Efficient Estimation of Word Representations in Vector Space

1 Citation

Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

https://arxiv.org/pdf/1301.3781.pdf

2 Abstract

We train word embedding vectors on a 1.6B word corpus in less than a day of training time. The resulting vectors do well in word similarity tasks.

3 Introduction

Learning how to turn words into short vectors in an unsupervised manner can provide a useful word representation for tasks where labeled data is limited.

The neural network language model (NNLM) is one way to do this. It takes N one-hot encoded words (each length V), looks up (i.e. matrix multiply) its embedding in a learnable embedding matrix ($V \times D$), and uses a hidden layer (size H) to predict the next word in the corpus. This is computationally expensive though.

4 Model Architectures

The training time of a model is $O = ETQ$ where E is embedding vector length, T is number of epochs, and Q is an architecture-dependent quantity.

NNLM gets $Q = ND + NDH + HV = O(HV)$. With hierarchical softmax, we can turn HV into $H \log_2 V$. You can also use a Recurrent NNLM (RNNLM), which can take into account a longer history. It has $Q = HH + HV$. Again HV turns into $H \log_2 V$ with hierarchical softmax.

We present models trained in the DistBelief system using Adagrad.

5 New Log-linear Models

The continuous bag of words model (CBOW) is like NNLM, but doesn't have a hidden layer. Basically, we tried to predict the word w_t given w_{t-2}, w_{t-1},$\mathrm{w}_{t+1}, w_{t+2}$. First, we compute $s = \sum_{i \in \{-2,-1,1,2\}} w_{t-i}$, where the words are one-hot encoded. Then, we multiply by an embedding matrix to get Ws and then we classify this with hierarchical softmax to predict w_t. This gets $Q = ND + D^2 \log_2 V$

The continuous skipgram model aims to predict a word's context from the word (CBOW does the opposite). We take compute Ew_t and classify it to predict a context word. We select a context word by randomly sampling from a word within a $C = 5$ word window centered on w_t. We sample such that more distant words are less likely. This has $Q = CD^2 \log_2 V$

6 Results

We evaluate our models with an analogy task. Basically, if word a (e.g. "big") is to b (e.g. "biggest") as c (e.g. "small") is to d (e.g. "smallest"), then we expect: $b - a + c = d$, where the variables are the word vectors.

We create a dataset with 9K syntactic (e.g. small:smallest = big:biggest) and 10K semantic (e.g. Paris:France = Madrid:Spain) analogy questions.

We train on a 6B word Google News corpus with a 1M word vocabulary. We started with a subset of the training data with 30K words, but it turns out you need to increase both for better performance. We use SGD (0.0025) for 3 epochs with linear learning rate decrase.

We find the NNLM beats the RNNLM. CBOW beats NNLM on syntactic task and ties on semantic tasks. Skipgram does a little worse than CBOW on syntactic and much better on semantic. CBOW and Skipgram both beat NNLM and RNNLM.

We try the Microsoft Sentence Completion Challenge (i.e. pick one of five words to complete the sentence) and find that an ensemble of Skipgram and RNNLM sets state of the art (58.9% accuracy).

7 Examples of the Learned Relationships

We learn some pretty good relationships. Also, for the analogy task, instead of providing $a : b$ and c, you can get a 10% accuracy improvement if you provide several $a : b$ analogies for a single c. That is, instead of Paris:France and asking for the analog for Madrid (should be Spain), you can provide Paris:France, Berlin:Germany, London:UK, etc.

8 Conclusion

Our Skipgram and CBOW models beat RNNLM and NNLM, set state of the art on Sentence Completion, and are cheaper to train. We provide an analogy dataset.

Note, we also provide a single-machine C++ implementation to train the word vectors.