COSI 231a Project 2 Report

Travis Litke

November 2025

1 Introduction

The experiments in Project 2 were designed to evaluate how hyperparameter and model architecture affect model performance. To complete this project, I implemented the dataset and training/eval loops to include a BLEU scoring evaluation, and then tuned a baseline model by performing a grid-search over 36 unique hyperparameter configurations.

2 Methods

2.1 The Transformer Architecture

The model used in this project is an Encoder-Decoder neural network that trains Multi-Head Attention matrices. Text input is tokenized, encoded, and vectorized, and then passed to the model which begins with the encoding layer(s), where input is given a sinusoidal positional embedding and then passed to a masked self-attention layer. This output goes to a cross-attention layer, and then is passed through a feedforward network and normalized. The decoder layer(s) then use attention scores to generate a sequence of tokens based on a test example. In this case, the model is trained to recover sentences with articles removed and verbs lemmatized. During the training loop, the model learns where articles have been most likely removed, and which verbs are improperly conjugated according to the training examples. This information is reflected in the attention scores that are calculated in the encoder and decoder layers. The outputs from applying attention scores to the input embeddings and output embeddings are passed through a feedforward layer and then activated to return next-token probabilities in the output sequence. The generate function is responsible for either a "greedy" generation where it simply returns the most-probable token, or performs a "beam-search" where it maintains n most probable next tokens and returns the most likely complete sequence.

2.2 Attention Mechanism & Masking

Attention scores start as randomly initialized weight vectors of dimension sequence length x embedding size. Each row of Q, K, and V correspond to token

embeddings. Q is calculated by multiplying the X input matrix by the Q weight matrix, while K is calculated by multiplying the X input matrix by the K weight matrix. Q is then multiplied by K-transpose, and the resulting attention scores are normalized by dividing each element by the square root of the attention dimension, which can be computed by dividing the model dimension by the number of attention heads. This normalization prevents runaway gradients in the softmax function, where all of the attention scores are softmaxed. The V matrix is computed by multiplying input matrix X with a randomly initialized weight vector Wv. The attention scores are multiplied by V. This final matrix is then passed through a linear layer to create an output that can be passed to the rest of the model for error calculation and backpropagation.

Masking is used to cancel out the effect of certain tokens. It could be used to ignore the effects of a padding token. It is also used in autoregressive models to prevent attention scores from being affected by future tokens. So the first token in an input stack can only attend to itself, while the second token can only attend to itself and the token that came before it, so on and so forth. The attention mask is applied over scaled attention scores, before the matrix is multiplied by its Value matrix.

2.3 Hyperparameters and Training Procedures

The first step of this project was to generally tune the model by performing a grid search. The search was performed over all combinations of the following values:

- Model Depth: 1, 2, 4
- Model Dimension: 128, 256
- Attention Heads: 2, 4, 8
- Feedforward dimension: 256, 512
- Dropout: 0.2
- Batch size: 64

I decided to optimize parameters for the highest BLEU score since that was the most important metric for model performance in this experiment. Unfortunately the highest BLEU score did not correlate to the lowest average loss in either training or evaluation steps of the training loop. The highest BLEU score resulted from a model depth of 4 encoder/decoder layers, a model dimension of 128, 2 attention heads, and a feedforward layer dimension of 512, as shown in figure 1.

Model Depths	Model Dimensions	Attention Heads	Feedforward Dimensions	Avg Train Loss	Avg Eval Loss	BLEU Score:
1	128	2	256	3.970991542	2.268399085	0.4436927704
1	128	2	512	3.906459141	2.169548203	0.4416457332
1	128	4	256	3.93051835	2.20472017	0.4438931079
1	128	4	512	3.900031891	2.150089779	0.4328845596
1	128	8	256	4.02733797	2.287385284	0.4423076042
1	128	8	512	3.952619552	2.204741252	0.4433976611
1	256	2	256	3.11737978	1.489647779	0.4405837328
1	256	2	512	3.090079197	1.48958257	0.444063526
1	256	4	256	3.088508366	1.476128007	0.4421515925
1	256	4	512	3.020979488	1.460565708	0.43263386
1	256	8	256	3.108861851	1.546797428	0.4401196582
1	256	8	512	3.025032981	1.453009332	0.443064364
2	128	2	256	3.941444176	2.102569385	0.4452755567
2	128	2	512	3.912578903	2.071726013	0.4393332381
2	128	4	256	3.960004787	2.151540046	0.4423241232
2	128	4	512	3.911880271	2.097976204	0.442573481
2	128	8	256	4.012285386	2.29237963	0.4452366923
2	128	8	512	3.960776404	2.178096042	0.439568621
2	256	2	256	3.124423238	1.407962739	0.4400710842
2	256	2	512	3.009273465	1.294158987	0.438447103
2	256	4	256	3.070425617	1.367669803	0.4410021554
2	256	4	512	**2.968127032**	1.315076987	0.4408680048
2	256	8	256	3.201760194	1.454110631	0.4442810803
2	256	8	512	3.018960639	1.389965359	0.4619910877
4	128	2	256	4.001598426	2.085616471	0.463946042
4	128	2	512	4.0084471	2.077900372	**0.4699942842**
4	128	4	256	3.979769057	2.096697046	0.4558262987
4	128	4	512	4.038186418	2.135688846	0.4590492427
4	128	8	256	4.122052225	2.317832324	0.456240629
4	128	8	512	4.18972898	2.361531807	0.4592927017
4	256	2	256	3.349210524	1.445724381	0.4568424835
4	256	2	512	3.119592048	1.346217239	0.4623338607
4	256	4	256	3.366589293	1.413882268	0.4582255632
4	256	4	512	3.135615667	**1.278242031**	0.4581841996
4	256	8	256	3.408131894	1.423185509	0.4482460797
4	256	8	512	3.176792338	1.315288784	0.4537801301

Figure 1: Grid Search Results

3 Results

3.1 Experiment 1: Positional Encoding vs. Positional Embedding

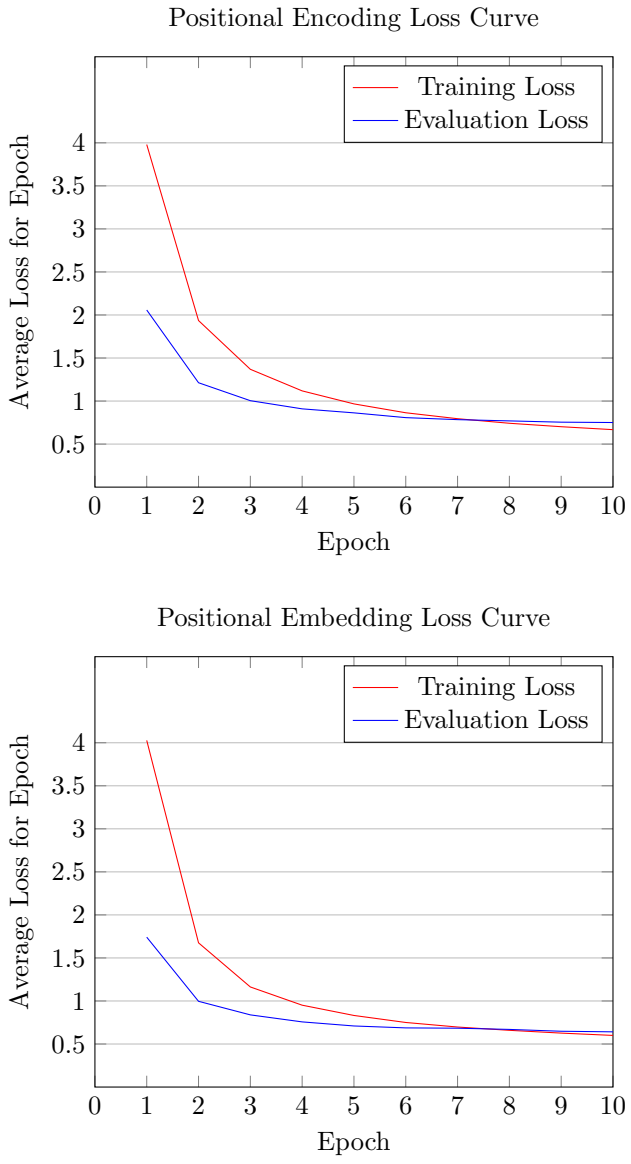
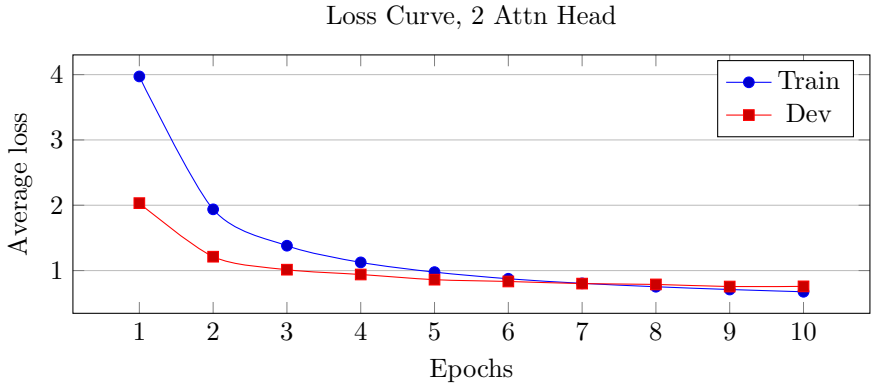


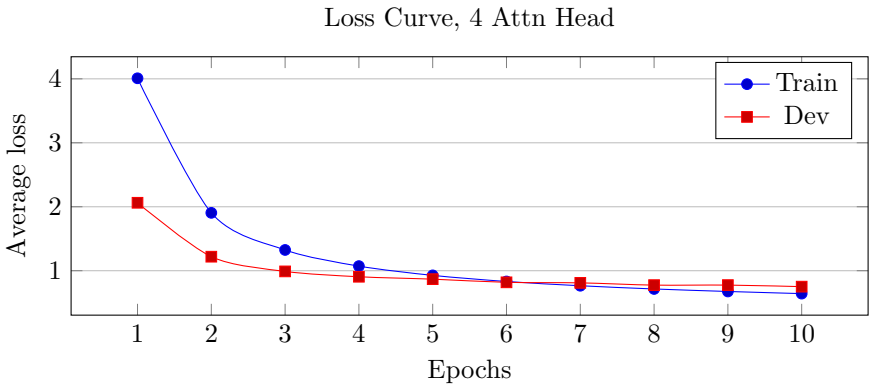
Figure 2: Loss curves on Positional Encoding and Positional Embedding results

While not significantly better, there was lower average loss when using learnable positional embeddings during training. The test set BLEU score for the positional encoding model was 45.44, while the test set BLEU score for the positional embedding model was 45.75.

3.2 Experiment 3: Model Architecture

3.2.1 Number of Attention Heads





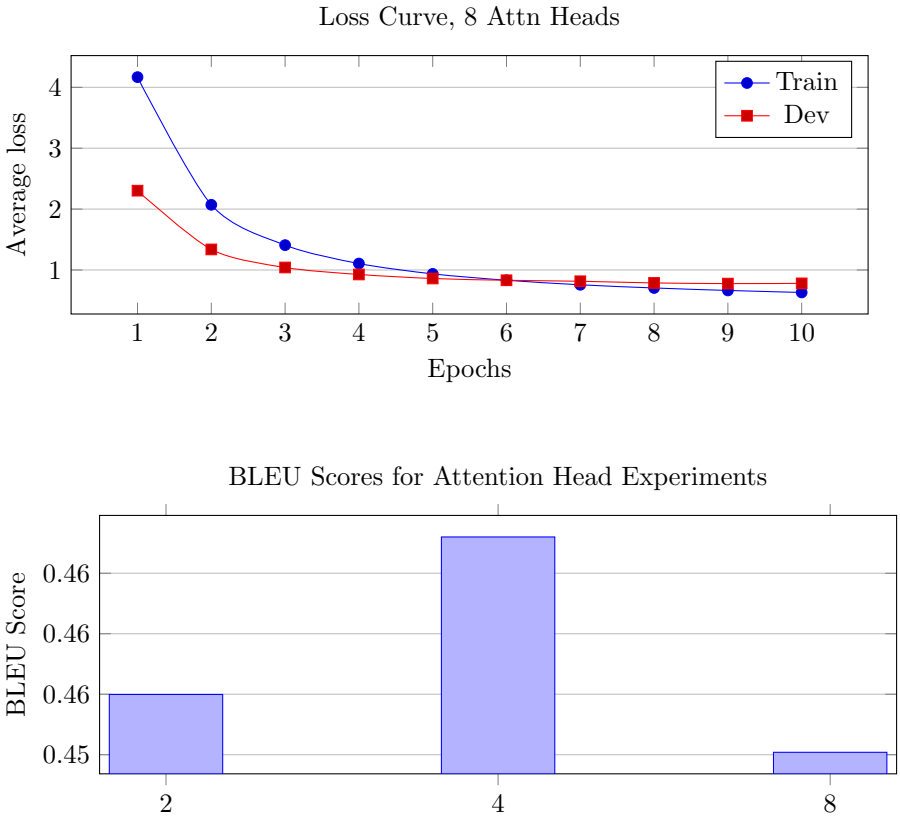
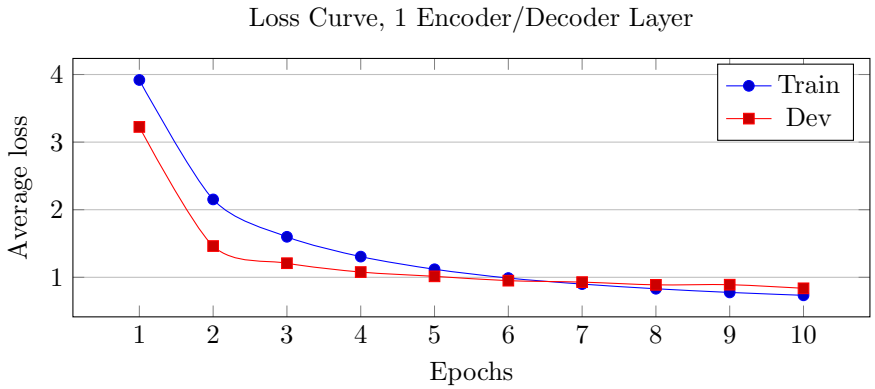
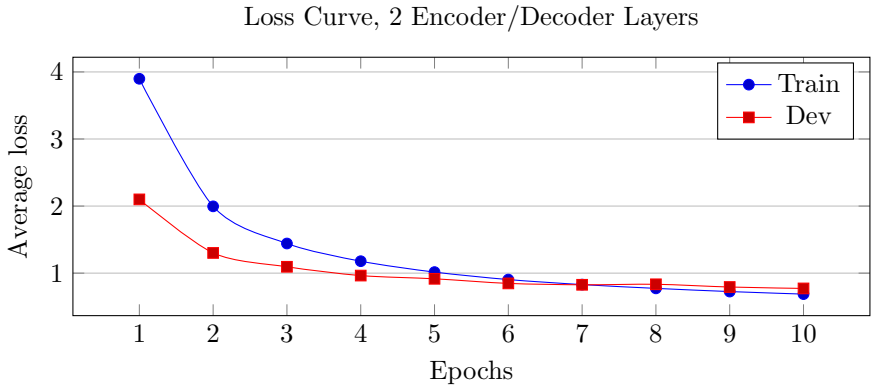


Figure 3: Performance by Number of Attention Heads

The models behaved similarly with differing attention head numbers, except that as the number of attention heads increased, signs of overfitting started showing sooner. This supports the results of my grid search which found that 2 attention heads resulted in a higher BLEU score than other configurations. The highest BLEU score in this experiment was achieved with 4 attention heads, and a BLEU score of 0.4576.

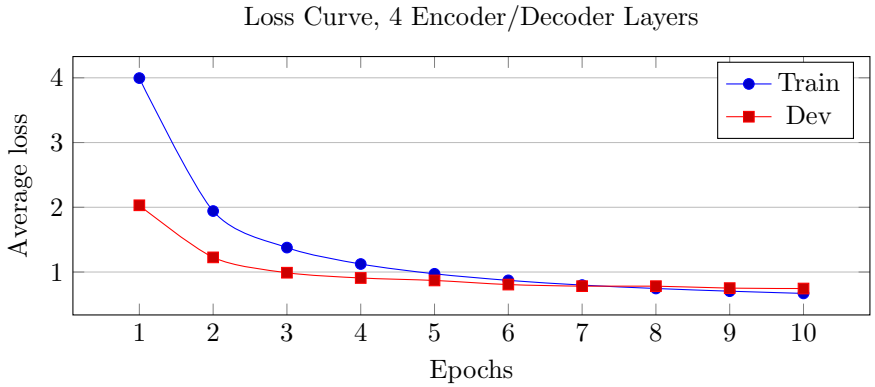
3.2.2 Encoder/Decoder Depth





Sample hypothesis:[”’, ’long’, ’time’, ’,’, ’indeed’, ’!”’] Sample reference: [”’, ’long’, ’a’, ’time’, ’,’, ’indeed’, ’!”’]

Figure 4: Random sampling of BLEU evaluation, 1 layer



Sample hypothesis: ['he', 'was', 'a', 'shepherd', '-', 'boy', 'for', 'his', 'father', ',', 'and', 'day', '–', 'often', 'night', '–', 'he', 'was', 'out', 'in', 'the', 'fields', ',', 'far', 'from', 'home', ',', 'watching', 'over', 'the', 'sheep', '.'] Sample reference: ['he',

'was', 'shepherd', '-', 'boy', 'for', 'his', 'father', ',', 'and', 'all', 'day', '–', 'often', 'all', 'night', '–', 'he', 'was', 'out', 'in', 'the', 'fields', ',', 'far', 'from', 'home', ',', 'watching', 'over', 'the', 'sheep', '.']

Figure 5: Random sampling of BLEU evaluation, 2 layers

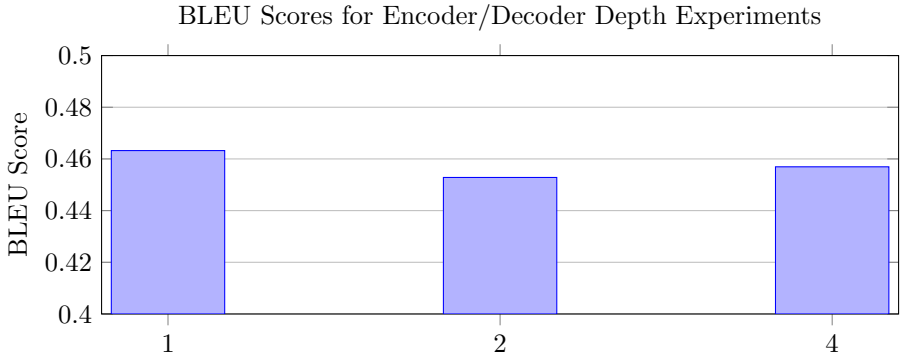


Figure 6: Performance by Number of Encoder/Decoder Layers

Sample hypothesis: ['7', ':', '26', 'neither', 'shalt', 'thou', 'bring', 'abomination', 'into', 'thine', 'house', ',', 'lest', 'thou', 'be', 'cursed', 'things', 'like', 'it', ':', 'thou', 'shalt', 'utterly', 'a', 'token', 'it', ',', 'thou', 'shalt', 'utterly', 'abhor', 'it', ';', 'for', 'it', 'is', 'a', 'thing', '.'] Sample reference: ['7', ':', '26', 'neither', 'shalt',

'thou', 'bring', 'an', 'abomination', 'into', 'thine', 'house', ',', 'lest', 'thou', 'be', 'a', 'cursed', 'thing', 'like', 'it', ':', 'but', 'thou', 'shalt', 'utterly', 'detest', 'it', ',', 'and', 'thou', 'shalt', 'utterly', 'abhor', 'it', ';', 'for', 'it', 'is', 'a', 'cursed', 'thing', '.']

Figure 7: Random sampling of BLEU evaluation, 4 layers

The experiments involving model Encoder/Decoder depth showed a high dev set loss with only 1 layer and improved loss with 2 and 4 layers. This experiment also showed signs of the model beginning to overfit data after epoch 7. Adding additional layers increased training time only marginally, by maybe 1 or 2 minutes, but resulted in better performance over fewer training epochs. Unexpectedly, the model with only one Encoder/Decoder layer returned the highest BLEU score.

4 Analysis

The results of experiment 1 suggest that learnable positional embeddings carry more information and since they can be adjusted during a training loop, result in higher model performance. During the training loops for both positional encoding and positional embeddings, the loss curve graphs show a cross at epoch 8 that suggests the models were heading towards overfitting, as the loss on the dev set was consistently lower than training set loss until epoch 8, when the loss started to rise.

5 Conclusion

While small, there were quantifiable differences between model architecture results. The data suggests that allowing for more learning will generally result in higher evaluation metrics, with a higher risk of overfit. This could be corrected by stopping training early on a signal such as the one I encountered where the dev set loss rises higher than the training set loss. The small variations in BLEU scores across the experiments are likely due to the abbreviated grid search. Being able to train only one epoch for all hyperparameter combinations most likely did not yield the most optimal settings for the rest of the experiments. Better results could be achieved by testing more hyperparameters such as dropout, learning rate, and batch size, which were all kept constant during this project. Time and compute power were massive limitations. A full grid search would have taken nearly six days on my home computer. With each epoch taking approximately 30 minutes, and ten epochs per experiment, the total time taken

to run this entire project was over 58 hours, not including the time it took to calculate BLEU scores. I will be planning accordingly for the next project.