Text Analytics HW2

Name: Du, Sophie

NetID: cdp4205

Problem 1

Compare Embedding Sizes

(Refer to code part 1: Compare Embedding sizes)

In this section, we compare the impacts on model outputs of ***different embedding sizes***, while keeping window size and model type the same. Here we experiment on both CBOW and skip-gram models with default parameters but different embedding sizes of 100 (default), 200 and 300. We evaluate the embeddings by hand-picking 5 words ('disease', 'treatment', 'drug', 'effective', 'patient') and manually reviewing their closest neighbors in terms of cosine similarity.

The embedding size represents the number of dimensions of the embeddings, which refers to the **length of each word vectors**. Theoretically speaking, larger embedding sizes (longer vectors) can **store more information** since they have more possible states and the representations can be more accurate to distinguish among similar words (words with similar meanings are spatially close to each other). Based on the experiment result, when size=100, the word 'treatment', for example, its most similar word is 'drug' with 0.92 cosine similarity; when size=500, the most similar words still include 'drug' but with a little higher cosine similarity as 0.94, and the value could fluctuate randomly. Also, as the embedding size increases, the training time increases as well, so practically there is not much benefit to go beyond a size of 300-500, and in some applications even smaller vectors work fine.

Compare Window Sizes

(Refer to code part 2: Compare Window Sizes)

In this section, we compare the impacts on model outputs of ***different window sizes***, while keeping the embedding size and model type the same. Here we experiment on both CBOW and skip-gram models with default parameters but different window sizes of 2, 5 (default) and 10. We evaluate the embeddings by hand-picking 5 words ('disease', 'treatment', 'drug', 'effective', 'patient') and manually reviewing their closest neighbors in terms of cosine similarity.

The window size represents the maximum distance between a target word and words around the target word, which refers to the **context** words we are looking at. When window size increases, *more context information will be included*. For example, 'Patients take drugs for the treatment', if window size is 2, the vector of word 'patients' is directly affected by the word 'take' and 'drugs'; if window size is 5, then the vector of word 'patients' can be affected by three more words 'for', 'the', 'treatment'. By 'affected', it means it will pull the vectors of three words closer. In this case, the word vectors highly depend on the context we are using for training. In other words, if window=2 can already capture the context of a word, then if window=5 is chosen, it will **decrease the quality of the learnt model**, as useless information is included, and vise versa.

Compare Models

In this section, we generally compare two Word2Vec models, ***CBOW vs skip-gram***, based on how their functionalities.

- Firstly, CBOW model takes the context of each word as input and predicts the word corresponding to the context, while skip-gram takes the target word and predicts the context surrounding the target words.
- Secondly, when using same corpus and keeping all other parameters the same (window size, embedding size), CBOW is generally running faster than skip-gram.
- Lastly, As skip-gram is predicting context, it works well with small amount of data and could represent rare words well. On the other hand, CBOW has better representations for more frequent words as it takes the context as input and is able to recognize and predict words with higher frequency.

Relevant Code

- Code can be found here (%3Chttps://github.com/chuandu2/cdp4205_msia414_2019/blob/homework2/Text_Analytics_Lab2_Sophie.pdf)

Problem 2

	Model Summary	**Learning Approaches**	**Key Contributions**	**Training Corpus Size**
Word2Vec	The training objective of Skip-gram model is to find word representations that are useful for predicting the surrounding words in a sentence or a document. By subsampling frequent words during training results in a significant speedup and improves accuracy of representations of less frequent words, as it aggresively subsamples words with high frequency while preserving the ranking of the frequencies.	First, we find words that appear frequently together, and infrequently in other contexts. Then, we use a simple data-driven approach where phrases are formed based on unigram and bigram counts, and the bigrams with score above the chosen threshold will be used as phrases. Another approach is to simply represent the phrases with a single token.	These representations exhibit linear structure that makes precise analogical reasoning possible. The subsamping of frequent words results in both faster training and better representations of uncommon words. Also, the Negative sampling algorithm is an extremely simple training method which learns accurate representations especially for frequent words.	Train Skip-gram models with phrase based corpus ~30 billion words

	Model Summary	Learning Approaches	Key Contributions	Training Corpus Size
BERT	There are two steps: pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For fine-tuning, BERT is first initialized with pre-trained parameters, and all parameters are fine-tuned using labeled data from downstream tasks. For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings.	To train a deep bidirectional representation, we simply mask some percentage of the input tokens at random, and then predict those masked tokens. If the ith token is chosen, we replace the ith token with (1) the mask token (80%), (2) a random token (10%), (3) the unchanged ith token (10%). Then Ti will be used to predict the original token with cross entropy loss. To train a model that understands sentence relationships, we pre-train for a binarized next sentence prediction task that can be trivially generated from any monolingual corpus. Then during fine training, for each task, we plug in the task-specific inputs and outputs into BERT and fine-tune all parameters.	BERT is effective for both fine-tuning and feature based approaches. A distinctive feature of BERT is its unified architecture across different tasks, which is a multi-layer bidirectional Transformer encoder. The input representation is able to unambiguously represent both a single sentence and a pair of sentences, where a ‘sentence’ can be an arbitrary span of contiguous text, rather than an actual linguistic sentence. BERT enables even low-resource tasks to benefit from deep unidirectional architectures.	Use BooksCorpus (800M words) and English Wikipedia (2,500 words) for BERT pre-training

	Model Summary	Learning Approaches	Key Contributions	Training Corpus Size
ELMo	ELMo word representations are functions of the entire input sentence. They are computed on top of two-layer biLMs with character convolutions, as a linear function of the internal network states, allowing to do semi-supervised learning. It is a task specific combination of the intermediate layer representations in the biLM. Vectors derived from a bidirectional LSTM is trained. The higher level LSTM states capture context-dependent aspects of word meaning while lower-level states model aspects of syntax.	First, we run biLM and record all of the layer representations for each word. Then we let the end task model learn a linear combination of these representations. To add ELMo to the supervised model, we first freeze the weights of the biLM and then concatenate the ELMo vector with Xk and pass the ELMo enhanced representation into the task RNN. For some tasks we also include ELMo at the output of the task RNN by introducing another set of output specific linear weights. Also, it is beneficial to add a moderate amount of dropout to ELMo and in some cases to regularize the ELMo weights by adding a parameter to the loss.	The pre-trained model is modified to support joint training of both directions and add a residual connection between LSTM layers. Including ELMo at the output of the biRNN in task-specific architectures improves overall results for some tasks and introducing ELMo at input layer allows the model to attend directly to the biLM's internal representations. ELMo enables to model both complex characteristics of word use (syntax and semantics), and how these uses vary across linguistic contexts.	Trained biLM on ~30 million sentences