

Problem 1:

In the cell which runs both the `get_vocab` and `encode` method, we see a portion of the input text which appears to be a script for a Shakespeare play, a list of unique characters of letters and symbols under `Vocab`, and an encoding where the text "Romeo and Juliet" where each character is mapped onto integers. Since there is a numeric mapping for each character it implies that the model we are building doesn't really know words, sentences, grammar, etc. Instead it's learning patterns within a string of characters. Since each character has an integer mapping we think that the model is predicting which character is going to come next by assigning probabilities to each of the 66 characters in the vocab and choosing the next letter with the greatest probability.

Problem 2:

After running the transformer language model for 20000 steps I've noticed for the first thousand steps, the training loss and test loss decreases linearly. During the first thousand steps, it appears that both the losses are approximately the same. Over the next couple thousand steps the decrease in both losses appears to still be linear, however the slopes for each loss becomes much less steep. Beyond 5000 steps there is an increase in steps appears to barely decrease both losses. After the first 1000 steps, the loss in training slowly becomes lower than the test loss as more steps are taken.

Problem 3:

The `generate` function works by predicting the next character one character at a time by using probabilities for each character in our vocab. These probabilities are calculated by getting the raw scores and softmaxing them. The line `logits, = self(tokens[:, -ctxsize :])` is used so that the characters are used so that we only use the most recent characters in order to predict the next character. We limit the amount of characters used to predict the next character so that the context for the prediction is recent and relevant to the characters directly behind it.

Problem 4:

The variable `device` controls if the model is trained using the GPU or the CPU. When we remove the `device = device` when we try training our open we get an error and this error happens because there is a type mismatch since PyTorch requires all the tensors that are interacting with each other to be on the same device. Our model is on the GPU, but the input tensor is on the CPU.

Problem 5:

In the training loop, the variables `X`, `y`, `logits`, and `loss` all have specific dimensions and purposes for how the model processes the data. `X` is the input batch, and its dimensions are `[batch_size, ctx_size]`. The `batch_size` refers to how many sequences are being processed at once, and `ctx_size` is the length of each sequence in the batch. The input sequences in `X` are made up of tokenized characters, where each character has been mapped to an integer. The variable `y` has the same dimensions as `X` because it's basically the target sequences that the model is learning to predict. It's just a shifted version of `X` where each position represents the next character in the sequence, so the model can learn what comes next. The `logits` variable, which is the output of

the model, has dimensions [batch_size, ctx_size, vocab_size]. Here, vocab_size is the total number of unique characters in the dataset. For each position in the input sequence, logits contains a vector of raw scores that represent the model's prediction for the next character across all characters in the vocab. Finally, loss is a scalar value that measures how far off the model's predictions are from the actual target values in y. This is calculated using cross-entropy loss, which is used to guide the model's training by adjusting its weights to minimize this value over time. Overall, the batch dimension lets multiple sequences to be processed at once, which makes training more efficient.