COSI134A FALL 2019
Project4: Train a neural network parser *Jiawei Zhang*

This project is my implementation of train neural network parser using Penn Treebank data. Specifically, we use the encoder-decoder framework and attention mechanisms realized by tensorflow/nmt. On average, the program takes 8 hours to train on 40,000 train steps, including training and outputting predicted sentence structure of test file. The model recall is around 90%.

Code structure

- **preprocess_output.py** preprocess raw data and output them (input and output of vocab, train, dev, test) in the format that can be processed by tensorflow/nmt
- **postprocess_output.py** postprocess the test output to compare with gold file by EVALB
- **processor.py** load data and contains some preprocess and postprocess method
  linearize_parse_tree() linearize tree
  ex. (S (NP (N I)) (VP (V sleep)) (. .)) → (S (NP N)NP (VP V)VP .)S
  postprocess() match output and origin sentence to
  (S (NP N)NP (VP V)VP .)S + I sleep. → (S (NP (N I)) (VP (V sleep)) (. .))

Folder Structure

- nmt
  - nmt – folder contains nmt code
  - tmp[1] – folder contains experiments result
    - ex. luong, ba
- data
  - train, dev, test – folders with gold data
  - preprocess – folder contains pre-processed data.
    - File: train.en, train.parse, dev.en, dev.parse, test.en, test.parse for training and predicting; gold_output_test for EVALB
  - postprocess – folder contains post-processed data of different models
    - ex. test_luong, test_ba

How to run?

Use attention = bahdanau for example

1. **Get files ready** python preprocess_output.py (cd Proj4 folder)
2. **Run nmt program in command line with different parameters** (cd nmt folder)
   CUDA_VISIBLE_DEVICES=0
   python -m nmt.nmt
   --attention= bahdanau
   --src=en --tgt=parse
   --vocab_prefix=../data/preprocess/vocab
   --train_prefix=../data/preprocess/train
   --dev_prefix=../data/preprocess/dev

---

[1] We didn't include this file in hand in files for they are too large. We upload different models' output.

--test_prefix=../data/preprocess/test
--out_dir=./tmp/ba
--num_train_steps=40000
--steps_per_stats=100
--num_layers=2
--num_units=128
--dropout=0.2
--metrics=bleu

where we can specify the model parameters, attention = luong, bahdanau; beam_width = 5, 10, 20; dropout = 0.2, 0.5, 0.8

3. **Postprocess output test file** python postprocess_output.py ba
4. **Evaluate model performance with EVALB** (cd EVALB folder)
   ./evalb -e 50000 ../data/preprocess/gold_output_test ../data/postprocess/test_ba

We store the output file in ./data/postprocess and named them by according model.

The best model is with parameter attention = luong, num_layers = 2, num_units = 128, num_train_steps = 40000, beam_width = 0, dropout = 0.5

Experiments

1. Attention mechanism

	Luong	bahdanau	No attention
Recall	90.56	89.00	71.54
Precision	92.72	90.41	72.69

Luong attention mechanism has higher accuracy. Both attention mechanism's accuracy is significantly higher than model with no attention.
parameters: num_layers = 2, num_units = 128, num_train_steps = 40000, beam_width = 0, dropout = 0.2

2. Reverse source input

	Reverse	No reverse
Recall	75.74	90.37
Precision	77.16	92.16

As we can see, reverse source input has much lower accuracy. It's not a good feature.
parameters: attention: luong, num_layers = 2, num_units = 128, num_train_steps = 40000, beam_width = 0, dropout = 0.2

3. Beam width

	5	10	20	0
Recall	88.46	90.93	87.54	90.56
Precision	89.32	91.51	89.88	92.72

We can see that when we do beam search, a width of 10 provides the best result. However, the best result is still than width = 0.
parameters: attention: luong, num_layers = 2, num_units = 128, num_train_steps = 40000, dropout = 0.2

4. Drop out

	0.2	0.5	0.8
Recall	90.37	91.27	85.58
Precision	92.16	91.41	88.95

As we can see, drop out ratio of 0.5 has highest accuracy, it avoids both overfitting and underfitting.
parameters: attention: luong, num_layers = 2, num_units = 128, num_train_steps = 40000, beam_width = 0

Above all, attention mechanism does help us in predicting sentences structures. Reverse source input doesn't help. Drop-out ratio of 0.5 increases accuracy. We run program on the server. However, we couldn't make it run with GPU, which accounts for long running time.