Project 2

Peter Williams

November 29, 2025

1 Introduction

For this project, the task was to implement a sequence-to-sequence transformer model that can recover original sentences from processed text. In pursuit of that task, our goal was to expand the existing model and tokenizer code to include code that would process the dataset, and run experiments so that we could evaluate performance of the model with various hyperparameter and model configurations.

Broadly, my approach to this project was to first build out the dataset class, and test that the transformer model and dataset worked well together. After verifying initial compatibility, I ran a grid search over several combinations of model parameters to find the best set of base hyperparameters to use for later experiments, including variations in model depth, dimension, number of attention heads, the feedforward dimension, and batch size. With the grid search completed, I ran three sets of experiments: one comparing fixed sinusoidal functions and learnable positional encodings; one comparing greedy decoding and beam search for model generation after training; and two comparing the effect of the number of attention heads and the number of layers in the encoder/decoder on model performance. The main metrics used for evaluation across all experiments were performance on a held-out test set and BLEU score.

2 Methods

2.1 Transformer

At a high level, a transformer consists of an encoder and decoder, each of which is made up of several encoder or decoder layers. First, some input is passed into the encoder after being embedded into some feature representation, and being combined with some positional encoding. Then it progresses through the encoder layers, which are described below. Then the output is embedded and positionally encoded before being passed through the decoder layers. Finally, there is another linear layer and softmax to get output probabilities for the vocab and output some token which is chosen by a greedy or beam search to select from among the most likely tokens, as defined by the softmax's probability distribution.

Each layer in the encoder is made up of a self-attention layer which then adds and normalizes its output with the input, which is passed into a feed-forward layer, which is again added and normalized before passing to the next encoder layer. Each decoder layer works the same, except with an additional layer of encoder-decoder cross-attention and normalization between the self-attention and the feed-forward network.

The transformer uses two types of attention, self-attention, and cross-attention, which primarily differ in their inputs, although both function similarly. I will describe self-attention first, and then show how encoder-decoder cross-attention differs. Each attention layer has three sets of weights, labeled the key (K), query (Q), and value (V) matrices, which are used to create three different vectors from the input sequence vectors. Then a score for each word pair in the input is calculated by taking the dot product of the key and query vector for each word, which essentially gives another vector of scores for each word compared to each other word in the input. Those scores are scaled by the square root of the dimension of the key vectors used, and then the softmax is calculated to determine how much each word relates to the word at that position. Finally, the softmax output is multiplied with the original value vector. The transformer builds on this by using multi-headed attention, so that there are multiple heads, or sets of KQV weights, which then let the layer learn to attend to multiple different things in the input sequence at once.

The encoder-decoder cross-attention in the decoder layer differs from self-attention primarily by looking at how the output sequence attends to the words in the input sequence. So the key and value vectors come from the output sequence that the decoder is processing, and the query vector comes from the input sequence that the encoder processed. This allows the output to attend to the words in the input sequence, in addition to the words around it. Additionally, the output sequence uses masking to make the output sequence more usable in actual generation. Because the transformer uses attention for parallelization, it expects a full sequence of the max length, which can be achieved by padding the input. However, for actual generation, the model won't have the tokens that occur "after" the current target token, so the model uses masking to "cover" each token that occurs afterwards in a special token "¡MASK¿" so that the model isn't learning to predict the current word based on future words, only based on the previous words in the output sequence, although the input can use the whole sequence.

2.2 Implementation

The main transformer module was already implemented for this project using PyTorch. Each piece of the model as described above was organized into individual classes that were combined into the final encoder-decoder architecture according to standard PyTorch practices, including initialization and forward functions. Additionally, greedy decoding and beam search functions were also provided for later experimentation. One modification I made to the encoder and decoder classes is to add the option to use learnable positional encodings instead of fixed positional encodings. By passing in a boolean upon model initialization, the user can decide which of the two to use in both the encoder and decoder.

To format the dataset so that it is acceptable input for the model, I built a dataset class that inherits from PyTorch's Dataset class. When initialized with a file, tokenizer object, and max sequence lengths, the class loads the raw data from the given file, tokenizes and encodes each sequence pair, and then trims and/or pads each sequence so that they all include special tokens for start and end of sentence, as well as a padding token so that each sequence is the same max length.

With the dataset prepared for training, I then implemented functions for training a single epoch, testing a single epoch, and computing the BLEU score. Training and testing follow standard practice for a PyTorch model: for training, set the model to train, move it to the device, initialize the loss, and then iterate through batches in the dataloader, updating the loss and propagating it backward through the model; evaluation follows the same steps with the model in evaluation mode and without propagating the loss back through the model. The function to compute BLEU score functions similar to evaluation, except just using the model's generate function to generate output. That output is then decoded and special tokens are filtered out, and that hypothesis is compared to the target output using the NLTK package to compute the BLEU score value for each sample.

With core training and evaluation methods built, I implemented further helper functions combining these into larger functions to train the whole model for as many epochs as desired, then evaluate that model by computing loss on the test set and generating example output for qualitative analysis, and finally to save the model, results, and loss curve. I used all of these functions to then run the grid search, and all three experiments.

For the grid search, I compared 72 total combinations, experimenting with the batch size, model dimension, number of attention heads, dimension of the feedforward network, and number of layers for the encoder and decoder. The possible values are shown in Table 1. A model was trained with each possible hyperparameter configuration combination for 5 total epochs, with the same values for dropout, learning rate, and maximum sequence length for each model (0.1, 0.001, and 50, respectively).

Table 1: Grid Search Hyperparameter Options

Hyperparameter	Options
Encoder/Decoder Layers	1, 2, 4
Model Dimension	128, 256
Number of Attention Heads	2, 4, 8
Feedforward Dimension	256, 512
Batch Size	32, 64

In experiment one, I compared the effect of different positional encodings on model performance. Using the same hyperparameters as the best model from the grid search, I trained a model using fixed sinusoidal encodings (which were also used for each model in the grid search), and then trained a model using learnable positional encodings. I compared the performance of the two models using performance on the test set, measured using loss and BLEU score.

Next, I compared the effect of greedy decoding and beam search for generation. I started by training a model with the same hyperparameters as the best model from the grid search. Using that model, I then generated output using both greedy decoding, and beam search decoding with a beam width of 3, and computed BLEU score for each type of generation. I also compared the time taken to generate output and compute BLEU score for each approach. Because of the time required for generation, I only compared greedy and one beam search instead of testing multiple beam widths.

The final experiment is similar to the grid search earlier, but only comparing the performance of models trained with several options for number of attention heads and model depth. Using the best hyperparameters for all but those two parameters, I trained three models experimenting with the number of attention heads (2, 4, or 8), and three more models experimenting with the number of encoder/decoder layers (1, 2, 4). Within each category, I compared the performance of each model using BLEU score and test set loss.

3 Results

3.1 Grid Search

Table 2: Top 10 Transformer Model Configurations from Grid Search

Test Loss	Batch Size	Model Dim	Number of Heads	Feedforward Dim	Transformer Layers
0.617	32	256	8	256	4
0.62	64	256	4	512	4
0.622	64	256	8	256	4
0.629	32	256	8	512	4
0.63	64	256	8	512	4
0.643	64	256	2	512	4
0.651	32	256	4	256	4
0.651	64	256	2	256	4
0.657	32	256	2	512	4
0.657	32	256	4	512	4

Because I had 72 hyperparameter configurations that I tested, only the top 10 are included in the table below. Model performance was measured by calculating the loss on the test set after training.[1] Each configuration trained for 5 epochs, with a learning rate of 0.001, and dropout of 0.1. As Table 2 shows, the hyperparameter configuration that performed the best according to loss on the test set has a batch size of 32, model dimensions of 256, 8 attention heads, a feedforward network dimension of, and 4 layers each for the encoder and decoder. The training and dev set loss curves for this model are shown in Figure 1.

3.2 Experiment One

The loss curves for each model from experiment one are seen in Figure 2 and Figure 3. Final performance metrics on the test set, including loss and BLEU score, as well as the total number of parameters for the model are in Table 3. As shown, both models converged with very similar loss and BLEU scores on the test set. The model with learnable positional encodings performed slightly better than the sinusoidal encodings, by 0.007 on test loss and 0.004 on BLEU score, but it also has nearly 26 thousand more parameters.

[1]BLEU scores should roughly correlate with test loss; there was a bug with its calculation during the grid search, and generation and calculation take too long to run it again for the grid search. It was fixed for all remaining experiments.

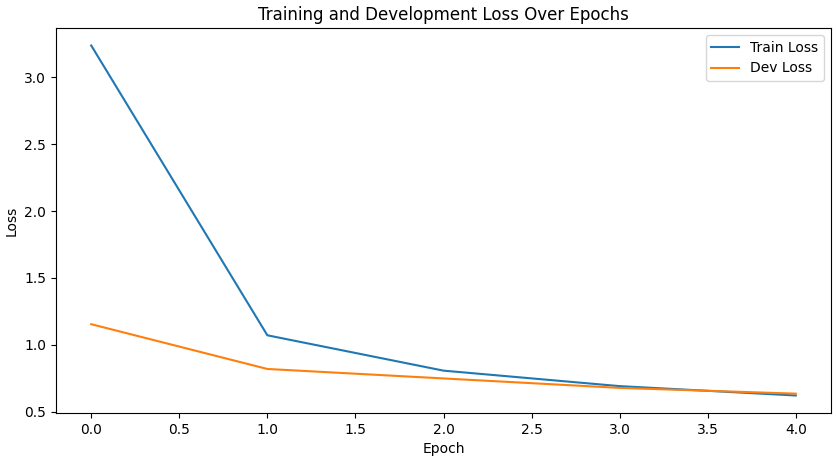

Figure 1: Training and dev set loss curves for the best model configuration from grid search

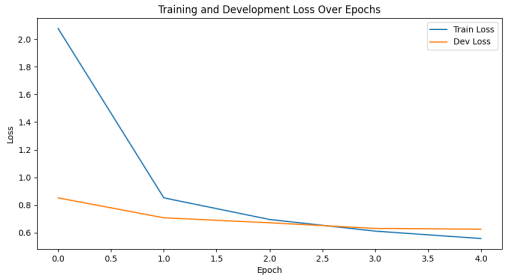

Figure 2: Training and dev set loss curves for model trained with learnable positional encodings

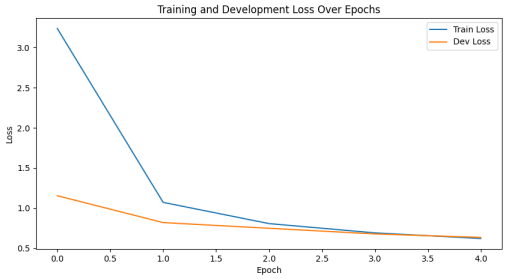

Figure 3: Training and dev set loss curves for the model trained with fixed sinusoidal positional encodings

Table 3: Comparison of Positional Encoding Methods

Method	Test Loss	BLEU Score	Parameter Count
Sinusoidal	0.6394	0.8567	31,863,657
Learnable	0.6324	0.8520	31,889,257

Table 4: Example Outputs from Sinusoidal vs. Learnable Positional Encoding Models

Input	Reference	Sinusoidal Output	Learnable Output
old married man – quite good for nothing .	an old married man – quite good for nothing .	old married man – quite good for nothing .	the old married man – quite good for nothing .
length breadth height of it be equal .	the length and the breadth and the height of it are equal .	the length and the breadth the height of it was equal .	lengths the breadth height of it is equal .

3.3 Experiment Two

The model trained for experiment two uses the same hyperparameters as the best model from the grid search; the loss curve is shown in Figure 4. Using that model, I generated model outputs on the test set with both greedy decoding and with beam search with a beam width of three. Example outputs for each approach are shown in Table 6. Comparisons of average BLEU score, sequence length, and time efficiency for each approach are shown in Table 5. Beam search takes significantly longer, but does lead to a slight improvement in BLEU score. In looking qualitatively at the outputs, many of the differences between greedy and beam are very small, although they make different mistakes. Neither approach is significantly better consistently from what I saw looking through examples of the output. Each one can introduce errors that the other did not generate.

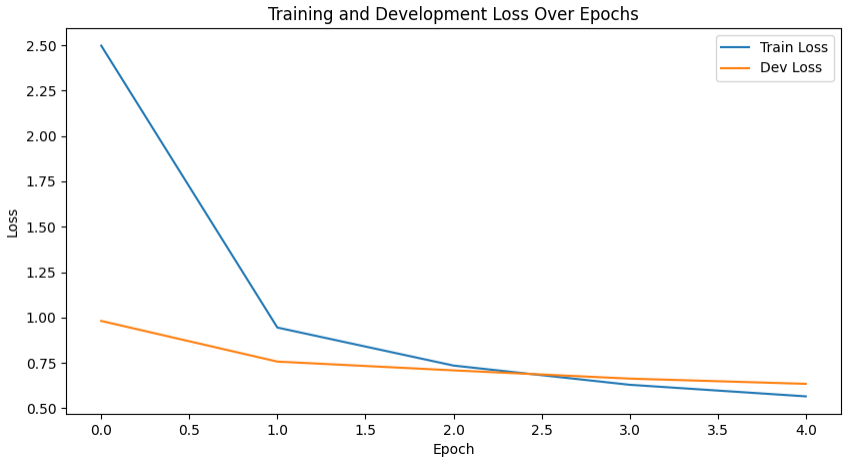

Figure 4: Training and dev set loss curves for experiment two model

Table 5: Decoding Method Comparison

Method	Test Loss	BLEU Score	Decoding Time (sec)	Average Sequence Length
Greedy Decoding	0.6394	0.8567	993.01	22.28
Beam Search (3)	0.6394	0.8576	4965.95	22.21

3.4 Experiment Three

The test loss and BLEU score for the models trained with variable number of attention heads are in Table 7. As shown, performance increases as the number of heads increases, although this also necessarily increases the number of computations performed.

The test loss and BLEU scores for the models trained with variable model depth are in Table 8. Again, the models with more layers perform better, both in learning measured by test set loss and in generation

Table 6: Representative Examples: Greedy vs. Beam Search Outputs

Input	**Reference**	**Greedy Output**	**Beam Output**
at least if be otherwise , there be not want other motif much more influential with him .	or at least if this were otherwise , there were not wanting other motives much more influential with him .	at least if this be otherwise , there is not wanted the other motives much more to enable with him .	at least if this be otherwise , there is not wanted the other motives much more to enable with him .
4 : 8 for gird you with sackcloth , lament howl : for fierce anger of lord be not turn back from u .	4 : 8 for this gird you with sackcloth , lament and howl : for the fierce anger of the lord is not turned back from us .	4 : 8 for gird you with sackcloth , lament howling : for fierce anger of the lord is not turned back from us .	4 : 8 for gird you with sackcloth , lament howling : for fierce anger of the lord is not turned back from us .
length breadth height of it be equal .	the length and the breadth and the height of it are equal .	the length and the breadth the height of it was equal .	the length breadth the height of it was equal .
14 : 35 if they will learn thing , let them ask their husband at home : for it be shame for woman speak in church .	14 : 35 and if they will learn any thing , let them ask their husbands at home : for it is a shame for women to speak in the church .	14 : 35 if they will learn any thing , let them ask their husband at home : for it is a shame for a woman to speak in the church .	14 : 35 if they will learn any thing , let them ask their husband at home : for it is a shame for a woman to speak in the church .
he have find thing which modern people call impressionism , which be name for final scepticism which can find floor universe .	he had found the thing which the modern people call impressionism , which is another name for that final scepticism which can find no floor to the universe .	he had found the thing which the modern people called the impressionism , which is named for the final scepticism which can find the floor universe .	he had found the thing which the modern people called the impressionism , which is named for the final scepticism which can find the floor universe .
piedro saw that his father be in passion , know that he be beaten because he be find out be rogue , rather than for be one .	piedro saw that his father was in a passion , and knew that he was beaten because he was found out to be a rogue , rather than for being one .	piedro saw that his father was in passions , knowing that he was beaten because he was found out to be a rogue , rather than for being one .	piedro saw that his father was in a passion , knowing that he was beaten because he was found out to be a rogue , rather than for being one .
it be high time go , for pool be get quite crowd with bird animal that have fall into it : there be duck dodo , lory eaglet , several other curious creature .	it was high time to go , for the pool was getting quite crowded with the birds and animals that had fallen into it : there were a duck and a dodo , a lory and an eaglet , and several other curious creatures .	it was high time to go , for the pool was getting quite a crowd with the bird animal that had fallen into it : and there was a duck dodo , and the lory to the eaglet , several other curious creatures .	it was a high time going , for the pool was getting quite crowd with the bird animal that had fallen into it : and there was a duck dodo , and the lory eaglet , several other curious creatures .
i be sure she saw me , she look away directly , take notice ; they go quite farther end of shop ; i keep sit near door !– oh !	i am sure she saw me , but she looked away directly , and took no notice ; and they both went to quite the farther end of the shop ; and i kept sitting near the door !– oh !	i am sure she saw me , she looked away directly , and took notice ; and they went quite farther end of the shop ; and i kept sit near the door !– oh !	i am sure she saw me , she looked away directly , and took notice ; and they went quite farther end of the shop ; and i kept sit near the door !– oh !
you rascal !	you rascal !	you rascal !	you rascal !

measured by BLEU score.

Table 7: Effect of Number of Attention Heads on Model Performance

Number of Heads	Test Loss	BLEU Score
2	0.6754	0.8463
4	0.6509	0.8560
8	0.6377	0.8571

Table 8: Effect of Transformer Depth on Model Performance

Depth	Test Loss	BLEU Score
1	0.7964	0.8381
2	0.6780	0.8537
4	0.6310	0.8590

4 Analysis

Overall, the project went fairly well. There were a few issues early on with the implementation of the training loops, particularly in computing the BLEU score, and in setting up cuda correctly. I figured out how to get cuda set up with my computer in time to use my GPU for the grid search, and for all further experiments, which significantly reduced the time required for training, and allowed me to run the grid search at all. The BLEU score function was incorrectly passing in the reference as a single list instead of a list of lists only for the initial grid search, as mentioned above, and was fixed for the later experiments that depended on BLEU score for the results and analysis. One major thing that made the experiments run smoothly was further modularizing the code required for training and evaluating each model, beyond even the functions to train and evaluate a single epoch. I implemented additional functions for training a model, evaluating a model, creating the datasets, saving the results, and a function for the grid search and each experiment. The major options were then added to the main function as command line arguments, which made running different configurations easy and repeatable.

From the experiment one results, it may seem at first glance that adding additional parameters in the form of learnable positional encodings would be worth the extra performance, but there are some tradeoffs that need to be considered for each potential application. First, the additional performance is a very small percent of the overall performance at the cost of nearly 26 thousand more parameters. Additional parameters may enable the model to learn patterns positionally that are not there by using the sinusoidal encodings, but at the cost of additional space and training time, if it even improves performance at all, which likely would need to be considered and tested for each model. Additionally, by looking at some of the example outputs, the learnable model's output doesn't consistently outperform the sinusoidal model, or avoid/fix certain problems that might occur more frequently in the sinusoidal output. It would require more in-depth exploration of sample outputs and careful consideration of a task's requirements to decide whether sinusoidal or learnable positional encodings are more beneficial to the task at hand. At least for this task, it seems that the learnable positional encodings may not be worth the additional memory, compute power, and time required to train the model using them.

The experiment two results lead us to a similar conclusion. The beam search results show a slight improvement in model performance on the test set, as measured by loss and BLEU score, but it also takes nearly five times as long to complete the generation and decoding on the test set. It's possible that beam search with a wider beam would lead to further improvements, but it also would likely increase the time even more. With limited time and compute power, I couldn't test further widths with the beam search. As it stands, it appears that beam search may lead to some improvements in generation, but at the cost of significantly more time, which may not be worth it for every task, which can be verified further by examining sample outputs for each new task the model is applied to. For this task, it appears that greedy decoding performs well enough and significantly faster to continue using a greedy decoding method over the beam search.

In experiment three, the results of the tests seem more straightforward to interpret. It seems clear that increasing the number of attention heads or increasing the number of layers increases the model's performance, as shown by both the test set loss and the BLEU scores. However, as the model gets deeper or adds more heads, the performance gain decreases in both metrics. Again, as in the first two experiments, it seems that whether the additional depth or heads are worth the additional computations required is a matter of testing the model for each specific task. If the task depends on every bit of possible performance, then increasing the depth and number of heads to eke out every possible gain may be useful; if the task is accomplished to a satisfactory degree with fewer layers or heads, then it may be worth the fewer computations required by decreasing the model size. With the rest of the training configuration, the model didn't exhibit any signs of overfitting, but that is another thing to be aware of when training the deeper models for a longer time.

5 Conclusion

In this project, I experimented with several different hyperparameter configurations in a grid search, and specific model architecture configurations in three different experiments: fixed sinusoidal vs learnable positional encodings, greedy vs beam search decoding, and model depth in number of encoder/decoder layers, and number of attention heads. I found that learnable positional encodings, beam search, and deeper models all perform better than their counterparts, but at the expense of more computations or time for minimal improvements on this task.

The main limitations of this project were the lack of compute power and time. Compute power especially limited the number of models that could be trained in the grid search, which had a number of different parameters that were held the same across all experiments instead of having options to compare. Several of the "best" hyperparameter options, as tested by the grid search, were also at the high end of the range of values tested for that hyperparameter, so further experiments would use values above that high end to check if performance continues to improve. This was also evident in experiment three, where the deepest model and the model with the most attention heads performed the best; further experiments would test even deeper models and even more attention heads to find where performance begins to decrease or have greatly diminished returns in performance for the compute power used. Limited time mostly constrained experiment two and any other testing involving generation, decoding, and the BLEU score calculation, which took significantly longer than training most other models, since I was able to parallelize the training with cuda. With more time, I would have run beam search decoding with several different beam widths to see how well those results compared to the beam search-width 3 and the greedy decoding.

Overall, the results of these experiments suggest that checking different hyperparameter and model configurations is valuable and necessary for each new task a model is applied to. In particular, some options such as beam search or learnable positional encodings that appear "better" in a vacuum compared to their earlier options like greedy decoding and sinusoidal positional encodings, may have a higher cost in compute power or time required in exchange for minimal improvements to the model's performance on the task at hand.