Brian Tran, Connor Whynott, Tyrone Xue

Assignment 3

Dataset and Preprocessing

The dataset that we used consisted of over 167-thousand tab-separated English-French sentence pairs. These sentences ranged from short and simple phrases such as “call me” and “we’ll go” to much longer and more complex sentences such as “how much money did you spend on your last holiday?” This extensive dataset was provided to us in a .txt file by Professor Chen. As for loading and preprocessing this data, we first had to read in the tab-separated file line by line and split each line into English and French sentences. We then used SentencePiece for byte pair encoding, or BPE, to create a vocabulary of subword units with a size of 8,000. We landed on this size because it was large enough that it ensured an efficient handing of more rare words and retained semantic meaning across the dataset but not too large where it would cause issues regarding computational power. We then split this dataset into a training and validation set with an 80-20 split. To ensure uniform input lengths during our model training, we padded sequences to the maximum length within each batch. Finally, we implemented Dataset and DataLoader from PyTorch to handle batching and shuffling during training.

Model Architecture

We implemented a standard Transformer architecture using PyTorch Lightning for a streamlined training and evaluation. For our encoder, we used multi-head self-attention mechanisms and feed-forward layers. Positional encodings were added to input embeddings to retain sequence order information. For our decoder, we incorporated masked self-attention and encoder-decoder attention layers. This allowed our model to create context-aware French translations for the English input sequences. For our output layer, we mapped the decoder’s output to the target vocabulary by using a linear transformation. This was followed by a softmax activation.

Training Strategies and Experimental Setup

We trained our model on an NVIDIA RTX 3070 GPU which allowed us to get a runtime of about seven to eight minutes per epoch. This means that our training of eleven epochs, we had a total run time of about 90 minutes, on average. For our optimizer, we decided to use an Adam optimizer with a learning rate of .0001. We did not implement a scheduler to dynamically change our learning rate. We had a previous model with a scheduler, but found this model to perform better. For our batch size, we landed on 32 to balance speed and resources efficiency. We also implemented early stopping to help prevent overfitting by using PyTorch Lightning callbacks that monitored our validation loss and gradient clipping to prevent gradient exploding. We found this setup to leverage our GPU hardware and balance computing time and model performance.

Evaluation Results

Our train and validation loss can be seen in Figure 1 below. Our BLEU curve can be seen in Figure 2 below. The maximum BLEU we achieved over 11 epochs was 0.14096. The W&B dashboard for this run of our model can be found here.

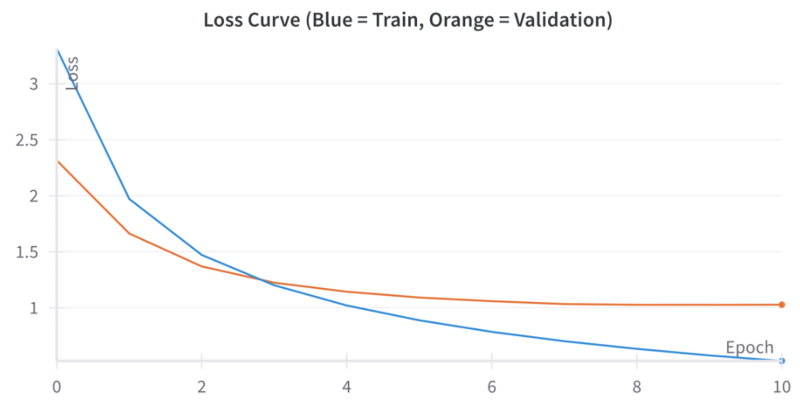

Figure 1: Loss Curve

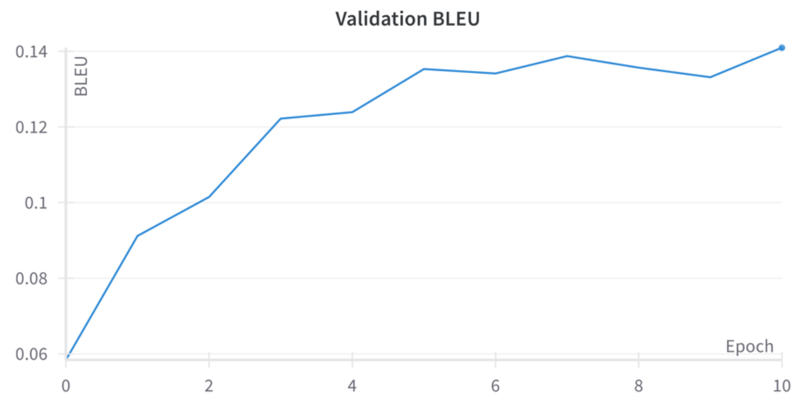

Figure 2: BLEU Curve

Challenges

Debugging was certainly challenging due to the long running time of the code. While training on the powerful GPU reduced training times compared to an M1 MacBook (about 8 vs. 40 minutes per epoch), iterative debugging and hyperparameter tuning was limited. These runtime limitations also prevented us from thoroughly exploring alternative hyperparameter configurations, potentially leaving performance improvements untapped. Neither of these challenges could necessarily be overcome. This project would require a lot more time that we did not have to really improve our model's performance.

Our biggest challenge that we faced was figuring out the inference file. After many hours of trying to figure out, we opted to go with an alternative to running the inference file. Please see the README document on how to run it. Please understand that we spent countless hours on the inference file and decided that this was the best that we could provide.