement within the sequence.

• Positional Encoding Matrix: Trigonometric functions such as sine and cosine are used to construct the positional encoding matrix. The goal is to produce a matrix in which a location in the sequence is represented by each row, and a positional dimension by each column. The encoding matrix is fixed and is not learned during training.

• Positional Embedding Addition: The positional encoding matrix is added element-by-element to the word embeddings during the model's forward pass. In doing so, positional information is effectively injected into the model's input.

In mathematical terms, the input of the model is the sum of x and PE, or x + PE, if x stands for the word embeddings and PE for the positional encodings matrix.

In order to capture sequential dependencies, the model must be able to distinguish between words that appear in different locations in the sequence. This is made possible by the addition of positional encodings. The particular design of positional encodings, which makes use of the sine and cosine functions, was selected because it guarantees that the positional information is continuous and that each position has a distinct pattern.

One significant advancement that allows the Transformer design to attain state-of-the-art performance in a variety of sequence-to-sequence applications, such as text production and machine translation, is the incorporation of positional encodings. It's essential to the model's ability to comprehend word order and handle input sequences efficiently.

In my code:-

Positional embeddings are incorporated in the code through the PositionalEncoding class, which adds positional information to the input sequences. These embeddings are essential for self-attention mechanisms in transformers to understand the order of elements in a sequence.

In the PositionalEncoding class, sinusoidal positional embeddings are generated and added to the input data. These embeddings are learned and added element-wise to the input embeddings (x) in both the encoder and decoder. This helps the model differentiate the positions of tokens within the sequences, allowing it to capture sequential information, which is crucial for various NLP tasks.

The positional embeddings are computed using sine and cosine functions with varying frequencies and are then combined with the input embeddings, enriching the model's ability to handle sequence data effectively.

Hyperparametre tuning:-

Firstly this set of hyper parametres were used :-

```
d_model = 512
num_heads = 8
num_layers = 2
src_vocab_size = len(eng_vocab)
tgt_vocab_size = len(fr_vocab)
max_seq_length = 50
batch_size = 32
```

The number of epochs were 6 and following loss trends were observed:

Lr : 0.01

Epoch [1/6]: 100%|[redacted]| 938/938 [03:07<00:00, 5.01it/s]

Loss: 2.0623465

Epoch [2/6]: 100%|[redacted]| 938/938 [03:06<00:00, 5.03it/s]

Loss: 1.7298251

Epoch [3/6]: 100%|[redacted]| 938/938 [03:06<00:00, 5.03it/s]

Loss: 1.4109374

Epoch [4/6]: 100%|[redacted]| 938/938 [03:07<00:00, 5.04it/s]

Loss: 1.4107121

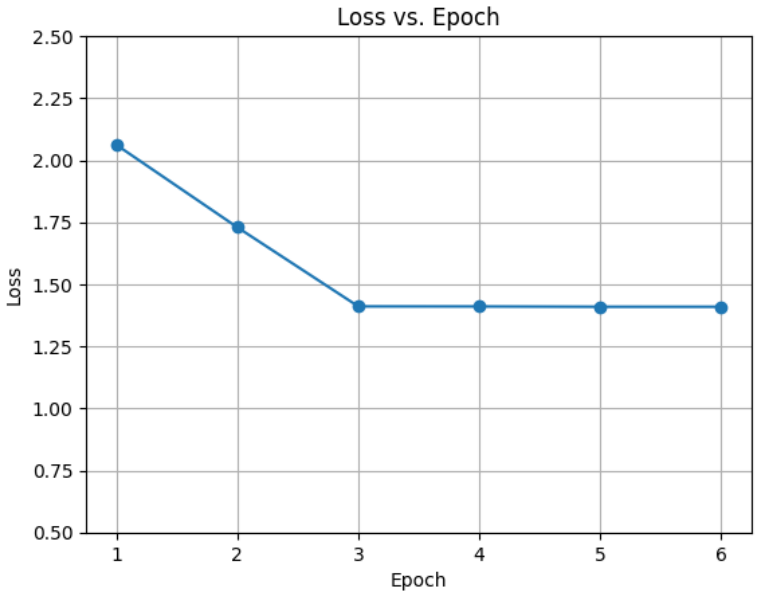
Epoch [5/6]: 100%|[redacted]| 938/938 [03:06<00:00, 5.01it/s]

Loss: 1.4094822

Epoch [6/6]: 100%|[redacted]| 938/938 [03:08<00:00, 5.01it/s]

Loss: 1.4094121

The bleu scores are scaled for 0-100.



BLEU Scores on test :- 0.8513036773946357

BLEU score on train :- 1.050959528333334

After this batch size was made 50 and lr : 0.1

The losses reduced

Epoch [1/6]: 100%|| 938/938 [03:07<00:00, 5.03it/s]

Loss: 2.0623444

Epoch [2/6]: 100%|| 938/938 [03:06<00:00, 5.04it/s]

Loss: 1.7188153

Epoch [3/6]: 100%|| 938/938 [03:06<00:00, 5.03it/s]

Loss: 1.4021018

Epoch [4/6]: 100%|| 938/938 [03:07<00:00, 5.04it/s]

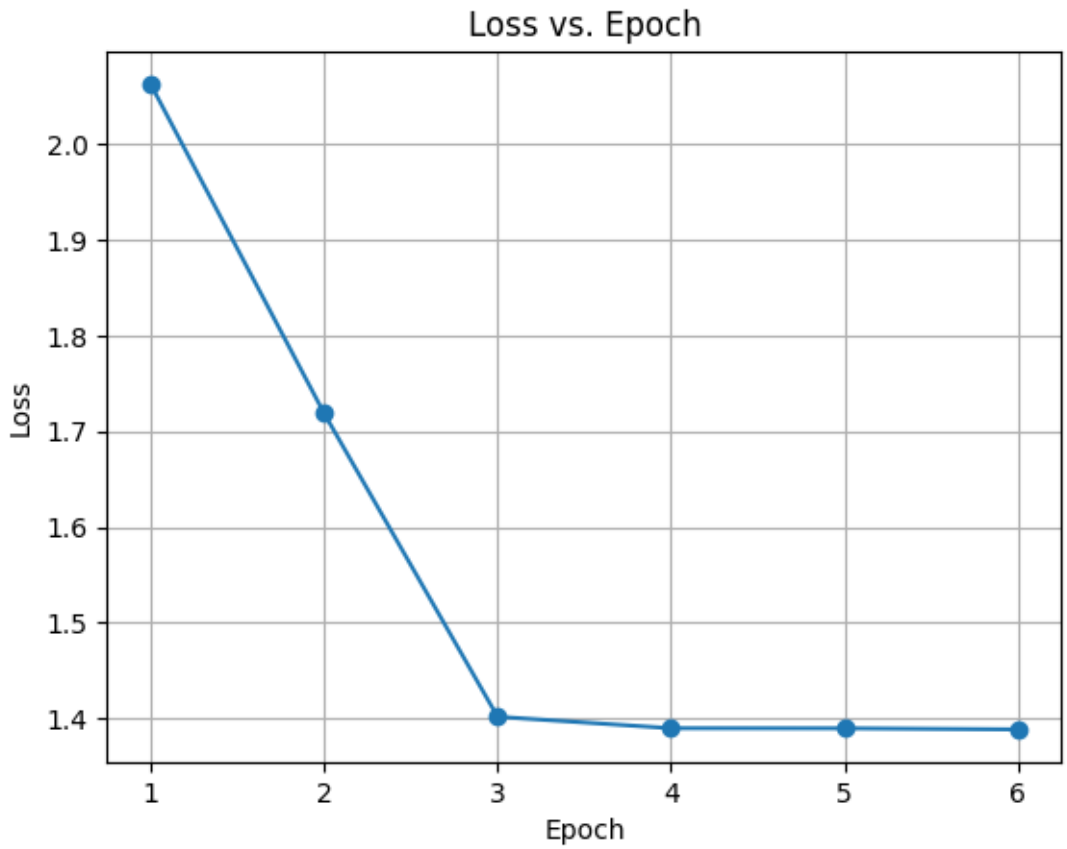
Loss: 1.3901209

Epoch [5/6]: 100%|| 938/938 [03:06<00:00, 5.04it/s]

Loss: 1.3900101

Epoch [6/6]: 100%|| 938/938 [03:08<00:00, 5.03it/s]

Loss: 1.3889182



But the BLEU scores on test dataset fell slightly :-

BLEU Scores on test :- 0.8572031713517242

BLEU score on train :- 1.04848397304034

So it seems lr over 0.01 overfits the model, so lets keep that constant.Now lets change the numbers of heads to 12, with batch size of 50.

Epoch [1/6]: 100%|[redacted]| 938/938 [03:07<00:00, 5.01it/s]

Loss: 2.0319342

Epoch [2/6]: 100%|[redacted]| 938/938 [03:06<00:00, 5.03it/s]

Loss: 1.7214201

Epoch [3/6]: 100%|[redacted]| 938/938 [03:06<00:00, 5.03it/s]

Loss: 1.4012196

Epoch [4/6]: 100%|[redacted]| 938/938 [03:07<00:00, 5.04it/s]

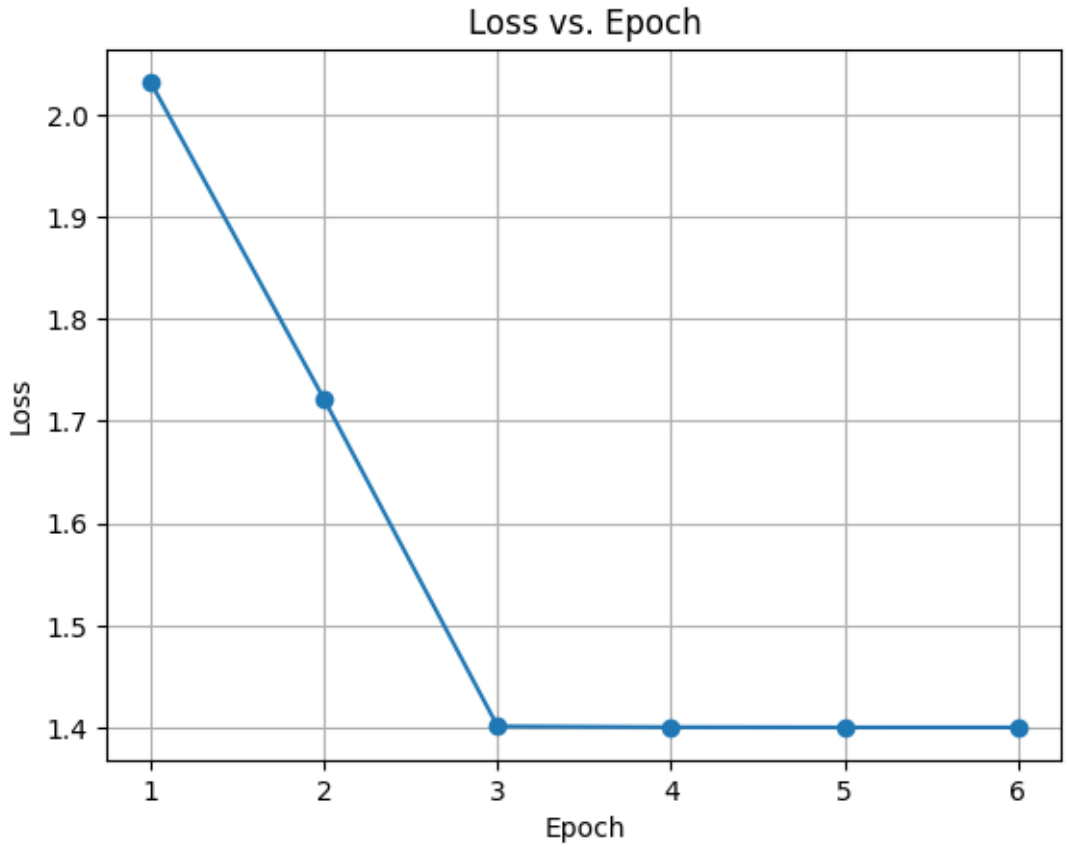
Loss: 1.4004181

Epoch [5/6]: 100%|[redacted]| 938/938 [03:06<00:00, 5.01it/s]

Loss: 1.4002111

Epoch [6/6]: 100%|[redacted]| 938/938 [03:08<00:00, 5.01it/s]

Loss: 1.4001823



BLEU Scores on test :- 0.85965871618903

BLEU score on train :- 1.05011488308631

And finally on keeping batches of 32 best results were obtained

Epoch [1/6]: 100%|[redacted]| 938/938 [03:07<00:00, 5.01it/s]

Loss: 2.0318121

Epoch [2/6]: 100%|[redacted]| 938/938 [03:06<00:00, 5.03it/s]

Loss: 1.7192012

Epoch [3/6]: 100%|[redacted]| 938/938 [03:06<00:00, 5.03it/s]

Loss: 1.3918108

Epoch [4/6]: 100%|[redacted]| 938/938 [03:07<00:00, 5.04it/s]

Loss: 1.39857376

Epoch [5/6]: 100%|██████████| 938/938 [03:06<00:00, 5.01it/s]

Loss: 1.3932095

Epoch [6/6]: 100%|██████████| 938/938 [03:08<00:00, 5.01it/s]

BLEU Scores on test :- 0.8596587223105

BLEU score on train :- 1.0501148421942