Homework 4

(Self-attention Layer as Fully Connected Layer)[1] We talked about self-attention in the class. Assume a sequence of N inputs, each a d-dimentional vector. Denote the input as a matrix $\mathbf{X} \in \mathbb{R}^{N\times d}$, self-attention matrix as $\mathbf{A} \in [0,1]^{N\times N}$. The output is also $N \times d$:

$$\mathbf{Y} = \mathbf{A}\mathbf{X} \tag{*}$$

In constrast, with a fully connected layer, we treat the entire input sequence as a vector. In other words, we would consider input as $\mathbf{x} = \text{vec}(\mathbf{X}) \in \mathbb{R}^{dN}$, where $\text{vec}(\cdot)$ concatenates columns of a matrix into a long vector.

1. (10')) Express the output $\mathbf{Y}$ as obtained by a fully connected layer, *i.e.*, write Eq. (*) into

$$\text{vec}(\mathbf{Y}) = \mathbf{M}\text{vec}(\mathbf{X}).$$

   You may find Kronecker product useful.

2. (10') Discuss the form of $\mathbf{M}$. Can we replace a self-attention layer with a fully connected layer?

(Language Model) In this exercise, we play with different neural language models. We will follow this code repo, and obtain PPL's of different models on wikitext-2 test set.

When training and testing neural language models, we typically adopt the following data batching scheme:

- Concatenate all words into a long string. This ignores sentence, paragraph and chapter boundaries.
- Reshape the long string according to batch size. So for example, if the (concatenated) text string is "a b c ... z", using batch size 4, we will reshape it into

$$\begin{array}{c} \text{batch} \longrightarrow \\ \text{timesteps} \downarrow \end{array} \begin{bmatrix} a & g & m & s \\ b & h & n & t \\ c & i & o & u \\ d & j & p & v \\ e & k & q & w \\ f & l & r & x \end{bmatrix}.$$

- Feed segments of length "bptt" (back-propagation through time). In the above example, if using "bptt=2", we can create the following input-output tuple,

$$\left(\begin{bmatrix} a & g & m & s \\ b & h & n & t \end{bmatrix}, \begin{bmatrix} b & h & n & t \\ c & i & o & u \end{bmatrix} \right)$$

  as the first batch. And

$$\left(\begin{bmatrix} c & i & o & u \\ d & j & p & v \end{bmatrix}, \begin{bmatrix} d & j & p & v \\ e & k & q & w \end{bmatrix} \right)$$

  as second batch, and so on.

[1] Refer to Bishop Deep Learning book Excercise 12.6.

Of course, this batching scheme breaks the long sequence, resulting in some transitions (e.g., f to g) never learned. It also introduces a hyper-parameter "bptt", which upper bounds the context length we can model. However, the main advantage is its efficiency. To see that, consider another batching scheme that packs 4 sentences of different lengths into a batch. We would have to introduce padding tokens to make it a rectangular-shaped batch, and the computation at padding tokens are wasted.

1. (10') Run the code with its default configurations to obtain test PPL's for LSTM and transformer models. Include all intermediate results, e.g., training and validation losses.

2. (20') We now vary the "bptt" length, while keeping all the other configurations same as question 1. Plot test PPL's of LSTM and transformer for a range of "bptt" lengths, and discuss the results.

3. (30') In this question, we always set word embedding dimension equal to the hidden dimension, *i.e.*, use the same value for "emsize" and "nhid" when calling main.py. Let us keep the default "bptt", and vary "nhid" and "nlayers". Report test PPL's for all ("nhid", "nlayers") tuples. It is suggested to visualize[2] the PPL on 2D grids of "nhid"-"nlayers" coordindate, with a colorbar. Discuss the results.

4. (20') **(Length Generalization)**[3] Take the best 'nhid"-"nlayers" configuration you get in question 3. Apply the corresponding model to test data, but with a varying "bptt". This simulates a situation with length mismatch between training and testing. Discuss the results.

5. (60')**(Bonus)** LSTM models compress all past history into a hidden state, which can be reused for next batch of input. See the "repackage_hidden" function in main.py. Transformers however doesn't have this functionality, which could be one disadvantage. Can we design a hybrid architecture that combines LSTM and transformer? Implement your idea, and compare results with non-hybrid architectures.

[2]See this blogpost for example
[3]To learn more, read this paper.