Distributed Representations of Words and Phrases and their Compositionality

1 Citation

Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.

http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-composit

2 Abstract

We improve the Skipgram model. First, by sampling frequent words less often, we improve training speed. Second, we replace hierarchical softmax with negative sampling. Finally, we show how you can automatically discover phrases (e.g. "New York", "Boston Globe") and learn vectors for them too.

3 Introduction

The Skipgram model can train on 100B words in a day because it does not use dense matrix multiplication. It does well at the analogy task (e.g. the word nearest (cosine distance) to the $vec(Spain) - vec(Madrid) + vec(Paris)$ is $vec(France)$).

By sampling frequent words less, we get 2-10x training speedup. Negative sampling is also faster than hierarchical softmax. Finally, we automatically learn phrases.

4 The Skip-gram Model

Skipgram aims to predict a word's context given the word. That is, for a T word corpus with a context window c, we aim to maximize:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

The Skipgram approximates this with:

$$p(w_O|w_I) = \frac{\exp\left((v'_{w_O})^T v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left((v'_w)^T v_{w_I}\right)}$$

where v_w and v'_w are the input and output representations of word w. This is epensive to compute because of the summation in the denominator. To simplify, we can use hierarchical softmax, where we compute a Huffman binary tree representing the output word. This cuts the W into a $\log_2 W$. Basically, if $n(w, j)$

is the j^{th} node on the path from root to w, if $L(w)$ is length of path from root to w (so $n(w,1) = root$ and $n(w, L(w)) = w$), and $ch(w)$ is an arbitrary fixed child of w, then hierarchical softmax is:

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma(\mathbb{I}[n(w, j+1) = ch(n(w, j))] {v'_{n(w,j)}}^T v_{w_I})$$

A better approach is to use negative sampling, where we approximate the above probability with:

$$\log p(w_O|w_I) = \log \sigma({v'_{w_O}}^T v_{w_I}) + \sum_{i=1}^{K} \mathbb{E}_{w_i \sim P_n(w)}[\log \sigma(-{v'_{w_i}}^T v_{w_I})]$$

we use k between 5-20 for large datasets and k between 2-5 for small datasets. For $P_n(w)$, we sample with probability proportinal to the unigram probability raised to the power 3/4 (i.e. $(U(w))^{\frac{3}{4}}$).

Frequent words like "the" and "is" are not as useful to training as rarer words, so we when we sample words, we sample according to

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

we pick $t = 10^{-5}$.

5 Empirical Results

We evaluate on the analogy dataset. We train on 1B Google News words with a 692K word vocabulary (we throw away words occuring ≤ 5 times).

Negative sampling beats hierarchical softmax. Sampling frequent words less speeds up training and improves the word vectors.

6 Learning Phrases

To learn phrases, we assign scores to each bigram:

$$score(w_i, w_j) = \frac{count(w_i w_j) - \delta}{count(w_i) \times count(w_j)}$$

where δ prevents usage of infrequent words. We take bigrams scoring above a threshold and turn them into a single token. We can then apply this same procedure again, and again, and again (decreasing the threshold each time).

We then evaluated on a phrase analogy task. To do this, we use a 33B word dataset and entire sentence as context.

7 Additive Compositionality

The word vectors can be seen as representing the log probability of their context. Thus, we can sum them together to "multiply probabilities". Thus, "Russia" + "river" will select contexts that are near "Russia" AND "river". So, "Volga River" should be close to "Russia" + "river".

8 Comparison to Published Word Representations

Skipgram beats all other word vectors on the analogy task. It is trained on a much larger dataset, but actually trains faster than many of the other word vectors.

9 Conclusion

Negative sampling, sampling frequent words less, and learning phrases allowed us to train Skipgram on a huge dataset quickly and produce great word vectors.