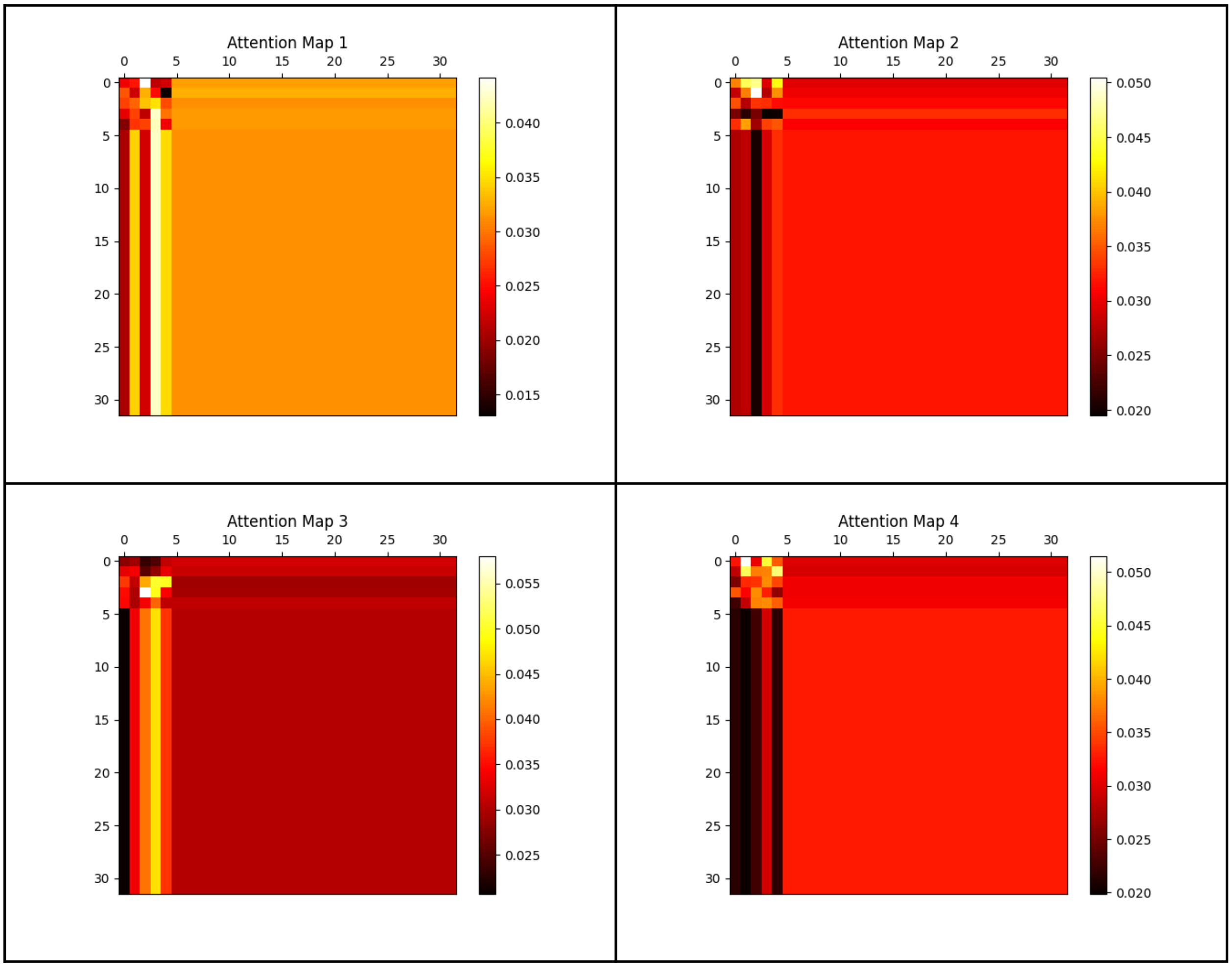
PA2 Write Up

Mehul Maheshwari

1 Encoder with Trained Classifier

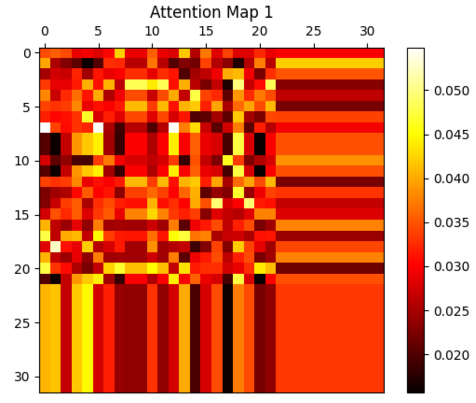
Attention Diagrams and Observations

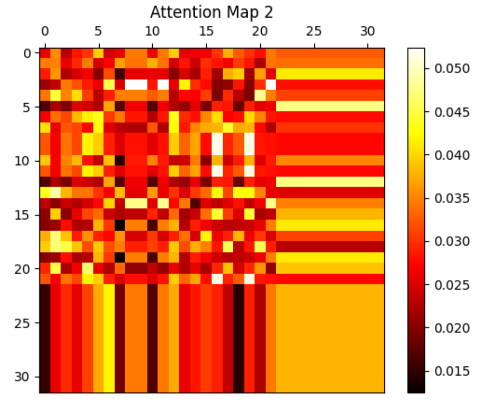
Given the test sentence “This is a test sentence”, we can see the following results on the sanity check function:

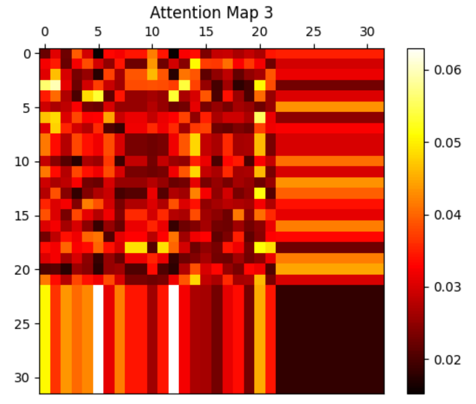


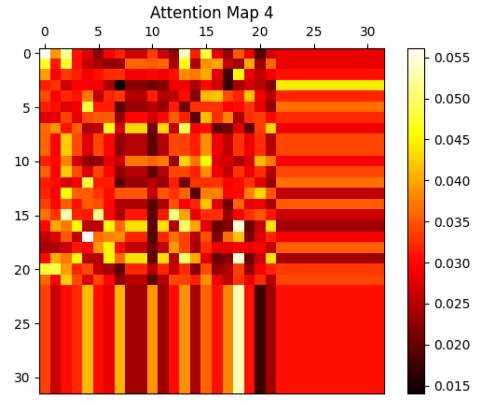
As we expect, each of the rows sum up to 1. Since this particular sentence was not very long, the rest of the 32 space is not being used, but we can see weights are being dynamically changed to make predictions. With a longer sentence we should expect to see a more filled attention map.

Lets try the test sentence “This is a much longer, and perhaps unnecessarily verbose test sentence, designed to show the capabilities of the attention mechanism”





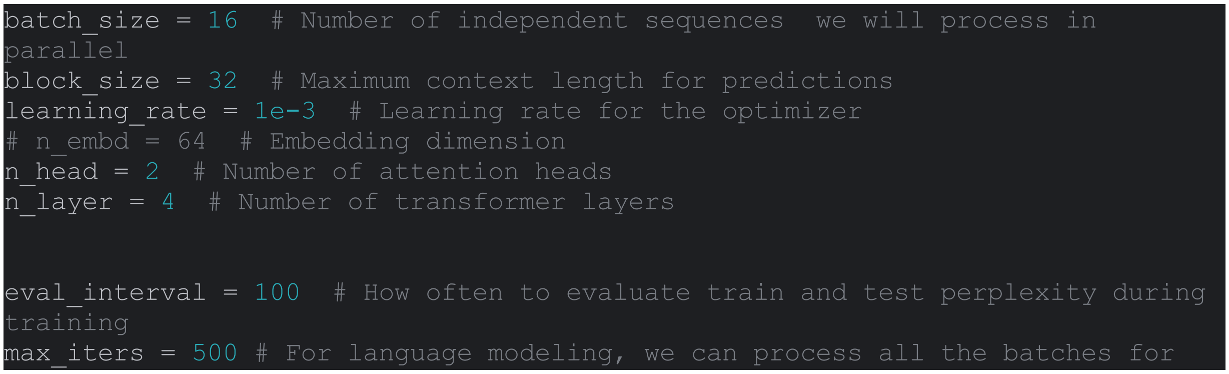




As expected, these attention maps are a lot more full, and we can now more clearly see how the different weights are scaled with different attentions.

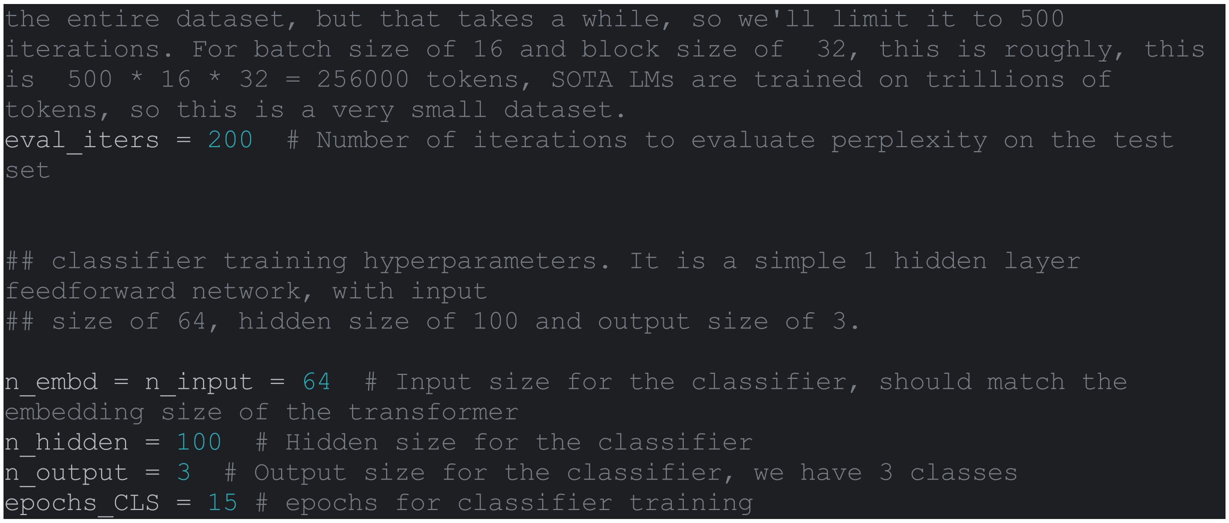
Accuracies by Epoch

Now let's test a classifier that is trained using the encoder. Note that the encoder and classifier use the exact same parameters as in the starter code, which is the following:



```
batch_size = 16  # Number of independent sequences  we will process in
parallel
block_size = 32  # Maximum context length for predictions
learning_rate = 1e-3  # Learning rate for the optimizer
# n_embd = 64  # Embedding dimension
n_head = 2  # Number of attention heads
n_layer = 4  # Number of transformer layers


eval_interval = 100  # How often to evaluate train and test perplexity during
training
max_iters = 500 # For language modeling, we can process all the batches for
```

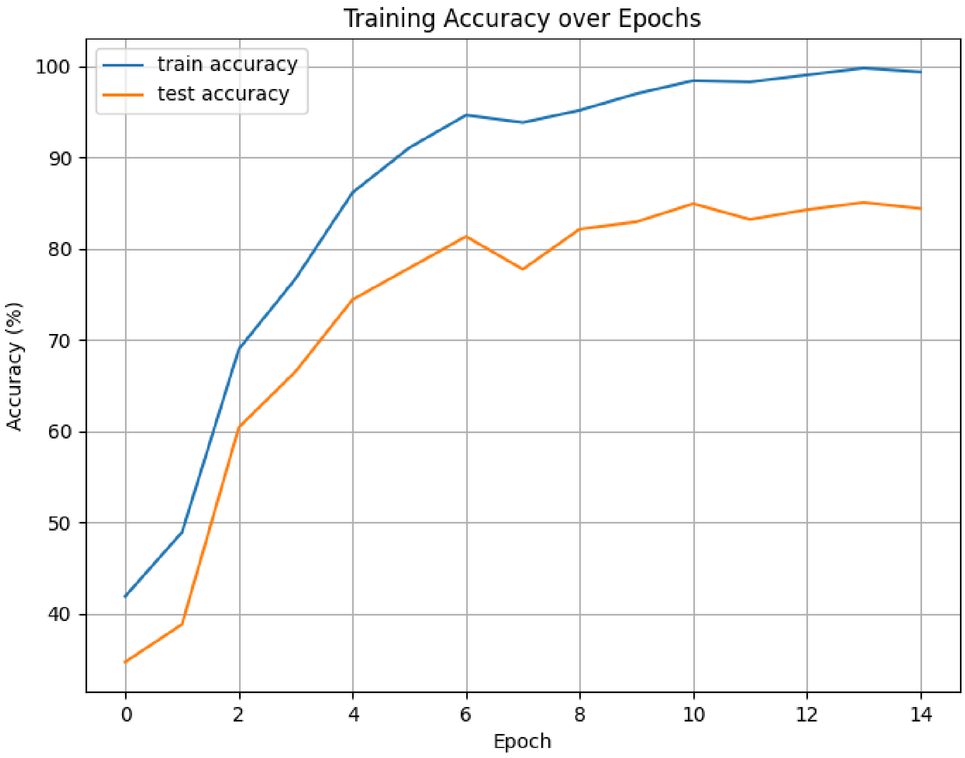


```
the entire dataset, but that takes a while, so we'll limit it to 500
iterations. For batch size of 16 and block size of  32, this is roughly, this
is  500 * 16 * 32 = 256000 tokens, SOTA LMs are trained on trillions of
tokens, so this is a very small dataset.
eval_iters = 200  # Number of iterations to evaluate perplexity on the test
set


## classifier training hyperparameters. It is a simple 1 hidden layer
feedforward network, with input
## size of 64, hidden size of 100 and output size of 3.

n_embd = n_input = 64  # Input size for the classifier, should match the
embedding size of the transformer
n_hidden = 100  # Hidden size for the classifier
n_output = 3  # Output size for the classifier, we have 3 classes
epochs_CLS = 15 # epochs for classifier training
```

I then ran the classifier model and got the following results:



Final Accuracy on testing set, Epoch [15/15]

Loss: 0.013630636036396027, Train Accuracy: 97.70554493307840%, Test Accuracy: 86.4%

This follows our expectations for how the model should perform (training is better than testing, and final testing is about mid 80's).

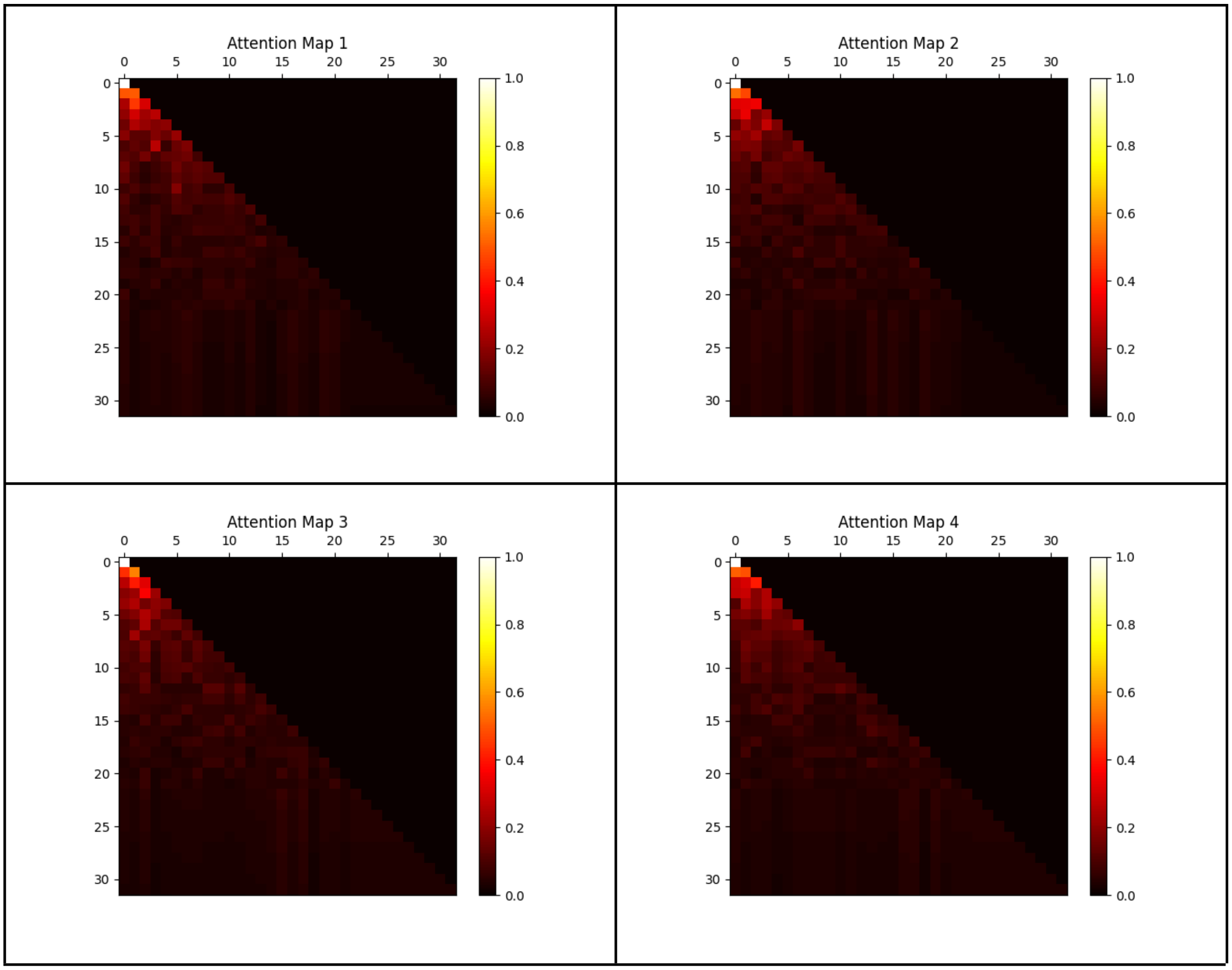
2 Decoder and Prediction

Setup

For this part, I implemented a transformed decoder that uses self-attention masking. As specified, the feedforward hidden dimension was 100, input/output dim was n_embed, and the activation was ReLU.

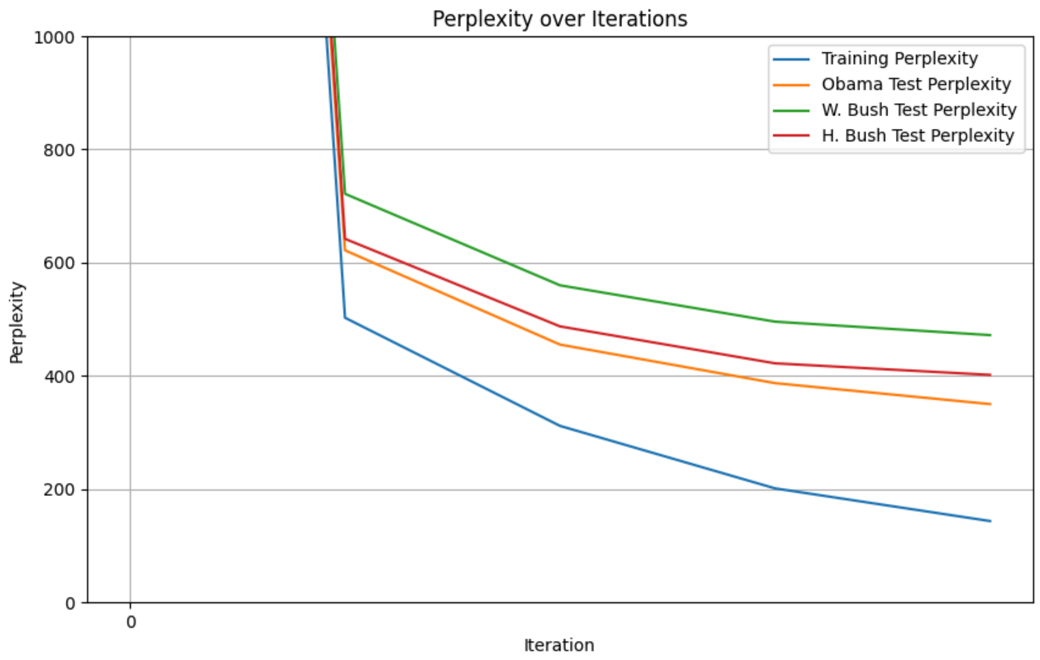
Note that for this question, I only trained on the training set, but I recorded the perplexities of the test sets as well. This is the same as pre-training on the training, and reporting the final perplexities, with the only difference being in my implementation you can also see the performance on the test sets through the iterations.

Sanity Check / Attention Maps



Here we can see that the attention maps are lower triangular. This is happening because the mask is cutting off any tokens that come from after the current token we are at, which prevents the decoder from ‘peeking’ at the word we are trying to predict.

Results



Training Perplexities by 100 iterations:
[6393.73, 502.88, 311.97, 201.67, 142.18]

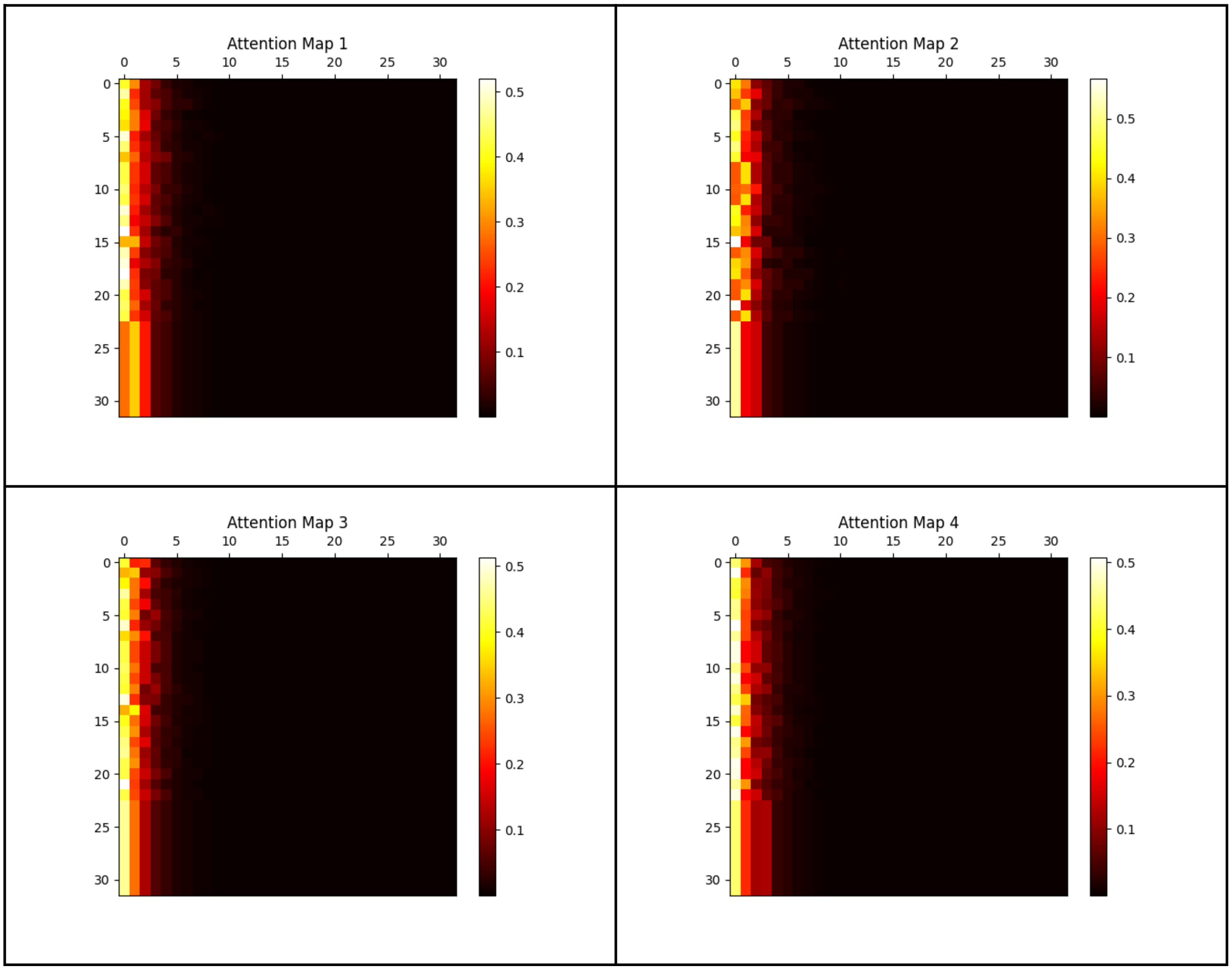
Final Obama Test Perplexity: 349.95
Final W. Bush Test Perplexity: 471.83
Final G. Bush Test Perplexity: 401.59
Number of parameters in the decoder: 961803

Because the initial perplexities were in the thousands, I scaled down the graph to make it a little bit easier to read. From these numbers, what we expected to see is happening (training perplexity is lower, and the rest of the datasets are higher). Perplexity is a measure of e^(mean loss), which means if the perplexity is higher, the next word is harder to predict. It could be due to sentence structure, vocabulary, longer sentences, or a combination of all three factors that make W. Bush’s speeches' more unpredictable as compared to the others.

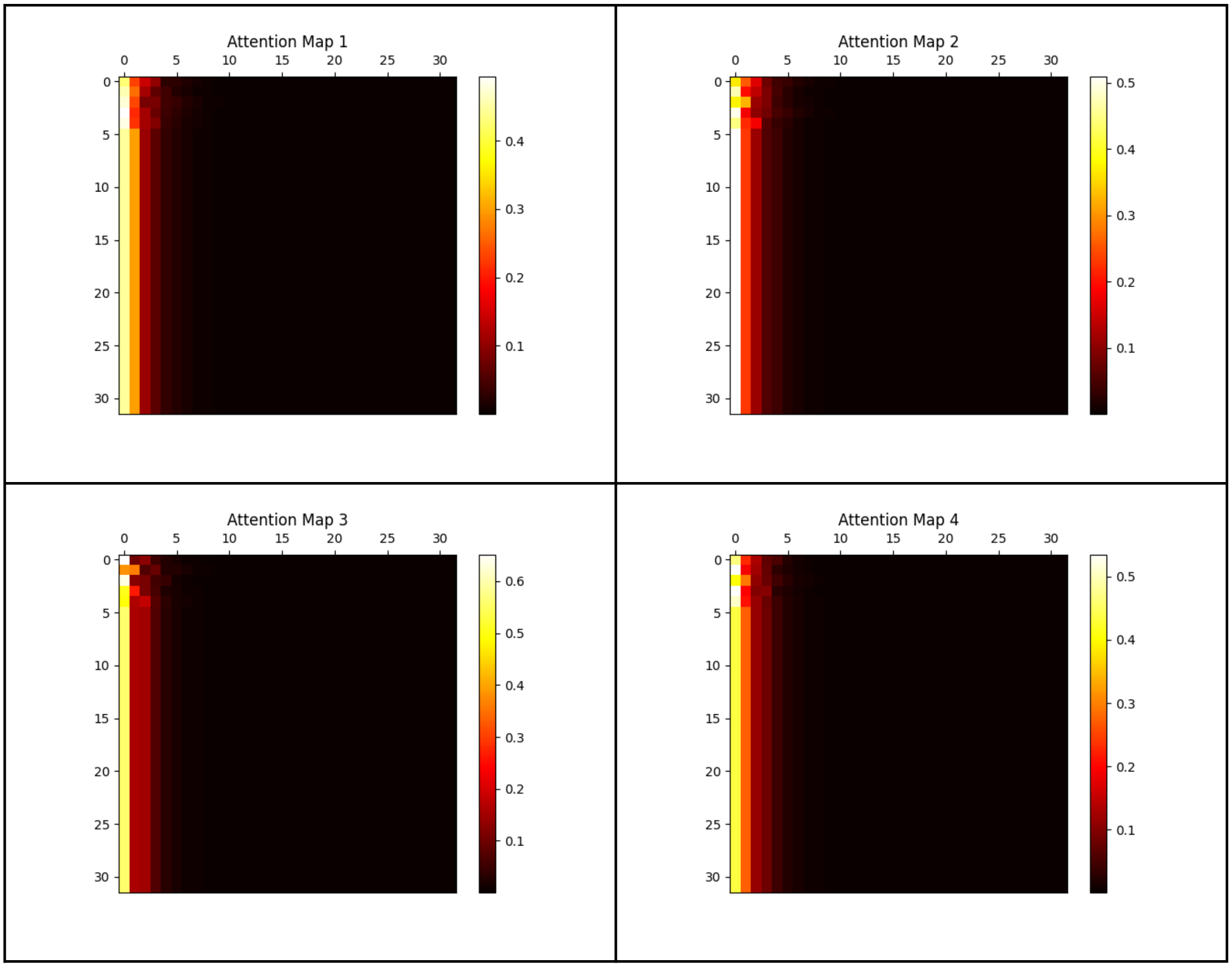
3 Architectural Exploration

For this section, I decided to explore using AliBi encodings instead of positional encodings. The benefit of AliBi encodings is that they scale effectively with longer sentences by weighing nearby tokens more. I kept most of the implementation factors the same between this and question 1 so that I could really see the difference that the alibi encodings make on the model.

To effectively demonstrate its abilities, let's take a look at how the attention maps differ for the long test sentence. As compared to the original attention map, we would expect to see that the attention is focused on more recent tokens. Here is the new attention map with positional encodings:

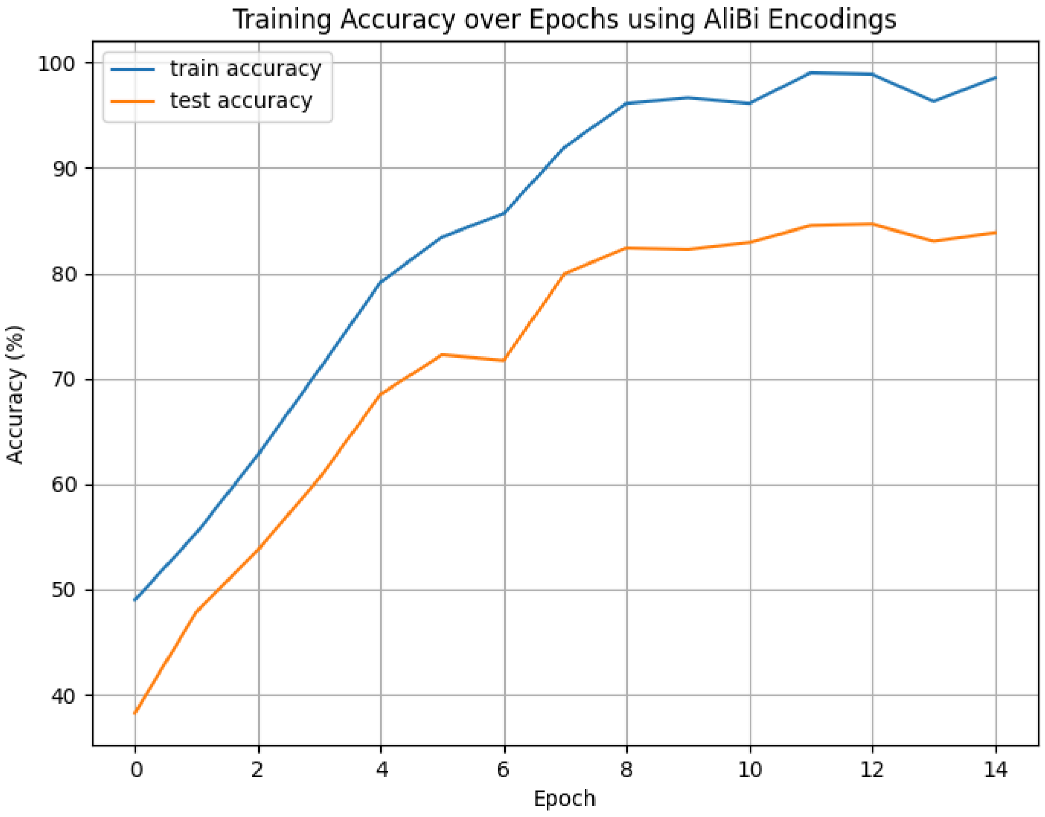


There is a big difference here! Even though the sentence size is much larger, we can see the focus of the weight is really with the first few tokens, and not so much with what happens later (which is what we expected). To really illustrate this, let's see what happens if we use the small test sentence from part 1.



With the smaller sentence size, we can see that the attention maps clearly have the focus on the first few tokens which mirrors the large sentence size. This is completely different from how the positional encoding encoder worked.
You can also see the difference in sentence sizes between these two results. In the long sentence results, around 22 tokens down from the y axis, we see that the variation stops which indicates the sentence has ended whereas in the short results, this occurs around 4 tokens in.

Lets see how the alibi encoding model fares on the training and test sets.



```
Epoch [15/15]: Train Accuracy: 98.56%, Test Accuracy: 83.86%
```

Recall, our model from part 1 had a final test accuracy of 86.5% which means that our alibi encoding did not make a significant difference in the classification accuracy. This however, does not mean the alibi did not help. After all, the classifier we used stayed the same, so since there was no architectural change, it makes sense that the classification model's ability remained the same.

Furthermore, the AliBi encoding brings benefits that cannot be represented via accuracy. For example, since most of the weighting is at the nearby tokens, we can actually save on space by using alibi encodings as opposed to positional encodings. We can also guarantee stable results, even if sentence size increases (we know this because of the attention maps), whereas we cannot say the same for the positional encoding model which clearly showed drastic variations.