Independent Project #3

By Jibang Wu, jw7jb

1. RNN from scratch:
   Implementation in tensorflow: `simple-rnnlm.py`
   use library LSTM cell, yet the rnn logic is implemented with a for loop on the sequence
   Since Tensorflow use static computational graph, the sequence length was preset to the max sequence length in the dataset, and every sequence was padded to this length, though the loss will not include these padded part.
   learning rate was set to 0.1
   This implementation is slow without gou optimization; In the later part of the assignment, to focus on the parameter tuning and loss function improvement, it thus use the cudnn version lstm instead.
2. Perplexity:
   Implementation in tensorflow: `jw7jb-perplexity.py`
3. Perplexity result:
   - With learning rate of 0.1
     after 1 epoches, the perplexity on training set is 1928.8, the perplexity on dev set is 882.3
     after 10 epoches, the perplexity on training set is 644.2, the perplexity on dev set is 545.6
     The log probabilities on the test data is written in `jw7jb-tst-logprob.txt`, where the format is `token\t{log probability}`
4. Stacked LSTM:
   Implementation in tensorflow: `jw7jb-stackedlstm-rnnlm.py`
   - 1 layer LSTM
     after 1 epoches, the perplexity on training set is 1928.8, the perplexity on dev set is 882.3
     after 10 epoches, the perplexity on training set is 644.2, the perplexity on dev set is 545.6
   - 2 layer LSTM
     after 1 epoches, the perplexity on training set is 1947.5, the perplexity on dev set is 1033.5
     after 10 epoches, the perplexity on training set is 804.2, the perplexity on dev set is 652.7
   - 3 layer LSTM
     after 1 epoches, the perplexity on training set is 1941.5, the perplexity on dev set is 1041.2
     after 10 epoches, the perplexity on training set is 907.1, the perplexity on dev set is 738.6
     So stacked LSTM under current experiment setting cannot improve perplexity
5. Optimization:
   Implementation in tensorflow: `jw7jb-opt-rnnlm.py`
   - SGD with learning rate 0.1
     after 1 epoches, the perplexity on training set is 1928.8, the perplexity on dev set is 882.3
     after 10 epoches, the perplexity on training set is 644.2, the perplexity on dev set is 545.6
   - Adam Optimizer with learning rate 0.1
     after 1 epoches, the perplexity on training set is 1645.8, the perplexity on dev set is 1622.8
     after 10 epoches, the perplexity on training set is 1458.0, the perplexity on dev set is 1457.4

- SGD with learning rate 0.1 and momentum 0.01
  after 1 epoches, the perplexity on training set is 1920.2, the perplexity on dev set is 897.9
  after 10 epoches, the perplexity on training set is 619.5, the perplexity on dev set is 524.5
  So SGD with momentum under current experiment setting improves the perplexity, while Adam Optimizer with learning rate 0.1 does not improve the perplexity

6. Model Size:
   Implementation in tensorflow: `jw7jb-model-rnnlm.py`
   - input/hidden dimensions 32
     after 1 epoches, the perplexity on training set is 1928.8, the perplexity on dev set is 882.3
     after 10 epoches, the perplexity on training set is 644.2, the perplexity on dev set is 545.6
   - input/hidden dimensions 64
     after 1 epoches, the perplexity on training set is 1820.4, the perplexity on dev set is 854.3
     after 10 epoches, the perplexity on training set is 537.2, the perplexity on dev set is 481.1
   - input/hidden dimensions 128
     after 1 epoches, the perplexity on training set is 1737.7, the perplexity on dev set is 840.8
     after 10 epoches, the perplexity on training set is 504.9, the perplexity on dev set is 462.8
   - input/hidden dimensions 256
     after 1 epoches, the perplexity on training set is 1682.1, the perplexity on dev set is 882.3
     after 10 epoches, the perplexity on training set is 484.8, the perplexity on dev set is 455.2
     So under current experiment settings, the increasing model size does improve the perplexity, and the optimal model size so far is 256
7. Mini-batch:
   Implementation in tensorflow: `jw7jb-minibatch-rnnlm.py`
   Yes, different mini-batch sizes make a difference. The best minibatch size is 16.
   So here I compared the minibatch size of 16, 24, 32, 64 under same learning rate 0.01 experiment settings.
   Interestingly, overfitting on training is also introduced by minibatch, which did not happen in previous experiments.
   Using minibatch, the performance is almost as good as those with larger hidden/input size. So minibatch shows its ability to better and quickly converge the model to optimiality.
   Note that padding is used to keep all sample in the btach has same sequence length. However, the loss and perplexity calculation did not take into these padding, since they were not part of the originial data.
   - Baseline
     after 1 epoches, the perplexity on training set is 1928.8, the perplexity on dev set is 882.3
     after 10 epoches, the perplexity on training set is 644.2, the perplexity on dev set is 545.6
   - mini-batch size of 16
     after 1 epoches, the perplexity on training set is 1573.5, the perplexity on dev set is 1002.0
     after 10 epoches, the perplexity on training set is 409.1, the perplexity on dev set is 461.9
   - mini-batch size of 24
     after 1 epoches, the perplexity on training set is 1736.9, the perplexity on dev set is 1180.9
     after 10 epoches, the perplexity on training set is 410.8, the perplexity on dev set is 453.1
   - mini-batch size of 32
     after 1 epoches, the perplexity on training set is 1683.7, the perplexity on dev set is 1166.98
     after 10 epoches, the perplexity on training set is 417.4, the perplexity on dev set is 472.9
   - mini-batch size of 64
     after 1 epoches, the perplexity on training set is 1806.5, the perplexity on dev set is 1448.7

after 10 epoches, the perplexity on training set is 544.9, the perplexity on dev set is 539.0