Erik Andersen
COSI 134A PA4
16 Dec 2019

PA4: An OpenSource NMT Tutorial

I. Code Structure

For this project, we used an open-source Neural Machine Translation available on *github* (tensorflow/nmt), so we did not have to implement attention ourselves. Therefore, the only code files that I submitted were *preprocess.py* and *postprocess.py*, which, as suggested by their names, preprocess and postprocess the data as needed.

a. *preprocess.py*: Prepares the ptb data by separating out sentences and linearized trees for use with tensorflow/nmt. For linearization, instead of stand-alone right parentheses, a tag is added to every ')' character, which helps show the bracketing agreement structure during training. Also, this file creates the vocabulary files to be used for the source and target. To run this file, type *python3.6 preprocess.py* in the pa4 directory. Optional flags include *--data_dir* (where the data is to be found) whose default is ./data, and *--do_reverse* (default is False), which is the boolean value indicating whether the encoder input should be reversed. If specified, this last flag should be either *True* or *False*, and no other value.

b. *postprocess.py*: Uses the sequences outputted by nmt (and the sentences) to create well-formed bracketed sequences that can be evaluated by EVALB. EVALB is used to evaluate both the dev and test set predictions. To run this file, type *python3.6 postprocess.py* in the pa4 directory. You can specify the out directory (from nmt) using the flag *--model_dir* and the *evalb* directory using the flag *--evalb_dir*. Finally, *--do_reverse* (default is False) can be specified if the preprocessed source input was reversed. This would reverse the source input back to the unaltered version. Note, however, that EVALB's accuracy calculations do not take into account the positions of the words in the tree, so the postprocessor *--do_reverse* flag may never need to be specified.

II. Experimental Overview and Settings

Before beginning the main experiments, I ran the tutorial as described online. I used this format to construct a fitting run mechanism for this project. Running from the outermost nmt directory, which I placed within my pa4 directory, the command line operations I used were as follows.

```
python3.6 -m nmt.nmt --src=sts --tgt=lnr --batch_size=64 --train_prefix=../data/train
--dev_prefix=../data/dev --test_prefix=../data/test --out_dir=../attention --num_train_steps=20000
--learning_rate=.35 --steps_per_stats=100 --num_layers=1 --num_units=64 --dropout=0.2 --metrics=bleu
--vocab_prefix=../data/vocab --attention=scaled_luong --src_max_len=154 --tgt_max_len=558
--tgt_max_len_infer=558
```

Please note that you should run this with python3.6, as I found problems running it with 3.7.

What follows is an explanation of some of these hyperparameters and why I chose the values seen above.

Hyperparameter (or Option)	Value	Explanation for Choice
--src	sts	This indicates the file extension for all source files. This is fixed in *preprocess.py* and should not be altered.
--tgt	lnr	This indicates the file extension for all target files. This is fixed in *preprocess.py* and should not be altered.
--batch_size	64	This was adapted from the default batch size parameter found in the starter code. The default appears to be 128 for nmt.
--train_prefix, --dev_prefix, --test_prefix	see above	This indicates the prefix of paths that contain the files to be used for the train set, dev set, and test set. (joined with the extensions from --src and --tgt). The prefix will depend on where the *nmt* repository sits in relation to

		the *data* file on any given system.
--out_dir	../attention	This is the directory to be created and filled with information output by the model. Again, this path will depend on where the *nmt* repository sits in relation to the *data* file on any given system.
--num_train_steps	20000	I attempted to set the number of training steps higher than the value in the tutorial (12000), but low enough to allow time for all experiments. At the given learning rate, the model seemed to reach a peak bleu score at around step 14000.
--learning_rate	0.35	I tried learning rate at the default value (1.0), 0.35, 0.25, 0.2, 0.1, 0.3, and 0.01. The default was clearly too fast and 0.01 was way too slow. A learning rate of 0.35 gave the highest 'best bleu score' (76.3 on the test set).
--steps_per_stats	100	This indicates the number of steps to be taken before a line is printed containing the step number, the learning rate, time per step, words per second (wps), perplexity, gN, (wps), the max bleu score, and the time. This number remains unchanged from the tutorial.
--num_layers	1	This indicates the number of layers for the neural network, and it was adapted from parameters found in the starter code.
--num_units	64	This number indicates the number of units in the output layer of the network. *64* was adapted from the starter code and was used in order to save time, as 128 and 512 slowed the training down considerably.
--metrics	bleu	This tells us that the model will use the BLEU score as a metric. This is left as is from the tutorial.
--vocab_prefix	../data/vocab	This is the prefix indicating files to be used for vocabulary. Again, this path will depend on where the *nmt* repository sits in relation to the *data* file on any given system.
--src_max_len	154	This value represents the maximum length of sequences found in the source files. As this parameter has a default of 50, it was set to the longest sequence found in the source sentences, which was 154. This number is calculated and printed in preprocess.py
--tgt_max_len	558	This value represents the maximum length of sequences found in the target files. As this parameter has a default of 50, it was set to the longest sequence found in the target sequences (linearized tag sequences), which was 558. This number is calculated and printed in preprocess.py.
--tgt_max_len_infer	558	This value represents the maximum length allowed for target sequences during inference. It was set to the same as tgt_max_len because otherwise, the model appears to use a value of twice the length of the source sequence, which is often too short.

Values not present in the table above (--attention, --dropout) are experimental hyperparameters that will be tweaked in the section below.

III. Experiments

For the experiments, we tweak the following: the method of attention used (or lack thereof), the dropout value, and the beam width hyperparameter. We also test whether reversing the encoder input has any effect on the accuracy output by EVALB.

We begin by running using different kinds of attention, ultimately testing without attention entirely.

	Normed bahdanau	Scaled luong	No attention

Best BLEU score	71.05	75.32	37.08
EVALB Tagging Accuracy (Dev, test)	68.16, 67.28	81.20, 81.45	23.25, 22.64

We can see that normed bahdanau and scaled luong attention both perform much better than using no attention at all. We set attention to scaled_luong and then continue. Next, we experiment with the dropout value. Values tested are as follows: 0, 0.2, 0.3, 0.5, 1.0.

	Dropout Value			
	0.0	0.2	0.3	0.5
Best BLEU score	73.42	75.32	70.49	73.71
EVALB Tagging Accuracy (Dev, test)	65.63, 66.92	81.20,81.45	74.60, 73.93	77.06, 77.09

The worst tagging accuracy comes when we use a dropout of 0.0, and the best appears at 0.2. We set dropout=0.2 and then experiment with the beam width hyperparameter. Values tested are as follows: 0, 10, 25, 50.

	Beam Width			
	0	10	25	50
Best BLEU score	75.32	76.33	73.98	73.17
EVALB Tagging Accuracy (Dev, test)	81.20,81.45	83.74, 83.08	83.62, 84.30	82.45, 82.09

We can see that having beam width of 10, 25, and 50, all do better than having beam width of 0, but the best seems to be in the 10-25 range. Since a beam_width of 10 gave the highest average between BLEU score and tagging accuracy, we use that value, even though a beam width of 25 gives the best tagging accuracy on the test set. Finally, we reverse the encoder input (source sentences), using beam width of 10, as well as all set parameters above, and see what effect that has on the accuracy.

	Unaltered	Reversed
Best BLEU score	76.33	75.20
EVALB Tagging Accuracy (Dev, test)	83.74, 83.08	73.04, 72.95

It appears that using our parameters, the reversed source input did a fair bit worse on tagging than using the unaltered source input with all other hyperparameters being the same. As a last datapoint, we include one last experiment, done with all the same parameters as above, except with --batch_size=128, --num_units=128, --num_layers=2, and --beam_width=0. For this experiment, the best bleu score achieved was 82.15, the tagging accuracy on the dev set was 85.59, and the tagging accuracy on the test set was 84.89. Using a beam width > 0 and keeping the number of units at 128 or raising it (to 512, for example), we can hypothesize based on the above data that we might be able to achieve a tagging accuracy close to 90%, which is approximately the accuracy achieved by state-of-the-art sequence to sequence models using attention.