Deep Learning for Natural Language Processing
Assignment 4

The Hong Kong University of Science and Technology, Spring 2021

Deadline: Thu 23:59, 15th Apr 2021

1 Introduction

The goal of this assignment is to perform Language Modeling based on Recurrent Neural Network model. You first need to load and clean and analyze the data, i.e., extract the words and their corresponding embedding. Furthermore, implement many-to-one word prediction model based on a Recurrent Neural Network (RNN) model using PyTorch.

For this assignment, we are going to use a The Microsoft Research Sentence Completion Challenge Dataset (https://github.com/HLTCHKUST/DLNLP-HW4-Language-Modeling) from Sherlock Holmes novels. We already provide a subsets of the original challenge, however if your machine is not powerful enough you can further reduce the number of file. In case you do so, please report the files name used for the training.

2 Data

Please include the following subsections in your report, and implement your solution using **Dataset** and **DataLoader** of the pytorch library. Submit these as single file **dataloader.py**.

2.1 Preprocessing

If you look at the statistics from above, you can see that the original data is quite noisy. Just as you should have done in Assignment 1, try to clean the data as much as possible, while minimizing loss of information. Briefly list all the methods you used to clean the data in the report, and submit your code as **preprocess.py**.

2.2 Train-Valid Split

We normally split the training dataset into Training and Validation, as you would have learned in previous assignments. Do not forget to handle this during this homework as well. A convenient way to do this in a stratified manner is to use **sklearn** library.

2.3 Dataloader output

Since in the next section is required to implement a Many-to-One model for Language Modeling, and the Dataset class should implement preprocessing accordingly. For instance, if we have a sentence "I love black and blue cats":

- Many–to–One:

```
X data = [[SOS, I],
          [SOS, I, love],
          [SOS, I, love, black],
          [SOS, I, love, black, and],
          [I, love, black, and, blue]]
```

```
y_data = [[love],
          [black],
          [and],
          [blue],
          [cats]]
```

As you can notice, you need to include padding token (i.e. "PAD") to make all the input to the same size. Notice that the corpus is not divided in sentences, but is plain text. It is completely up to you to choose how to split (e.g. in sentences or with a window of n tokens). Many ways are valid, please justify your design choice.

3 Implement RNN with PyTorch

Implement Recurrent Neural Networks.

```
class RNN_M2O(torch.nn.Module):
    def __init__(self):
        super(RNN_M2O, self).__init__()
        emb = nn.Embedding(vocab_size , emb_size)
        # TODO
    def forward(self, inputs):
        # TODO
        return out
```

You should use nn.Embedding (Embedding Lookup), as in the previous assignments. Implement also the loss function (e.g. CrossEntropyLoss) and the optimizer (e.g. Adam). Submit this task as **RNN.py**.

4 Results and Analysis

Create a training function and a test function and submit a single file main.py. Language modeling task is evaluated using the perplexity score. Implement such score for both model and implement an early stopping criteria using such score in the validation set. (Hint perplexity is simply e^{loss}, where loss is the average loss value).

4.1 Hyper-parameters

Tune 3 combinations of the following parameters in the RNN model. You are free to implement any regularization technique (e.g. Dropout or L2 reg.).

- Learning Rate: [0.01, 0.001, 0.0001]
- Hidden Dimension Size: [128, 256, 512]
- Number of Hidden Layers: [1, 2]

Report the validation perplexity and discuss the results achieved. Test the best Hyper-parameters for the RNN model and discuss which hyper-parameter combination achieve the lowest perplexity in the test set.

5 Bonus

Transfer Learning with Pre-trained Word Embeddings: Try to initialize nn.Embeddings with pre-trained word2vec, GloVe, FastText. Report and discuss whether this helps in this task.

Language Generation: RNN language model can be used as a text generator. This can be implemented by using the predicted word at time t as RNN input at time t + 1. Thus starting from the SOS we can generate text which will result similar to the corpus. Try to implement such setting and generate a short paragraph.

6 Submission

Turn this in by **Thu 23:59, 15th Apr 2021** by emailing your solution and code to hkust.hltc.courses@gmail.com, with subject line "Deep Learning for NLP 4". Remember to state your name, student ID in the email. The zip should only include the following materials:

- Short report of the assignment in pdf format (maximum 3 pages).
- A zipped folder which contains: **dataloader.py**, **statistics.py**, **preprocess.py**, **RNN.py**, **main.py**, and any other files you have used in your experiments. Document it well with comments.