Text Generation with the Transformer Decoder

Background

After implementing the details about MLP and CNN in python with Numpy and PyTorch, we believe that you have already learned basic skills in deep learning. In this homework, we will implement a language model with the decoder of Transformer to generate sentences.

To train a language model, we minimize the cross-entropy loss conditioned on the ground-truth prefixes. To be specific, given a sentence $\{x_0, x_1, \cdots, x_T\}$, the loss should be formulated as follows:

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T} \log P(x_t|x_{<t}) \tag{1}$$

Usually, x_0 is a special token, which indicates the beginning of a sentence (i.e., the so-called `bos` token); x_T is a special token, which indicates the end of a sentence (i.e., the `eos` token). We regard `<|endoftext|>` as both the `bos` and `eos` tokens in this homework. We also use `<|endoftext|>` to fill the useless space for a batch of sentences, where the short sentences are padded to match the longest one.

The language model should be trained in the Teacher-Forcing mode (See the slides of RNN). We first convert the input sentence $X = \{x_0, x_1, \cdots, x_{T-1}\}$ to the embedding sequence $E = \{e_0, e_1, \cdots, e_{T-1}\}$. Then, the Tranformer decoder tranforms the embedding sequence E into hidden states $H = \{h_0, h_1, \ldots, h_{T-1}\}$. h_t is used to predict the next token, i.e. x_{t+1}. To be formal,

$$P(x_{t+1}|x_{<t+1}) = \mathrm{Softmax}(\mathrm{Linear}(h_t)) \tag{2}$$

At last, we can calculate $\mathcal{L}$ following Equation (1) and optimize it with Adam optimizer.

In this homework, you should finish all **TODO** parts in codes, which includes several parts:

- The base architecture in `model_tfmr.py`, including the following several parts:
  - The mask matrix for attention weights (i.e., the `_attn` function), which restricts the model to attend on only the prefix tokens.
  - Functions for splitting and merging attention heads (i.e., the `_split_heads` function and `_merge_heads` function). If interested, we encourage you to think about the superiority of multi-head attention w.r.t. single-head attention when conducting experiments.
  - Connections of different modules in the Transformer Block (i.e., the `forward` function of the `TfmrBlock` class). Note:
    - Different from the original Transformer paper[5], we do not include an encoder in this homework.
    - Different from the original Transformer paper[5], we implement Pre-Norm which is widely used in GPT-style models. Please refer to Page 39 in lecture 8 for more details.
  - Positional encoding in the `forward` function of the `TfmrModel` class. Note that you should consider the length of the cache hidden states (i.e., `past_key_values`), which are used during inference.

  Note: You are not allowed to use the pre-defined modules in `torch.nn`, `torch.nn.functional` or any APIs provided by `HuggingFace`, etc. for implementing the above modules (except `torch.nn.Softmax` or `torch.nn.functional.softmax` for computing attention weights).

- Training Loss in `model_tfmr.py` and the perplexity in `main.py`: You should calculate the loss for the language model (Equation 1), which is exactly the corresponding perplexity after the `exp` operation.
  Note: The lengths of a batch of sentences may be different, where we add paddings for short sentences to match the longest one. However, you must **NOT** calculate losses for paddings. You should calculate **the average loss of each token in a mini-batch** in the following procedure: you first calculate the loss of each sequence in the batch by averaging on the tokens of each sequence, and then calculate the loss of the batch by averaging on all the sequences in the batch.
- Top-p decoding in `model_tfmr.py`: You should implement the top-p decoding strategy (Nucleus Sampling [1]). The decoding strategy is used in Free-Run mode for inference, where the model should read its generated prefix and predict the next token. Specifically, we first take a `<|endoftext|>` token as the first step of Transformer's input. Then, the next token distribution can be obtained by decoding strategies. For the **random decoding strategy** with temperature τ, we randomly take a sample from the following distribution:

  $$P^*(x_t|x_{<t}) = \text{Softmax}(\text{Linear}(h_{t-1})/\tau) \tag{3}$$

  Then x_t can be used as the next step's input. We repeat the process until an <eos> token is generated.

  For **top-p decoding strategy** with temperature τ, the next token distribution should be formulated as follows:

  $$P^{**}(x_t|x_{<t}) = \begin{cases} P^*(x_t|x_{<t}) / \sum_{x\in\mathcal{D}_t}(P^*(x|x_{<t})) & x_t \in \mathcal{D}_t \\ 0 & x_t \notin \mathcal{D}_t \end{cases} \tag{5}$$

  where P^* is from Equation (3); For **top-p decoding**, $\mathcal{D}_t$ is the minimal set satisfying $\sum_{x\in\mathcal{D}_t} P^*(x_t|x_{<t}) \geq p$. That is, in each decoding step, you should randomly choose tokens from **the top words** whose sum of probability is larger than p. See [1] and our RNN slides for details.

We encourage you to rewrite part of the codes in `main.py` to use TensorBoard to visualize your experimental results.

Requirements

- python >=3.6
- PyTorch >= 1.1
- nltk == 3.5 (Higher versions may lead to throwing errors during evaluation)

Dataset Description

Microsoft COCO (MSCOCO) dataset is an image caption dataset, where we only extract the sentences and ignore images for language generation. We extract 25,000 sentences and split them into the training / validation / test parts, containing 15,000 / 5,000 / 5,000 sentences respectively.

Don't upload this data file in your homework submission!

Metric Description

We use 4 metrics to evaluate your result:

- Perplexity: $\mathrm{PPL} = \exp(-\frac{1}{T}\sum_{t=1}^{T}\log P(x_t|x_{<t}))$. Lower perplexity indicates better performance (you should implement this metric by yourself).
- Forward BLEU, Backward BLEU, Harmonic BLEU: BLEU is a metric first used for machine translation, which evaluates the model by n-gram precision [2]. We use three variants of BLEU for our task [3]: Forward BLEU measures fluency, Backward BLEU measures diversity, and Harmonic BLEU is their harmonic average indicating overall performance. A larger BLEU score indicates better performance.

The evaluation can be very slow, so we do not evaluate the models on the BLEU metrics when training. You should choose the checkpoint with the **lowest perplexity** on the validation set, and evaluate it by these four metrics.

Python Files Description

- `main.py` contains the main script to run and evaluate the whole program.
- `model_tfmr.py` contains the script for model implementation.
- `configuration.py` defines the hyper-parameters for the model.
- `config.json` specify the hyper-parameters for the model.
- `tokenizer.py` defines the BPE tokenizer of GPT2 [4]. The vocabulary is saved under the `tokenizer` directory.
- `data` contains `train.txt`, `dev.txt` and `test.txt`, which are the training set, development set (validation set) and test set, respectively.
- `pretrained checkpoint`: `checkpoints/pretrained_ckpt.bin` and `checkpoints/config.json` is the checkpoint and config of the pre-trained 3-layer Transformer. You can download them from THUCloud.

Command

- `Train from scratch`: `python main.py --name NAME`. Run the model with the experiment name (default: run).
- `Fine-tune a pre-trained model`: `` `python main.py --name NAME --prertain_dir `` `PRETRAIN_DIR`. Fine-tune a pre-trained model. `PRETRAIN_DIR` is a directory containing `pretrained_ckpt.bin` and `config.json`.

- `Test`: `python main.py --test NAME`. Load the best model for the experiment NAME, and evaluate it with the four metrics.

See `main.py` for more arguments.

NOTE: After you run the test command, you'll get the result and a file named `output.txt` with generated sentences. **You should choose the best experiment and submit this file as your final result**.

Report

You should conduct the following experiments in this homework:

1. Train two models, where **one is trained from scratch** (i.e., parameters are randomly initialized) and the **other one is trained with parameters initialized from the pre-trained checkpoint** (constructed from GPT-2 base). We call them **Tfmr-scratch** and **Tfmr-finetune**, respectively.
    - Plot the loss value of Transformer (using the default hyper-parameters) against every epoch until converging during training (on both training parts and validation parts) for them.
    - Choose the best model on the development set and report the test results on 4 metrics (Perplexity, Forward BLEU, Backward BLEU, Harmonic BLEU).
    - Compare and analyze the results of two models.

    (**Note:** In order to save time, we set the default number of Transformer blocks to 3, which is smaller than the default number of Transformer blocks in GPT2-base, i.e., 12. Therefore, we only take the **first 3 layers** of GPT2-base to construct the pre-trained checkpoint used in this homework)
2. Compare the generation results on the test set of **Tfmr-scratch** and **Tfmr-finetune** combined with various decoding strategy (**random**, **top-p** and **different temperature** τ). Try at least 4 combinations:

    (random, $\tau = 1$), (random, $\tau = 0.7$), (top-p=0.9, $\tau = 1$), (top-p=0.9, $\tau = 0.7$) on both trained models, **which means you need to run the evaluation for as least 8 times** and report the corresponding results.
3. Randomly choose 10 sentences for each strategy for both Tfmr-scratch and Tfmr-finetune. Answer the questions:
    1. Are there any grammar errors? List typical errors in your report.
    2. Which strategy/model generate the best sentences?
    3. Are the 4 metrics (Perplexity, Forward BLEU, Backward BLEU, Harmonic BLEU) consistent with your judgment?
4. Describe your final network with the hyper-parameters (if you change them) and decoding strategies. Report the result on 4 metrics, and submit the corresponding generation result with your report. **You should rename the generation result to `output.txt`, which contains 5000 sentences.**
5. By carefully reading the source code of Transformer decoder, answer the following questions **briefly** :
    1. Compare Transformer and RNN from at least two perspectives such as time/space complexity, performance, positional encoding, etc.
    2. Regarding the inference time complexity, answer the following question.
        1. During inference, we usually set `use_cache` in `model_tfmr.py` to `True`. What is the argument used for?

2. Denote the whole sequence as $L = (l_0 = <|\text{endoftext}|>, l_1, l_2, \cdots, l_T)$, please give the inference **time complexity** when decoding the token l_t ,i.e., the t-th loop in the `inference` function of `model_tfmr.py` when decoding the first example, and the whole time complexity for decoding the whole sequence L. We denote the hidden state dimension as d (so that the dimension of the intermediate state of the feed forward layer is $4d$), the number of heads in multi-head attention as n, the number of hidden Transformer blocks as B, the vocab size as V.
3. Based on your analysis of the question No 2., in which case the self-attention module dominate the time complexity? And in which case the feed-forward layer is dominant?

3. Discuss the influence of pre-training regarding the generation results, convergence speed, etc. Considering the experimental setup (the training task, data, pre-trained checkpoints, etc.), does the influence of pre-training meet your expectation?

Bonus (≤ 2):

- Try to use the BPE tokenizer to tokenize several sentences. Discuss the superiority of the BPE tokenizer compared with splitting by space.
- You can take different 3 layers from the original 12 layers in GPT2-base, and observe the performance. For example, taking the last 3 layers or the 1-st, 6-th and 12-th layers instead of simply the first 3 layers may lead to different generation results.
- Discuss the effect of the number of heads used in multi-head attention, etc. You can refer to `config.json` to obtain inspiration.

Note that you should conduct elaborate experiments and carefully analyze the results to get the bonus.

NOTE: To save your time, the default hyper-parameters should provide a reasonable result. However, you can still tune them if you want to do more explorations.

NOTE: When using the default hyper-parameters, it will cost about 5G GPU memory and 5 minutes per epoch. If you use CPU for training, it will cost 15 minutes per epoch. The validation ppl should be less than **60** after the first epoch when training from scratch. Accordingly, You can stop training in time and correct your code if the validation ppl is larger than 60 after the first epoch.

NOTE: TensorBoard may be helpful when you plot figures in your experiment.

NOTE: The perplexity on the validation set should be **lower than 28 for Tfmr-scratch** and **lower than 22 for Tfmr-finetune**.

NOTE: If you are out of memory when testing your model, see the comments in the `evaluate` function of `main.py`. Setting `cpu_count=0` in `FwBwBleuCorpusMetric` will solve the problem but become much slower.

NOTE: You are **NOT** allowed to use other advance neural network packages, e.g., `fairseq`, `texar`, `Transformers`.

Code Checking

We introduce a code checking tool this year to avoid plagiarism. You **MUST** submit a file named `summary.txt` along with your code, which contains what you modified and referred to. You should follow the instructions below to generate the file:

1. Fill the blanks in the codes. Notice that you should only modify the codes between `# TODO START` and `# TODO END`, and the other changes should be explained in `README.txt`. **DO NOT** change or remove the lines starting with `# TODO`.
2. Add references if you use or refer to online codes, or discuss with your classmates. You should add a comment line just after `# TODO START` in the following formats:
    1. If you use a code online: `# Reference: https://github.com/xxxxx`
    2. If you discuss with your classmates: `# Reference: Name: Xiao Ming Student ID: 2018xxxxxx`

    You can add multiple references if needed.

    Warning: You should not copy codes from your classmates, or copy codes directly from the Internet, especially for some codes provided by students who did this homework. In all circumstances, you should at least write more than 70% codes. (You should not provide your codes to others or upload them to Github before the course ends.)

    警告：作业中不允许复制同学或者网上的代码，特别是往年学生上传的答案。我们每年会略微的修改作业要求，往年的答案极有可能存在错误。一经发现，按照学术不端处理（根据情况报告辅导员或学校）。在任何情况下，你至少应该自己编写70%的代码。在课程结束前，不要将你的代码发送给其他人或者上传到 github 上。
3. Here is an example of your submitted code:

    ```
    def forward(self, input):
      # TODO START
      # Reference: https://github.com/xxxxx
      # Reference: Name: Xiao Ming Student ID: 2022xxxxxx
      your codes...
      # TODO END
    ```

4. At last, run `python ./code_analyze/analyze.py`, the result will be generated at `./code_analyze/summary.txt`. Open it and check if it is reasonable. A possible code checking result can be:

    ```
    ########################
    # Filled Code
    ########################
    # ..\codes\layers.py:1
        # Reference: https://github.com/xxxxx
        # Reference: Name: Xiao Ming Student ID: 2018xxxxxx
        your codes...

    ########################
    # References
    ```

```
########################
# https://github.com/xxxxx
# Name: Xiao Ming Student ID: 2018xxxxxx

########################
# Other Modifications
########################
# _codes\layers.py -> ..\codes\layers.py
# 8 -         self._saved_tensor = None
# 8 +         self._saved_tensor = None # add some thing
```

Submission Guideline

You need to submit a report document, codes, the model output, and the code checking result, as required as follows:

- **Report:** well-formatted and readable summary to describe the results, discussions, and your analysis. Source codes should *not* be included in the report. Only some essential lines of codes are permitted. The format of a good report can be referred to as a top-conference paper.
- **Codes:** organized source code files of your final network with README for extra modifications or specific usage. Ensure that TAs can easily reproduce your results following your instructions. **DO NOT include model weights/raw data/compiled objects/unrelated stuff over 50MB**.
- **Model Output (Important):** Submit the generation result which contains 5,000 sentences generated by your model.
- **Code Checking Result**: You should only submit the generated `summary.txt`. **DO NOT** upload any codes under `code_analysis`. However, TAs will regenerate the code checking result to ensure the correctness of the file.

You should submit a `.zip` file named after your student number, organized as below:

- `Report.pdf/docx`
- `summary.txt`
- `codes/`
  - `*.py`
  - `output.txt`
  - `README.md/txt`

Deadline

November 12th

TA contact: 顾煜贤, guyx21@mails.tsinghua.edu.cn

Reference

[1] Holtzman, Ari, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. "The curious case of neural text degeneration." ICLR2020.

[2] Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002, July). BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics* (pp. 311-318).

[3] Shi, Z., Chen, X., Qiu, X., & Huang, X. (2018). Toward diverse text generation with inverse reinforcement learning. *arXiv preprint arXiv:1804.11258*.

[4] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. Improving Language Understanding by Generative Pre-Training.

[5] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.