# Speech Transformer NLP Classifier

Hayden Kwok

May 2024

## 1 Introduction

In this report, I will go over the application and investigation of the transformer architecture components in the corresponding codebase. Using speeches from US politicians as the dataset, this project entails creating and training transformer encoder and decoder models for natural language processing tasks.

The first challenge is to anticipate which politician will deliver a particular speech segment by implementing and training a transformer encoder with a feedforward classifier. To do this, a transformer encoder must be made, high-dimensional embeddings must be produced, and a classifier must be used to provide predictions based on the embeddings.

In order to predict the next word in a sequence, the second task focuses on pretraining a word-level transformer decoder on a language modeling problem using an autoregressive approach akin to GPT. During training, the model employs masked self-attention to prevent future token peeking, and perplexity measures are used to assess the model's performance on several speech datasets. Furthermore, using alterations to the model's architecture and hyperparameters, I experimented with numerous architectural modifications, such as alternative positional encodings and sparse attention patterns, to enhance the classifier's accuracy or the decoder's confusion.

Speech clips from George W. Bush, George H. Bush, and Barack Obama make up the dataset, which allows the encoder to do a three-way classification task and the decoder to perform a word prediction task. The pretraining of the transformer decoder, architectural research, and transformer encoder and classifier implementation are all covered in this study. The model design, training procedure, assessment metrics, and empirical findings are covered in detail in each area. The report seeks to present a thorough comprehension of transformer models as well as the learnings from this task.

## 2 Encoder Trained With Classifier

Here, I implemented a feedforward neural network classifier that takes in embeddings of the text data from the transformer encoder.

## 2.1 Sanity Check

I employed a sanity check to ensure that the attention mechanism is functioning properly: rows sum to 1 and to analyze the resulting attention matrices. After running the code, the checks pass; here are the resulting graphs of the first and eighth attention maps:

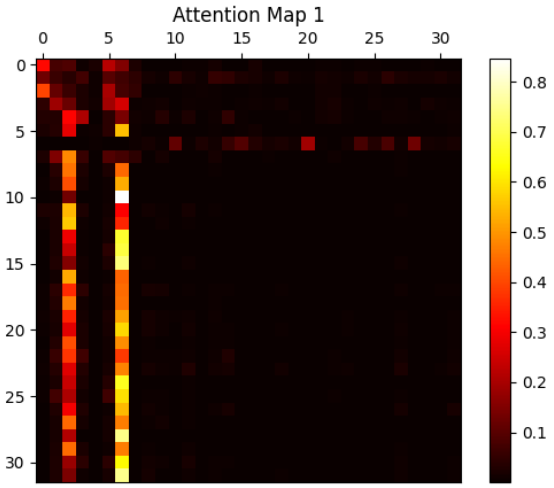


Figure 1: Attention Map 1

The attention matrix for layer 1 shows a dispersed pattern of attention, where a less dense spread is seen across multiple rows and columns with high attention scores. This indicates that at the early stages of the sequence processing, the model is attending to a broader range of tokens, and the more prevalent presence of prominent vertical and horizontal lines, suggests that the model is giving more attention to specific tokens (around the 2-6 range) and could indicate higher importance of the early stages to understand the input. Overall, the attention map shows that the model is learning to effectively distribute its focus across various parts of the input in the early layer which will allow the future output of predictions to be more accurate.

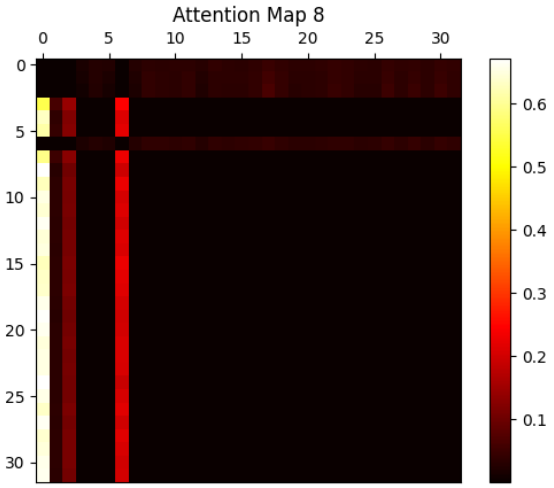


Figure 2: Attention Map 8

The attention matrix of the 8th layer focuses more deeply on the depth of a certain set of tokens (still within the 0-6 range) which indicates that the model has learned to identify those sections as more crucial parts of the sequence during its processing of deeper layers. The horizontal and vertical lines, similarly but more intensely than layer 1, show that the model is effectively narrowing down its focus to the most relevant parts of the input, which indicates its successful learning process and allows the output classification to be more accurate.

## 2.2 Evaluation

The model is trained using a dataset with 880,146 parameters and a vocabulary size of 5,755.

| Epoch | Train Loss | Train Acc (%) | Test Acc (%) |
|---|---|---|---|
| 1 | 1.081 | 44.646 | 33.333 |
| 2 | 1.070 | 44.646 | 33.333 |
| 3 | 1.021 | 53.872 | 48.667 |
| 4 | 0.951 | 56.549 | 48.000 |
| 5 | 0.893 | 58.222 | 52.533 |
| 6 | 0.810 | 65.440 | 56.133 |
| 7 | 0.741 | 66.874 | 58.000 |
| 8 | 0.656 | 77.294 | 66.533 |
| 9 | 0.584 | 80.736 | 68.000 |
| 10 | 0.523 | 84.512 | 72.400 |
| 11 | 0.427 | 85.038 | 74.933 |
| 12 | 0.376 | 90.583 | 79.733 |
| 13 | 0.301 | 92.878 | 80.933 |
| 14 | 0.245 | 94.598 | 81.867 |
| 15 | 0.178 | 94.646 | 81.200 |

Table 1: Train and Test Accuracies per Epoch

Based on Table 1, we see a decrease over time in the training loss, from 1.081 in epoch 1 to 0.178 in epoch 15, which has subsequently led to an upward trending increase in the training accuracy of our model (44.646% to 94.646%). This signifies that are model is learning and is properly classifying the embeddings. Although our testing accuracy shows a corresponding upward trend, a notable gap between testing and training accuracies forms at the later epochs, with training accuracy continuing to increase while testing increases comparably slowly, suggesting a potential situation of overfitting. To address this, in the Performance Improvements section I employ dropout to the models to reduce overfitting and maintain high accuracy within both the training and testing data.

# 3 Pretraining Decoder Language Model

## 3.1 Sanity Check

I employed a sanity check to ensure that the attention mechanism is functioning properly: rows sum to 1 and to analyze the resulting attention matrices. After running the code, the checks pass; here are the resulting graphs of the first and eighth attention maps:

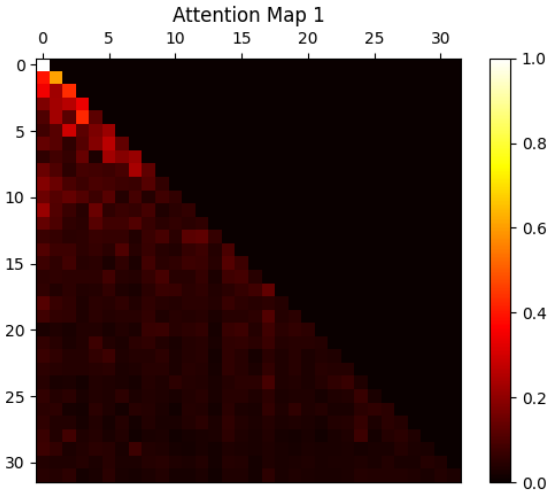


Figure 3: Attention Map 1

Similarly to the encoder's attention matrix 1, the attention matrix 1 for the decoder emulates a similar dispersion throughout the diagonal with a higher intensity at the start of the sequence. This shows that the model is attending to the broad scope of the sequence and is looking at the initial tokens to better capture a wider range of contextual information that can improve the model's understanding at later layers of the sequence with lesser context. On a linguistics level, it could affect the output by gathering a more diverse set of contextual cues, which then sets a strong foundation to predict parts of the speech in subsequent layers.

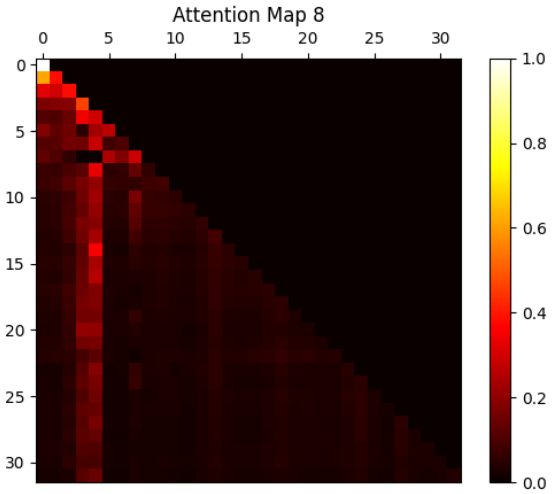


Figure 4: Attention Map 8

In the 8th attention matrix of the decoder, we see a somewhat similar pattern to the encoder of concentration and selectivity within relevant parts of the sequence. This shows that the model is now able to make more accurate predictions based on the given context, and with the refinement in deeper layers allows the capturing of long-term dependencies and key relationships within the data. Now that the model has a more thorough understanding of the potential input sequence, the effect on the output would be that the model is more likely to generate more coherent and contextually appropriate sequences by leveraging the more focused attention of deeper layers.

## 3.2 Evaluation

In a language modeling challenge, the decoder is trained to predict the following word in a sequence based on the words that came before it. Each vocabulary word's probability is included in the decoder's output. The training aims to reduce the cross-entropy loss between the sequence's true next word and these predicted probabilities.

The model is trained with a total number of parameters at 863,243 and a vocabulary size of 5,755. Due to time constraints, we restrict the training in our language modeling base setting to 500 iterations (batches). The block size is set to 32 and each batch has a size of 16. As a result, during training, about 256,000 tokens are processed and as a result, the perplexity numbers are not expected to be particularly low.

| Iteration | Train Loss | Train Perplexity |
|---|---|---|
| 100 | 6.478 | 585.963 |
| 200 | 5.956 | 419.052 |
| 300 | 5.501 | 288.210 |
| 400 | 5.265 | 213.702 |
| 500 | 5.129 | 165.630 |

Table 2: Training Perplexities at Different Iterations

| Iteration | Obama Perplexity | H.Bush Perplexity | W.Bush Perplexity |
|---|---|---|---|
| 500 | 425.5246 | 451.6575 | 578.1260 |

Table 3: Test Perplexity at Iteration 500

Table 2 showcases the training data for each 100 iterations. We can see here that the training loss decreases steadily while the perplexity follows in that same trend, indicating that the model is effectively learning, and increasing its accuracy in predicting the next token in the sequence.

Based on Table 3, we can see from the final perplexities of the three datasets after 500 iterations that the perplexities vary with Obama, H.Bush, and W.Bush coming in from lowest to highest respectively. We can infer that the model's training dataset aligns more closely to that of the Obama dataset, meaning on a high level that the language structures and language are more similar between the two. On the contrary, the higher perplexity for the W.Bush dataset indicates that the language could be more complex or be less represented similarly in the training data.

# 4 Bonus Exploration

## 4.1 Architectural Exploration

Unfortunately, due to extenuating circumstances, I was not able to undergo experiments for this section.

## 4.2 Performance Improvements

In this section, we have previously identified that the existing Econder Classifier model has demonstrated overfitting which we will employ dropout (at 0.1) to attempt to reduce it.

I also increased the epochs by 15, totaling 30, and increased the hidden size by 1000 to give the model more room to improve per epoch and to better emphasize the usage of dropout.

| Epoch | Train Loss | Train Acc (%) | Test Acc (%) |
|---|---|---|---|
| 1 | 1.094 | 44.694 | 33.467 |
| 2 | 1.066 | 47.658 | 38.933 |
| 3 | 1.029 | 44.933 | 33.733 |
| 4 | 0.993 | 56.262 | 48.133 |
| 5 | 0.929 | 60.564 | 50.933 |
| 6 | 0.883 | 63.671 | 52.533 |
| 7 | 0.868 | 70.124 | 58.933 |
| 8 | 0.753 | 79.111 | 68.400 |
| 9 | 0.677 | 79.924 | 68.133 |
| 10 | 0.591 | 85.468 | 73.200 |
| 11 | 0.536 | 87.715 | 74.933 |
| 12 | 0.469 | 90.488 | 75.867 |
| 13 | 0.399 | 92.352 | 79.333 |
| 14 | 0.369 | 93.881 | 79.733 |
| 15 | 0.338 | 95.124 | 81.200 |
| 16 | 0.295 | 96.415 | 82.400 |
| 17 | 0.284 | 97.323 | 83.733 |
| 18 | 0.232 | 97.658 | 82.800 |
| 19 | 0.205 | 97.562 | 82.933 |
| 20 | 0.199 | 95.937 | 82.533 |
| 21 | 0.153 | 98.757 | 85.867 |
| 22 | 0.190 | 97.228 | 83.733 |
| 23 | 0.161 | 98.470 | 84.133 |
| 24 | 0.138 | 98.423 | 85.467 |
| 25 | 0.140 | 99.044 | 84.533 |
| 26 | 0.117 | 98.757 | 84.533 |
| 27 | 0.099 | 99.522 | 85.333 |
| 28 | 0.113 | 99.235 | 85.733 |
| 29 | 0.103 | 99.522 | 85.600 |
| 30 | 0.109 | 99.713 | 86.933 |

Table 4: Train and Test Accuracies per Epoch

Based on Table 4, the model shows significant improvement over 30 epochs, with training loss decreasing from 1.094 to 0.109 and training accuracy increasing from 44.694% to 99.713% (previously 94.646%). In terms of evaluation, the testing accuracy improves from 33.467% to 86.933% (previously 81.20%), indicating better generalization to unseen data. The additional epochs lead to further gains in both training and testing accuracy, while the increased hidden dimensions allowed the model to capture more complex patterns. The dropout slightly mitigated overfitting, which overall promoted better generalization. These changes are anticipated to result in higher accuracy and more better performance.

Although the performance improvement is very minimal (<10%), the gap

between the testing and training accuracies appears to stabilize and remain constant. To take this a step further, it would be reasonable to employ early stopping methods to prevent the training from continuing after it has reached its peak and stabilized, reducing further overfitting.

As expected, the sanity check passes and the graphs are similar, demonstrating a change from breadth to depth in terms of attention to specific tokens.

I referenced the nanoGPT implementation for this PA2. [1]

# References

[1] Andrej Karpathy. nanogpt. `https://github.com/karpathy/nanoGPT/blob/master/model.py`, 2023. Accessed: 2024-05-19.