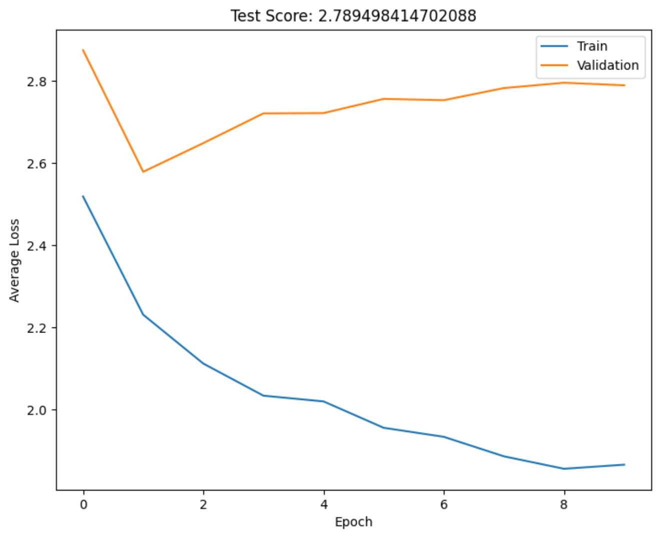
Transformer Report

Hyperparameter Testing

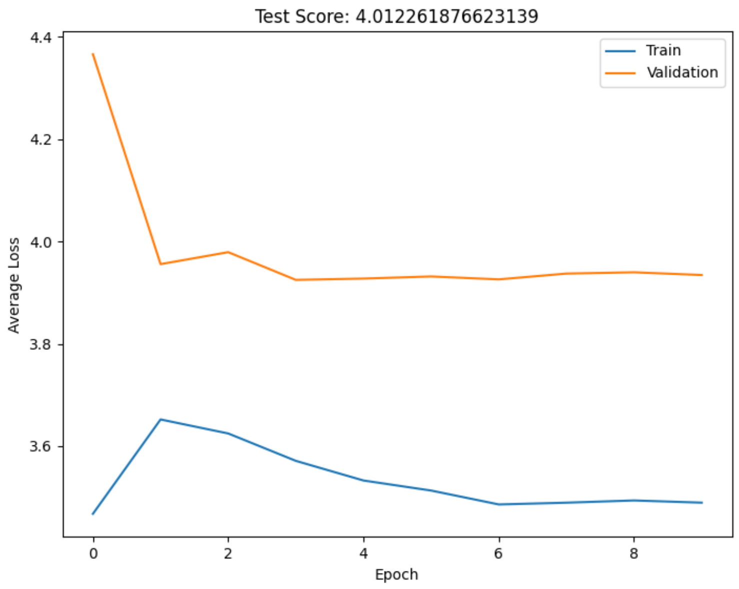
Epochs 10 - Heads 2 - Dropout 0.1 - Learning Rate 10^-3 - Batch Size 10 - Embed Size - 300

The less number of heads may be responsible near stable validation loss



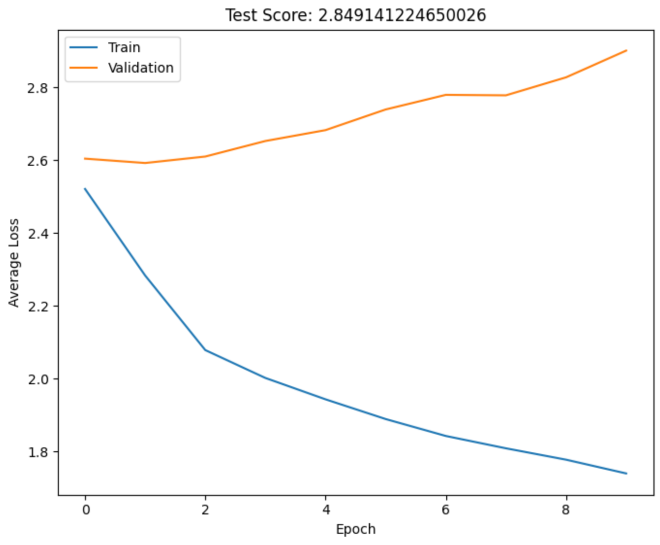
Epochs 10 - Heads 8 - Dropout 0.1 - Learning Rate 10^-3 - Batch Size 10 - Embed Size -512

Here the embedding size was increased and also the number of heads. The configuration is identical to the original “Attention is All you need” paper. Looks like due to the large number of parameters, it requires more than 10 epochs to converge, although the train and val loss seem to be flatlining.



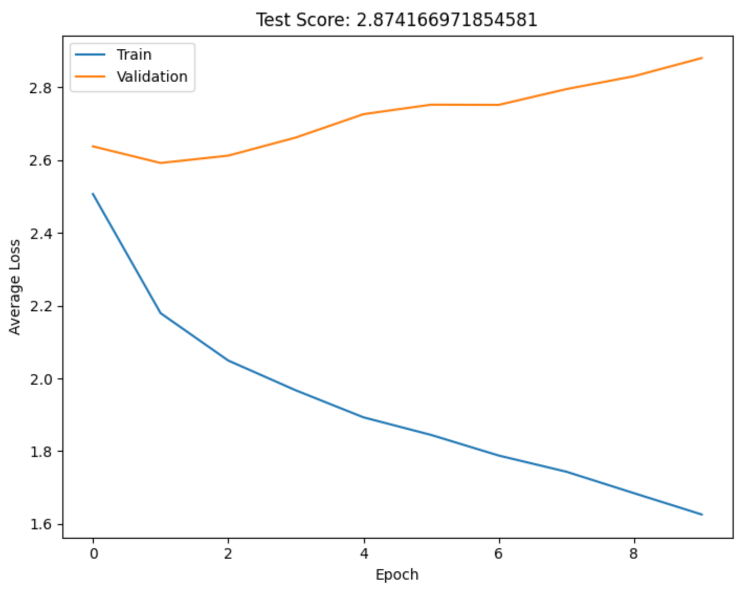
Epochs 10 - Heads 6 - Dropout 0.1 - Learning Rate 10^-3 - Batch Size 10 - Embed Size -300

The validation loss increases marginally due to more heads, and thus be overfitting to the data



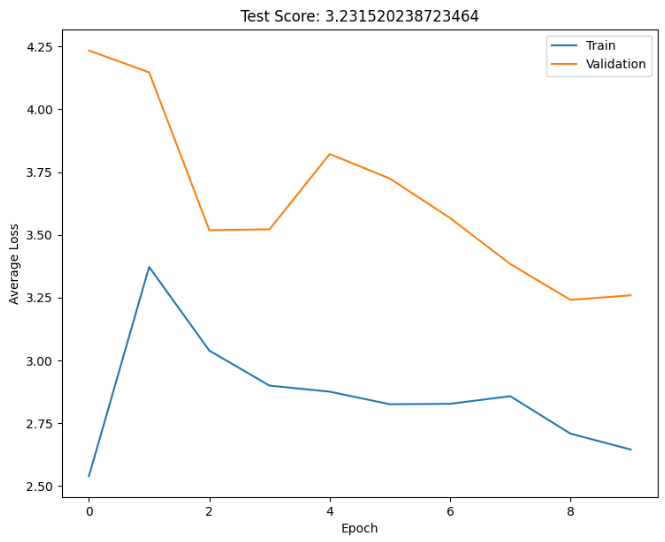
Epochs 10 - Heads 6 - Dropout 0.3 - Learning Rate 10^-3 - Batch Size 10 - Embed Size -300

Changed the dropout to 0.3, but the model still seems to be overfitting.



Epochs 10 - Heads 4 - Dropout 0.1 - Learning Rate 10^-3 - Batch Size 10 - Embed Size -300

Here the number of heads seems perfect for the data, as validation continues to decrease along with train loss. The validation loss is not too far away from the train loss, indicating it hasnt overfit the data. The test loss is also **3.23**

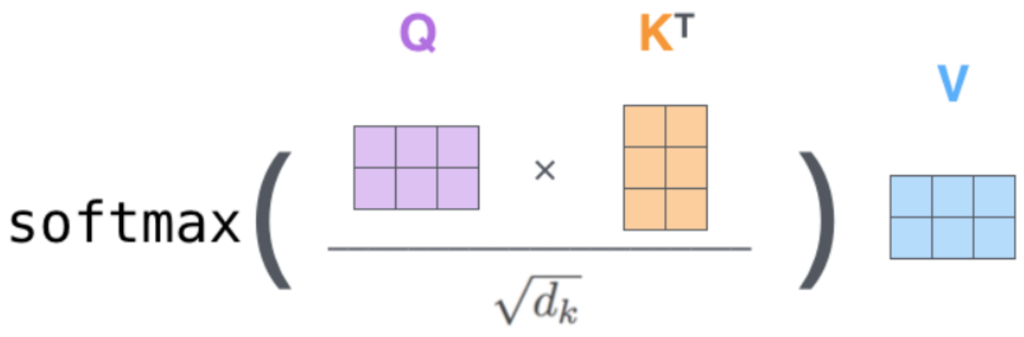


The model config highlighted in Red was chosen to be the best model after hyperparameter tuning.The number of heads may indicates the number of ways/angles the model can pay attention to the words, separate Q,K,V matrices for each head, hence separate attention. Dropout prevents overfitting.

All these models had internal feed-forward NN size as 1024

A1

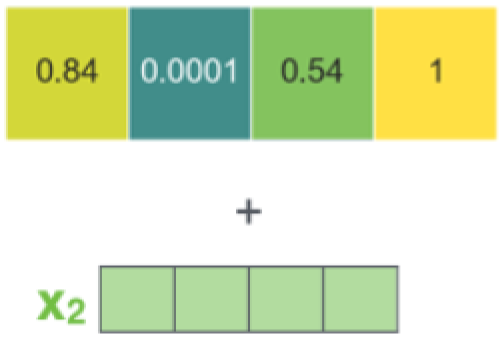
Self Attention helps the model categorize and quantify what part of the sentence is important for that particular word. Each word in the sentence essentially pays attention to other words in the sentence and aggregates contextual information. Each words asks what information is relevant to me in this sentence. Self Attention helps the transformer in capturing long range(since the word **directly** pays attention to a word far away, unlike in RNNs) and contextual dependencies in the sentence. Self Attention helps in weighing the words in the sentence wrt to a particular word in the sequence, hence effectively paying attention dynamically to different words.



Self attention is highly parallelizable, in contrast to Sequential Language models like RNNs and LSTMs, since each word at each time step pays attention to the entire sentence in one go instead of aggregating information. This also helps in paying attention bi-directionally, where weights are given to words before and after the word in focus. This parallizability helps transformers scale well to train on enormous amounts of data, yielding very good language models.

A2

Transformers use positional encodings in addition to word embeddings because they lack the inherent understanding of the order or position of words in a sequence. Unlike recurrent neural networks (RNNs) or convolutional neural networks (CNNs), transformers process input sequences in parallel, which makes them highly efficient but doesn't naturally capture sequential order. Positional encodings are used to address this limitation by providing the model with information about the position of each element in the sequence.



Positional Encodings are calculated for every position in the sequence and are added to every word in the sequence, thus imparting positional information into the embedding of a word at a particular position. The positional encodings can be calculated in many ways, but the two most popular ways are

Sin and Cosine Positional Encoding

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

pos - position of the word

i - index of an element in the word embedding / dimension index

Learned Positional Encodings - In this a learnable vector is added to the word embedding, this learnable vector through training encodes the positional information of the word.