Attention-based Encoder-Decoder Parsing

Jonne Saleva *

2019-12-17

Code Structure

The structure of this project is a little more convoluted than the previous projects – sorry about that. The root of the zip file contains two folders `ptb_processed_data`, and `nmt_fork` which contain processed training data and a fork of the `nmt` repository, respectively. Inside the `nmt_fork` folder is the starter code provided for this project, specifically in `./nmt_fork/starter_code`.

The actual training is performed using the various shell scripts in the `nmt_fork` folder. The same is true for inference and evaluation which can be run from there as well. As for postprocessing, it is necessary to navigate into the `starter_code` folder and run `postprocess.sh`.

As for pre and post-processing, the preprocesing code can be found in `starter_code/create_nmt_data.py`. The code creates the folder `starter_code/training_data` whose files were then manually renamed to match `ptb_processed_data`. The post-processing only alters the number of parentheses in the decoded tree so that the open and close parentheses match in number. No insertion of tokens is performed since that it not necessary for decoding.[1]

Experimental Settings

Overall, in the interest of time and also since we ran out of AWS credits, we perform the following 10 experiments. We run the 5 different attention mechanisms availa ble through the `nmt.py` script: `noattention`, `bahdanau`, `normed_bahdanau`, `luong`, and `scaledluong`. We combine these attention mechanisms with two decoding mechanisms: `greedy` and `beam10`, the latter of which indicates a beam search decoding with beam with 10.

*jonnesaleva@brandeis.edu

[1] I know you did it Ben, and I would too if I didn't have a flight to Hong Kong in a few hours...

Results

We summarize the results in terms of tables created from `nmt_fork/evaluation/all_eval.csv` using the `pandas` library in Python.

Below are the results in terms of bracketing F1 score and tagging accuracy when considering **all** sentences:

		bracketing_f1	tagging_acc
attention	decoding_scheme		
scaled_luong	beam10	84.96	95.20
	greedy	83.87	93.25
luong	greedy	73.51	82.46
noattention	beam10	38.78	43.40
	greedy	38.42	44.00
bahdanau	greedy	34.92	38.86
	beam10	33.06	38.18
normed_bahdanau	beam10	23.30	29.85
	greedy	19.47	23.09
luong	beam10	13.59	22.39

Below are the results in terms of bracketing F1 score and tagging accuracy when considering only **short** sentences:

		bracketing_f1	tagging_acc
attention	decoding_scheme		
scaled_luong	beam10	84.98	95.24
	greedy	83.53	93.42
luong	greedy	73.70	82.65
noattention	beam10	38.78	43.40
	greedy	38.47	44.05
bahdanau	greedy	35.26	39.22
	beam10	33.06	34.26
normed_bahdanau	beam10	23.35	29.89
	greedy	19.52	23.12
luong	beam10	13.86	22.81

Discussion

As we can see, the results are fairly similar whether we choose to evaluate short sentences or everything together. An interesting thing to note is that for unscaled `luong` attention, changing the decoding seems to hurt the performance a LOT. With more time, I would look into this more deeply – it seems like such a large performance change that there may be something wrong with my code.

The main takeaway from the tables above seems to be that scaled Luong attention is the clear winner. Surprisingly, the "no attention" setting was *not* the worst performing experiment so it is hard to draw inferences like "attention is always better" based on these results alone.

All in all, I feel disappointed that true to time and financial constraints I was not able to run more experiments. It would have been nice to try, for instance, how well a GRU does viz-a-viz an LSTM, or how using a bidirectional LSTM would influence performance. However as I mentioned in my footnote, I have a flight to Hong Kong tomorrow morning, and thus cannot take advantage of the extension...

Please have mercy while grading – thank you for a great semester!!