Neural Networks, Homework 3

Mihnea-Alexandru Vîrlan

1 Processing the dataset

Since Shakespeare's plays are organized into verses, I mainly followed the recommended approach outlined in the homework's description.

At first, for every combination of Play's title, followed by the line number, I concatenated all the verses belonging to the corresponding number. The concatenation was done until meeting a new different line number. I also removed the newlines separated the verses, and added a newline after concatenating all the verses that make up a player line. The processing was done sequentially, from the top of the dataset, sample by sample. This approach also assures that I do not combine play lines from different acts, or scenes, even if they correspond to the same line number. Also, before each line of a character, its name is prepended such that, the corpus contains lines in the form of $< Character\_name >:< Actual\_line >$.

85% of the corpus was used for training, and 15% for testing. The split is guaranteed not to have parts of a play in training and the rest in testing.

2 Implementing the model

The individual components of the model were implemented according to the paper "Attention is all you need" [1]. The source code `gpt_components.py` contains the implementation of the model with its main parts: the multi-head attention, the Transformer layers, the sinusoidal embeddings and a manual implementation of the layer normalization.

For the experiments, the following parameters have been used for the big and small model

1. Big model
   (a) 8 attention heads
   (b) 8 Transformer layers
   (c) feature dimension $d_{model} = 512$
2. Small model
   (a) 6 attention heads
   (b) 7 Transformer layers
   (c) feature dimension $d_{model} = 256$

3 Experiments

For all the experiments involved a learning rate schedule was used, being made of a linear warmup with a number of steps depending on the tokenization method, and a cosine annealing after the warmup steps, until the end of the training. Also, the number of epochs involved was 8, and the optimizer used was Adam with Weight Decay of 0.01, with 0.01 being the default weight decay in Pytorch's AdamW. The batch size was set to 32 for all the experiments.

When using the subword tokenization, 50 warmup steps were used and every element of the batch contained pieces of text ammounting 128 words that were tokenized accordingly. Using the character tokenization, 500 warmup steps were used, and every element of the batch contained 62 consecutive letters. In the latter case, after adding the tokens for the beggining of sequence and end of sequence, a batch element will have 64 tokens, 62 of which are letters.

3.1 The big model

3.1.1 Loss plots

Figure 1: Train loss curve: solid line - subword, dashed line - char

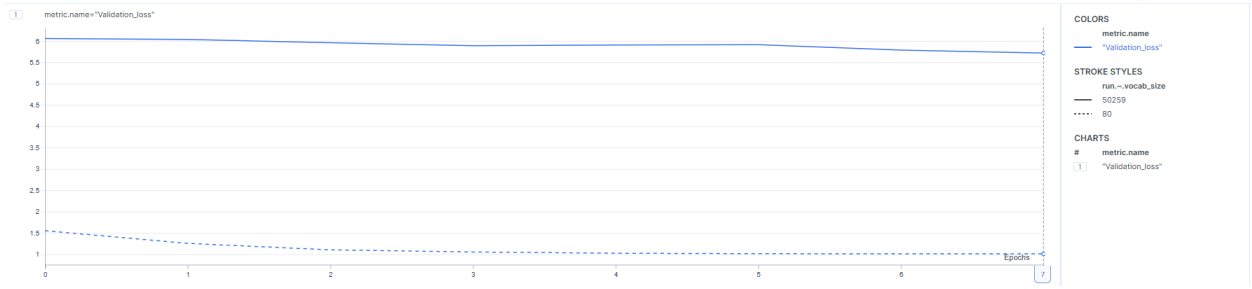

Figure 2: Validation loss curve

The above figures of the loss plots for training and testing show that the model handles the individual characters better than subwords.

3.1.2 Perplexity plot

Figure 3: Perplexity plot

The perplexity plot is correlated to the validation losses evolution. With the subword tokenization one having a value way higher than the character one, we may conclude that the big model struggles to learn the patterns of Shakespearean plays with regards to the subword than characters, leading to the model predicting characters better.

3.2 Small model

During the experiments with the small model, similar patterns in loss evolution were observed, as well as perplexity.

3.2.1 Loss plots

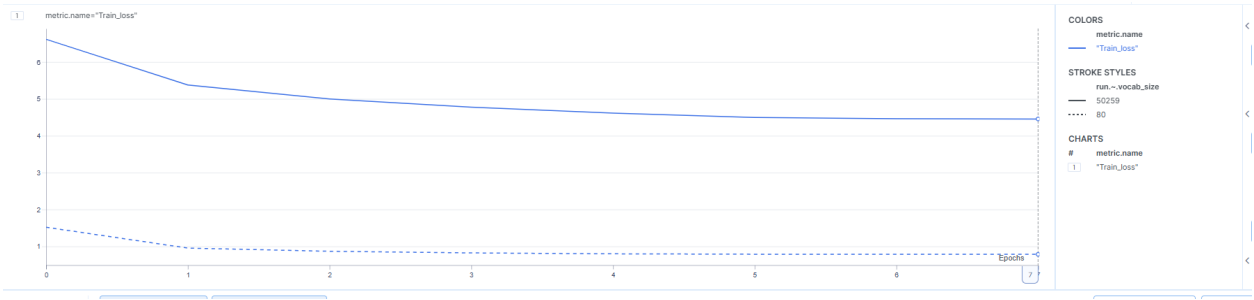

Figure 4: Train loss small model

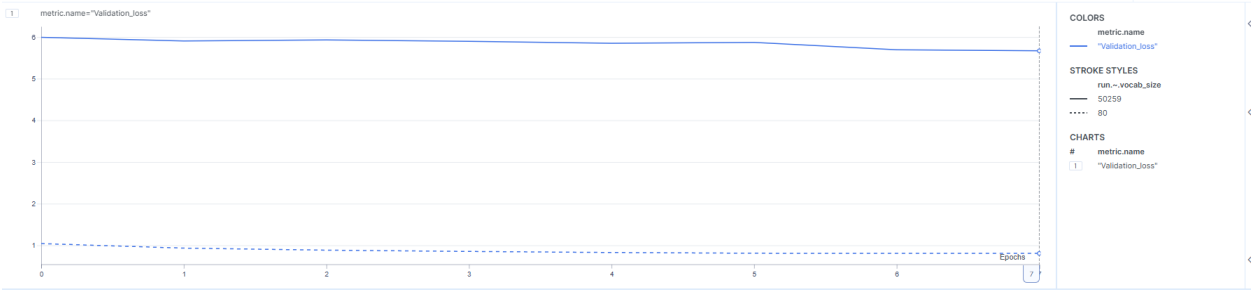

Figure 5: Validation loss small model

3.2.2 Perplexity plot

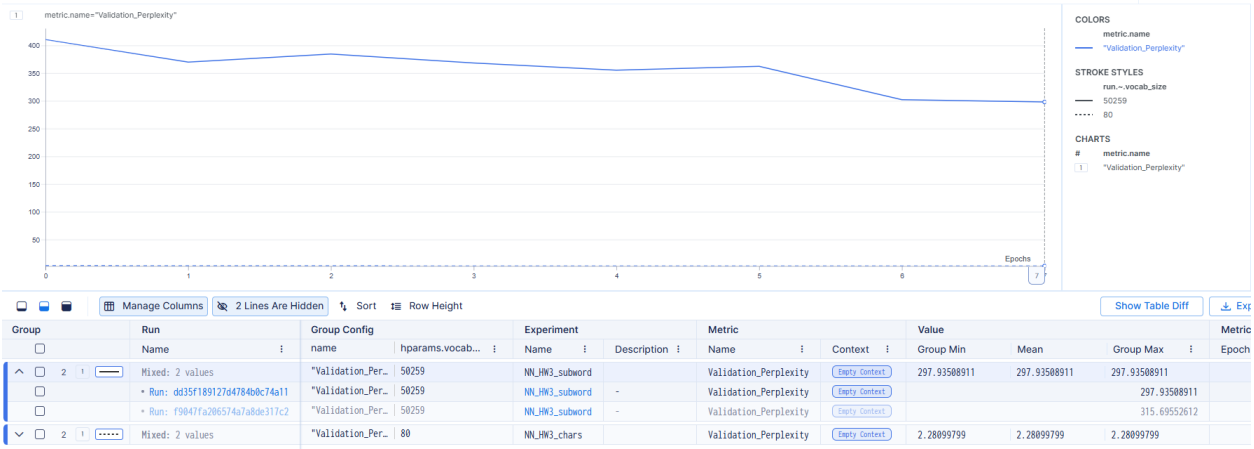

Figure 6: Validation perplexity small model

3.3 Comparison of perplexities

Now, for each tokenization scheme we will show the differences in perplexities

3.3.1 Subword

Figure 7: Subword Perplexities Blue-small model, Green-the big model

Interestingly, this smaller model appeared to have a better perplexity, showing that, in this case a increased number in attention heads and transformer layers may produce an overfitting.

3.3.2 Character

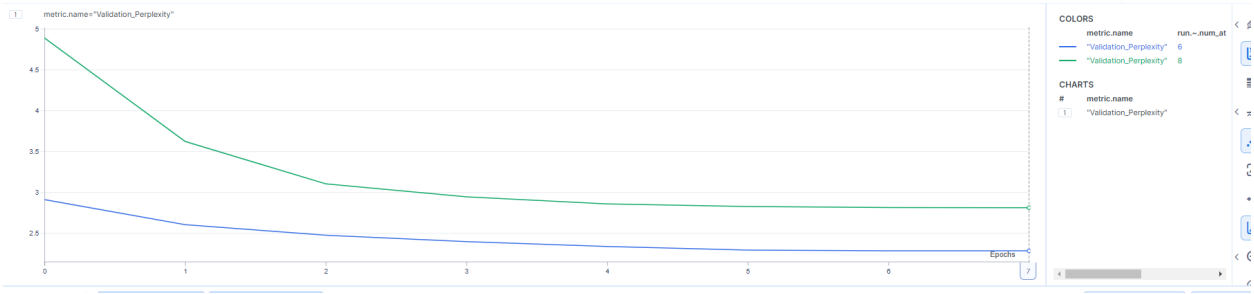

Figure 8: Character perplexities Blue small model, Green - the big model

The pattern is therefore met again in the case of character perplexities, even if they are in the range 1.6-3.6.

4 BLEU/ROUGE scores

For testing the quality of the generated texts, I took the first 10 lines from the test set. For each metric there will be a table. For this, I used the perplexity and bleu scores from torchmetrics, and in the case of ROUGE, I took only the fmeasure of each of the ROUGE1,ROUGE2,ROUGEL.

4.1 BLEU1

	Subword	Char
Large GPT	0.0677	0.0005
Small GPT	0.0474	0.0028

4.2 BLEU2

	Subword	Char
Large GPT	0.014	0
Small GPT	0.0092	0

4.3 BLEU3

	Subword	Char
Large GPT	0	0
Small GPT	0	0

4.4 BLEU4

	Subword	Char
Large GPT	0	0
Small GPT	0	0

4.5 ROUGE1

	Subword	Char
Large GPT	0.1281	0.0678
Small GPT	0.0898	0.0710

4.6 ROUGE2

	Subword	Char
Large GPT	0.0283	0.0167
Small GPT	0.0102	0.0161

4.7 ROUGEL

	Subword	Char
Large GPT	0.1196	0.0638
Small GPT	0.0825	0.0679

We are observing something counterintuitive: altough the larger GPT did not perform well with regards to the perplexity, we see that it does tend to obtain better, although very low scores. This may underline the fact that a perplexity on characters does not guarantee a high quality text.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL `https://arxiv.org/abs/1706.03762`.