Homework 4 Part 1 (Summer Version)

Language Modeling using RNNs

11-785: Introduction to Deep Learning (Fall 2019)

OUT: **June, 2019**
DUE: **September, 2019**

Start Here

- **Collaboration policy:**
  - You are expected to comply with the University Policy on Academic Integrity and Plagiarism.
  - You are allowed to talk with / work with other students on homework assignments
  - You can share ideas but not code, you must submit your own code. All submitted code will be compared against all code submitted this semester and in previous semesters using MOSS.
- **Overview:**
  - **Part 1**: All of the problems in Part 1 will be tested by yourself. You can download the starter code from Course Page.

1 Introduction

In part 1 of homework 4, we will be training a recurrent neural network on the WikiText-2 language modeling dataset. You will practice using a recurrent network to model and generate texts.
The below sections will describe the dataset and what your model is expected to do. You are responsible for the organization of your codes.
Here are the papers and blogs we recommend you to read:
1. Regularizing and Optimizing LSTM Language Models for information on how to properly construct, train, and regularize an LSTM language model.
2. CS224n: Natural Language Processing with Deep Learning for the basic ideas of language model and Recurrent Neural Networks.
3. 11747: Nerual Networks for NLP for the references for RNNs and language model.

Our tests are extremely easy and you only have to make the negative log-likelihood decrease. These tests require that you train the model by yourself, generating the predictions and plotting the loss curves. Details follow below.

1.1 Files in the Handout

The template provided to you is in the form of a Jupyter notebook. There are TODO sections in the notebook that you need to complete. The Classes provided for training your model are provided to help you organize your training. Ideally, you wouldn't need to change the rest of the notebook, as these parts integrate your blocks of code to run the training, save models/predictions and also generate plots. However, if you do choose to (maybe to implement early stopping for example), be careful. Every time you run training, the notebook creates a new experiment folder under `experiments/` with a `run_id` (which is CPU clock time for uniqueness). All your model weights, predictions will be saved here. The notebook trains the model, prints

the NLL on the dev set and creates the generation and prediction files on the test dataset, per epoch.
Your solutions will be tested locally by your self.
`make runid=<your run id> epoch=<epoch number>`
You can find the run ID in your Jupyter notebook (its just the CPU time when you ran your experiment). You can choose the best epoch using `epoch_number`.

2 Dataset

A pre-processed WikiText-2 dataset is included in the template tarball.

- `vocab.npy`: a NumPy file containing the words in the vocabulary
- `vocab.csv`: a human-readable CSV file listing the vocabulary
- `wiki.train.npy`: a NumPy file containing training text
- `wiki.valid.npy`: a NumPy file containing validation text

The vocabulary file contains an array of strings. Each string is a word in the vocabulary. There are 33,278 vocabulary items. The train and validation file contain an array of articles. Each article is an array of integers, corresponding to words in the vocabulary. There are 579 articles in the training set. For example, the first article in the training set contains 3803 integers. The first 6 integers of the first article are `[1420 13859 3714 7036 1420 1417]`. Looking up these integers in the vocabulary reveals the first line: `= Valkyria Chronicles III = <eol>`.

2.1 DataLoader

To make the most out of our data, we need to make sure the sequences we feed into the model are different every epoch. An easy way to do this is by using Pytorch's `DataLoader` class but overwriting the `__iter__` method. What this method should do is to:

1. Randomly shuffle all the articles from the WikiText-2 dataset.
2. Concatenate all text in one long string.
3. Run a loop that returns a tuple of `(input, label)` on every iteration with `yield`. (look at iterators in python if this sounds unfamiliar)

3 Training

You are free to structure the training and engineering of your model as you see fit. Follow the protocols in the paper as closely as you are able in order to guarantee maximal performance. Refer to the paper for additional details and please ask for clarification on Piazza. Your model will likely take around 3-6 epochs, to achieve a validation NLL below 5. Performance reported in the paper is 4.18, so you have room for error. Data is provided as a collection of articles. You may concatenate those articles to perform batching as described in the paper. It is advised to shuffle articles between epochs if you take this approach.

3.1 Language Model

In traditional problem of language modelling, a trained language model would learn the likelihood of occurrence of a word based on the previous words. Therefore, the input of your model is the previous texts. Of course, language models could be operated at different levels, such as character level, n-gram level, sentence

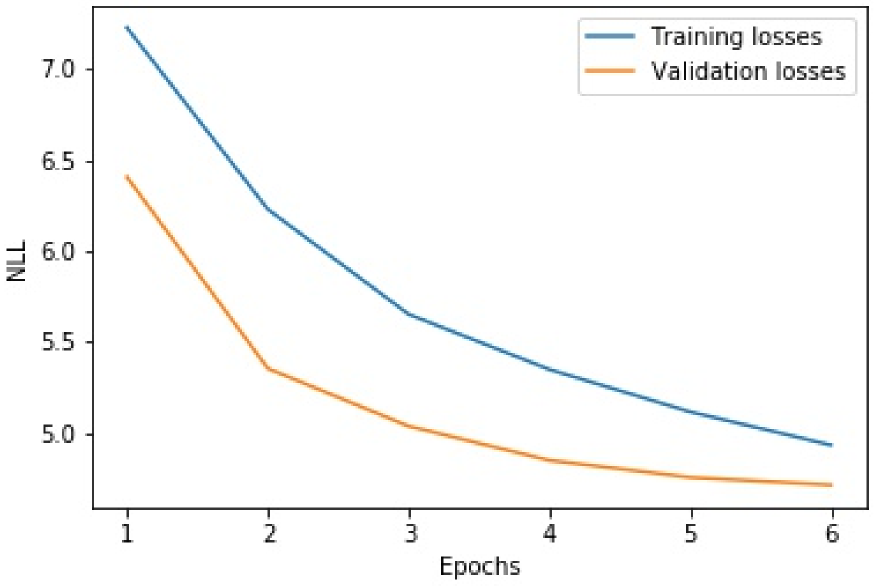

Figure 1: A successful training plot sample

level and so on. In the part 1, the character level is recommended. And it would be better to choose to use the fixed length input. You do not have to use packed sequence as the input.

4 Problems

4.1 Prediction of a Single Word (50 points)

Complete the function `prediction` in class `TestLanguageModel` in the notebook. This function takes as input a batch of sequences, shaped `[batch size, sequence length]`. This function should use your trained model and perform a forward pass. Return the scores for the next word after the provided sequence for each sequence. The returned array should be `[batch size, vocabulary size]` (float). These input sequences will be drawn from the unseen test data. Your model will be evaluated based on the score it assigns to the actual next word in the test data. Note that scores should be raw linear output values. Do not apply softmax activation to the scores you return.

4.2 Generation of a Sequence (50 points)

Complete the function `generation` in the class `TestLanguageModel` in the notebook. As before, this function takes as input a batch of sequences, shaped `[batch size, sequence length]`. Instead of only scoring the next word, this function should generate an entire sequence of words. The length of the sequence you should generate is provided in the forward parameter. The returned shape should be `[batch size, forward]` (int). This function requires sampling the output at one time-step and using that as the input at the next time-step. In the future, you will learn more about these details. If your outputs make sense, they will have a reasonable NLL. If your outputs do not reasonably follow the given outputs, the NLL will be poor.

5 Testing

In the handout you will find a template Jupyter notebook that also contains tests that you run locally on the dev set to see how your network is performing as you train. In other words, the template contains a test that will run your model and print the generated text. Good luck! May the global optimum be with you!