Embeddings trained with selective attention

Dilip Thiagarajan
dt372@cornell.edu

Amar Shah
ahs268@cornell.edu

Abstract

Continuous bag-of-words (CBOW) models for language generally don't consider how significant words in the context are when examining the focus word (the word being predicted). While work has been done to alleviate this, both in training the embeddings themselves or in jointly training for a downstream task, the parametrization requires a dependence on relative position within the context window. We propose a variant of the CBOW model involving a kernel matrix that would explicitly learn relationships between the context and focus to emphasize selectively. We compare results between our model and the standard CBOW model (word2vec) on a variety of tasks, including word similarity (common words and rare words), sentiment analysis, and named entity recognition.

1 Introduction

In language modeling, tokens have gone through different eras of common featurizations - before, tokens would often be represented based on their n-gram probabilities, for some arbitrary n. With the re-emergence of neural networks and the corresponding advantages of their learning capacity, tokens are commonly featurized now using embedding models such as word2vec. The premise of these models is to determine a representation for each word that allows it to more accurately predict its context, whether it be with the continuous bag-of-words model (CBOW), which predicts the averaged context given the input word, or the skip-gram model, which predicts all of the words in the context.

In both of these models, each context word has the same weight in determining a quantitative representation of a word, which isn't ideal, especially when it comes to commonly used words such as stop-words and articles. To alleviate this, we want to use attention as an augment when learning these embeddings. This has been done to some extent in (2), but their approach applied attention based on the position of each context word with respect to the input word. We want to examine the improvement that can be obtained by looking at the specific word-pairs (input and context) rather than by position. To do so, we introduce a kernel matrix in the embedding space that would learn to attenuate the weighting of a word in the context window.

2 Related Work

2.1 word2vec

word2vec (1) introduced the idea of learning word embeddings based purely on the context of each word. In doing so, it introduced two different models - the continuous-bag-of-words model, and the skip-gram model. In the continuous bag-of-words model, the goal is to use the averaged representation of the context window $C(w)$ to predict the current word w, i.e.

$$\sigma\left(\frac{1}{2c}\sum_{w_c\in C(w)} f(w_c)\right) \approx w$$

Code can be found at https://github.com/dthiagarajan/self_attention_embeddings.

where w_c, w would be one-hot vectors corresponding to the index of each word in the vocabulary. The skip-gram model is similar, except instead of averaging over the context window, each context word is used as a separate example for learning the current word w. This model proved to be very effective for providing an initial representation for individual words for many tasks, such as sentiment analysis, automatic summarization, named entity recognition, translation, and more.

While the use of `word2vec` in further tasks comes at no extra cost, there are some concerns to be mentioned. First is the size of the context window to be used - using a small context window causes the model to lose out semantic relations it otherwise would have learned, while a larger context window can be more computationally expensive. Additionally, the only method for the representation for a word to become more similar to a semantically related word is for it to appear more times in the context of that related word. There is no notion of selective focus between words that occur infrequently in the overall vocabulary, but frequently in each others context.

2.2 Better Word Representations with Variable Attention

Variable attention (2) is an attempt at alleviating the lack of selective focus in the context windows for learning embeddings. To do so, the continuous bag-of-words model is extended by weighting the average of the context windows using an attention weight specific to each word type (unique word in the vocabulary). Explicitly, the goal now becomes:

$$\sigma\left(\frac{1}{2c}\sum_{w_c\in C(w)}\alpha_{i(w_c)}(w_c)f(w_c)\right)\approx w$$

where $i(c)$ is the index of w_c in the context window of w, and $\alpha_i(w_c)$ is the attention given to the word w_c at position i in the context window. The attention weights for each word type w defines a distribution for each possible relative position i in the context window, and the context window average is modified accordingly. The authors perform some experiments to qualify their claim that this model improves on more syntactically focused tasks, such as part-of-speech induction, while remaining competitive on more semantically focused tasks, such as sentiment analysis. Moreover, an added benefit of this model is that larger context windows can be used, as the attention weights would most likely attenuate positions further away from the center of the context window.

2.3 GLoVe

The GLoVe word embedding model (3) is based on the importance of word co-occurences with other words and believes that sliding window approaches lose the information carried by global counts for words and those outside the current reading frame while also discrediting traditional bag of words factorization models like LSA because they don't have any contextual weight. The happy medium proposed by GLoVe is to count co-occurences of words and store them in a matrix while doing a few tricks to optimize for 0 co-occurences and setting a regression style weight function to improve efficiency. The actual embeddings themselves are learned with a non linear hyperparameter α to help scale the regression function with scaling function f to avoid overweighting words like and, the and underweighing sparse co-occurences as such:

$$J=\sum_{i,j=1}^{V}f(X_{i,j})(w_i^T w_j-\log X_{ij})^2$$

where w_i and w_j are the word and context word vectors and X is the co-occurence matrix. This allows for efficient, but powerful word embeddings that incorporate both local context as well as global co-occurences of words when creating vectors and is commonly better than $O(V^2)$ runtime when $\alpha = 1.25$ and keeps it competitive with online algorithms. GLoVe was intended to perform well on the word analogy task due to the impact of the co-occurence matrix, but it has relatively good performance on other tasks like word similarity and named entity recognition that's on par with CBOW.

2.4 fastText

fastText (4) introduces an extension of the classic word2vec model, which performs well but doesn't explicitly model word morphology. Specifically, the authors propose a character n-gram model, where each character n-gram is represented by a vector, and the vector for a word is the sum of the n-grams that consist it.

The extension is based on the skip-gram model: given a sequence of words $w_1, ..., w_T$, the objective of the model is to maximize the probability

$$\sum_{i=1}^{T} \sum_{c \in context(w_i)} \log p(w_c | w_i)$$

which is done essentially using a vector lookup layer and a softmax layer to get the probabilities. The problem can also be framed as several binary classification tasks for each context position c, known commonly as negative sampling. The crux of the extension is to represent each word as a bag of character n-grams, as well as special characters for representing prefixes and suffixes generally. Interestingly, the whole word is also included (with these special symbols pre/appended). Accordingly, the vectorizations for all these n-grams are used in the summation of the loss function as an additional sum within the sum over context words. This language model was then evaluated on various tasks such as word similarity, and word analogies.

3 Model

3.1 Specification

Our model implements a variant on the variable attention embedding model (2), but instead of using position as a key for the attenuation of a context embedding, the following modeling assumption is employed: given w as the one-hot vector trying to be learned (i.e. the centering of the context window), and $C(w)$ being the set of one-hot vectors corresponding words w_c in the context window of w, and $f(w)$ being the embedding for w

$$\left(\frac{f(w_c)^T K f(w)}{\sum_{w_c \in C(w)} f(w_c)^T K f(w)} w_c \right) \approx w$$

Written more succinctly, the context embeddings are attenuated using the the softmax of

$$f(w_c)^T K f(w)$$

for every word in the context where K defines a kernel for the context embeddings in the embedding space. This matrix K should be indicative of semantic relationships based on the embedding space after training on the training data.

3.2 Parameter Selection and Tradeoffs

We chose a context size of 2 words to be similar to `word2vec` as we want enough context about the frame to avoid assuming the context from just 2 words total while not losing the importance of having context in the first place with a window too big. This choice is made on the basis of training efficiency but given enough time we'd like to further study how this new kernel choice contributes to understanding underlying semantic meaning. Having 4 words total does increase the computational run-time but we believe the extra contextual background is worth the trade off and any more might be overkill.

We chose a batch size of 256 for the interim because we wanted to maximize how quickly we could train given the limited access to GPUs at the moment and would like to play with smaller batches when more resources become available. A larger batch means we may not get the optimal results as a result of SGD; however, we can still get a reasonable benchmark for now and optimize later.

Embedding dimension is one of the most important parameters and we chose 50 as a baseline. Although having an embedding size above 100 or 200 would be very costly and might not provide any more context we think starting here is a safe bet. The kernel matrix K mentioned above also has parameters quadratic in the embedding dimension, sow e believe that increasing embedding dimension will aggressively overfit, so we expect 50-200 to be optimal for learning.

Learning rate is a common hyperparameter that can be effectively tuned for training neural networks. However, given the time constraint, we choose to do 5 epochs of training (as done in `word2vec`) using SGD with a learning rate of 0.001 for fair comparison.

Model, Epoch, Embed Dim	50 Dim	100 Dim	200 Dim
Sel 0	6.99	6.30	6.47
Sel 1	6.97	6.21	6.86
Sel 2	7.00	6.33	7.19
w2v 1	7.20	6.31	6.29
w2v 2	7.17	6.25	6.53
w2v 3	7.22	6.26	6.48

Table 1: WordSim-353 (MSE, 5163 examples)

Negative sampling is something that we can also include to optimize how the model learns. The premise of negative sampling is to subsample word embeddings that are affected by backpropagation if they are not involved in the context of the current word. The `word2vec` chooses to subsample words by assigning them probabilities approximately proportional to their word frequency, so that words that are more common are more frequently excluded from this backpropagation consideration, while words that are less common are less likely to be excluded. This is reasonable, given that a less frequently occurring word is more likely to be semantically significant, so its absence in a context window attributes more semantic meaning than if a more common word is absent from a context window.

4 Data and Methods

Initially, for all downstream tasks, our model will be compared with the `word2vec` model as a frozen featurization of the tokens being considered. Accordingly, each task essentially becomes learning a linear classifier/regressor on top of this featurization. To compare our embedding model with standard models such as `word2vec`, we will first evaluate our models on word similarity tasks such as WordSim-353 and the Stanford Rare Word Similarity dataset. We will then use our word embeddings as an initialization for various other tasks, such as sentiment analysis (which can be evaluated using datasets such as the IMDB movie review dataset, or Senti), named-entity recognition (with the CoNLL-2003 dataset), and part-of-speech tagging (using Penn Treebank English data).

For dealing with unknown words, we took the most infrequently occuring n-gram in each unknown word (where $3 \leq n \leq 7$) and used that as a substitute for our word. This methodology and the choice of possible values of n were arbitrary, and we found that changing the value of n to be much larger had minimal differences in the downstream tasks (including rare word similarity, where this would likely be the most significant choice).

Given that these methods don't leverage the kernel matrix of our model, the experiments we will conduct after that will be to examine how much information is added when using the kernel matrix. Accordingly, we will concatenate features to the embedding featurization for each token using the kernel matrix. This will vary from task to task, and will be detailed in the results section accordingly.

5 Results

The results we have obtained thus far are shown below. Specifically, the results on the word-similarity task WordSim353 can be found in Table 1, with the mean-squared errors (MSE) for all models trained for 3 epochs shown. In Table 2, we show the MSE for all models on the rare word similarity task provided by Stanford. In Tables 3 and 4, we show the accuracy for all models on the IMDB sentiment analysis dataset, with Table 3 showing the accuracies on negative and positive examples separately, while Table 4 shows the overall accuracies on all examples.

6 Discussion

We separate the discussion of our results by task before returning to an overall evaluation.

Model, Epoch, Embed Dim	50 Dim	100 Dim	200 Dim
Sel 0	11.02	11.99	15.69
Sel 1	10.99	11.97	15.78
Sel 2	11.00	11.96	15.86
w2v 1	10.90	12.07	15.48
w2v 2	10.93	12.04	15.67
w2v 3	10.92	12.10	15.51

Table 2: Rare Word Similarity (MSE, 22095 examples)

Model, Epoch, Embed Dim	50 Dim	100 Dim	200 Dim
Sel 0	(0.122, 0.927)	(0.221, 0.858)	(0.277, 0.855)
Sel 1	(0.093, 0.931)	(0.206, 0.893)	(0.277, 0.837)
Sel 2	(0.219, 0.834)	(0.151, 0.889)	(0.286, 0.834)
w2v 1	(0.151, 0.907)	(0.206, .834)	(0.177, 0.872)
w2v 2	(0.141, 0.917)	(0.273, 0.817)	(0.235, 0.824)
w2v 3	(0.225, 0.830)	(0.215, 0.851)	(0.254, 0.855)

Table 3: Sentiment Analysis (Neg/Pos Accuracy)

6.1 WordSim-353

In this task, we believed that selective attention as a framework would show separation from standard `word2vec` embeddings, as training the original embeddings would weigh words with better underlying relations more in the altered CBOW model. Practically, we believed that would translate to stronger weighting to word pairs such as 'porcupine' and 'sharp', as they would likely appear in similar contexts in each of their occurrences.

However, empirically, the results are quite comparable, as can be seen in Table 1. Furthermore, it seems that `word2vec` tends to outperform our model in higher dimensions. We believe this occurs because the kernel matrix jointly trained when learning the embeddings is not used during this downstream task, and the higher dimensional embeddings might rely on the information capacity of this kernel matrix for tracking relationships in the latent space of the embeddings.

Moreover, specific to the problem, we find that the selective attention model tends to identify pairs of nouns as being similar. It's possible that this occurs due to the nature of the altered CBOW model we use to train the embeddings in the selective attention model - infrequent words are possibly emphasized more if they appear in context or as focus word.

6.2 Rare Word Similarity

While the rare word similarity task and the generic word similarity task are very similar in nature, the key difference in the execution of learning in these tasks lies in how we decided to handle unknown words. Given that this methodology was used for handling unknown words with both `word2vec` and our model, the results are comparable, with a mean error of around 3 to 3.5. Therefore, this task was hard to evaluate differences in performance, but given that the kernel matrix is still not being

Model, Epoch, Embed Dim	50 Dim	100 Dim	200 Dim
Sel 0	0.510	0.528	0.555
Sel 1	0.497	0.537	0.547
Sel 2	0.515	0.507	0.550
w2v 1	0.515	0.508	0.512
w2v 2	0.515	0.535	0.518
w2v 3	0.517	0.522	0.543

Table 4: Sentiment Analysis (Overall Accuracy)

leveraged for training for the downstream task, the subword information being used is very minimal. Therefore, an adaptation that could work for future work involving our model and this task could be to use the bag-of-words methodology used in (5).

6.3 Sentiment Analysis

Looking at Table 3, we see that both models are atrocious at learning to classify negative examples, but perform reasonably well with positive examples. Given that the classes are balanced in the dataset used for this task, the model is definitely not predicting just positive all the time, so it's difficult to say why this is occurring. One hypothesis for this is that because stop-words and common words were not removed from reviews, they caused noise during the training of the model, resulted in predictions that were close to ambiguous between positive and negative examples. Upon inspecting the class probabilities, this seems to be the case - for positive examples, the class probabilities are very close to 1 for positive and 0 for negative, but for negative examples, both probabilities are very close to 0.5. One simple way to alleviate this would be to remove those words, but another would be to weight on infrequent words occurring in the original vocabulary.

Another explanation for why this might be occurring is due to the higher frequency of proper nouns related to the movies occurring in these reviews, and, due to how we are handling unknown words, it's possible that we are accidentally picking up an n-gram from these words that are completely unrelated to the context of the given sentence. An easy way to handle for this is to handle unknown words differently, by just removing them from consideration. Given that the review vocabulary set is probably similar to the vocabulary set of training, we'd expect that this would not drastically degrade the results for this task.

6.4 Overall Evaluation

From examining these downstream tasks, one problem we noticed, but did not expect, was how this novel attention mechanism seems to pay more attention to words with similar embeddings. By not including the kernel matrix in any of the downstream tasks, it becomes difficult for the model to factor in these contextual clues. Accordingly, future work involving improvements to this model will involve including of the kernel matrix, and later jointly training the kernel matrix and the embeddings for the downstream tasks. We've tried some initial experiments involving inclusion of the kernel matrix, but we were having some trouble with exploding gradients, so we weren't able to have some initial results regarding that in time for the submission of these results.

Another issue we noticed in all these tasks, but was more reasonable to expect, was the resolution of unknown words. For future work, we will try to incorporate the CHARAGRAM model, or something related, to alleviate this issue. One way to examine how well subword information is being realized in our model would be to inspect the embedding space, while also looking at the most similar words for words with significant stems composing the word.

Finally, one big reason for the difficulty including the kernel matrix in downstream tasks is due to the amount of matrix multiplications that would be required for training and inference in those tasks. Training cycles for our model's embeddings took 9 hours per epoch for the 50-dimensional embedding, to 14 hours per epoch for the 200-dimensional embedding. This was not practical given the duration of our project, so for future work, we will experiment with optimizations concerning the kernel matrix, including:

- Treating the kernel matrix as an outer product of two d-dimensional vectors. This can further be realized as a context lookup and a focus lookup, similar to the treatment used in Transformer networks (6).
- Using the embedding w as well as the kernel matrix multiplied with w as an additional featurization describing the word more thoroughly in the latent space.
- For tasks inferring from a variable number of words, using a softmax style weighting of the quadratic forms of each of the words. Given the focus word w, and all other words w_o, take the softmax of xKw for $x \in w_o$, and use that as a weighting for w in whatever combination is used for inference downstream.

References

[1] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. & Dean, J. Distributed representations of words and phrases and their compositionality. In Proc. Advances in Neural Information Processing Systems 26 3111–3119 (2013).

[2] Wang Ling, Lin Chu-Cheng, Yulia Tsvetkov, Silvio Amir, Ramon Fernandez Astudillo, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015a. Not all contexts are created equal: Better word representations with variable attention. In Proc. EMNLP.

[3] Pennington, Jeffrey, Richard Socher, and Christopher Manning. ”Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

[4] Bojanowski, Piotr, et al. "Enriching word vectors with subword information." arXiv preprint arXiv:1607.04606 (2016).

[5] John Wieting and Mohit Bansal and Kevin Gimpel and Karen Livescu, Charagram: Embedding Words and Sentences via Character n-grams, CoRR, abs/1607.02789, 2016

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in Advances in Neural Information Processing Systems, 2017, pp. 6000–6010.