CS5740: Assignment 2

https://github.com/cornell-cs5740-21sp/a2--a2-group-19

Kae-Jer Cho kc2225
Haomiao Han hh696
Yuxin Zhang yz2726

1 Introduction (5pt)

This report aims at exploring different methods for generating word embeddings, so as to achieve high accuracy in computing word pair similarities on both development and test dataset. The data our team used is the 1 million sentences of the billion word language model benchmark dataset as well as the dependency parser analysis on that 1 million sentences.

We took the Word2Vec general approach in building embeddings for each word, which generally relies on two things: the $(word, context)$ collections as the training dataset, and the model for data training. For the first part, we used both the linear and syntactic ways to build the $(word, context)$ collection. Also, we experimented with different window sizes in building linear context datasets. For the second part, we experimented with different embedding dimensions, epoch size, and batch size in running the model. The best embedding model is the skipgrams-based model with a score of 0.531 on the development set.

2 Model (10pt)

The model first reads the input dataset and performs preprocessing on the dataset. Afterwards, $(word, context)$ pairs are generated using either based on linear context (skipgram) or syntactic structure of the sentence. We use negative sampling in our model; for each $(w_{observed}, c_{observed})$ pair observed in the input data, we generate k negative $(w_{observed}, c_{random})$ pairs that we consider as not observed in the input data. The purpose of negative sampling is to make the training process more efficient, as in each epoch the model only need to adjust $k + 1$ weights for each word in the vocabulary.

To generate the random $(w_{observed}, c_{random})$ pairs, we sample the entire vocabulary using a method that over samples words that appear most often in the training data. Specifically, the probability of each context word $cr \in V$ is selected is:

$$p_{cr} = \frac{(N_{cr})^{0.75}}{|V|} \tag{1}$$

where N_{cr} is the number of times that word cr appears in the corpus, and $|V|$ is the vocabulary size of the corpus. Existing papers [1] have pointed out that exponentiating N_{cr} with 0.75 works best empirically, since this will somewhat increase the chance of selecting words that do not appear very often in the corpus.

Given the architecture of our model, we want to maximize the likelihood of observing an observed $(word, context)$ pair and minimize the likelihood of observing a $(word, context)$ that is not observed in the original dataset. Thus, we derive the following objective function of our model:

$$log\ \sigma(v_{co}^T\ v_{wo}) + \sum_{i}^{k} log\ \sigma(-v_i^T\ v_{wo}) \tag{2}$$

where v denotes a word vector, co denotes an observed $context$, wo denotes an observed $word$, and k denotes the number of negative samples. The first term (i.e. the term before +) seeks to maximize the probability of occurrence of an observed $(word, context)$ pair, while the second term seeks to minimize the probability of occurrence of all k random $(word, context)$ pairs that are previously generated by negative sampling.

3 Experiments (12pt)

We experiment with different configurations of parameters in order to derive the best performance. Firstly, we experiment with two different ways of deriving $(word, context)$ pairs: one based on skipgrams and another one based on syntactic structures. Within the skipgrams model, we further experiment with the amount of training data by adjusting the window size w. We also experiment with the dimension of the word embeddings. Within the syntactic-based model, we further experiment with dimension of embedding, number of epochs, batch size.

4 Data

Data (5pt) For the $(word, context)$ training dataset built upon syntactic-based approach, we build each $(word, context)$ pair in the 1M training dataset based on the syntactic annotation for each word in each sentence. That is to say, only words analyzed as syntactically or semantically relevant to the target word in a particular sentence are considered as context, regardless of its distance from the target word. For example, given sentence "What a world cup", where where word token is annotated as (1,What,4), (2, a, 4), (3,World,4), (4,Cup, 0), the $(word, context)$ is built as: $(What, Cup)$, (a, Cup), $(World, Cup)$, $(Cup, What)$, (Cup, a), $(Cup, World)$.

Under the skipgram-based approach, we generate skipgrams for each word in every sentence base on the specified window size. For example, given sentence "I ate pizza for lunch today" and window size $w = 2$, we have the following skipgrams for the word $pizza$: $(pizza, I)$, $(pizza, ate)$, $(pizza, for)$, and $(pizza, lunch)$.

Effective vocabulary size is calculated from all the word tokens of which at least one $(word, context)$ is generated upon. Under the skipgram-based approach, the effective vocab size is all the word tokens in all sentences after rare words substitution (see next section for details). In the syntactic-based context dataset generation, only word tokens of which a $(word, context)$ pair can be found is counted in the vocabulary since the syntactic context for a word token might not exist after preprocessing stage.

Vocabulary Size	262416
Training set size	353
Dev set size	1000000
Test set size	2034

Table 1: Input Data Statistics

Pre-processing and Handling Unknowns (10pt) Before building the $(word, context)$ pair dataset, we first preprocessed and tokenized the 1M raw text sentences. Preprocessing includes converting all characters to lower cases, removing the numerical characters, removing the special characters, and removing stop words. Those procedures are conducted since we are only interested in embedding character-based words, and that we would like to only use semantic-indicative words to build context. We use the entire dataset for generating $(word, context)$ pairs since more data enriches the contextual information in the training dataset and allows for more complex models in training.

Context	Window size	Effective vocab size	# of (c,w) pairs
linear	2	79397	44877554
linear	3	79397	64673904
linear	5	79397	99222716
syntactic	N/A	225473	15528490

Table 2: Input Data Statistics (model dependent)

Unknown words handling is a two-step process. In the first step, during preprocessing, we set a certain "removal threshold" and replace each word with its frequency lower than this threshold by the "$\langle UNK\rangle$" token. This substitution operation is also performed with all $(word, context)$ pairs. The disadvantage of doing so is that less frequently observed words will get removed from the training dataset, thereby losing some contextual information since those words also serve as a context for other words. On the other hand, treating less frequently observed words all as "$\langle UNK\rangle$" provides rich contextual information on unknown words in general, which would potentially result in better generalization in the test dataset where there are true unobserved words.

In the second step, after training finishes and the word embedding is generated, we perform additional handling for unknown words in `reduce.py`. Specifically, for each unknown word $word_{unknown}$, we conduct the following operations: word stemming using both `PorterStemmer` from `NLTK` and manual stemming (removing "ly" and "ish" suffixes); finding synonyms of $word_{unknown}$ using `wordnet`; and finding hypernyms and hyponyms of $word_{unknown}$. After each operation, we search our vocabulary to see if there is a match ($word_{matched}$). If a match exists, we then assign the vectors of $word_{matched}$ to $word_{unknown}$. If, after all the operations, a match could not be found, we assign the vectors of the "$\langle UNK\rangle$" token to $word_{unknown}$.

5 Implementation Details (3pt)

The two main parts of our implementation are preprocessing and model building. Data preprocessing details are discussed in the previous section: first, we use `nltk` library as the stop word resource and `pandas` library to help construct and process data. We set our removal threshold to 2 when removing less frequent words. Second, We separately use linear and syntactic methods to build data collection as our model's input data. We then assign an index number for each word, and convert each $(word, context)$ pair from string to numbers. Third, we apply `PyTorch` library to build and train our model, which we try several approaches like implementing a DataLoader-like function to generate data batches and subclassing `pytorch.nn.Module` to create a two-layer neural network model that implements the vector calculation and loss calculation. New negative samples are generated dynamically in each batch in the training process, and we generate 4 negative samples for each observed $(word, context)$ pair. We use Adam as the gradient descent optimizer, with learning rate = 0.0015.

Context	Window size	Epoch	Embedding dim	Batch size	Dev accuracy rate	UNK handling
linear	2	8	100	512	0.438	Method 1
linear	2	8	100	512	0.506	Method 2
linear	3	8	100	512	0.518	Method 2
linear	3	8	200	512	0.484	Method 2
linear	3	8	200	512	0.488	Method 3
linear	5	8	100	1024	0.531	Method 3
syntactic	/	4	100	512	0.359	Method 1
syntactic	/	20	100	4096	0.448	Method 1
syntactic	/	25	200	4096	0.494	Method 1
syntactic	/	25	200	4096	0.489	Method 3
syntactic	/	25	300	8192	0.442	Method 3

Table 3: Development set word-pair similarity correlation with different model experimentation. Unknown words handling methods are as follows: Method 1 = only assign the vector of "$\langle UNK \rangle$" token to all unknown words; Method 2 = use word stemming and synonym finding; Method 3 = use word stemming and synonym/hypernym/hyponym finding.

6 Results and Analysis

Test Results (3pt) We are able to achieve a maximum score of 0.17302 on the skipgrams model (window size = 5, embedding dimension = 100) and 0.117 on the dependency-based model. We acknowledge that there is a considerable difference between our development results and test results. This is likely due to insufficient handling of unknown words.

Development Results (15pt) Our best score on the development set is 0.531 using a skipgrams mode with window size = 5, embedding vector dimension = 100, batch size = 1000, and with UNK handling Method 3. With these different combinations of hyperparameters and UNK handling methods, we find that larger window size and larger batch size works better, and UNK method 3 works better than UNK method 1 and 2. Intuitively, with our training data, generating the skipgrams with a larger window size helps the model to capture the context with words. However, a larger embedding vector dimension does not improve the performance here. Comparing between different UNK handling methods, Method 3 stands out because it tries to look up the unknown words by several techniques like word stemming, finding synonyms/hypernyms/hyponyms, which maps unknown into related words in our embeddings.

Qualitative Analysis (10pt) We seek to figure out the difference between our prediction score and the human similarity score. We find some pairs with low prediction scores but with a relatively high human similarity score, which indicates a poor prediction for this pair from our model, shown in Table 4. These pairs are pretty similar by the meaning, leading to a high human score. The frequency difference of the words in the pair might be a reason for the undesirable performance: for example, from our training data, we get 2293 occurrences of "sex" and only 6 for "fuck"; 2193 for "dollar" and 199 for "buck"; 10499 for "money" and 1238 for "currency". This affects how our model builds embeddings.

Pair	Prediction	Human
(fuck, sex)	0.59	9.44
(dollar,buck)	0.30	9.22
(money,currency)	1.36	9.04

Table 4: Prediction failures

7 Conclusion (3pt)

In this assignment, we implemented skipgrams and dependency-based models for word embeddings. Our experiments and results show that the skipgrams-based model works better, achieving a score of 0.17302 on the test set. Qualitatively, we found that some of the errors we made are likely due to the lack of availability of certain words in the training data.

8 Reference

[1] `https://fajieyuan.github.io/papers/WSDM2018.pdf`