Report

z5089812, Xuan Tong

Abstraction

After testing multiple times, the average performance I've got is around 9. Aspects I've taken into account are preprocessing, batch generation, and similar words extraction, I'll walk through these three aspects in detail.

Methodolody

Preprocessing

This project is dedicated to find out similar words for adjectives, many words have nothing to do with this purpose. And with powerful spacy, we can get information like position of speech, we can have the lemmaed and lowercased. Operations I implemented in this phase are:

1. Transform words of POS tag 'NUM', 'SYM', 'CONJ', 'PREP', 'PRON', 'PPRON', into the tag name directly, since words like 'and', 'or', 'it', 'the' are not capable to produce any implication for the meaning of adjectives.
2. Skip words with POS tag of 'PUNCT', 'SPACE', 'PART', these characters will mess most of operation, so filter them
3. Have words of POS tag 'ADJ', 'NOUN', 'VERB' labeled with corresponding tags, so we can easily to apply extra operations to these three relatively important types of word in batch generation and similar word extraction.
4. Also have all other words lowercased and pick their lemma form.
5. Add '-EOF-' tag to processed words list, which indicate the end of file. Will flush the buffer after encountering this tag.
6. Add '-EOF-' tag to processed words list, which indicates the end of sentence. Didn't add extra operations toward this tag, but surprisingly, it learns the meaning of this tag in some extent.

Batch Generation

This part refers a bit to the given word2vec demo. Use a sliding window to extract context word of the center word, for each sliding window, make it generate 'num_sample' (center word, context word) pairs, implementation worth noticing are:

1. Use subsampling trick, the words tend to appear in high frequency, give it low probability to be chosen as context word, formula as following:

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \cdot \frac{0.001}{z(w_i)}$$

2. Skip all the 'EOF' tag, and after each sliding window generation, if there exist the word 'EOF', flush the buffer, move the 'data_index', and fill in the buffer with words from next file
3. If the center word is an adjective, before randomly picking context words, always try to find the VERB before this word and the NOUN after this word within the sliding window, prefer to add pairs like this to batch.

Similar ADJ word extraction

This part is fairly easy, operation I've implemented are as following:

1. Since the purpose is trying to find similar adjectives, so only pick ADJ out of embedding file, by using the suffix tag added to adjectives in the preprocessing phase.
2. Make use of 'most_similar' function of genism model
3. If target is not in the embedding file, simply return a list of common see adjectives.

Result and Conclusion

In the very beginning, results are barely over 3. After implementing all the mentioned implantations, my findings are 'ADP' takes large portion of word appearance, we'd better leave that in original text instead of replacing them with merely 'ADP', also after implementing subsampling, the performance get improved by approximately 1.5, which is quite significant. I also implemented a version where it only generates training pair of words sit within one sentence, however that didn't give me the expected output, can't explain why that get outperformed by the coarser version.

Above are my implementations for this project, after doing this project, I get a rough idea of word embedding and word2vec model.