# Text Generation with RNN

## 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 RNN 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 <go>, which indicates the start of a sentence; $x_T$ is a special token <eos>, which indicates the end of a sentence. There is also two other special tokens: <unk> means the token is not in our vocabulary; <pad> is used for filling 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 $E = \{e_0, e_1, \cdots, e_{T-1}\}$ (by looking up the pretrained word vector). Then, the RNN encode 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 specific,

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

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

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

- Basic RNN Cells in `rnn_cell.py`: You should implement `GRUCell` and `LSTMCell` following our implementation of `RNNCell`. **Note:** You are not allowed to use the predefined RNNs in `torch.nn` or `torch.nn.functional` for this part, e.g. `torch.nn.LSTM`, `torch.nn.GRU`, `torch.nn.LSTMCell`, `torch.nn.GRUCell`.
- Training Loss in `model.py`: You should calculate the loss for the language model. **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 should **NOT** calculate losses for paddings. You should calculate **the average loss of each token in a mini-batch**, which means that you should divide the sum of cross-entropy loss for the samples in a mini-batch by the length of all valid tokens.
- Top-p decoding in `model.py`: You should implement the top-p decoding strategy (Nucleus Sampling[1]) following our random decoding strategy. The decoding strategy is used in Free-Run mode for inference, where the model should read its generated prefix and predict the next token. To be specific, we first take a <go> token as the first step of RNN's input. Then, the next token distribution can be obtained by decoding strategies. For the **random decoding strategy** with temperature $\tau$, we randomly take a sample from the following distribution:

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

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 $\tau$, the next token distribution should be formulated as follows:

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

where $P^*$ is from Equation (3); $\mathcal{D}_t$ 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
- cotk == 0.1.0

  cotk is a package to load the dataset and provide metrics for language generation. You can install it by `pip install cotk==0.1.0`. You can see homepage at `https://github.com/thu-coai/cotk` and the document at `https://thu-coai.github.io/cotk_docs/`

## 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. We also provide pre-trained word embeddings in `./wordvec/vector.txt`.

## Metric Description

We provide 4 metrics to evaluate your result:

- Perplexity: $PPL = exp(-\frac{1}{T}\sum_{t=1}^{T}\log P(x_t|x_{<t}))$. Lower perplexity indicates better performance.
- 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. 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 lowest perplexity on validation set, and evaluate it by these four metrics.

## Python Files Description

- `main.py` contains the main script to run the whole program.
- `model.py` contains the script for model implementation.
- `rnn_cell.py` contains various basic RNN cells.

## Command

- `Train`: `python main.py --name NAME`. Run model with the experiment name (default: run).
- `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 an 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. Plot the loss value of one-layer RNN with 3 kinds of rnncells (i.e., RNNCell, GRUCell, LSTMCell) against every epoch during training (on both training parts and validation parts). Report the test results on 4 metrics (Perplexity, Forward BLEU, Backward BLEU, Harmonic BLEU). Compare and analyze the performance of 3 kinds of rnncells.
2. Choose the best model and try different decoding strategies in inference (at least including, random sampling with temperature=1, random sampling with temperature=0.8, top-p sampling with p=0.8, top-p sampling with p=0.8 and temperature=0.8). Report the test results on 4 metrics. How the decoding strategies influence the generated results?
3. Read the sentences generated by the different decoding strategies above. Randomly choose 10 sentences for each strategies and list them in your report. Is there any grammar errors? Which strategies generate the best sentences? Discuss whether the 4 metrics (Perplexity, Forward BLEU, Backward BLEU, Harmonic BLEU) are consistent with your judgement?
4. Describe your final network with the hyperparameters (if you change them) and decoding strategies. Report the result on 4 metrics, and submit `output.txt` with your report.

**Bonus** ($\leq 2$):

- (0.5 point bonus) Use the basic RNN cells to construct a multi-layer RNN. Plot the loss value of two-layer RNN against every epoch during training (on both training parts and validation parts). Report the test result on 4 metrics. Compare and analyze the performance of one-layer RNN and two-layer RNN.
- Other explorations: For example, discuss the effect of pretrained word vector, train a character-level language model. You should conduct elaborate experiments and carefully analyze the results to get the bonus.

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

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

**NOTE:** The perplexity of 3 cells on validation set should be lower than **25**, or some penalty will be imposed on your score.

**Note:** When implementing basic RNN cells, you are **NOT** allowed to the predefined RNNs in `torch.nn` or `torch.nn.functional`, e.g. `torch.nn.LSTM`, `torch.nn.GRU`, `torch.nn.LSTMCell`, `torch.nn.GRUCell`.

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

## 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 codes. Notice you should only modify the codes between `# TODO START` and `# TODO END`, the other changes should be explained in `README.txt`. **DO NOT** change or remove the lines start with `# TODO`.
2. Add references if you use or refer to a online code, 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 ended.)

    **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: 2018xxxxxx
        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 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 `output.txt` 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 1st**

**TA contact:** 黄斐, huangfei382@163.com

## 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*.