Assignment 3: RNN for NLP (due April 1, 2020, 11.59 pm EST)

The goal of this assignment is to learn how to implement and use recursive neural network for classification and for language modeling using Pytorch. Pytorch is one of the most popular deep learning frameworks in both industry and academia, and learning its use will be invaluable should you choose a career in deep learning. You will be using Pytorch for this assignment, and instead of providing you source code, we ask you to build off a couple Pytorch tutorials.

The goals of this assignment are:

1. To learn to do classification with recursive neural network.
2. To learn to use the recursive neural network as a language model to generate text.

The materials provided in this zip file are:

```
|-- data
|  |-- cities_train/
|  |  |-- af.txt
|  |  |-- ...
|  |  |-- za.txt
|  |-- cities_val/
|  |  |-- af.txt
|  |  |-- ...
|  |  |-- za.txt
|-- cs505_hw3.pdf
```

Deliverables

Here are the deliverables that you will need to submit in a zip file. Please be sure to follow this structure. Also, make sure that you do not create additional structures(for example, by putting the contents below in a top level folder, and zipping the folder itself).

```
|-- code
|   |-- ...
|   |-- ...
|   |-- README.txt
|   |-- data/
|   |   |-- ...
|   |   |-- ...
|-- writeup.pdf
```

- The code must be in Python3, as mentioned in the beginning of the course.
- The `README.txt` must explain how to run your code. **The grader must be able to run your code for your submission to be graded.**
- The `data/` directory will contain any data that your code uses(whether we provide it, or whether you find it yourself). This is important to make sure any grading script we write has no trouble running your code.

1 Classification Using Character-Level Recurrent Neural Networks (40 points)

Read through the tutorial here that builds a char-RNN that is used to classify baby names by their country of origin. Make sure you can reproduce the tutorial's results on the tutorial's provided baby-name dataset before moving on.

Modify the tutorial code to instead read from the City Names dataset. The tutorial code problematically used the same text file for both training and evaluation, which is not a great idea. For the city names dataset, we provide you separate train and validation sets.

All training should be done on the train set and all evaluation (including confusion matrices and accuracy reports) on the validation set. You will need to change the data processing code to get this working. Specifically, you'll need to modify the code in the 3rd code block to create two variables `category_lines_train` and `category_lines_val`. In addition, to handle Unicode, you might need to replace calls to open with calls to `open(filename, "r", encoding="utf-8")`.

NOTE: you'll want to lower the learning rating to 0.002 or less or you might get NaNs when training.

Attribution: the city names dataset is derived from Maxmind's dataset.

Complete the following analysis on the city names dataset, and include your finding in the report.

1. (**10 points**) Write code to output accuracy on the validation set. Include your best validation accuracy in the report. Discuss where your model is making mistakes and use a confusion matrix plot to support your answer.
2. (**10 points**) Modify the training loop to periodically compute the loss on the validation set, and create a plot with the training and validation loss as training progresses. Is your model overfitting? Include the plot in your report.
3. (**10 points**) Experiment with the learning rate (at least 5 different learning rates). You can try a few different learning rates and observe how this affects the loss. Use plots to explain your experiments and their effects on the loss.
4. (**10 points**) Experiment with the size of the hidden layer or the model architecture (at least 5 different sizes and/or modifications). How does this affect validation accuracy? (**Bonus 5 points** for those

whose validation accuracies are in the top-3 of the class)

2 Text generation using char-RNN (20 points)

In this section, you will be following more Pytorch tutorial code in order to reproduce Karpathy's text generation results. Read through his blog post here, and then use this code to base your own code on (I cannot find the file shakespeare.txt in the code's GitHub, so I use a different shakespeare.txt file from here to step through the code. Since the input files are different (different lengths, different contents), the outputs aren't exactly the same as in the tutorial. That's okay, the main thing is to understand what the code is doing).

You will notice that the code is quite similar to that of the classification problem. The biggest difference is in the loss function. For classification, we run the entire sequence through the RNN and then impose a loss only on the final class prediction. For the text generation task, we impose a loss at each step of the RNN on the predicted character. The classes in this second task are the possible characters to predict.

Experimenting with your own dataset. Pick some dataset that interests you. Here are some ideas (but you can find and use other datasets as well):

- ABC music format
- Donald Trump speeches
- Webster dictionary
- Jane Austen novels

Include a sample of the text generated by your model (**5 points**), and give a qualitative discussion of the results. Where does it do well and where does it seem to fail? (**5 points**) Report perplexity on a couple validation texts that are similar and different to the training data (**10 points**).

NOTE: If you have this error: `RuntimeError: dimension specified as 0 but tensor has no dimensions` while running the code, you can modify some lines of the train function with:

```
...
loss += criterion(output, target[c].unsqueeze(0))
...
return loss.data.item() / args.chunk_len
```