Homework 4 Part 1
Language Modeling using RNNs

11-785: INTRODUCTION TO DEEP LEARNING (SPRING 2020)

OUT: **April 5, 2020**
DUE: **May 3, 2020, 11:59 PM ET**

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 graded on Autolab. Download the starter code from Autolab.

1 Introduction

In Homework 4 Part 1, we will be training a Recurrent Neural Network on the WikiText-2 Language Modeling Dataset.

You will learn how to use a Recurrent Network to model and generate text. You will also learn about the various techniques we use to regularize recurrent networks and improve their performance.

The below sections will describe the dataset and what your model is expected to do. You will be responsible for organizing your training as you see fit.

Please refer to Regularizing and Optimizing LSTM Language Models for information on how to properly construct, train, and regularize an LSTM language model. You are not expected to implement every method in that paper. Our tests are not overly strict, so you can work your way to a performance that is sufficient to pass Autolab using only a subset of the methods specified in the aforementioned paper.

These "tests" require that you train the model on your own and submit a tarball containing the code capable of running the model, generating the predictions and plotting the loss curves. Details follow below.

1.1 Autograder Submission and Fun Facts

`hw4/training.ipynb` is a Jupyter Notebook of the template provided to you.

Within the Jupyter Notebook, there are `TODO` sections that you need to complete.

The classes provided for training your model are given to help you organize your training code. You shouldn't need to change the rest of the notebook, as these classes should run the training, save models/predictions and also generate plots. If you do choose to diverge from our given code (maybe 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 of your model weights and predictions will be saved there.

The notebook trains the model, prints the Negative Log Likelihood (NLL) on the dev set and creates the generation and prediction files on the test dataset, per epoch.

Your solutions will be autograded by Autolab. In order to submit your solution, create a tar file containing your code. The root of the tar file should have a directory named hw4 containing your module code. You can use the following command from the handout directory to generate the required submission tar.

```
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 Language Modeling Dataset is included in the template tarball and includes:

- `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 files each 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.

As an 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 to be: `= 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. You should use Pytorch's `DataLoader` class but overwrite the `__iter__` method.

The `__iter__` method should:

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 to, in order to guarantee maximal performance.

The following regularization techniques will be sufficient to achieve performance to pass on Autolab. Refer to the paper for additional details and please ask for clarification on Piazza. It may not be necessary to utilize all of the below techniques.

- Apply locked dropout between LSTM layers
- Apply embedding dropout
- Apply weight decay
- Tie the weights of the embedding and the output layer
- Activity regularization
- Temporal activity regularization

Your model will likely take around 3-6 epochs, to achieve a validation NLL below 5.0. The Autolab tests require a performance of 5.4. 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 language modelling, a trained language model will learn the likelihood of the occurrence of a word based on the previous words. Therefore, the input of your model is the previous text.

Of course, language models can be operated on at different levels, such as character level, n-gram level, sentence level and so on. In Homework 4 Part 1, we recommend using the word level representation. You may try to use character level if you wish. Additionally, it would be better to use a "fixed length" input. (This "fixed length" input is not the most efficient way to use the data, you could try the method in the paper: "4.1. Variable length backpropagation sequences"). Lastly, you do not have to use packed sequence as the input.

$$P(w_1, ..., w_n) = P(w_1) \prod_{i=2}^{i=m} P(w_i | w_{i-1}, ..., w_1)$$

4 Problems

4.1 Prediction of a Single Word (50 points)

Complete the function `prediction` in class `TestLanguageModel` in the Jupyter 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. Required performance is a negative log likelihood of 5.4. The value reported by autolab for the test dataset should be similar to the validation NLL you calculate on the validation dataset. If these values differ greatly, you likely have a bug in your code.

4.2 Generation of a Sequence (50 points)

Complete the function `generation` in the class `TestLanguageModel` in the Jupyter 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. Please refer to Recitation 7 for additional details on how to perform this operation. Your predicted sequences will be passed through tests that we have curated, and the NLL of your outputs will be calculated.

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. Required performance is a negative log likelihood of 3.0.

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. If that generated text seems like English, then the test on Autolab will likely pass. Good luck!