DS-GA 1011 Fall 2019
Neural Conversation Modeling

In this assignment, you will work on conversation modeling. There are three parts in this work: modeling, decoding, interactive chat.

Important: Submit a single ipynb file. We provie you an example ipynb.

Important: you can use any materials and the code from the conversation modeling lab (11/6) to complete this assignment. Baseline model is a pretrained model from the lab (11/6).

Important: you can use any open-source code which helps you to complete this assignment. However, **you must cite this code appropriately (link is enough)**.

1 Modeling

1.1 Visualizing the attention

In this part you will analyze the existing attention mechanism in your baseline model **from the lab**. The baseline model performs an attention step over the whole input representation obtained from the encoder model.

Your task is:

1. Check the baseline model code and find how attention weights are logged. Make sure to understand how this type of attention works.
2. Implement a function which takes in the attention weights and plot the corresponding heat map.
3. Make sure that on the axes of the attention map you show the actual tokens from sequences.

What you present: select 5 examples (from validation set) and print: input, target, attention map. Be ready to explain how this attention is computed on every time step.

1.2 Transformer Encoder

The goal of this part is to plug-in a transformer encoder instead of RNN and make it work.

Important: you are not required to outperform the RNN-based baseline model. The idea is to understand the differences in those encoders.

Transformer encoder should have at least:

1. Positional embedding (please use one as it was presented in Masked Language Model lab).
2. Attention mask for self-attention.
3. Transformer encoder (stack of layers). To simplify your work, please use pytorch official transformer implementation.

In the end the output from Transformer Encoder should be similar to RNN encoder (the last time step hidden will not exist). Just assume no initial hidden in the Decoder RNN.

With 256 embedding and hidden size, 2 layers and 4 heads you could get approx. 36.6 validation PPL in 30 epochs.

What you present: training / validation curves for both loss and PPL. Be ready to explain what type of positional embedding you use and how it works. Be ready to explain why self-attention mask is essential for this kind of mini-batches.

2 Decoding strategies

In this part you will implement a stochastic decoding strategy and an additional heuristic for beam search.

Important: use the pretrained baseline model from lab for all tasks in this part.

2.1 Nucleus sampling

Reference to the original paper: `https://openreview.net/pdf?id=rygGQyrFvH`

Please read section 3.1 for details. Here we give the needed excerpt:

$$P'(x|x_{1:i-1}) = \begin{cases} P(x|x_{1:i-1})/p' & \text{if } x \in V^{(p_{\text{nucleus}})} \\ 0 & \text{otherwise} \end{cases}$$

$$p' = \sum_{x \in V^{(p_{\text{nucleus}})}} P(x|x_{1:i-1})$$

where $V^{(p_{\text{nucleus}})} \subset V$ is a top-p vocabulary which is defined as a smallest subset such that:

$$\sum_{x \in V^{(p_{\text{nucleus}})}} P(x|x_{1:i-1}) \geq p_{\text{nucleus}}$$

Your task is:

1. Implement nucleus sampling as a function (similar to greedy and beam from the lab).
2. Select a single input sample from the validation set and produce **100 decodings** for the following choices of nucleus parameter $p_{\text{nucleus}} = \{0.1, 0.5, 0.9\}$. In total there will be $100 * 3$ predictions. Compute the average log-probability of generated sequences for each value of p_{nucleus}. How it differs? Make a plot showing the difference.
3. For the same prediction from previous step compute how many unique tokens were produced for each value of p_{nucleus}. How it differs? Make a plot showing the difference.

What you present: plots you prepared for this task.

2.2 N-gram blocking for beam search

Reference to a relevant paper: `https://arxiv.org/pdf/1705.04304.pdf`

Here you will implement a so called n-gram blocking heuristic which aims to reduce amount of repeated ngrams in the prediction.

Important: for full credit you need to implement this as part of beam search. However, if you struggle with a beam, you can make it for a greedy search for partial credit.

Important: your implementation should be dependent on a particular order n.

This is an approximate route you can follow to implement it:

1. Make a function which takes an active hypothesis and computes all seen ngrams in this hypotheses.
2. Using all seen ngrams from previous step, compute which expanded tokens would produce a repeated ngram.
3. Penalize those tokens from being chosen in `torch.topk` function. For example, you can set its log-prob score close to negative infinity.

What you present: given a single example from validation set (print it as well), produce a prediction using a fixed beam size 20 and different orders of ngram blocking $n = \{1, 2, 3\}$. For each output, show 10 best candidates. Make sure that your candidates do not have repeating ngrams inside.

For partial credit you can use greedy search function instead of beam (as noted in the beginning).

3 Interactive chat

In this part you will implement a logic around your model and data loader such that it can handle a real chat interface with your model in Jupyter.

Important: you can use a fixed persona from validation set. For the decoding, use any of the strategies you have implemented in this homework.

Your task is:

1. Make a logic which iterates over the input to your model (given by you!) and output given by your model. In other words, it should properly concatenate all previous dialogue history with new user input and model's outputs.
2. Make a logging function which will dump the actual input you provide on each dialogue turn, such that you can confirm that the input you feed looks correct (notice there is new line token between each dialogue turn in the data)

What you present: You will start the interactive session and perform a dialogue with it (at least 3 turns). After that your code should be able to print all inputs which were used by your model.

To avoid timing issues, keep some already created dialogue in your ipynb so we could skip performing the dialogue part.