Assignment 4 Report
DS-GA-1008: Deep Learning, Spring 2015

Aditya Ramesh
`_@adityaramesh.com`

1 Answers to Questions

1. The solution to the problem from the nngraph tutorial is implemented in `nngraph/nngraph_handin.lua`.
2. `i` corresponds to x_t, the input to the LSTM unit; `prev_c` to c_{t-1}, the state of the LSTM cell at time $t-1$; and `prev_h` to h_{t-1}, the output of the LSTM cell at time $t-1$.
3. `create_network` returns the "rolled" version of the RNN.
4. `model.s` stores the LSTM cell states and outputs of each layer of the network, evaluated at each element of each sequence in the current batch. `model.ds` stores the gradients of the output with respect to the LSTM cell states and outputs at time $t+1$, when performing BBTT at time t. `model.start_s` is used to store the LSTM cell states and outputs after processing the last element of the previous sequence. It is reset to zero after one pass through the training set.
5. If the norm of the gradient is greater than `params.max_grad_norm`, the gradient is rescaled to have norm `params.max_grad_norm`; an alternative is *gradient clipping.*
6. The script uses Stochastic Gradient Update (SGU), and the gradients are computed using BBTT.
7. With the addition of the extra output node, each call to `backward` will initiate backpropagation from *both* the criterion module and the output node. To nullify the gradient contributions from the output node, we supply a matrix of zeros for its output gradients.

2 Character-Level Model Description

I anticipated that the training time would be the main bottleneck during experimentation. My focus was to try to improve the time to convergence using a variant of the baseline model. To this end, I tried combinations of the following strategies:

- Removing unnecessary sources of model complexity.
- Increasing the batch size.
- Using adaptive optimization algorithms.

One of the main challenges while trying to improve time to convergence was that the optimizer would easily diverge. The default weight decay strategy implemented in the script works well for small batch sizes of around 20, but decreases the learning rate too eagerly when larger batch sizes are used. I tried replacing the replacing the default optimization procedure with the following:

- AdaDelta with Nesterov Accelerated Gradient (NAG). The learning rate was always set to one, the momentum was chosen to be in the set $\{0.9, 0.95, 0.99\}$, and the decay constant was chosen to be in the set $\{0.95, 0.99\}$.

- RMSProp with NAG. Each time the optimizer diverged, I manually terminated the script, decreased the learning rate, and restarted from the version of the model with the best training accuracy.

None of these strategies was effective in accelerating time to convergence when compared to SGU for the batch sizes 50, 100, and 200. In the best case, the optimizer got stuck at around 830 training perplexity, at which point it was no longer able to make progress.

Graves (2013) achieves a perplexity of 122 on the Penn Treebank Test Set using a character-level LSTM RNN with only one layer. This suggests that increasing the complexity of the model while keeping the sequence size constant is probably not the best use of time and computing resources. One reason for this may be that the data set is relatively small, so models are prone to overfitting. Following this reasoning, I trained two models using the default optimizer. Dropout was not found to be helpful for this task.

1. Batch size 20; sequence length 50; one layer; LSTM cell size 200; initial learning rate 1; learning rate decayed every 4 epochs.
2. Batch size 20; sequence length 100; one layer; LSTM cell size 400; initial learning rate 1; learning rate decayed every 4 epochs.

Model 2 achieved a validation perplexity of 252.469 after 15 epochs, while Model 1 achieved a validation perplexity of 492.017 after 14 epochs.

3 Future Ideas

Given the success of the model reported in Graves (2013), I thought of possible reasons for why the performance of the baseline is so poor in comparison. There are three notable differences between Graves' implementation and that of Zaremba:

1. Graves uses "peephole" connections from the LSTM cell to the gates.
2. Graves incorporates skip connections from the input to the LSTM units for several of the models described in his thesis. Graves' model for the Penn Treebank Test Set consists of only one LSTM layer, so this may not be relevant.
3. Graves uses gradient clipping instead of gradient scaling.

Item 1 in particular may allow the network to make better use of its memory from previous timesteps.

The current implementation does not shuffle the sentences after each epoch, and the contextual memory of the network is only cleared after one pass through the entire training set. I think that it would make more sense to shuffle the sentences after each epoch, and clear the network's memory after processing each sentence. This scheme would complicate data processing, but is something that I think is worth trying in the future.