CSE 256 PA2 Report - Transformers

Yashowardhan Shinde (A69027295)

Part 1.4: Sanity Checks

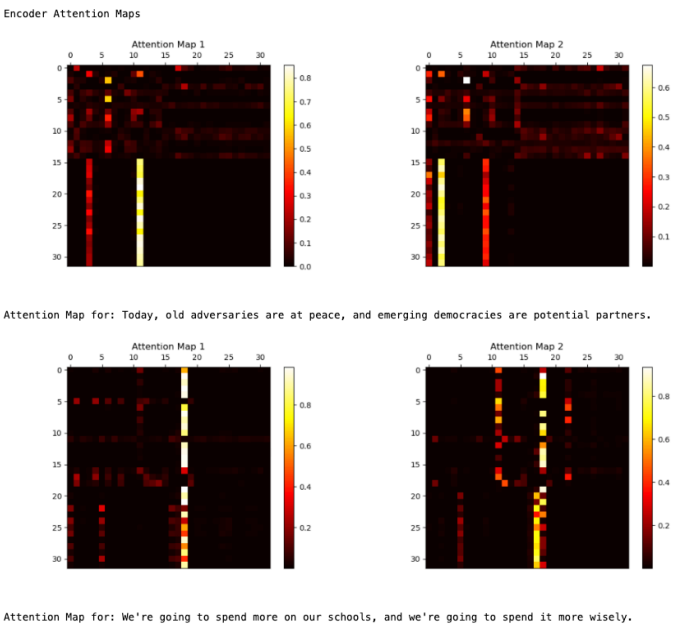

Figure 1. Attention Maps for Encoder

After implementing the encoder architecture, I extracted the attention maps from the last attention block. These attention maps provide insight into how each token in the input sequence attends to others during the encoding process. In Figure 1, which depicts two example sentences, it's evident that not all tokens in the sequence exhibit strong correlations with one another. Instead, only a few tokens demonstrate significant correlations with others, while the majority of the attention map appears dark.

This observation underscores the selective nature of attention within the transformer model. Rather than uniformly attending to all tokens in the sequence, the model focuses its attention on specific tokens deemed relevant for understanding the context and semantics of the input. This targeted attention mechanism enables the model to capture intricate dependencies and relationships between words while efficiently processing input sequences of varying lengths.

Part 1.5: Evaluation

Epoch	Train Acc (%)	Train Loss	Val Acc (%)
0	46.32	1.074	39.47
1	51.34	1.033	52.13
2	63.53	0.944	58
3	67.45	0.817	63.47
4	74.24	0.681	69.07
5	80.21	0.558	73.6
6	62.48	0.507	59.87
7	86.9	0.509	77.6
8	89.82	0.376	78.8
9	92.88	0.309	80.27
10	94.84	0.224	82.8
11	96.75	0.165	85.6
12	97.04	0.144	83.73
13	97.56	0.118	85.73
14	98.52	0.055	85.87
Final	**98.52**	**-**	**85.87**

Table 1. Results for Encoder Training

Part 2.3: Sanity Checks

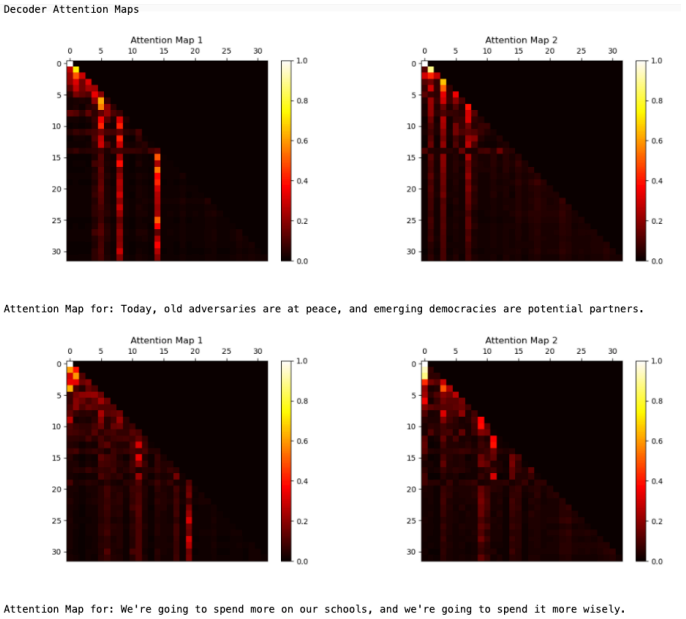

Figure 2. Attention Maps for Decoder

Similar to the encoder I extracted the attention maps from the last attention block of the decoder. Using two example sentences as input, the attention maps revealed notable differences compared to those generated by the encoder. We can see that the upper triangular matrix of each map is dark. This discrepancy arises primarily from the use of masked attention within the decoder architecture. In masked attention, each token is restricted to attending only to preceding tokens, ensuring that during training, the model can only access information from earlier parts of the sequence. This constraint is crucial for tasks like language modeling, where the goal is to predict the next word in a sequence. Consequently, the attention scores within the decoder exhibit higher values for preceding tokens, reflecting the model's reliance on past context to make accurate predictions.

Part 2.4 Evaluation:

Iteration	Perplexity
0	6310.94
100	583.26
200	458.21
300	323.66
400	242.04
500	182.01

Table 2. Perplexity Values of Decoder every 100 Iters.

Dataset- Step 500	Perplexity
Train	182.0117188
Test Obama	379.9954529
Test HBush	429.994812
Test WBush	487.1153259

Table 3. Perplexity Values of Decoder of Different Datasets

I think the reason behind the difference in perplexities is:

- **Dataset Characteristics:** Each test dataset (Obama, HBush, WBush) may have different linguistic characteristics, vocabulary sizes, and styles of language. Therefore, the model may perform differently on each dataset depending on how well it generalizes to the unseen data.
- **Domain Specificity:** The language used in speeches by different politicians (Obama, H. Bush, W. Bush) may vary significantly based on their political ideology, audience, and historical context. The model might be better at capturing the language patterns of certain politicians or political eras than others.

Part 3.1 Architecture Exploration

For this part, I implemented the attention mechanism discussed in AliBi by Press et al. I basically removed the positional encodings from the encoder and decoder network and added the bias mask in the attention layer as discussed in this paper.

In the AliBi mechanism proposed by Press et al., the traditional approach of adding positional encodings to the word embeddings in the transformer model is replaced with a static, non-learned bias added after the query-key dot product in the attention sublayer. ALiBi does not add positional embeddings to word embeddings; instead, it biases query-key attention scores with a penalty that is proportional to their distance. This bias is designed to introduce a form of inductive bias into the attention mechanism, promoting better generalization.

Technically, the bias term $m.bias\_matrix$ is added to each head-specific attention score, and it is defined as a function of the position of the query token within the sequence. Specifically, the bias term is computed as $m.[-(i-1), \ldots, -2, -1, 0]$, where i represents the position of the query token within the sequence. This bias term effectively prioritizes information from preceding tokens in the sequence during the attention computation. By introducing a static bias term based on token positions, AliBi aims to mitigate the need for learning positional encodings, simplifying the model architecture while maintaining or improving performance. The mathematical expression for this can be given using the equation:

$$softmax(q_i K^T + m \cdot [-(i-1), \ldots, -2, -1, 0]),$$

where scalar m is a head-specific slope fixed before training. This operation is shown in Figure 3.

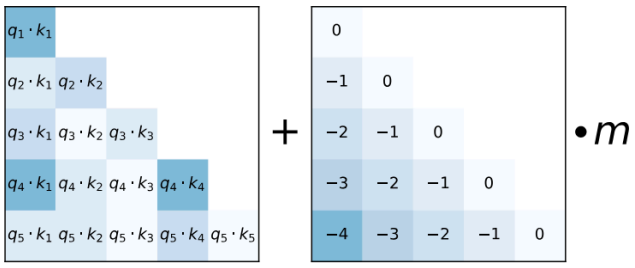

Figure 3. AliBi Method

To adapt the AliBi mechanism to the encoder network, I generated a symmetric mask matrix using the lower triangular mask matrix used for the language model (decoder). The new mask $Z = X + X^T$, where X is the lower triangular matrix used for the decoder. This symmetric mask matrix is then applied in the attention mechanism of the encoder, ensuring that each token can only attend to all other tokens in the sequence.

For the encoder + classifier model, the results were comparable to the original transformer used in Part 1. The results are shown in Table 4.

Epoch	Train Acc %	Train Loss	Val Acc %
0	48.85	1.07	47.73
1	60.99	0.98	57.87
2	67.93	0.86	65.87
3	68.21	0.71	66.13
4	78.11	0.56	74.53
5	90.73	0.45	82
6	88.67	0.38	78.8
7	93.83	0.31	83.73
8	93.31	0.28	84
9	95.41	0.21	83.87
10	95.89	0.17	85.33
11	97.75	0.12	84.27
12	98.09	0.07	86.13
13	98.23	0.06	86.8
14	97.56	0.07	85.33
Final Results	**97.56**		**85.33**

Table 4. Results for AliBi Encoder Training

As we can see the results are quite similar to that of our earlier method. AliBi Encoder + Classifier produces a slightly lower accuracy than the original architecture.

However, the story is different when comparing the results for the language modeling task. Here, AliBi Decoder outperforms our original architecture from Part 2 the results are as shown in Table 5 and Table 6.

Iteration	Train Perplexity
0	6444.65
100	513.58
200	320.85
300	211.33
400	151.9
500	115.03

Table 5. Perplexity Values of AliBi Decoder every 100 Iters.

Dataset - Steps 500	Perplexity
Train	115.03
Test Obama	340.74
Test HBush	391.41
Test WBush	444.81

Table 6. Perplexity Values of AliBi Decoder of Different Datasets

Here, we can see a significant improvement in the perplexity scores of the Language Modelling Decoder. This shows that AliBi performs better than the original architecture in Part 2.

Part 3.2 Performance Improvement

In the previous section, I was able to get performance improvement for the Decoder Model used for language modeling. Here, I tried to improve the accuracy of the Ecoder+Classifier Model using Hyperparameter tuning. I analyzed 3 different hyperparameters on 2 different learning rates (0.001, 0.003), and I was able to achieve a better accuracy than the Encoder + Classifier model in Part 1. I also used a different optimizer here, I used the AdamW optimizer instead of Adam.

The hyperparameters I explored were Network Depth (Number of Attention Blocks), Number of Attention Heads, Dimension of Hidden Layer in the Classifier, and Learning Rate. The results for all these are shown in Table 7. The train vs validation accuracy curves for all these combinations can be seen in Figure 3.

Config	Hyperparameter	Learning Rate	Val Acc %
1	Depth 3	0.003	84.133
2	**Depth 4**	**0.003**	**87.067**
3	Depth 5	0.003	86.4
4	Hidden Dim: 64	0.003	84
5	**Hidden Dim: 128**	**0.003**	**86.667**
6	Hidden Dim: 256	0.003	81.867
7	Num Heads 2	0.003	83.333
8	**Num Heads 4**	**0.003**	**86.133**
9	Num Heads 8	0.003	85.2
10	Depth 3	0.001	81.867
11	Depth 4	0.001	82.133
12	**Depth 5**	**0.001**	**86.667**
13	Hidden Dim: 64	0.001	82.8
14	**Hidden Dim: 128**	**0.001**	**86.133**
15	Hidden Dim: 256	0.001	84.533
16	Num Heads 2	0.001	84.267
17	Num Heads 4	0.001	84.4
18	**Num Heads 8**	**0.001**	**86.667**

Table 7. Results of Hyperparameter Tuning for Encoder + Classifier Model

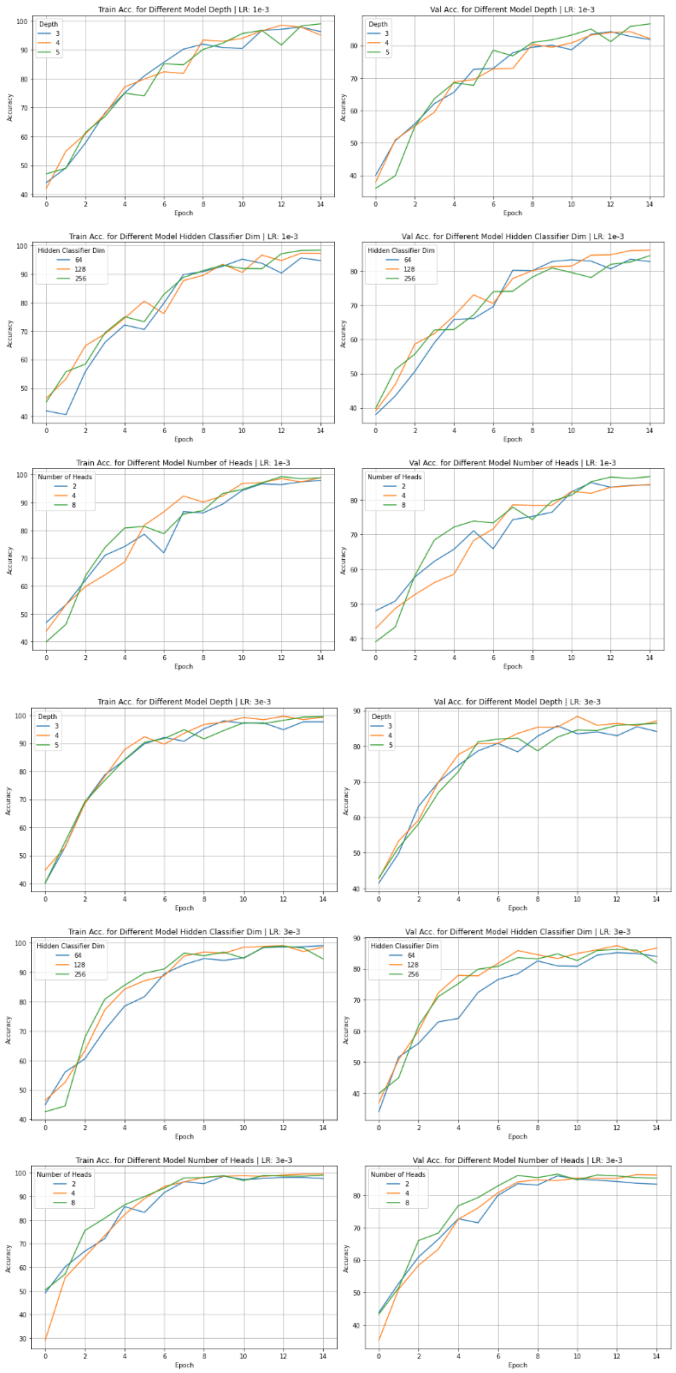

Figure 3. Training vs Validation Accuracy Curves for Different Sets of Hyperparameters.

The trends in the validation accuracy indicate that configurations with a higher model depth tend to yield better performance, with configuration 2 (depth 4) achieving the highest accuracy of **87.067%.** Increasing the hidden dimension size also generally improves accuracy, as seen in configurations 5 and 14 in Table 7. However, excessively large hidden dimensions, such as in configuration 6, can lead to a decrease in accuracy. Moreover, the number of attention heads has a varying impact on accuracy, with configuration 8 in Table 7 (4 heads) performing better than configurations 7 (2 heads) and 9 (8 heads) in Table 7 but not consistently across all configurations. Additionally, **a higher learning rate (3e-3)** generally results in better accuracy compared to a lower learning rate (1e-3).

From the analysis, it's evident that six combinations of hyperparameters outperformed the Encoder+Classifier Model from Part 1 in terms of accuracy. After a thorough examination of these results, I identified a specific set of hyperparameters that consistently yield superior performance. These optimal hyperparameters include a **learning rate of 3e-3, a model depth of 4, 4 attention heads, and a default hidden layer dimension of 100**. Notably, this configuration consistently achieves a validation accuracy ranging from **86.99% to 87.1%**, surpassing the performance of our original model from Part 1.

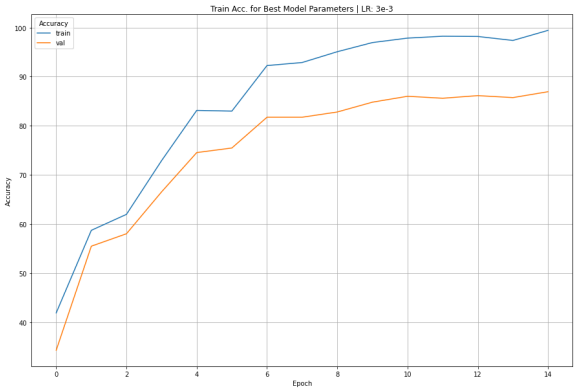

Figure 4. Training and Validation Accuracy Curve for Best Model.

The train and validation accuracy curve for this best model is shown in Figure 4.

Part 1.1: Encoder Implementation
Part 1.2: Feedforward Classifier Implementation
Part 1.3: Joint Encoder & Classifier Training
Part 2.1: Decoder Implementation
Part 2.2: Decoder Pretraining

All the above sections can be found in my code. If the plots are not clearly visible please check out the Jupyter Notebook "**Experiments.ipynb**" in my code. It has all the plots arranged systematically.