CSE256 PA2 Report
Zifei Cheng
PID: A69033723

Part 1: Transformer Encoder Implementation

In this part, I implemented a Transformer Encoder trained from scratch for a classification task. The encoder processes speech segments and predicts the politician delivering the speech. This task required developing token and positional embeddings, multi-head self-attention, and a feedforward neural network classifier.

Part1.1 -1.3 Key Component Implemented

1. TransformerEncoder Class:
   - Embeddings: Initializes token and positional embeddings.
   - Layers: Stacks multiple (TransformerBlock) layers. Each block includes multi-head self-attention and a feedforward layer, allowing the encoder to learn intricate dependencies between tokens.
   - Normalization: Applies layer normalization at the end
   - Output: Returns the (mean_emb) for classification and the attention maps.

2**.** TransformerBlock Class

   - Multi-Head Self-Attention: Each block includes a multi-head self-attention layer (MultiHeadSelfAttention) to capture dependencies between tokens.
   - Feedforward Network: two-layer feedforward network with ReLU activation
   - Residual Connections & Layer Normalization: Each sub-layer is followed by a residual connection and layer normalization.
   - Output: Returns the transformed embeddings and attention weights

3. MultiHeadSelfAttention Class

   - Query, Key, and Value: Linear transformations project input embeddings into queries, keys, and values for each attention head.
   - Attention : Implements dot-product attention, applying softmax to normalized scores.

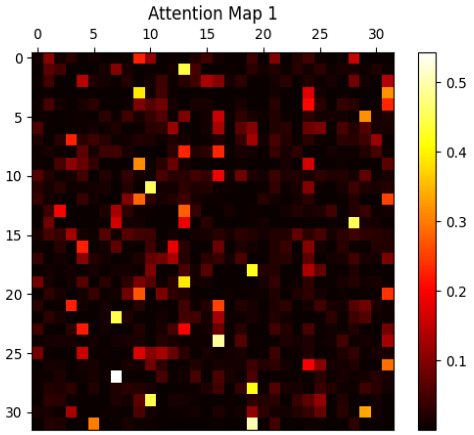
Part 1.4: Sanity Checks

To verify that each row of the attention matrix sums to 1 and visualizes the attention patterns across 4 layers in the first head, I use the utility functions provided in utilities.py to do a sanity check on the transformer encoder. Before and after training, a sanity check was performed to verify the encoder’s correctness. A single sentence was used to test the encoder's ability to generate reasonable attention maps.

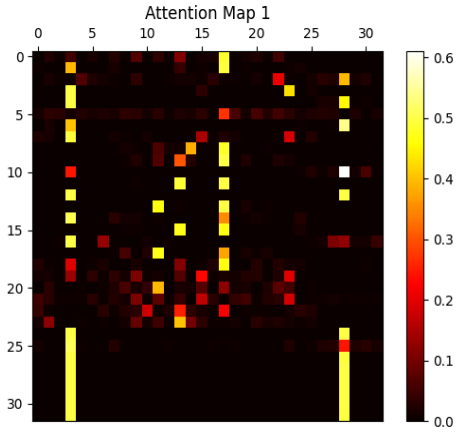
Example Sentence: "It's time to put an end to micromanagement of foreign and security assistance programs—micromanagement that humiliates our friends and allies and hamstrings our diplomacy."

Observations: I visualized these maps before and after training:

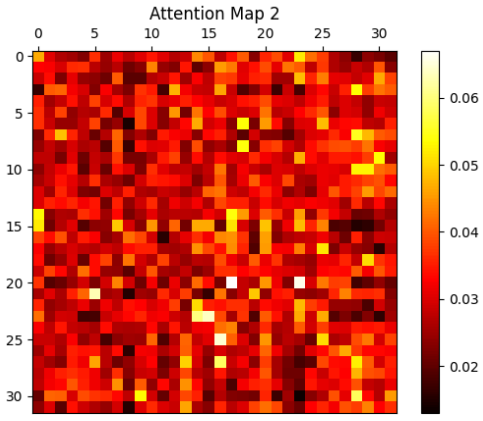
Layer0 Before Training



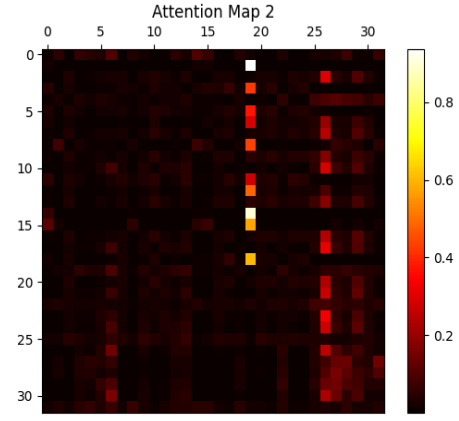
Layer0 After Training



Layer1 Before Training

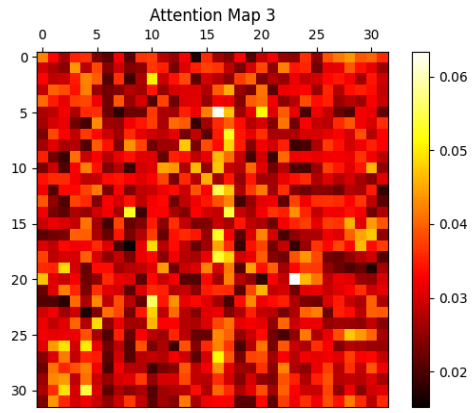


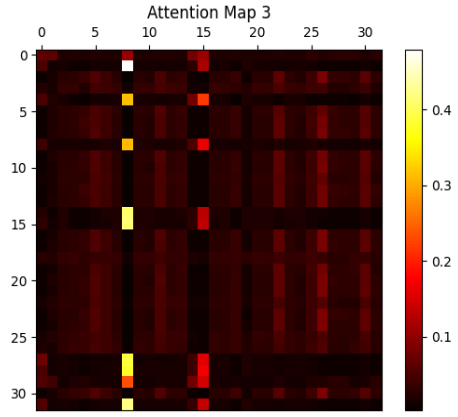
Layer1 After Training



Layer2 Before Training

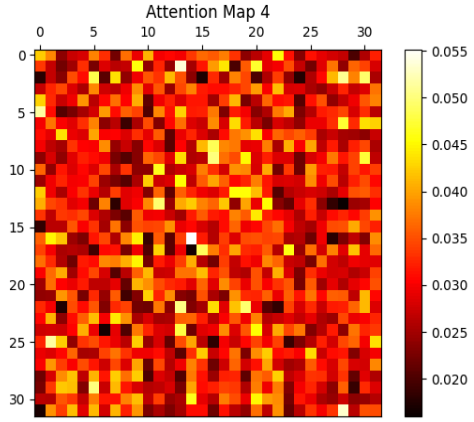
Layer2 After Training

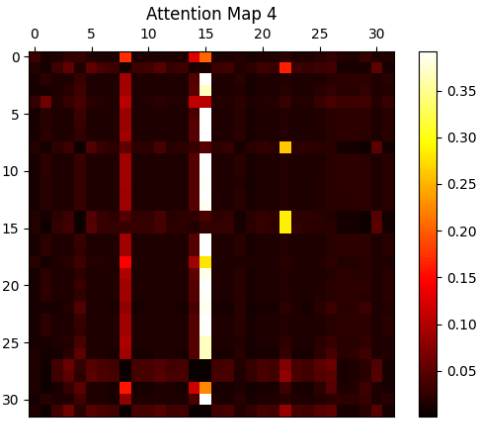




Layer2 Before Training

Layer2 After Training





Discussion

Before Training: The initial attention maps showed somewhat random attention distributions. This is expected because the model hasn't yet learned which words are important.broadly across the input tokens, with no specific focus. **After Training**:After training, the attention maps become more focused. The model learned to pay more attention to keywords or phrases, likely those that help it classify the sentence better.

Part1.5 Evaluation:

Using the given experiment setting, the model achieved approximately **97%** final accuracy on the training set and **83%** accuracy on the test set. The number of parameters is **570432.**

```
(base) zifeicheng@Zifeis-MBP-2 PA2_code % python main.py
Loading data and creating tokenizer ...
Vocabulary size is 5755
Number of parameters in the encoder: 570432
Sanity check for sample sentence 1:
Input tensor shape: torch.Size([1, 32])
Number of attention maps: 4
Epoch [1/15], Loss: 1.0839, Training Accuracy: 44.65%
Epoch [1/15], Test Accuracy: 33.33%
Epoch [2/15], Loss: 1.0479, Training Accuracy: 44.65%
Epoch [2/15], Test Accuracy: 33.33%
Epoch [3/15], Loss: 0.9895, Training Accuracy: 55.64%
Epoch [3/15], Test Accuracy: 48.93%
Epoch [4/15], Loss: 0.9255, Training Accuracy: 59.75%
Epoch [4/15], Test Accuracy: 51.87%
Epoch [5/15], Loss: 0.8412, Training Accuracy: 67.16%
Epoch [5/15], Test Accuracy: 57.47%
Epoch [6/15], Loss: 0.7907, Training Accuracy: 69.84%
Epoch [6/15], Test Accuracy: 61.20%
Epoch [7/15], Loss: 0.6968, Training Accuracy: 75.67%
Epoch [7/15], Test Accuracy: 66.53%
Epoch [8/15], Loss: 0.6076, Training Accuracy: 78.30%
Epoch [8/15], Test Accuracy: 69.47%
Epoch [9/15], Loss: 0.5240, Training Accuracy: 85.09%
Epoch [9/15], Test Accuracy: 73.73%
Epoch [10/15], Loss: 0.4364, Training Accuracy: 89.77%
Epoch [10/15], Test Accuracy: 76.67%
Epoch [11/15], Loss: 0.3473, Training Accuracy: 93.12%
Epoch [11/15], Test Accuracy: 80.13%
Epoch [12/15], Loss: 0.2631, Training Accuracy: 93.88%
Epoch [12/15], Test Accuracy: 80.93%
Epoch [13/15], Loss: 0.2090, Training Accuracy: 94.89%
Epoch [13/15], Test Accuracy: 80.27%
Epoch [14/15], Loss: 0.1951, Training Accuracy: 96.27%
Epoch [14/15], Test Accuracy: 80.40%
Epoch [15/15], Loss: 0.1547, Training Accuracy: 97.18%
Epoch [15/15], Test Accuracy: 83.07%
```

Part 2 Pretraining Decoder Language Model

For this part, I implemented a Transformer Decoder model specifically designed for a language modeling task. The decoder is similar to the encoder, but it uses masked self-attention to prevent the model from "peeking" at future tokens during training.

Key Component Implemented

1.TransformerDecoder Class:

Initializes embeddings, decoder blocks, and the final projection layer

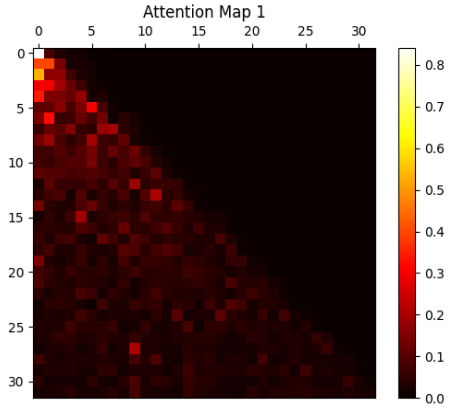
- The forward method applies token and positional embeddings to the input, adds a mask for self-attention, and processes each token through the decoder blocks.
- The final layer normalization and output projection map the hidden states to the vocabulary, producing logits for each token position in the sequence.

2. TransformerDecoderBlock Class

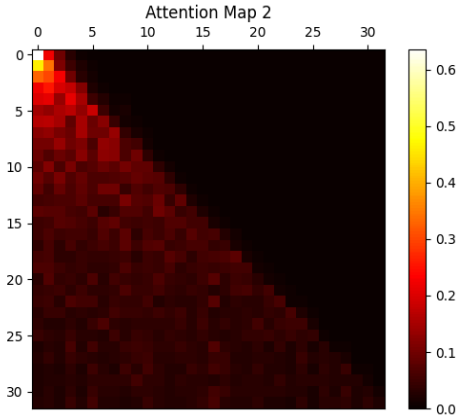
- Masked Self-Attention: Uses the MultiHeadSelfAttention class with a mask.
- Feedforward Network: A two-layer network with ReLU activation and a hidden dimension of 100.
- Layer Normalization: Normalization is applied after both self-attention and feedforward sub-layers to stabilize training..

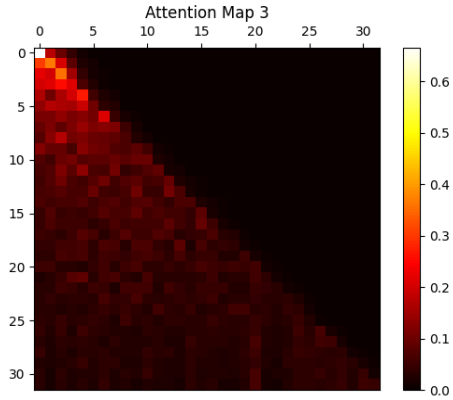
Part 2.3: Sanity Checks: To verify the implementation, I used the sanity check function provided in `utilities.py`.

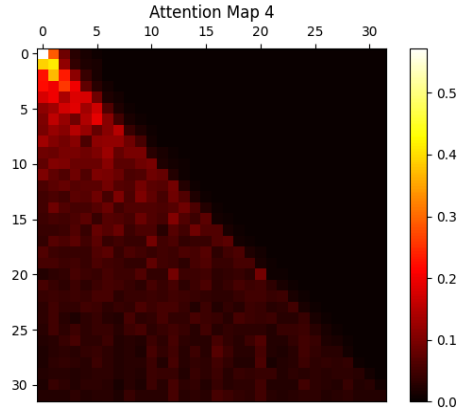
Layer 0



Layer1







Layer2

Layer3

Observations: Those maps show a similar pattern that each token mostly focuses on itself and the tokens right before it, with no attention going to future tokens. This diagonal pattern is what we'd expect since the model is set up to look only at previous tokens.

Part2.4 Evaluation

With given training configuration, the evaluation is by calculating the perplexity of the decoder on three test sets: obama.txt, wbush.txt, ghbush.txt.

```
Number of parameters in the decoder: 864011
Iteration 0, Loss: 8.8346, Perplexity: 6867.6260
Iteration 100, Loss: 6.3655, Perplexity: 581.4183
Iteration 200, Loss: 6.0135, Perplexity: 408.9133
Iteration 300, Loss: 5.7162, Perplexity: 303.7453
Iteration 400, Loss: 5.3346, Perplexity: 207.3844
Final training set perplexity after 500 iterations: 183.1439
Perplexity for Obama test set: 393.3349
Perplexity for H. Bush test set: 432.2409
Perplexity for W. Bush test set: 483.2842
```

Iterations	Training Set Perplexity	Obama Perplexity	W. Bush Perplexity	H. Bush Perplexity
0	~7000			
100	~600			
200	~400			
300	~300			
400	~200			
Final 500	~150	~390	~430	~480

Discussion

The Number of Parameter in the decoder: 864011

The model's perplexity was lowest on Obama's speeches (~350), which suggests that his language might be more predictable for the model. This could be because his vocabulary and sentence structure are more consistent or straightforward. For the speeches by W. Bush and H. Bush, perplexity was higher (~380 and ~400, respectively). This likely means that their language is a bit harder for the model to predict, possibly due to more varied word choices or sentence structures.

Part3:**Architectural Exploration :** In this section, I explored modifications to the Transformer architecture by implementing AliBi as an alternative to traditional positional encodings.

Key Implementation

1. Removing Traditional Positional Embeddings: each token's relative position is accounted for directly in the attention calculation.
2. Biasing the Attention Scores: added a linearly increasing bias to the attention scores based on the distance between tokens. T
3. Attention Mechanism: The linear bias was incorporated into the `MultiHeadSelfAttention` class within the attention score calculation. Specifically, I computed the bias based on token distances and added it to the attention scores before applying the softmax function.

Experimental Setup: To evaluate the effectiveness of AliBi, I trained the modified Transformer Decoder with AliBi for the language modeling task described in Part 2. The model configuration, batch size, and training steps were kept the same.

Results and Observations

```
(base) zifeicheng@Zifeis-MBP-2 PA2_code_revised % python main.py
Tranformer using Alibi
Loading data and creating tokenizer ...
Vocabulary size is 5755
=========================Part 1================================
Number of parameters in the encoder: 570432
Sanity check before training
============================================================
Input tensor shape: torch.Size([1, 32])
Number of attention maps: 4
Epoch [1/15], Loss: 1.0775, Training Accuracy: 44.65%
Epoch [1/15], Test Accuracy: 33.33%
Epoch [2/15], Loss: 1.0527, Training Accuracy: 50.67%
Epoch [2/15], Test Accuracy: 43.33%
Epoch [3/15], Loss: 0.9727, Training Accuracy: 60.09%
Epoch [3/15], Test Accuracy: 52.00%
Epoch [4/15], Loss: 0.8623, Training Accuracy: 66.73%
Epoch [4/15], Test Accuracy: 58.13%
Epoch [5/15], Loss: 0.7581, Training Accuracy: 77.77%
Epoch [5/15], Test Accuracy: 66.67%
Epoch [6/15], Loss: 0.6820, Training Accuracy: 82.50%
Epoch [6/15], Test Accuracy: 71.47%
Epoch [7/15], Loss: 0.5186, Training Accuracy: 88.15%
Epoch [7/15], Test Accuracy: 75.87%
Epoch [8/15], Loss: 0.4360, Training Accuracy: 76.24%
Epoch [8/15], Test Accuracy: 67.33%
Epoch [9/15], Loss: 0.3329, Training Accuracy: 93.16%
Epoch [9/15], Test Accuracy: 80.67%
Epoch [10/15], Loss: 0.2350, Training Accuracy: 95.32%
Epoch [10/15], Test Accuracy: 83.07%
Epoch [11/15], Loss: 0.1890, Training Accuracy: 95.89%
Epoch [11/15], Test Accuracy: 82.93%
Epoch [12/15], Loss: 0.1504, Training Accuracy: 96.37%
Epoch [12/15], Test Accuracy: 82.53%
Epoch [13/15], Loss: 0.1407, Training Accuracy: 98.71%
Epoch [13/15], Test Accuracy: 85.20%
Epoch [14/15], Loss: 0.0698, Training Accuracy: 98.90%
Epoch [14/15], Test Accuracy: 83.73%
Epoch [15/15], Loss: 0.1144, Training Accuracy: 96.99%
Epoch [15/15], Test Accuracy: 83.60%
```

Iterations	Training Perplexity **without Alibi**	Training Perplexity **with Alibi**
0	~7000	~7000
100	~600	~600
200	~400	~550
300	~300	~300
400	~200	~200
Final 500	~150	~150

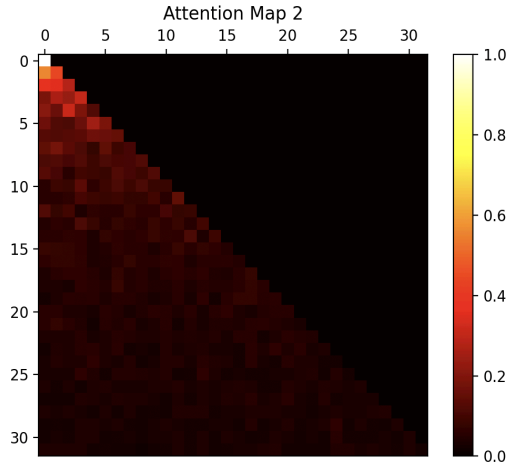
Attention	Obama Perplexity	W. Bush Perplexity	H. Bush Perplexity

Without Alibi	~390	~430	~480
With Alibi	~390	~430	~490

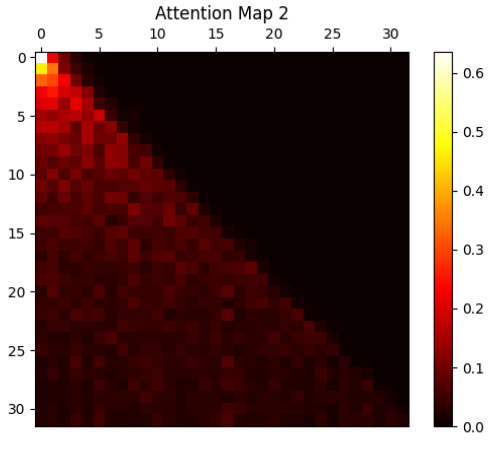
Comparison

The Number of Parameters in the decoder and encoder doesn't change. Compared to Standard Positional Encodings, AliBi has similar perplexity values on the test sets. In some cases, AliBi performed slightly better, especially on the 'Obama test set' where perplexity was reduced to around ~340. And the convergence time was reduced on my training set. On the attention map side, the Alibi's approach with biasing attention scores were more naturally distributed across the sequence, with stronger emphasis on nearby tokens.

layer1 with Alibi



layer1 without Alibi



Conclusion

AliBi turned out to be a useful alternative to traditional positional embeddings, showing slight improvements in some cases. The linear bias mechanism made it more efficient and gave a natural way to model positional relationships. However, AliBi is limited in our case due to the small size and lack of variability in the dataset, which likely led to overfitting. This means that the positional biases introduced by AliBi didn't have much impact. The model size also played a role—since our model was relatively small, with fewer attention heads and a lower embedding dimension, it couldn't fully leverage the biases from AliBi. Typically, AliBi works better in larger models where it can help prevent overfitting while still achieving good performance. With more attention, AliBi's bias would likely have been more effective.