CSE 256 Fall 2024, UCSD: PA2

Vedant Yogesh Deshpande
PID: A69032161

Part 1 : Encoder Trained with Classifier

Introduction

- This project focuses on building and training a transformer encoder along with a feedforward classifier to categorize speech segments, aiming to predict the politician most likely responsible for each segment. The transformer encoder captures intricate word relationships and contextual nuances within sentences, while the classifier leverages these encoded embeddings to make final predictions about the speaker.

Part 1.1: Transformer Encoder Implementation

- The encoder employs multi-head self-attention and positional embeddings to effectively capture word order and contextual relationships. In each attention head, scaled dot products are computed across tokens to yield distinct representations of word relationships, which are then normalized through a softmax operation to focus attention on relevant words.
- The encoder structure consists of multiple transformer blocks, each with LayerNorm, self-attention, dropout, and residual connections, providing stability and regularization during training. After processing through these layers, the embeddings are averaged to generate a single, context-rich vector that is then passed to the classifier.

Part 1.2: Feedforward Classifier Implementation

- The classifier is a straightforward feedforward network with a single hidden layer activated by ReLU. It receives the mean-pooled embeddings from the encoder, allowing it to interpret the encoder's outputs and produce probabilities for each politician. Using LogSoftmax, these probabilities indicate the model's confidence for each class, ensuring an interpretable and efficient output layer suitable for classification.

Part 1.3: Joint Training of Encoder and Classifier

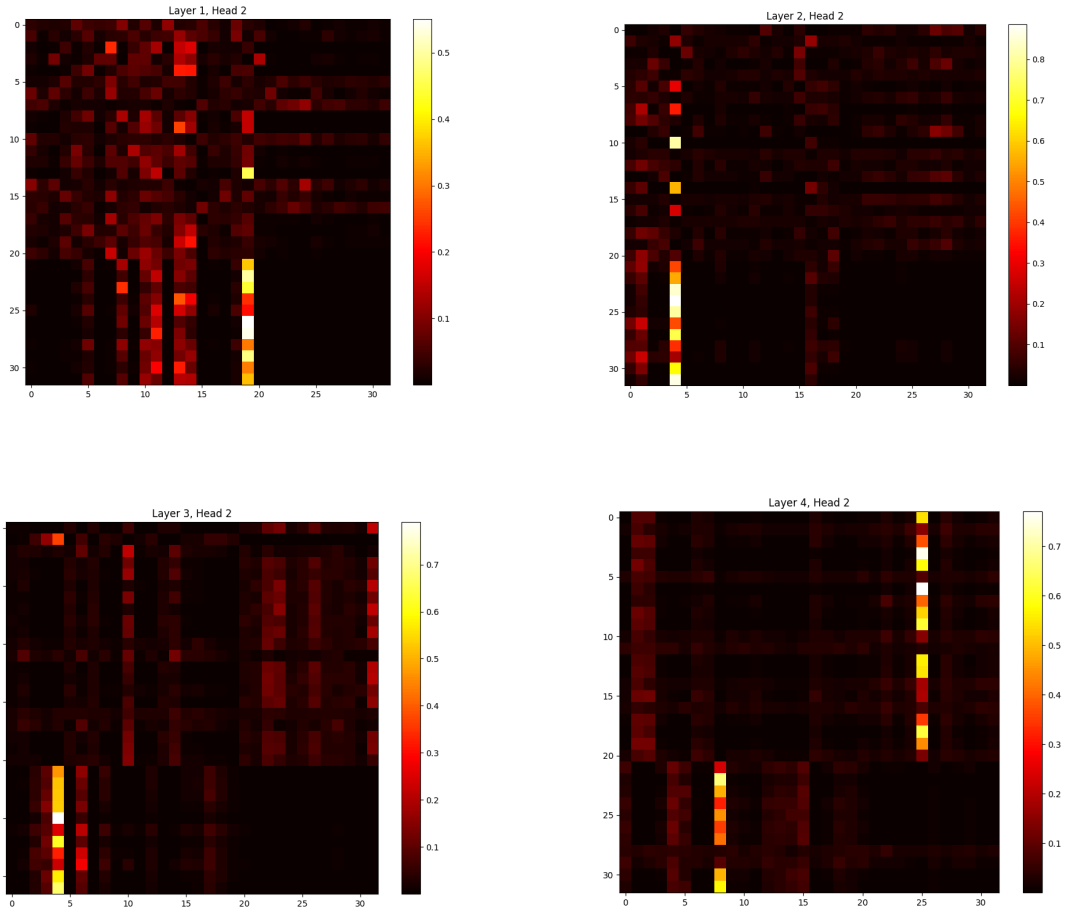
- The encoder and classifier are trained jointly with cross-entropy loss, aligning both components towards the classification objective. This end-to-end training

setup optimizes parameters within the encoder and classifier to enhance prediction accuracy, and monitoring accuracy over 15 epochs helped track model convergence and learning progress.

Part 1.4: Sanity Check and Attention Maps

- Sanity checks visualized attention matrices to confirm each row summed to one, verifying correct normalization of attention scores.

Figures 1-4: Attention maps for Layers 1, 2, 3 and 4 for Head 2



- Based on above figures, the following observations are made:
    1. The progressive refinement from scattered (Layer 1) to focused attention (Layer 4) suggests the model is learning to identify specific linguistic patterns or key words that are characteristic of different politicians.
    2. The high-intensity spots (yellow/white colors) become more pronounced and isolated in later layers, suggesting the model becomes more confident about which parts of the input are most relevant for classification.

3. The model's attention patterns evolve hierarchically across layers - from broad feature capture in Layer 1, culminating in highly selective final feature extraction in Layer 4.

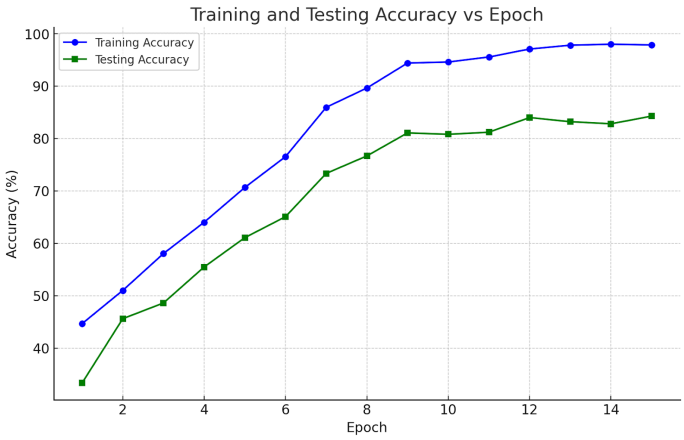
Part 1.5: Evaluation

- Starting with a training accuracy of 44.65% and a testing accuracy of 33.33% in epoch 1, the model's testing accuracy gradually improves to 84.27% by epoch 15. This steady increase indicates that the model is learning effectively, with the gap between training and testing accuracy remaining reasonably small by the end, suggesting good generalization.
- The model has a total of **576,339 parameters**.

Table 1: Training and Testing Accuracies

Epoch	Training Accuracy	Testing Accuracy
1	44.65	33.33
2	51.00	45.60
3	58.03	48.60
4	64.01	55.47
5	70.65	61.07
6	76.53	65.07
7	85.95	73.33
8	89.63	76.67
9	94.41	81.07
10	94.60	80.80
11	95.55	81.20
12	97.08	84.00
13	97.80	83.20
14	97.99	82.80
15	**97.85**	**84.27**

Graph 1: Accuracy vs Epochs



Part 2 : Pretraining Decoder Language Model

Introduction

- A transformer-based decoder was implemented for a language modeling task, aiming to predict the next word based on previous words. The decoder uses masked self-attention to prevent access to future tokens during training. It was pretrained on a limited dataset to minimize cross-entropy loss between predicted and true next words.

Part 2.1: Decoder Implementation

- **Masked Self-Attention:** A lower triangular matrix (tril) in the AttentionHead class ensures tokens only attend to themselves and preceding tokens, preventing lookahead during training.
- **Positional Embeddings:** The decoder combines positional embeddings with word embeddings, utilizing more context-rich information for making predictions.
- **Feedforward Layer:** After self-attention, each token passes through a feedforward neural network (ReLU activation), projecting it to a 100-dimensional hidden space.
- **Final Layer and Loss Calculation:** The lm_head layer projects embeddings to vocabulary-sized logits. Cross-entropy loss is computed between predicted and target tokens.

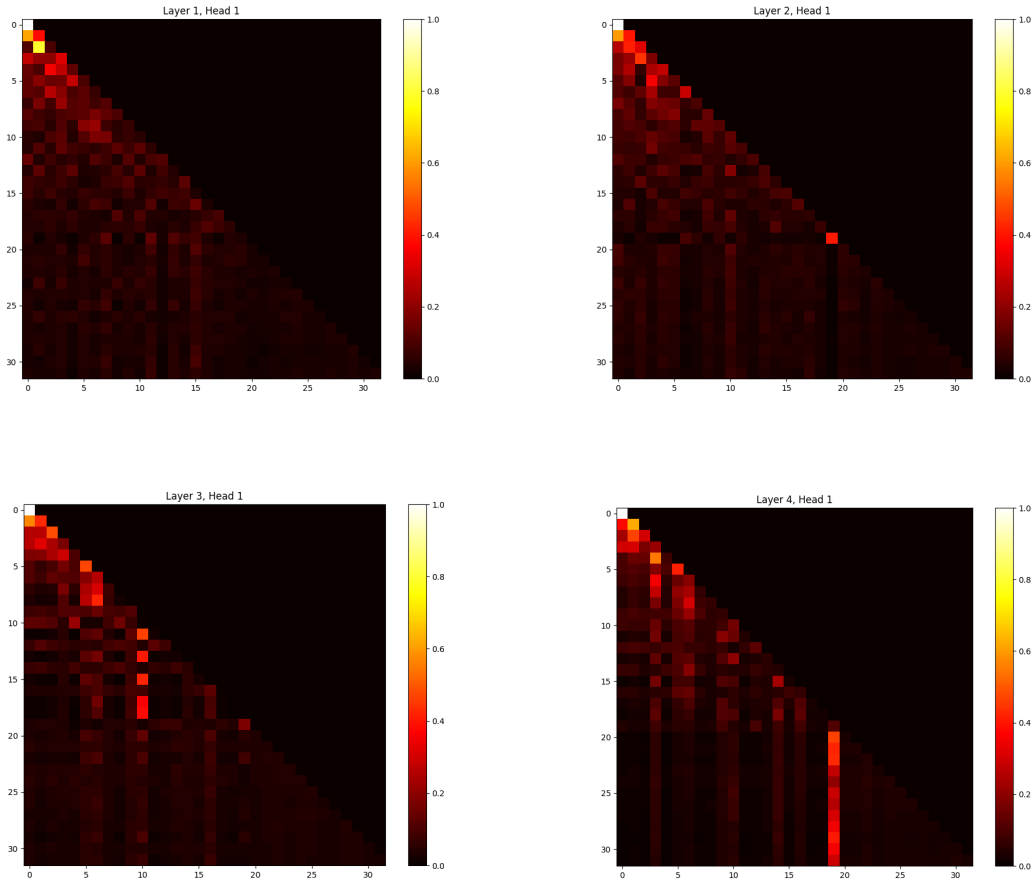
Part 2.2: Decoder Pretraining

- The decoder was pretrained on a language modeling task using cross-entropy loss. Training was performed on a small dataset (500 iterations, batch size 16, block size 32), processing about 256,000 tokens. The limited dataset led to higher perplexity, but was sufficient for evaluating masking and basic functionality.

Part 2.3: Sanity Checks

- The attention maps from Layer 1 to Layer 4 for Head 2 reveal consistent causal masking characteristic of decoder architectures, shown by the triangular pattern below the diagonal. All layers display strong diagonal attention patterns, with the most intense focus (brightest red) on immediate previous tokens in the top-left corner. As we move from lower to higher layers (Layer 1 to 4), the attention patterns become progressively more diffuse, suggesting a transition from local token relationships to more distributed attention across the sequence.

Figures 5-8: Attention maps for Layers 1, 2, 3 and 4 for Head 2



- Based on above figures, the following observations are made:
    1. In the above decoder attention maps, the upper triangular portion is masked (set to zero) since tokens can only attend to previous tokens, not future ones, ensuring causality in the language modeling task - each token can only use information from tokens that came before it.
    2. Attention evolves from scattered (Layer 1) to focused (Layer 4), indicating learned politician-specific language patterns.
    3. Later layers show stronger, isolated activations, reflecting increased model confidence in identifying relevant features.
    4. Attention progresses hierarchically from broad feature detection to precise feature selection across layers.

Part 2.4: Evaluation

- Training perplexity decreased consistently with each iteration, starting from 577.01 at iteration 100 and reaching 170.36 by iteration 500. This indicates gradual improvement in the model's ability to predict the next token.
- Testing perplexity was lowest for Obama at 373.27, indicating the model's relatively better performance on this text. Perplexity was higher for H. Bush at 423.66 and highest for W. Bush at 503.31, suggesting greater difficulty in predicting text for these figures.
- The model has a total of **943,739 parameters**.

Table 2 : Training Perplexities

Iteration	Training Perplexity
100	577.01
200	438.65
300	300.70
400	217.80
500	**170.36**

Table 3 : Testing Perplexities for each Politician

Politician	Testing Perplexity
H Bush	**423.66**
Obama	**373.27**
W Bush	**503.31**

- The differences in perplexity scores for sentences spoken by "H Bush" (423.66), "Obama" (373.27), and "W Bush" (503.31) likely reflect variations in language style, vocabulary, and structure in each speaker's phrasing. A lower perplexity score for "Obama" indicates his sentences may align more closely with patterns in the model's training data, making them easier for the model to predict. Conversely, "W Bush" has the highest perplexity, suggesting his sentence structures or word choices are less predictable or less frequently encountered by the model, leading to greater difficulty in prediction.

Part 3 : Architectural Exploration

WordPiece Tokenizer:

- The WordPiece tokenizer breaks words into subword units based on frequency, allowing uncommon words to be represented as combinations of smaller, common subwords.
- This approach balances vocabulary size with language coverage, improving handling of rare words and out-of-vocabulary terms in NLP models.

AliBi:

- AliBi (Attention with Linear Biases) is a positional encoding technique for transformer models that applies a linear bias to the attention scores, emphasizing closer tokens without fixed embeddings.
- This bias allows transformers to consider relative position information effectively, making it more adaptable to sequences of varying lengths.

CLS Token:

- The [CLS] token is a special token added at the beginning of input sequences in transformer models like BERT, used to represent the entire sequence's context.
- After processing, the embedding for this token is often used for classification tasks, summarizing information from all tokens in the sequence.

Evaluation Results:

Experimentation on Encoder

	Without exploration	Using CLS Token	Using WordPiece	Using Both
Training	97.86	96.41	98.56	98.13
Testing	84.27	83.33	84.4	86.8

- Using both [CLS] token and WordPiece tokenizer yielded the highest testing accuracy (86.8%), suggesting they complement each other well in capturing sequence context and handling rare words. Training accuracies were also high across configurations, with the WordPiece tokenizer alone reaching the peak at 98.56%.
- When compared to the SimpleTokenizer, the WordPiece tokenizer gives a similar accuracy result, as both methods are capable of effectively representing the underlying language structure. While WordPiece offers better handling of rare and out-of-vocabulary words by breaking them into subword units, the overall performance may be similar when the vocabulary is sufficiently covered by both

tokenizers, allowing the model to learn meaningful patterns regardless of the tokenization strategy.

- The accuracy difference between finding the mean of the embeddings and using the CLS token is minimal, indicating both methods are valid for classification. This small gap suggests both approaches are capturing essential features, with no clear advantage for either.

Experimentation on Decoder

	Without exploration	Using AliBi	Using WordPiece	Using Both
Training	170.36	117.50	153.45	103.60
H Bush	423.66	395.60	211.55	167.80
Obama	373.27	349.55	207.72	162.36
W Bush	503.31	472.15	245.37	196.01

- The table presents performance metrics for three configurations: AliBi, WordPiece, and both combined, on the training and testing datasets.
- The training data achieved scores of 117.50 with AliBi, 153.45 with WordPiece, and 103.60 with the combined method. W Bush showed the poorest results across all methods (472.15 with AliBi, 245.37 with WordPiece, and 196.01 combined). H Bush and Obama yielded similar patterns of performance, with AliBi showing the highest (worst) scores (395.60 and 349.55 respectively), WordPiece performing better (211.55 and 207.72), and the combined approach consistently providing the best results (167.80 and 162.36).
- The results show that the WordPiece tokenizer and AliBi method complement each other well by addressing different aspects. WordPiece improves handling of rare words through subword units, while AliBi enhances relative position awareness in the attention mechanism. Together, they enable better word representation and token ordering, leading to improved performance.
- WordPiece works well for language modeling due to its fine-grained tokenization, capturing context, but for classification tasks, it may dilute semantic meaning and increase sequence length, which can hurt performance. Coarse tokenization methods often provide clearer, more meaningful representations for classification.

References:

1. Let’s Build GPT from scratch - https://www.youtube.com/watch?v=kCc8FmEb1nY
2. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation - https://arxiv.org/abs/2108.12409
3. PyTorch Documentation - https://pytorch.org/docs/stable/index.html
4. ChatGPT for conceptual understanding
5. PPT provided in class