Word2Vec using Character n-grams

Deepti Mahajan[1], **Radhika Patil**[2], **Varsha Sankar**[3]
Stanford University, CA 94305
dmahaj@stanford.edu[1], radhikap@stanford.edu[2],
svarsha@stanford.edu[3]

Abstract

This paper outlines an approach to improving upon the word2vec skip-gram model. In order to obtain better word representations for morphologically rich languages and to yield more accurate results for rare or unseen words. A word representation is derived from the vector embeddings of its constituent n-grams, which are updated and learned in the training process. We train two models on an English corpus: one with 3-gram embeddings, and another with 2- to 6-gram embeddings. We evaluate the models using word similarity and word analogy tasks and comparing to a word2vec skip-gram baseline.

1 Introduction

The word2vec model, introduced by Mikolov et al.[1], has led to significant improvements to natural language processing (NLP) task performance. However, there is room for improvement in the word2vec word representations. In particular, it is difficult for good representations of rare words to be learned with traditional word2vec. In addition, there could be words in the NLP task that were not present in the word2vec training corpus. The traditional model would end up using the same vector ('UNK') for all the unseen words even if they are not related. This limitation is more pronounced in case of morphologically rich languages (like German, Turkish or Finnish) as they have a lot of rare words. Thus learning vector representations of subwords, or character n-grams, and incorporating them into the word vector representations would be beneficial as it takes into account the structure of the words. This report outlines our work in implementing the n-grams enhanced word2vec using skip-gram approach for English words. After obtaining modified word embeddings using different combinations of the constituent n-gram embeddings which were learned by the model, we evaluate the embeddings by intrinsic methods of word similarity and word analogy. The results are analyzed and compared with that of conventional skip-gram model baseline.

2 Related work

Recently, information about character subsequences of words are being incorporated into the word vector representations for improving its performance in a lot of applications. A recent paper by researchers at Facebook AI (Piotr Bojanowski, Edouard Grave, et al.) used the approach of learning character n-gram representations to supplement word vector accuracy for five different languages to maintain the relation between words based on their structure.[2] In a paper by Google, Neural Machine Translation of rare words is performed using subword units obtained by segmenting words using the byte-pair encoding compression algorithm. Google's Neural Machine Translation System divides words into limited set of common subword units ("wordpieces") to handle the translation of

rare words. It is reported to provide a balance between the flexibility of character-delimited models and the efficiency of word-delimited models and improves the overall accuracy of the system.[3] Another work on incorporating morphological information into word representations by Alexandrescu and Kirchhoff (2006), introduced factored neural language models, where words are represented as sets of features.[4]

3 Approach

In this section we discuss about the models used for word vector learning, the datasets used, and the overall procedure that we followed.

3.1 Model

The word vectors were trained first by using the traditional skip-gram model. Then word vectors using character n-gram information was learned. The models are described below.

3.1.1 Skip-gram Model

The continuous skip-gram model learns vectorial representations for each word w belonging to a given vocabulary {1,...,W}. The representations are trained to predict words appearing within the context window of a given center word. So given a large corpus of training data, which can be thought of as a sequence of words w_1, w_2, ... w_T, the skip-gram model maximizes the log-likelihood, i.e, the probability of a context word given a center word. This objective is given by;

$$\sum_{t=1}^{T} \sum_{c \in C_t} \log p(w_c|w_t) \tag{1}$$

where, C_t is the context window around the center word w_t. The probability of the context word given the center word is usually softmax over the scoring function as seen below;

$$p(w_c|w_t) = \frac{e^{s(w_t,w_c)}}{\sum_{j=1}^{W} e^{s(w_t,j)}} \tag{2}$$

Here, s is the scoring function, which is the dot product of the word vectors of the center and the context word.

The objective function that was minimized during training was given by the Noise Contrastive Estimation (NCE), denoted as J_θ. The equation for the loss is shown below, where Q is a noise distribution from which k noise samples are generated.

$$J_\theta = -\sum_{w_i \in V} log \frac{p(w_i|c)}{p(w_i|c) + kQ(w_i)} + \sum_{j=1}^{k} log \frac{p(\tilde{w_{ij}}|c)}{p(\tilde{w_{ij}}|c) + kQ(\tilde{w_{ij}})} \tag{3}$$

3.1.2 Character n-gram based model

The basic skip-gram model described above ignores the internal structure of the word. However this model incorporates information about the structure in terms of character n-gram embeddings. Each word vector is obtained by summing up the vector learned for that word and the embeddings of the n-grams that make up the word. The scoring function for this model is given by;

$$s(w, c) = \sum_{g \in G_w} \vec{z}_g^{\top} \vec{v}_c \tag{4}$$

G_w is the set of all n-grams present in word w and z_g corresponds to the vector associated with the n-gram g, while v_c is the word vector of the center word c. Thus by using this scoring function we learn the vectors associated with all n-grams as well as the words simultaneously.

3.2 Datasets

The text8 dataset was used for training. It consists of 100 MB (17 million tokens and 254 thousand unique words) of cleaned English text taken from Wikipedia articles. Of these, a vocabulary of 50,000 was built.[5]

For word similarity evaluation, the WordSimilarity-353 Test Collection was used. The dataset consists of a set of 353 English word pairs. It also includes mean word similarity scores obtained from human subjects.[6]

For word analogy evaluation, a dataset containing both semantic (e.g. France::Paris, Italy::Rome) and syntactic (e.g. small::smallest, large::largest) analogy pairs was used. The dataset was obtained from Googles word2vec code archives and contains 15,851 questions.

3.3 Procedure

3.3.1 Building word and n-gram vocabulary

From our training dataset, we generated a word vocabulary of 50,000 most frequent words while considered any other word as 'UNK' (Unknown). The n-gram vocabularies would each contain 26^n entries, which is a reasonable number for $n = 2, 3$. For higher values of n, the size of the vocabulary becomes too large and undesirable. Moreover, some of these n-grams may not occur in meaningful words at all. Thus we fixed the number of entries in the n-gram vocabularies (for $n > 3$) to be 20,000 and considered the most frequently occurring n-grams found in the words in our word vocabulary, as shown in Figure 1).

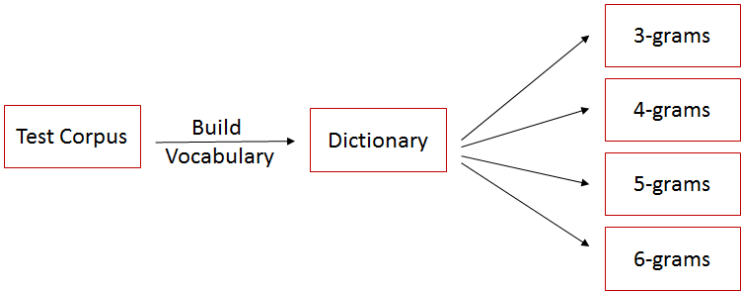

Figure 1: Converting words to character n-grams

3.3.2 Modifying the skip-gram model

To train the embeddings associated by the character n-grams, we mapped each word in the vocabulary to its constituent n-grams (For $n = 2, 3, 4, 5, 6$). We fixed the number of n-grams for each word to be 20 for each value of n in order to use matrix operations in our model. If a word had less than that many n-grams, then we did zero padding. On the other hand, we considered only the first 20 n-grams if it had more constituent n-grams. These mappings were computed and stored as a preprocessing step to speed up the process of training the actual embeddings.

In the model implementation, we considered the embedding matrices corresponding to each of the n-grams which are of size, number of elements in n-gram vocabulary by the embedding dimension. Apart from this, we also considered a word-only embedding matrix of size [vocabulary size, embedding dimension]. As a mini batch of inputs and labels were passed during each run, their mappings to each of the n-grams was also passed. Based on these mappings, the embedding vectors of all constituent n-grams were summed with the word-only embedding to form the required word vector representation, as shown in Figure2. This is then passed as an argument to the NCE loss function.

As the Optimizer (Stochastic Gradient Descent) minimizes the loss, all of these embedding matrices are updated after each run by backpropagation. After training for steps in the order of 100,000 the loss goes down considerably and the embeddings are saved for post processing.

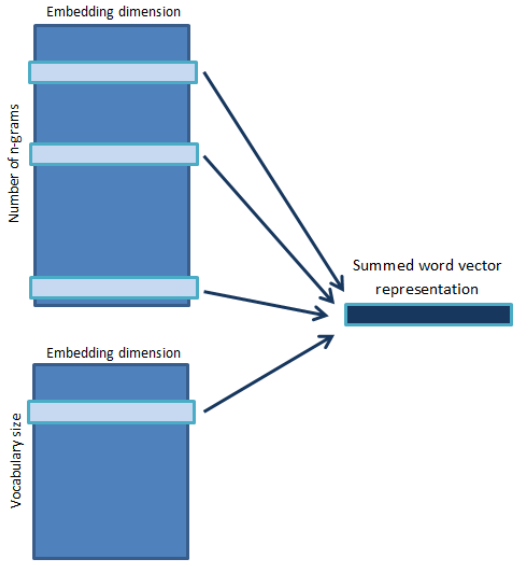

Figure 2: Computation of a word's vector representation from the n-gram (top) and word (bottom) embedding matrices

In the post processing, the final word embeddings are obtained by summing all the constituent n-gram embeddings and stored for easy lookup. Finally, these word vectors are evaluated by different methods.

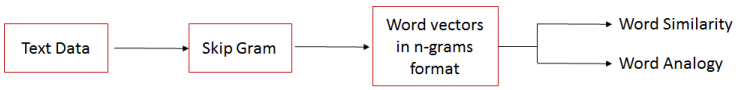

Figure 3: Model training and evaluation

4 Experiments and Results

We initially considered just 3-grams for each word and combined them with the word embeddings to check the performance. Finding that considering the subword structure caused improvement with the word similarity task, we moved on to try the all-gram approach which considered character n-grams for n = 2 to 6.

The graph in Figure4 shows the learning curve of the word2vec skip-gram model, with a final NCE loss of 4.63. The 3-gram and 2- to 6-gram models also had similarly converging learning curves.

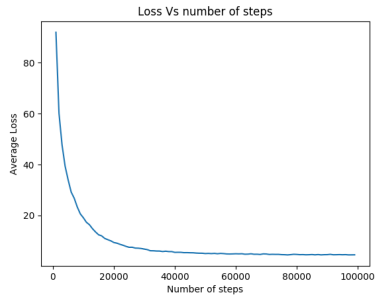

Figure 4: NCE loss curve of the word2vec skip-gram model

4.1 Intrinsic Evaluation

We evaluated our models using intrinsic evaluation–tasks which do not test a specific application of the resulting word vectors, but assess how well they capture word meaning. We do this using word similarity (finding a word's nearest neighbors) and word analogy (both semantic and syntactic) tasks.

4.1.1 Word similarity

In Word similarity based evaluation, the word embeddings are used to measure the similarity of a given pair of words. We calculate this as the cosine similarity of the two vectors. As we are using the WordSIM-353 dataset, in order to compare our similarity values with that of the human assigned scores, we scaled up the cosine similarity to a scale of 10 and then computed the Spearman's Rank Correlation coefficient between the two.

We also tried finding the top few similar words given an input word. Results for one such words by using the different models is shown below;

Example skip-gram similarity groupings:

Nearest to american: dasyprocta, positions, encyclopedia, nonstandard, slain, fostered, city, arbor

Example 3-gram word similarity groupings:

Nearest to american: americana, america, americans, americas, americo, erica, rican, african

Example 2- to 6-gram word similarity groupings:

Nearest to american: americana, america, americans, americas, americo, erica, comerica

Thus, the character n-gram based models performed better than the basic skip-gram model, and also captured the structure of the word (as evident from the output).

4.1.2 Word analogy

The word analogy accuracy was computed based on 12,158 questions–those containing words in our 50,000 word vocabulary. The questions were of form a:b::c:?. For evaluating our vectors, we obtained the difference between the vectors of a and b and added it to that of c. The cosine similarities between the result and all word embeddings were computed to find the most likely answer. If this word was same as that of d (the actual answer), a match was reported. The same was performed for all question-answer pairs and the percentage of matches was reported as the accuracy.

Table1 contains the quantitative results of word similarity and word analogy for the two models compared to the skip-gram baseline.

Table 1: Intrinsic evaluation results

Model	Word Similarity Correlation	Word Analogy Accuracy
word2vec skip-gram	22.32	34.2%
3-gram model	23.42	41.4%
2- to 6-gram model	25.14	43.1%

To quantify the effect of the word and n-gram based embeddings, we weighted them differently while summing to form the final word vector, and observed its performance of the resulting vector. The improvement in performance was only marginal and thus we haven't included it. However, a potential extension could be to make the model learn the optimal weights to associate with each of the n-gram embeddings and the word vector itself.

5 Conclusion

Our evaluation results indicate that the n-gram enhanced skip-gram models performed slightly better than regular skip-gram in the word similarity and word analogy tasks. However, it is worth noting that the skip-gram results are lower than the benchmark found in the paper by Bojanowski et al. [2] In fact, the skip-gram word2vec, when evaluated on the same similarity and analogy datasets, yielded a Spearman's correlation of 65 and a word analogy accuracy of 28%. Our model's loss did converge, as shown in Figure 4. Thus, we believe the discrepancy is due to the fact that their model was trained on a dataset of 50 million tokens. Our smaller dataset of 17 million may have led to suboptimal results. Since the addition of n-grams embeddings produced an improvement on these results, we can conclude that n-gram enhanced skip-gram can be useful for training over a limited amount of data. We can analyze this further by looking at the example word similarity groupings (in Section 4.1) for the word 'americans'. One of the skip-gram's nearest neighbors results were vaguely related ('city'). Both of the n-gram models yielded better results, which may have been due to the clustering of similarly-spelled words.

6 Future work

Our model adds value to word vector representations by incorporating a sense of meaning association with character n-grams along with words. In the future, the following could be implemented for betterment of the model.

6.1 Dataset size

Our results indicate improved performance in the 3-gram and 2- to 6-gram models from regular word2vec skip-gram. The relatively poor performance of word2vec may be explained by our small dataset (of 17 million tokens). In order to obtain optimal word2vec results and more accurately assess the performance of the character n-gram model, we would train on much larger datasets–at least 50 million tokens. This should, in theory, drastically improve the character n-gram results as well. In the text8 dataset, there was a higher chance of certain n-grams not existing in the training set. Thus, the vector representations for those n-grams were entirely dependent on the random initialization–which probably hurt performance. With a larger dataset, there would be a greater likelihood of most n-grams being updated in the training process.

6.2 Languages

We have performed training and evaluation only on English words. Considering that English does not have a lot of compound or combination words, while languages like German, Turkish, Finnish or Dutch have compound words as a regular feature. In order to assess the effect of our model on compound words, we would train our model on German tokens.

6.3 Extrinsic evaluation

While intrinsic evaluation allows us to observe whether our learned embeddings can capture some word meaning, it would be beneficial to also use the trained vectors for NLP tasks, such as sentiment analysis or named-entity recognition.

7 Team member contributions

The three group members contributed equally towards developing the model approach, project decision-making, and putting together the poster and paper. In particular:

- Deepti Mahajan contributed towards building the n-gram enhanced skip-gram model and wrote the word analogy evaluation code.
- Radhika Patil wrote the code that constructed the n-gram embedding matrices, found the character n-grams of each word, and mapped the word IDs to their corresponding n-gram IDs.

- Varsha Sankar contributed towards building the n-gram enhanced skip-gram model and wrote the word similarity evaluation code.

Acknowledgments

We would like to thank Professor Manning and Dr. Socher for a very interesting course, as well as our project mentor, Kevin Clark, for his advice.

References

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. ICLR Workshop, 2013.

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606, 2016.

[3] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, . Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Googles Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. ArXiv e-prints, 2016.

[4] Andrei Alexandrescu and Katrin Kirchhoff. 2006. Factored neural language models. In Proc. NAACL.

[5] Mahoney, Matt. "About The Test Data". Mattmahoney.net. N.p., 2011. Web. 20 Mar. 2017.

[6] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin, "Placing Search in Context: The Concept Revisited", ACM Transactions on Information Systems, 20(1):116-131, January 2002.