Programming Assignment 2: Report

Chinmay Sharma
A59023624
c1sharma@ucsd.edu

1 Part 1: Encoder Trained With Classifier

The classifier consists of a 4-layered Encoder which generates the embeddings and 2 fully connected layers with ReLU activation. Cross entropy loss has been used as the loss function with Adam optimizer. **Note:** I have initialization weights like Andrej Karapathy, so getting higher accuracy.

The classifier model has 576,539 parameters.

Epoch	Training Loss	Training Accuracy	Test Accuracy
1	1.0675	44.16	33.33
2	0.8871	56.5	56.8
3	0.6416	67.39	60.5333
4	0.4798	77.62	78.4
5	0.2738	89.77	84.4
6	0.161	94.78	85.4666
7	0.0928	97.08	84.9333
8	0.0539	98.47	86.6666
9	0.0414	98.94	87.0666
10	0.0331	99.09	87.1333
11	0.0134	99.61	87.7333
12	0.0186	99.37	87.2
13	0.0364	99.04	86.4
14	0.0226	99.23	87.8666
15	0.0144	99.66	87.8

Table 1: Training & Test metrics for Classifier model

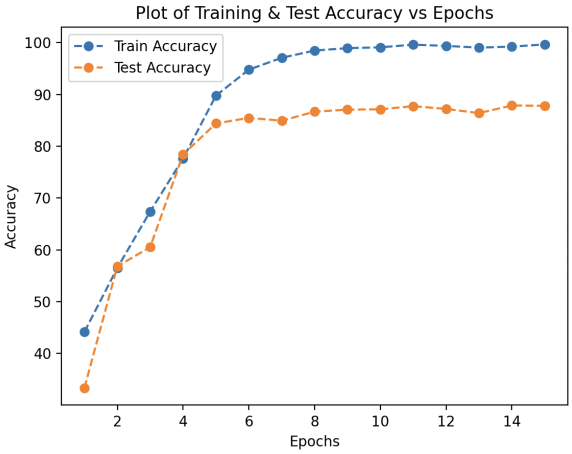

Figure 1: Training & Test accuracy over epochs

I visualized attention weights generated by the attention heads in the encoder, for the sentence: "These virtues give me an unshakable faith in America". In fig 2, which plots the attention weights for the first layer of the encoder, we can observe that the high attention weights are concentrated. This shows that in the first layer, the attention mechanism focuses more on local dependencies within the input sequence, as it has limited context available. It focuses on the immediate context, in both directions, of each token in the input sequence.

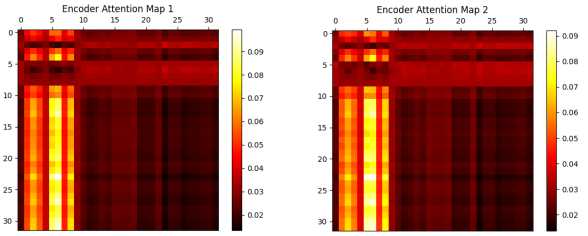

Figure 2: Attention matrices for both heads - Encoder Layer 1

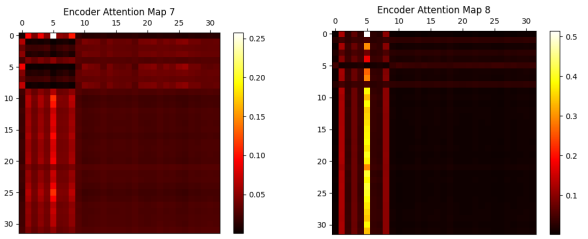

Figure 3: Attention matrices for both heads - Encoder Layer 4

Whereas, in the last encoder layer, attention weights are not concentrated but rather have high attention values only for certain tokens in the sequence. As shown in fig 3. We can see that token 5, i.e. "unshakable", as high attention from most of the other tokens in the input sequence. This indicates that the encoder has learned to attend selectively to the most informative parts of the sequence to generate a good representation (embedding).

2 Part 2: Pretraining Decoder Language Model

The decoder-only model has 4 decoder layers and is trained on the language modeling task, predicting the next word in a sequence given the previous words. The implementation uses Adam optimizer to minimize the cross-entropy loss.

The decoder-only model has 943,611 parameters.

Iteration	Perplexity			
	Train	Obama	H. Bush	W. Bush
100	575.3088	693.105	721.7043	808.4219
200	382.8657	516.2067	549.1379	621.8051
300	266.1291	417.7356	458.2095	525.0245
400	197.3809	363.0804	430.3409	483.0418
500	154.2479	348.0284	408.274	478.0699

Table 2: Training & Test Perplexity for Decoder-only Model: Feed-forward hidden dimension = 256

Iteration	Perplexity			
	Train	Obama	H. Bush	W. Bush
100	585.1399	715.4691	748.9787	829.4408
200	440.6043	584.6114	611.2496	693.2736
300	318.3407	472.2497	497.7435	574.2589
400	228.9188	410.3708	444.9504	513.491
500	180.8624	382.8425	417.144	490.9431

Table 3: Training & Test Perplexity for Decoder-only Model: Feed-forward hidden dimension = 100

Two experiments were performed by varying the hidden dimension of the feed-forward layer inside the decoder block. As we can see in Table 2 and Table 3, all the training and test (Obama, H. Bush, W. Bush) perplexities are lower for the decoder-only model with hidden dimension = 256 compared to the model with hidden dimension = 100. Therefore, for further analysis I used the model with feed-forward hidden dimension = 256. Fig 4 and 5 presents how perplexity decreases on training and test sets with iterations.

We observe that the perplexity numbers for test sets are different from one another. A potential reason is that the training data has a mixture of speeches from all the three politicians, which can result in the next token not being predicted in the exact speaking style for a given politician's test set.

Fig 6 and 7 present the attention weights from the decoder-only model for the sentence: "These virtues give me an unshakable faith in America". Since the decoder-only model uses masked multi-head attention, we can see that only the half left of the diagonal of the matrix contains attention

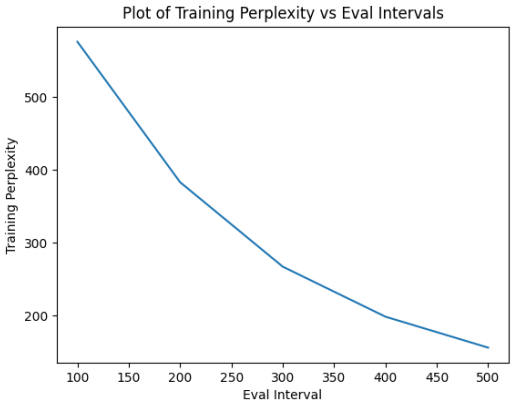

Figure 4: Training Perplexity over iterations

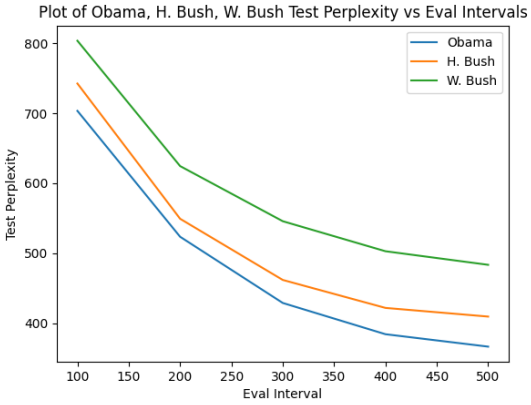

Figure 5: Test Perplexity over iterations

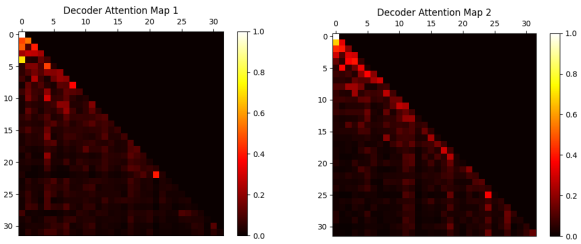

Figure 6: Attention matrices for both heads - Decoder Layer 1

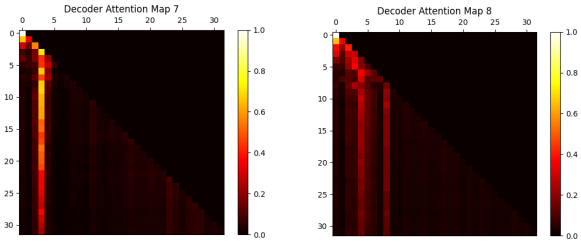

Figure 7: Attention matrices for both heads - Decoder Layer 4

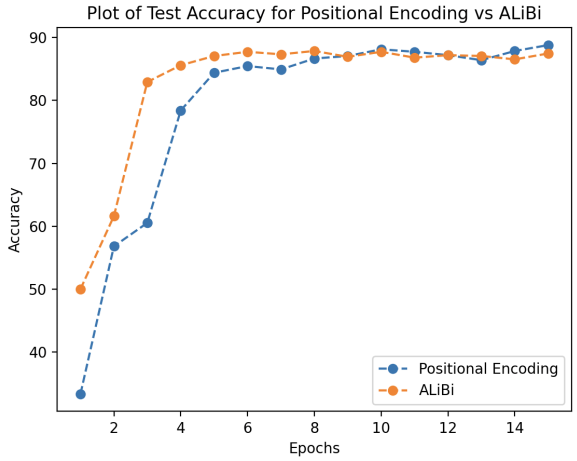

Figure 8: Test Accuracy of Classifier with Positional Encoding vs ALiBi in Encoder

weights. The high attention weights in the first layer, shown in fig 6, are more or less concentrated along the diagonal, indicating that each output token attends most strongly to the input token immediately preceding it. This is expected since in the first layer the model focuses on capturing short-range dependencies between nearby tokens.

On the other hand, the high attention weights in the last layer, shown in fig 7, are more distributed across multiple preceding input tokens. This denotes that the higher layers are able to consider broader context, beyond just the most recent token, when making predictions. Additionally, we observe a column with high attention weights that indicates that certain input token seems to get consistently high attention across many output tokens. This suggests that the model has identified a token as very importance for prediction.

3 Part 3: Exploration

3.1 Architectural Exploration

For this part, I removed the positional encoding from the encoder and decoder blocks and implemented Attention with Linear Biases (ALiBi) (Press et al., 2021), which is a more efficient position method. The attention head blocks in the original encoder/decoder were replaced by the ALiBi attention head blocks. ALiBi reduces training time without compromising the performance of the models. This can be corroborated by the experiments I conducted.

I compared two variants of the classifier model: (1) classifier with positional encoding and (2) classifier with ALiBi in encoder, to analyze the impact of ALiBi on performance and training time. As we can observe in fig 8, with ALiBi in the encoder of the classifier model, the test accuracy remains the same as that of the classifier with positional encoding in encoder. However, there is a 14% speedup in avg. training time of the classifier. I trained these two types of models five times and recorded the training time as shown in Table 4.

	Training Time (s)	
	Classifier with Positional Encoding in Encoder	**Classifier with ALiBi in Encoder**
	24.0387	20.9462
	27.6852	22.3246
	24.3258	21.6692
	25.078	22.7474
	25.4255	21.1484
Average	**25.31064**	**21.76716**

Table 4: Training times for Classifier with Positional Encoding vs ALiBi in Encoder

I performed a similar analysis for the decoder-only model with positional encoding vs ALiBi in decoder blocks. A 11.5% reduction in avg. training time was observed. Please refer to Table 5 for training time comparison.

Unlike the classifier, the decoder-only model achieved performance improvements as well with ALiBi. Compared to the model with positional encoding in decoder, the model with ALiBi in decoder blocks had less training perplexity throughout the training process. It also has lower test perplexity on each of the test sets. These results are shown in fig 9, where we can see that all the dashed lines (denoting decoder with ALiBi) have lower perplexity than their corresponding solid lines (denoting decoder with positional encoding).

	Training Time (s)	
	Positional Encoding in Decoder	**ALiBi in Decoder**
	35.3216	32.5361
	34.9233	30.6604
	36.0709	29.9158
	35.7318	32.7269
	36.2602	31.8983
Average	**35.66156**	**31.5475**

Table 5: Training times for decoder-only model with Positional Encoding vs ALiBi in Decoder

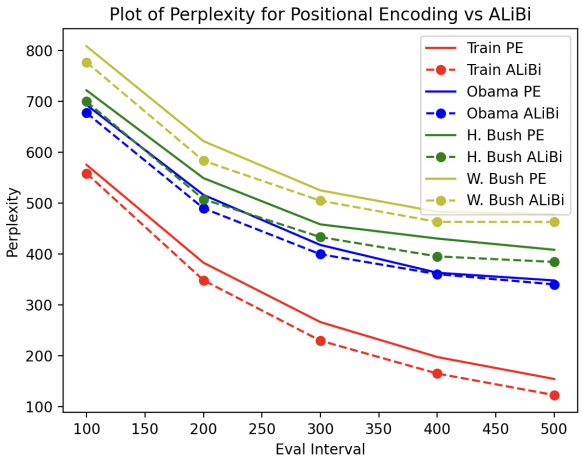

Figure 9: Training and Test perplexity of decoder-only model with Positional Encoding vs ALiBi in Decoder

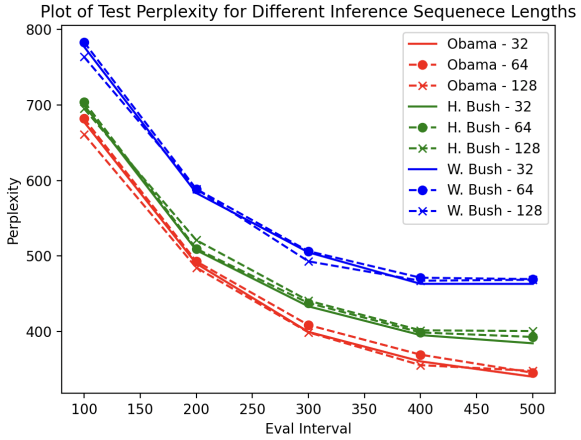

Figure 10: Test Perplexity of decoder-only model with different inference sequence lengths

The ALiBi paper also presents that the perplexity remains constant even if inference is performed on longer sequence length than the sequence length used for training the model. The decode-only model with ALiBi was trained on sequence length (block_size) = 32. So, to corroborate the paper's claim, I computed perplexity on the test sets with sequence length = 32, 64 and 128 during inference, i.e. same and longer than training sequence length. The test perplexities for all inference sequence lengths are plotted in fig 10. For example, "Obama - 64" in the plot's legend denotes the curve for Obama test set with inference length = 64. We can observe that for each of the individual test sets, the perplexity is similar for all inference sequence lengths, i.e. all three curves for a given test set overlap. Thus, confirming that with ALiBi we can train on shorter sequences but infer with longer sequences without degrading the model's performance.

3.2 Performance Improvement

I worked on improving the accuracy of the classifier model on the test_CLS.txt test set. If we look at fig 1, we can observe a significant gap between the training and test set accuracies. This indicates that the classifier model is overfitting on the training data. To reduce overfitting I introduced regularization by adding batch normalization and dropout in the classifier.

Batch normalization was only added to the classifier after the first fully-connected layer. Whereas, I added dropout in the classifier, after the first fully-connected layer, and the encoder, within the multi-head attention block and feed-forward network, as well as in the encoder block. I tested model performance by varying the dropout rate between 0.2 to 0.5 at 0.1 intervals. The best performing model was obtained with dropout rate = 0.2, which I have used for analysis here.

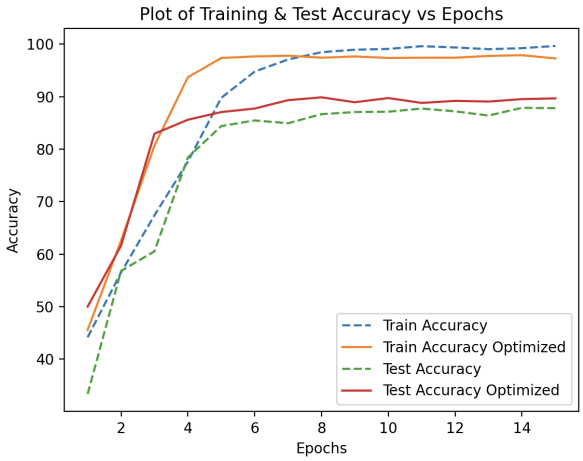

Figure 11: Comparison between Train and Test Accuracy of Original vs Optimized Classifier

Fig 11 contains plots of training and test set accuracies of original vs optimized classifier. We can see that by applying regularization techniques I have reduced overfitting, as the gap between train and test accuracy reduced in the optimized model. This gap is visibly more in the original classifier from part 1. The test accuracy also saw a 2% bump in the optimized classifier, as the test accuracy after 15 epochs for optimized classifier was 89.67 compared to 87.8 of the original classifier. Hence, adding batch norm and dropout to the classifier and encoder helped in performance improvement.