Gabriella Rich 11/17/2025

Assignment 3 Applied Machine Learning

Code: https://drive.google.com/file/d/19W1kIuMXuD-uCci-BY7RegW8E-MpCytZ/view?usp=sharing

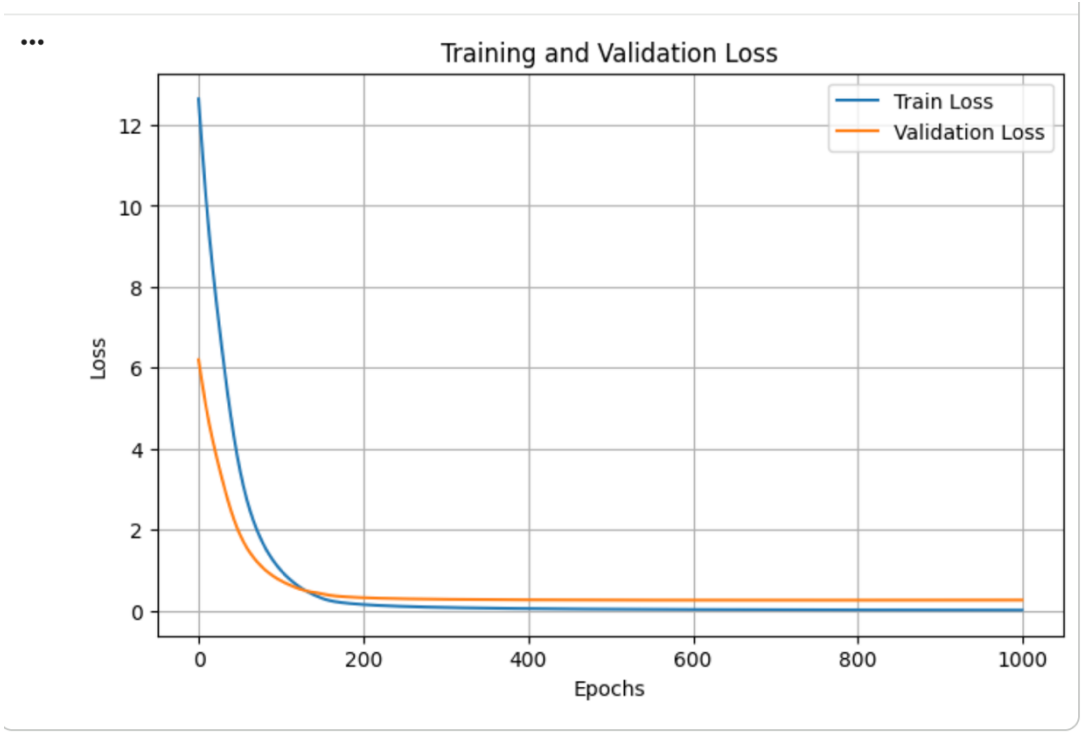
The goal for this assignment is to implement the core logic of a Transformer block from scratch and train this tiny Transformer on the Shakespeare Tiny Corpus dataset. It is a common corpus used for training and experimenting with different language models. Given the previous tokens can the transformer essentially predict the next token in the set. In order to train our Transformer first we have to work on splitting the text into tokens which are often sub words that have a language pattern that can be reused. In order to do this, we have to begin by using tokenization to split the text. For this assignment, I used Byte-Pair Encoding (BPE) with the help of chat GPT. I chose Byte Pair encoding over WordPiece and SentencePiece, but they are really similar in nature. Byte Pair encoding relies on a pre-tokenizer that splits the training data into words while WordPiece initializes the vocabulary to include every character present in the training data and learns the number of merge rules. After tokenization I split the text into overlapping sequences of inputs and targets and split up the data. Now it was time to begin building the transformer.

Building the transformer from scratch took me some time even with the help of chat GPT. I ran into a lot of issues which I wasn't aware of until I began the visualization phase. I began with the positional encoding class which gives the transformer the information about the order of tokens in a sequence. Positional encoding goes before the self-attention model because x needs to contain its positional information. Although a model can function without assigning explicit positional encodings to each token, the model's ability to understand data can become

impaired so I thought it better not to take that risk since without it is losses the order of information. Then I worked with Multi-Head Self Attention. This was the tricky part. Originally, I had worked with Self attention, but I found it easier to process multi-head self-attention when building the attention heatmaps later on. Mutli-Head Self Attention builds on self-attention and processes the same way as self-attention but in parallel to one another to trin the model to have a better understanding of the tokens. Then we apply a causal mask to ensure that the model does not look into the future and begin making up predictions. Then building the transformer block every block contains self-attention and chat GPT actually advised to add RMSNorm to prevent vanishing activations as it skips mean subtraction making it faster. Then within our Tiny Transformer class we use embedding to convert integer tokens IDs to the dimensional vectors and feed forward to multiply the inputs by weights to transform the matrices. It returns logits, which are used to record the loss and all_atn which will be used later on for the visualization of the tokenization. After the model is built, I can start to visualize the results and see what was going on.

First, I had to define clear training hyperparameters which chatGPT and the assignment specified for me. Starting with the learning rate, it controls how big each update step is during training and AI helped me work through what was a good value because it informed me that a learning rate that's too high causes the model to diverge while a learning rate that's too low causes the model to learn at a slow rate. The model metrics were specified in the code from class and I just adjusted the values to fit specific parameters (e.g vocab size = 500) and added in the sequence length and number heads coming from the Multi Head attention. Chat GPT added in the optimizer Adam to handle sparse gradients since it adapts the gradient size per parameter. Then we train it using cross-entropy loss to penalize the model when it assigns low probability to

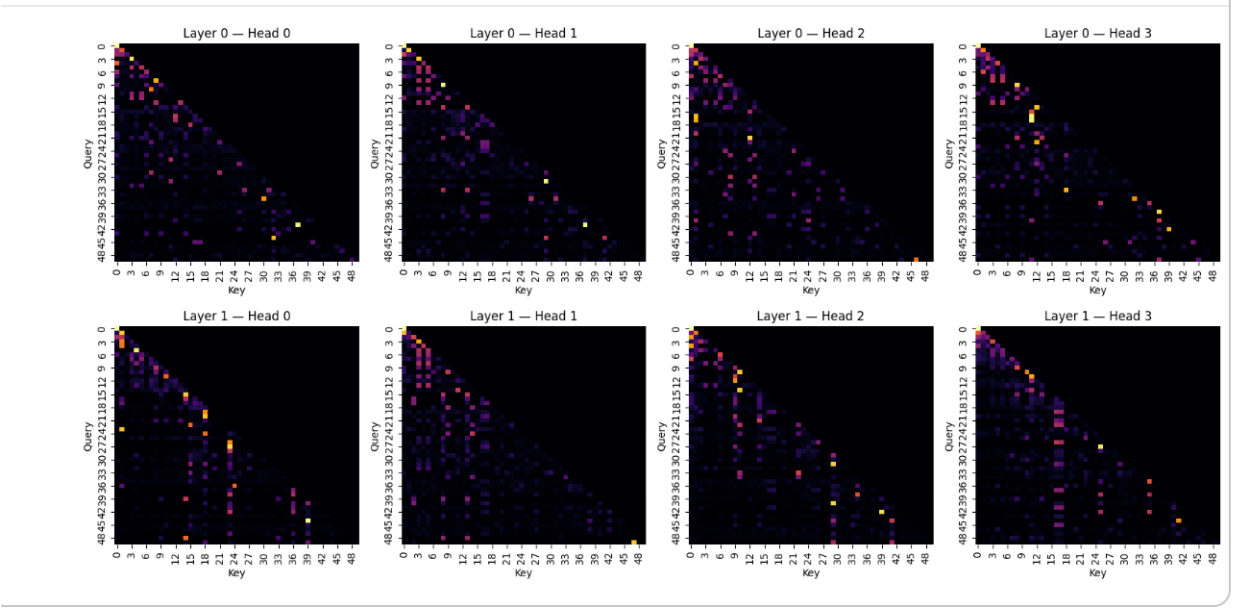
the correct next token which is important in training to make the correct predictions. To ensure everything was working smoothly it is important to visualize the training and validation loss curves over epochs. Below is the visualization of my final training loss and visualization but this is where I started having issues with the model.



After about 175 epochs, both of the losses for both curves go down to .2 and .3 respectively and remains stable around those points for the rest of the epochs. This means that the model is able to generalize well overall which means it is working as intended. When the curve was first plotted, I was getting that my training loss was 0.0 over every epoch. This was a major indicator that something was going wrong. Chat GPT helped me debug and figured out that my training data was not split properly so the targets were identical to the inputs so it was just copying itself and therefore there was no loss but there was also no training being done. No predictions made was bad for me and I worked to fix it. Then I plotted the attention heatmaps to visualize which tokens

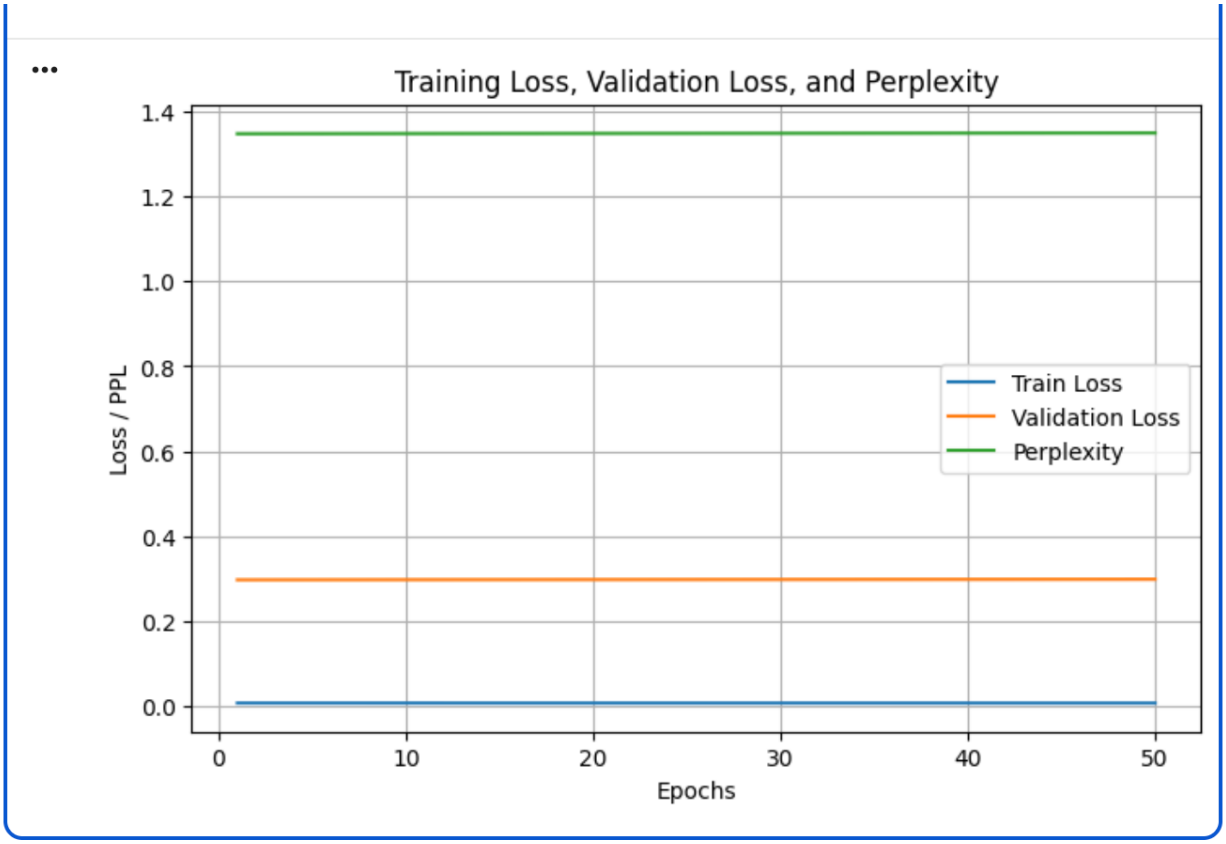
attend to which for layer 0-1 from the four heads 0-3. In the image below, the brightest spots (yellow and purple and red) reflect a strong diagonal and that for layer 0 heads they learn from their dependencies like the neighboring tokens but in layer 1 you can see some long range attention which reflects the structure that the model is using.

```
for layer_idx, attn in enumerate(all_attn):
    plot_multihead_attn(attn, layer_idx)
```



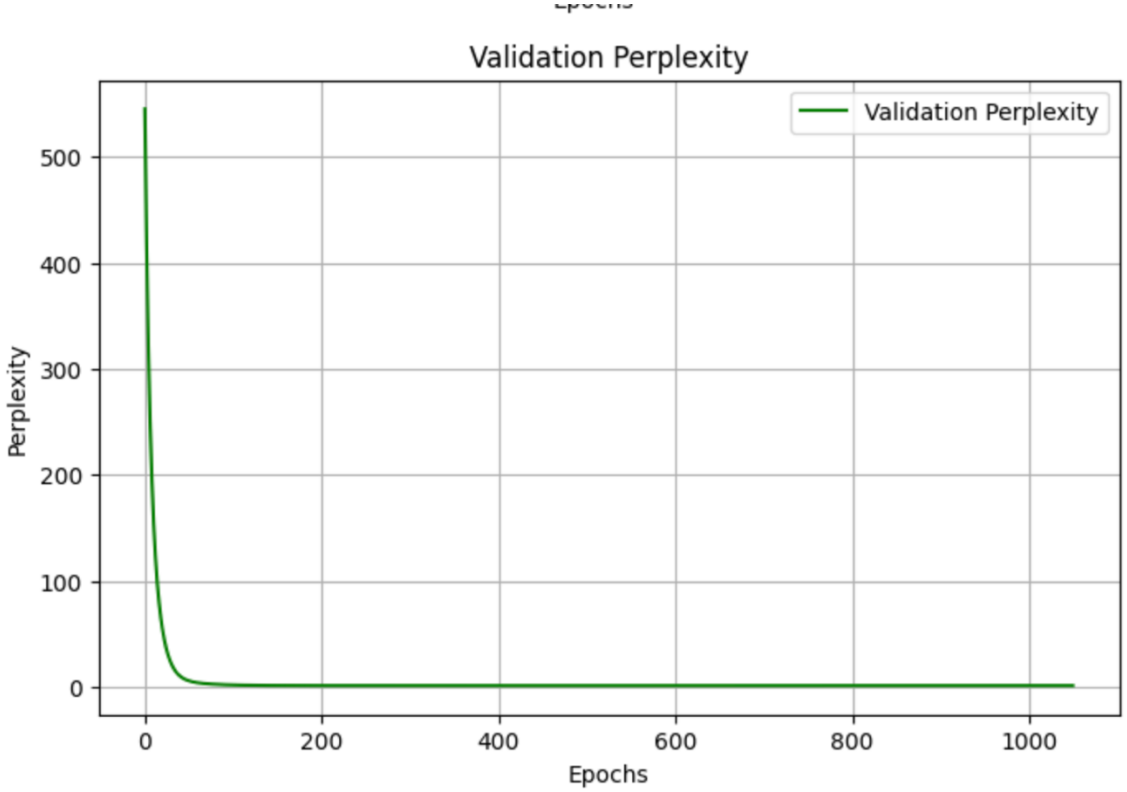
Another visualization tactic was to find and plot the perplexity on the validation set. Perplexity is a metric to measure the confidence of the model to predict the next token of the text sequence. Low perplexity is better because it means there are less surprises. A perplexity close to 1 over all epochs means that the model predicts every token almost perfectly. Since I am computing the perplexity values only from validation loss the values after convergence differ by really minimal amounts. I was pretty suspicious of this but every time I tried to fix this the line would either not show up at all in plots making visualization impossible or it would be completely flat. According to Chat GPT this means my model is completely fine but I was so

confused. Usually, lines are not completely flat especially given the fact that before I added perplexity training loss and validation were not flat. I'm not sure how to fix this issue and what it means. It was frustrating because Chat GPT is unwilling to debug something it deems as correct and I tried a few things such as decreasing the number of epochs, changing the scales to see close up differences, switching up the graph styles.



Thankfully I realized my mistake myself! I wanted to separate the first training and validation curve to make my report visually nicer and include the comparison to perplexity towards the end in order to compare interpretations. That ended up being my downfall because I was retraining already trained data, so nothing was changing for training loss, validation loss and perplexity remained the same the entire time. Luckily when I added perplexity to my original model I got

the below image outputted and was able to see that perplexity does start really high which is bad but eventually gets lower and lower which means that there are less surprises for the model.



There were several takeaways that I gathered from this project and working on building Tiny Transformers. One takeaway is that the bigger the dataset the harder the builder has to work to ensure that the model predicts correctly. Since this was such a small dataset with such a small vocab size and I only built a two Transformer blocks I was able to build a pretty accurate model but in reality, it's much harder. With larger dataset, the process of tokenization is incredibly important as it makes it easier for the model to process the data and predict correctly. Another takeaway I got from this is that it won't go the way you want all the time. I experienced an issue with runtime and not having enough memory to run some parts of this project. For example, at the end I wanted to test out my model using sample generations to predict the next token. I got it

to work luckily but unfortunately the constraint was that the model could only compute one singular next token prediction. Sadly, anything past one next token generation flags an immediate Runtime error in the code and I am still unsure how to fix that. Chat GPT informed me that this is because I made my sequence length maximum equal to 50 so I could increase that to get it to work or use sinusoidal positional encoding. Overall, it was able to predict the next 1 token which is decent. Building a Tiny Transformer from scratch helped me understand how to improve its performance and why each feature is needed in the first place.

I used mostly Chat GPT for the assignment. It generated most of the code for me or modified professor's code from class. I attempted to use Claude for debugging at one point for the perplexity confusion, but it ended up failing worse than Chat GPT because it wasn't aware of the scope of my work for the project and what was already performed. I wrote this entire paper on my own and analyzed the visualizations myself as well but Chat GPT was used to clear up confusion on certain concepts associated with the project.