Assignment 4 Deep Learning 2015, Spring

In this assignment you will train a RNN language model to do word-level and character level prediction. We use the Penn Treebank dataset, a classical small language modeling dataset. You will depart from Wojciech Zaremba's publicly available LSTM RNN code [1].

The format of this assignment is previous from different 3 assignments:

1. This assignment is done **individually**. Teamwork is allowed until the level of talking to each other to understand the provided code, and exchange ideas about architecture. It is forbidden to show someone else the code you wrote.
2. This assignment will provide more detailed instructions and will ask you to answer questions in your write-up. Structure your write-up as before, with an extra chapter "Answers to the questions". This is aimed to make the assignment easier to digest. Each answer should be maximum 2 sentences, no more than 40 words.
3. After following the instructions to get started with the code, it's business as usual and you try to get the best possible performance.

Word-level language model

Start with the LSTM starter code from my fork `https://github.com/tomsercu/lstm`. It has the pre-processed data included and fully working so you can just run `luajit main.lua`. As you can see, the preprocessing replaced unfrequent words by <unkn>, and inserted an <eos> tag for the end of a sentence. The error metric you optimize is perplexity, a standard language modeling metric.

nngraph

First of all, you will see that the code heavily relies on nngraph. Install nngraph as luarock and check out the examples [2] and the tutorial by Nando De Freitas [3].

Q1 Provide me with a file named nngraph_handin.lua that solves the problem from the nngraph tutorial.

The original code

Q2 In the LSTM code, look at the function `lstm(i, prev_c, prev_h)`. Relate these inputs to the equations in the paper [4]. (i = ..., prev_c = ..., prev_h = ...).

To understand the lstm cell better, I advise you to try to implement the lstm() function yourself as an exercise. This is just for your own benefit.

Q3 What does the function create_network() return? Unrolled or not?

Q4 What are model.s, model.ds, model.start_s? When is model.start_s reset to 0?

Q5 What form of gradient normalization is used?

Q6 What optimization method is used?

[1] original: `https://github.com/wojzaremba/lstm`, and my fork which you should clone: `https://github.com/tomsercu/lstm`

[2] `https://github.com/torch/nngraph`

[3] `https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/practicals/practical5.pdf`

[4] `http://arxiv.org/pdf/1409.2329v4.pdf` (Zaremba 2014)

Generating sequences

Implement a function `function query_sentences()` that reads from stdin a number of words, and the network continues the sentence.

Example of my function on the default 1-hour network:

```
Query: len word1 word2 etc
IN: 5 the committee decided to
the committee decided to <unk> it <eos> the <unk>

Query: len word1 word2 etc
IN: 10 the president of the
the president of the united states <eos> institutional investors are holding about eight auctions
```

Some hints:

1. Construct an inverse mapping in data.lua and add it to the return value of the script
2. You will need to add an output node to the gModule to retrieve the predictions. **Q7** How do you deal with the extra output in the backward pass?
3. Look at run_test() for inspiration
4. Look at a4_communication_loop.lua for an example of how to do the stdin/stdout.
5. At each timestep you can take the most probable word from your predictions, but because of the bias in word count in the corpus that will just become something like "<unkn> of the <unkn> <unkn> and <unkn> . . . ". You can make things more interesting by using torch.multinomial() to sample from the predicted distribution instead.

The main reason you should implement this is (1) it will be very useful for the submission (see below), (2) this is the perfect exercise to give you some more insight in the training procedure and model, (3) playing with your network talking gibberish is the coolest thing you'll do all day.

Character level language model

Now switch to a character level language model. To get started just uncomment the right paths in data.lua (they're already set for you). Secondly you also need to replace the calculation of perplexity by the following approximation [5]: instead of `exp(mean(nll)` use `exp(5.6 * mean(nll)`. The motivation for this funky metric is that 5.6 is the approximate average word length of the corpus.

Also since you don't have a test set, don't try to load it / transfer it to device.

As a baseline, use the 1-hour network and change the sequence length to 50. After 13 iterations you will get a validation perplexity around 322.

Suggested improvements to the character-level language model

Here are some ideas how you can improve over this baseline.

- Of course tune hyperparameters
- Change gradient clipping

[5] `http://arxiv.org/abs/1308.0850` (Graves 2013)

- Change optimization method
- SCRNN: `https://github.com/facebook/SCRNNs`
- Dynamic evaluation

Submission

Please send an email with title

```
[Deep Learning YOUR_NETID] YOUR_NAME A4 submission
```

to tom.sercu@nyu.edu.

Within that e-mail, please provide only a single statement that will **clone a git repository into a folder named your netid**. For example, I would write:

```
git clone https://github.com/tomsercu/lstm ts2387
```

Ways to provide me your repo: you could (1) use a private github or bitbucket repository to which you added me as collaborator (my github handle AND bitbucket handle is tomsercu) or (2) a hard path on mercer, but set the permissions of the paths correct, or (3) put your git repo in `$HOME/public_html` [6]. I will clone the repository on mercer, so you can use your local torch install if the permissions are set correctly. In the repository should be your write-up in pdf or html format, with filename `your_netid.pdf`.

Make a script `run.sh` that has the exact command to run your code. I expect something as simple as

```
#!/bin/bash
/path/to/your/personal/luajit main.lua
```

Your program should load your trained model and get in an interactive loop, in which it reads a line, containing a single character, and your program responds with a vector of length 50 containing log probabilities of the next character. The numbers are written to stdout in ascii and separated by spaces. This continues until end of file is reached. Before this interactive loop begins, you indicate you are ready by printing "OK GO".

Example (IN and OUT tags are just for clarity, not part of actual communication):

```
./run.sh
OUT: whatever junk your script writes
OUT: OK GO
IN : t
OUT: -9.123 -4.123 -12.15 ...
IN : h
OUT: -9.123 -4.123 -12.15 ...
IN : e
OUT: -9.123 -4.123 -12.15 ...
IN :
OUT: -9.123 -4.123 -12.15 ...
IN : p
OUT: -9.123 -4.123 -12.15 ...
```

My grading script is in the repo: a4_grading.py. You see that it reads the validate dataset, but I will replace this with an actual testset.

[6] `http://cims.nyu.edu/webapps/content/systems/userservices/webhosting` for hosting, `http://git-scm.com/book/en/v2/Git-on-the-Server-The-Protocols` for Dump HTTP

Hint: you need to call io.flush() in your luascript after printing OK GO and after every line. Otherwise your output gets buffered and the loop will be stuck.

Write-up

Your paper should contain the following details about your approach to the character-level RNN language model:

- description of the architecture (number and type of layers, size of input, etc.)
- description of the learning techniques applied (used dropout?, etc.)
- description of the training procedure (learning rate, momentum, error metrics used, train/validation split, training/validation/test error)

Evaluation

- 25% - Test set performance - at least beat the baseline.
- 25% - Correct answers to the questions.
- 25% - The rest of your paper, specifically what ideas and model architectures you tried beyond the one provided. Max three page paper plus one extra page for references - I encourage you to describe all your experiments and the motivation behind them. Please use a NIPS/CVPR template (LaTeX). You can choose to include the answers to the questions as a normal chapter or as an appendix.
- 25% - Simple, readable, commented, working code