The Transformer (From Scratch)

Advanced NLP : Assignment 2

Varun Edachali (2022101029)

October 2, 2024

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

1 Theory

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

1.1.1 What is the purpose of self-attention?

Attention helps models focus on the most relevant portions of input data. It is a machine learning method that determines the relative importance of each component in a sequence relative to other components in that sequence. Essentially, we weigh each component of a sequence and use their sum as a context vector, to create a more rich context for the prediction at the current step. This is as opposed to simply passing the hidden state of the previous step as input to the next, which can lead to the loss of information from far before.

Self Attention is a mechanism used in ML, to capture such dependencies and relationships *within* the input sequence. It allows the model to identify and weigh the different parts of the input sequence by attending to itself. This provides better context to create better predictions at the current step.

Sources

- Wikipedia
- h2o.ai

1.1.2 How does it facilitate capturing dependencies in sequences?

The context vector (mentioned above) is computed as a weighted sum of the *values*, where the weight assigned to each value is computed by a compatibility function of the *query* with the corresponding *key*. We calculate the score between the query and the key, sometimes called the *energy* through dot-product or otherwise. We then apply a softmax to normalise the score of each query across all the keys. These values then represent the weights to multiply the values with to yield the context vector.

Note that this is a super-set of a simple RNN context mechanism, as if we set all of the weights (currently given by $softmax(\frac{QK^T}{\sqrt{d}})$) to 0, apart from the last to 1, then we are using the context only from the last state.

Thus, we capture the dependencies by learning matrices W_q, W_k, W_v which are multiplied with the input to yield our keys, queries and values as above. The former two are multiplied to yield weights that are then multiplied with the latter to yield our attention scores, telling our model how much to focus on each part of the input.

The primary reason for the weight vectors is to yield optimal values of Q, K and V to optimally represent important parts of the input sequence, given the input.

Source

- StackExchange - qmzp, Sam Tseng
- StackExchange - mon

1.2 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.

1.2.1 Why do transformers use positional encodings in addition to word embeddings?

We need positional information because the meaning of entire sentences can change if the words are re-ordered.

In regular RNN models, we process the word at a given step only by taking the input of the previous step as input, so positional information is implicitly encoded into our process. However, in transformer blocks, we pass all of the inputs in parallel through the network for efficiency. A consequence of this approach is that each word may not be aware of its position, which can lead to faulty encodings and decodings. Thus, we explicitly augment the positional information, for more accurate processing.

Sources

- Slides
- Machine Learning Mastery

1.3 Explain how positional encodings are incorporated into the transformer architecture.

Positional encodings are incorporated into the transformer architecture by simple addition. Based on the paper, the reason for addition in particular (and not, say concatenation) seems to just be for simplicity. The transformer model can likely infer which sections represent the positional encoding and which the words themselves after training.

In the original paper, the positional encoding is represented using a sinusoidal representation.

$$PE(pos, 2i) = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

This may allow the model to generalise better to longer sequences than those encountered during training.

1.4 Briefly describe recent advances in various types of positional encodings used for transformers and how they differ from traditional sinusoidal positional encodings.

- Learned Position Embeddings: instead of deterministic sinusoidal embeddings, we allow the model to learn a unique embedding for each position during training. This can lead to task specific embeddings, and may be better on large task-specific datasets.
- Relative Positional Encodings: focus on the relative distance between tokens rather than their absolute positions. This allows the model to understand how far apart tokens are from each other. The difference is that sinusoidal encodings provide fixed positional information, which relative encodings enable the model to generalise better across different sequence lengths and variations by considering relative distance.

Source slides

2 HyperParameter Tuning

2.1 Analysis

Note The default values taken for the hyper-parameters in consideration are:

- *Dropout Rate:* 0.3
- *Embedding Dimension:* 240
- *Number of Layers:* 3
- *Number of Heads:* 2

These values are analysed by altering them one at a time, as below.

Each graph has the BLEU score of the test-set along the Y-Axis, and the value of the parameter in consideration in the X-Axis.

2.1.1 Dropout Rate

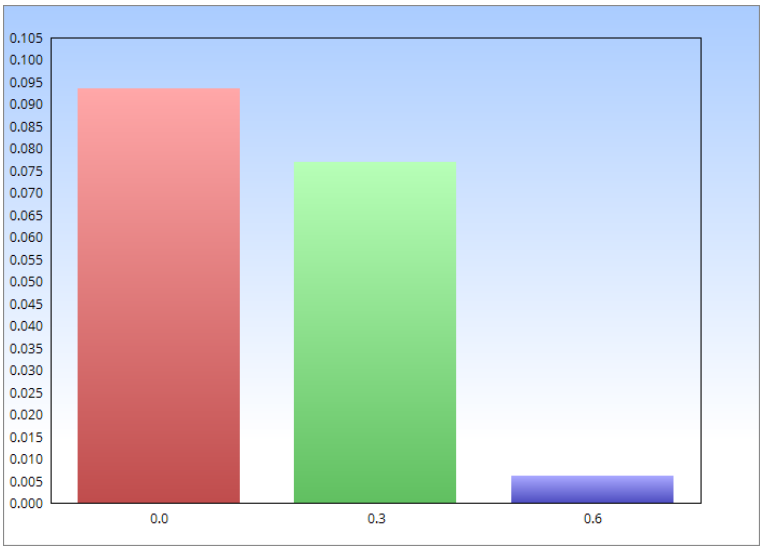

Figure 1: BLEU Score vs Dropout Rate

For this particular test set, a dropout rate of 0 yielded better results than 0.3. This likely points to a lack of diversity between the train, eval and test sets. Still, if we analyse the performance at the various epochs, we notice that a training rate of 0.3 leads to better results on the evaluation set, even over time.

With a dropout rate of 0, the model never improved on its loss on the evaluation set from the very first epoch. Thus, for longer-term training, I believe that the best choice would be something closer to 0.3 than 0.0, as it achieves good results without overfitting. I will invoke a loss graph here because it summarises my point:

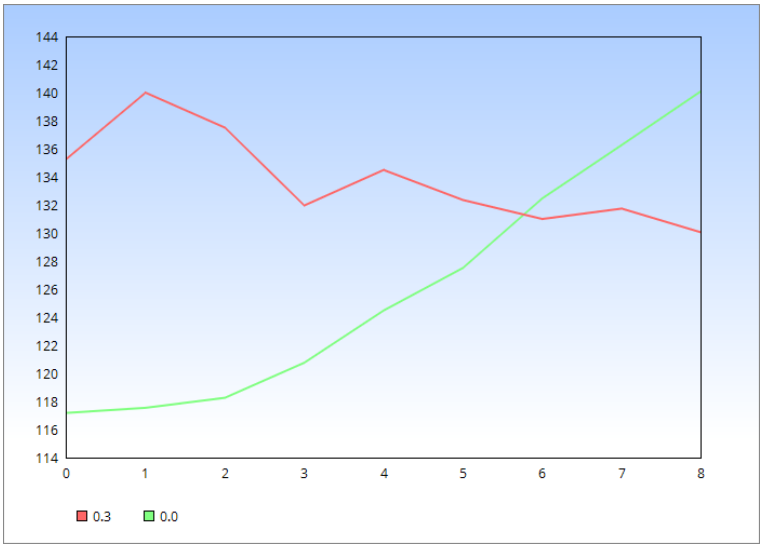

Figure 2: Eval Loss vs Epoch for various Dropout Ratios

Thus, a larger dropout ratio prevents overfitting and yields better and better results with time. I believe that with more time, if I could train the model on more epochs, a dropout ratio closer to 0.3 would yield better results on the data, despite the immediate data suggesting 0.0 is better, because of the above.

Final Choice: 0.15-0.25

2.1.2 Number of Heads

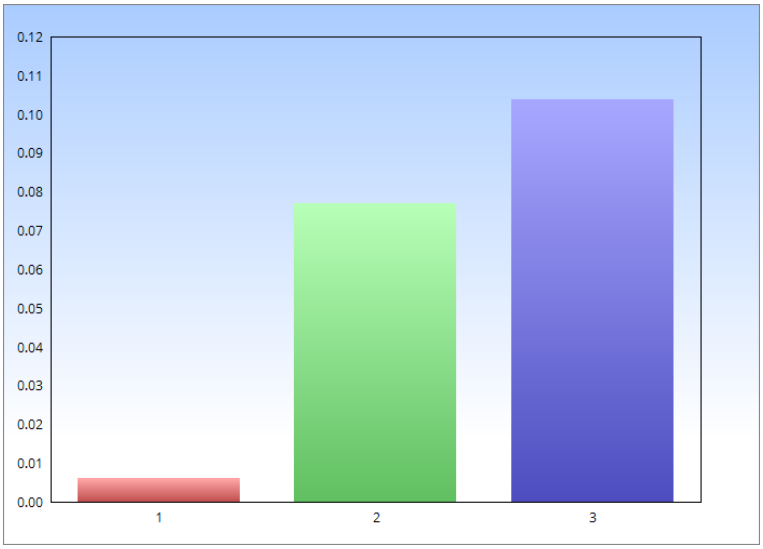

Figure 3: BLEU Score vs Number of Heads

We see very clearly that an increase in number of heads yields better results, likely pointing to more relations and context being captured within the input sequence by the various heads. I would choose as many heads as possible, subject to time and memory constraints. For this purpose, I would stick with 3 to 4 because I know it would yield results. Again, increasing it too much could lead to overfitting due to a large number of parameters.

Final Choice: 3-4

2.1.3 Number of Layers

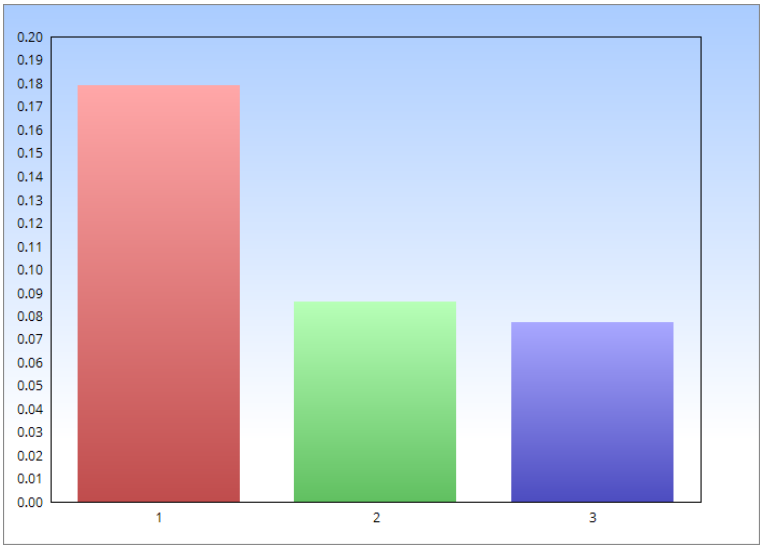

Figure 4: BLEU Score vs Number of Layers

We see much, much better performance when using a single layer than using multiple. This likely again points to a lack of diversity between within the training data. Using multiple layers may be leading to over-fitting on this training data due to an increase in the parameters to tune. For this particular dataset, I would stick to a single layer.

Final Choice: 1

2.1.4 Embedding Dimension

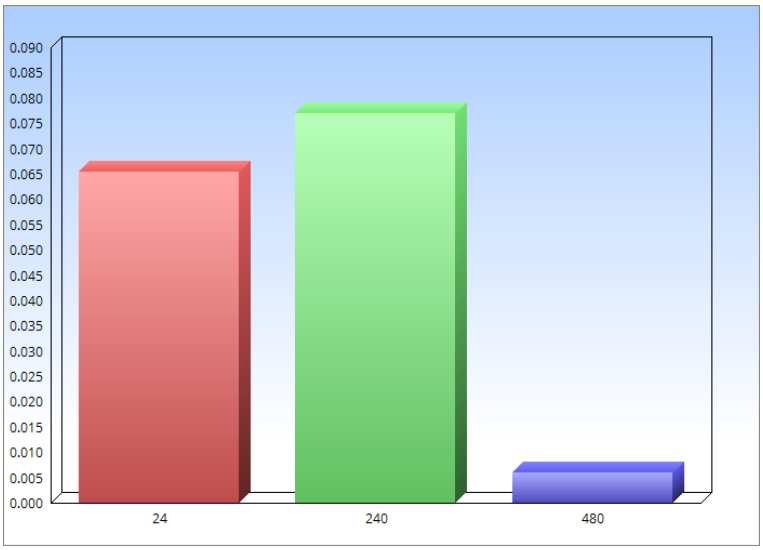

Figure 5: BLEU Score vs Word Embedding Dimension

The embedding dimension of 24 is likely not vast enough to adequately encode the entire vocabulary space, leading to better results with an embedding dimension of 240. A much larger embedding dimension probably introduces too many learnable parameters, leading to overfitting on the input data, and ineffective learning elsewhere.

I would have liked to experiment with, say, 120 based on these results, but I hit a time constraint.

The conclusion we can draw from this is that our vocabulary is not very diverse, as we can represent our data well even in a low-dimensional latent.

Final Choice: 180-240

2.2 Justification

Most of the above models lingered between BLEU scores of about 0.02 to 0.1, with a few breaching 0.1, reaching a **max of 0.17**. I will attempt to combine my learnings from the tuning to yield a better model.

- Using a dropout of 0.22, embedding dimension of 240, 1 layer and 4 heads gave a BLEU score of **0.2021** with about half the training time as previous models. This is already a far better result. In addition, it was continuing to converge even after 10 epochs. If I had the time or compute to train further, I believe it would yield even better results.

- To test the above hypothesis, I let it train for 25 epochs, and it seemed to train better, with the eval loss seeming to converge after about 21 epochs. This led to a BLEU score of **0.2297**, further justifying my analysis. I stopped here.

3 Sources

- Aladdin Person for the overall structure and organisation of classes
- Harvard - Annotated Transformer for the sinusoidal positional encoding (with modifications)
- Attention is All You Need for the understanding required to write custom train and test loops