Learning Better Word Embeddings with Morphological Knowledge

Ayush Jain
260673423, McGill University
ayush.jain@mail.mcgill.ca

Abstract

Word embedding were first introduced by (Yoshua Bengio et al., 2001) more than a decade ago, but are still one of the most exciting area of research in deep learning. It is one of the best places to gain intuition about deep learning's effective. Most word embedding approaches treat each word as an independent entity and fail to capture the explicit relationship among morphological variants of a word. Complex morphological derivatives are often represented poorly. Morphemes, being the smallest unit of meaning, can be used to overcome these shortcomings. In this report, I describe my observations from experiments on a neural network model using morphological knowledge. This model simultaneously learns word and morpheme embeddings, and derives better representation for rare words.

1 Introduction

Vector based word representations is a key sauce for success of many NLP systems in recent years, across tasks including named entity recognition, part-of-speech tagging, parsing, and semantic role labeling. Vector based representations find usage in deep neural network models and conventional feature-based models alike. Deep learning systems give each word a distributed representation, i.e., an embedding. Such distributed representation over word classes capture various dimensions of both semantic and syntactic information in a vector where each dimension corresponds to a latent feature of the word. As a result, a distributed representation has lower dimensionality, less susceptible to data sparsity, and can implicitly represent an exponential number of word clusters.

Despite the widespread use of word embeddings, most of the approaches found in literature treat each full-form word as an independent entity and fail to capture the explicit relationship among morphological variants of a word. Often rare words are composed of fairly common morphemes and still not using this fact for representation of such words is a serious shortcoming. The fact that morphologically complex words are often rare exacerbates the problem. For e.g. most of embeddings represent frequent words, such as distinct, well; they often badly model rare morphological derivatives like distinctiveness.

There are two important problems with rare words. First, it is difficult to obtain word embeddings for emerging words as they are not included in the vocabulary of the training data. Second, the embeddings for rare words are often of low quality due to the insufficient context information in the training data.

Fortunately, we might have a solution for problems pointed out above. Words which are semantically or syntactically similar often share some common morphemes such as roots, affixes, and suffixes. Earlier e.g. of distinct and distinctiveness share the same root. Morphological information coupled with previous word representation methods can bridge the gap between rare words and well-known words in learning word representations.

For this project, I replicate model proposed in Siyu Qiu et al. in (Siyu Qiu et al., 2014) using different methods and apply those resultant models to a new data set. Evaluations are based on Googles analogical reasoning task. Proposed neural network architecture incorporates morphological knowledge to create high-quality word embeddings. This model had two-fold advantages: first, it uses three types of co-occurrence information: word-word (conventional), word-morpheme, and morpheme-morpheme; second, since this new model learns word embeddings and morpheme embeddings simultaneously, it is convenient to learn representations for rare words from morpheme embeddings.

For the model, I segment words in the training data into morphemes, and then employed one hot encoding representations of words and their morphemes as the input to the neural network models. In the back propagation stage, the word embeddings, the morpheme embeddings, and other coefficients of model parameters are updated simultaneously.

The rest of this report is organized as follows. I briefly describe data set I used and pre-processing steps done over it in Section 2. In Section 3, I explain the methodology and models I implemented. Results are reported in Section 4. Lastly this work and future work are discussed in Section 5.

2 Data set and pre-processing

I used enwiki8 data set for training this model. This data set contains about 1.25 million words and represent the first 10**8 bytes of wikipedia. Data is freely available for download at (0). The data is UTF-8 encoded XML consisting primarily of English text. enwik8 contains about 25,000 articles and has 12.5 million words.

The data contains some URL encoded XHTML tags. However, hypertext links have their own encoding. External links are enclosed in square brackets in the form [URL anchor text]. Internal links are encoded as [[Wikipedia title — anchor text]], omitting the title and vertical bar if the title and anchor text are identical. Non-English characters are often coded directly as a UTF-8 byte sequence.

I used Matt Mahoneys text preprocessing script (0) to process the corpus. This script removed non-Roman characters and mapped all digits to English words. After this, punctuation symbols are processed out and every word in converted to lowercase. In addition to this, words occurring only once are discarded as learning meaningful representation from just one instance is very difficult.

3 Methodology

This section explains the architecture of model I implemented. It is heavily inspired from the CBOW model. In CBOW, a model aims to predict the central word using the surrounding words from a window sliding over the training data set. Words from corpus are represented in fashion similar to one hot encoding. For the forward pass, these input words from sliding window are mapped into the embedding space by the same weight matrix M, and then the embedding vectors are summed up to a combined embedding vector. Finally, summed embedding vector is decoded back to V-dimension space and the resulting vector is used to predict the central word. For back-propagation pass, error cost is propagated back to the network to update two weight matrices and other model parameters.

This model, while based on CBOW, has a modified input side - input layer accepts words and morphemes. To sum it up differently, vocabulary used by this model for the 1-of-v representation comprises words and their morphemes.

3.1 Morphological Analysis

Significant portion of this implementation is getting meaningful morphemes from such a large corpus of words. Since there are over 1.25 million words for the task, manual tagging of words is out of question. I was looking for a unsupervised model which could give high quality morphemes.

I finally ended up using Morfessor 2.0 (0), a public tool implemented based on the minimum description length algorithm (M. Creutz and K. Lagus, 2007). Morfessor 2.0 is, in principle, applicable to any string segmentation task. The task of the algorithm is to find a set of constructions that describe the provided training corpus efficiently and accurately.

For morphological task on enwiki8 data set, I used viterbi based algorithm to learn possible morphemes and then minimize cost function over it. Morfessor 2.0 gives fairly well separated morphemes but still segmented a few common words into completely wrong morphemes.I browsed over tagged data set to remove some errors.

3.2 Modified CBOW based Neural Network Model

On the input side, both the words and their morphemes are mapped into the embedding space by the same weight matrix. This matrix transforms the inputs from one-hot encoding (V-dimensional space) to embedding space (D-dimensional). Here V is size of the vocabulary and D is desired dimensionality of word embeddings. Similar to CBOW approach, then the weighted sum of the combination of word embeddings and the combination of

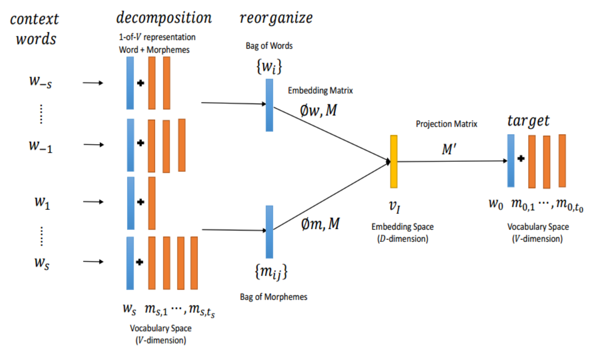

Figure 1: Final Neural Network Model

morpheme embeddings is calculate. Word embeddings and morpheme embeddings have different weights for the summation task.

On the output side, we transform the combined embedding vector back to one hot encoding (V dimensions) by another weight matrix and then predictions are made using softmax non-linearity to minimize error cost. Softmax gives the probability of word which maximizes the output prediction probability.

Error from the loss function is back-propagated throughout the network updating both weight matrices and model parameters. Convergence on the training data set gives the first weight matrix as learned word and morpheme embeddings.

Figure 1 from (Siyu Qiu et al., 2014) graphically explains model architecture.

3.3 Negative Sampling

Given that final model predicts the central word from the complete vocabulary, the computation cost for optimization term is proportional to the size of vocabulary and training time is too large. References to negative sampling approach can be found in related literature (T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, 2013) which speeds up the computation cycle. Instead of going over complete vocabulary, some fixed amount of random negative samples are used for each prediction task. These negative samples should be different for each entry from the vocabulary. Negative sampling converts the complete computation task to linear with the number of the negative samples.

3.4 Final Model

I tried two different ways to implement this model. First approach was to modify word2vec inputs and try to learn representation for morphemes along with words. Modified inputs combine words in the context window of size 5 and their corresponding morphemes. Since each word can have a different number of morphemes, final input is padded with a constant dummy word. Padded inputs are then forwarded to word2vec to learn on.

In this implementation, words and morphemes have same weights unlike described earlier section. This is because of the fact that word2vec doesn't allow for weighted summations.

Second approach I tried was of implementing a neural language model from scratch. This implementation is created on top of Theano library and also uses Lasagne framework (0) for creating neural layers. Initial input layer is present to accept input in vector form. This layer then converts inputs to one hot encoding format and passes it onto a hidden layer. This layer represents the initial weight mapping which transforms the V-dimensional input to the smaller D-dimension embedding space. There are V x D parameter in this layer to be learned. Word embedding vectors of inputs are then weighted summed. This summation gives vector representation of the central word. Next this embedding is converted back to V-dimensions and then a softmax non-linearity converts this into a word which maximizes the probability. Conversion on this architecture should give us the first weight matrix as a good word embedding map. Provisions are made for different weights for input embedding summation are implemented and these two parameters can even be learned the course of training.

Despite implementation seeming to be correct for second implementation, I'm not able to compile the theano evaluation function for the second layer of this network. First layer successfully generates word-embeddings but second layer receives null input from first. After two different attempts at writing this implementation from scratch - first using theano and second with lasange framework abstraction for layers, I can't seem to find the issue.

N.B. Implementation for first method using word2vec is working, small issues with second implementation tried from scratch.

4 Results

Performance of this model was evaluates over analogical reasoning task. Introduced in (T. Mikolov, K. Chen, G. Corrado, and J. Dean, 2013), this task has questions in the form a is to b is as c is to ? Assuming corresponding vectors to be a, b, and c, we will answer the question by finding the words with representation having the maximum cosine similarity to vector b a + c.

For evaluation purpose, I only considered tests where all the words are present in the vocabulary of my model, since the corpus I trained this model on, wiki8, is small. Also, simple word2vec trained only on word clusters is treated as baseline.

As can be seen in table, accuracy is improved in comparison to baseline model.

Table 1: Accuracy

Methodology	Accuracy
Baseline (vanilla word2vec)	19.80
morpheme2vec	22.27

Even without the results from second approach (logically should give better results), above results suggest morphological knowledge helps with word representation.

5 Discussion and Future work

The model I implemented is a novel neural network model to learn better word representations from text. The model leverages morphological information and uses different correlations to produce high-quality word embeddings, especially for rare words. Empirical experiments on an analogical reasoning task has shown better results than vanilla word embeddings.

- Data set I used for this training is relatively small compared to word corpus being used by other state-of-the-art models.

- Morphemes extracted were not accurate. Morfessor tried to separate words without any seed set or prior knowledge. Morphemes truly extracted based on word roots and affixes should fair a lot better.

New words being invented usually wouldnt be present a lot in Wikipedia corpus. News articles, social networking and blog post are more likely to have data. As a next step, it should be interesting to explore how this model performs on this kind of noisy data corpora.

English is limited in terms of morphology, and there is little for this model to learn on. This model should give better results with morphologically richer languages like Turkish. It will be interesting to see if this pans out well for morphologically rich languages which will be the real test for this model.

References

Yoshua Bengio et al. 2001. *A Neural Probabilistic Language Model.* Advances in Neural Information Processing Systems 13 (NIPS'00), 2001

Siyu Qiu et al. 2014. *Co-learning of Word Representations and Morpheme Representations.* COLING 2014, the 25th International Conference on Computational Linguistics, pages 141150, Dublin, Ireland, August 23-29 2014, ACL, vol. 70, no. 6, pp. 510523, 2014.

M. Luong, R. Socher, and C. Manning 2013. *Better word representations with recursive neural networks for morphology.* CoNLL-2013, 2013, pp. 104.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean 2013. *Distributed representations of words and phrases and their compositionality.* NIPS 2013, pp. 31113119.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean 2013. *Efficient estimation of word representations in vector space.* ICLR, 2013.

M. Creutz and K. Lagus. 2007. *Unsupervised models for morpheme segmentation and morphology learning.* ACM Transactions on Speech and Language Processing, Volume 4, 2007.

`http://lasagne.readthedocs.org/en/latest/`

`http://mattmahoney.net/dc/textdata.html`

`http://morfessor.readthedocs.org/en/latest/`