Home Assignment 1: Introduction Into Natural Language Processing

Essay

Around the time of article publication state-of-art in word-to-vector embedding were recursive neural network language models (RNNLM). Using embedding produced by ensemble of those and this ensemble itself, performance of 55.4% accuracy in Microsoft Research Sentence Completion Challenge. This challenge consists of selecting a word from the list of five to fill the one-word gap in the sentence. Accuracy shown by RNNLMs was the best, but computational complexity of those models was enormous. On the other side of complexity existed n-gram models, which were incredible when trained on a giant amount of data but couldn't produce significant results without it. This is clearly shown in the same benchmark, where most suitable, I suppose, n-gram model – 4-gram, showed only 39% accuracy.

"Efficient Estimation of Word Representations in Vector Space" strives to offer solution for this problem – lightweight in computational complexity but complex enough to capture important relations in (relatively) small quantities of data architecture – CBOW and Skip-gram.

Before diving into the not really complex structure of CBOW and Skip-gram, we need to know where Mikolov *et al*. (authors of the article) got their inspirations. As for neural network language models (NNLM), the first article with its description "A neural probabilistic language model" was posted in 2003 by Bengio *et al*. The main idea is to predict the next word using n words before it. To do this, we need to assign a vector of dimensionality D to each word in vocabulary (we get projection matrix P). Then, we should transform n last words to corresponding vectors from P at plug them into hidden layer, output of which is vector of length V (size of vocabulary) with applied softmax. The i-th number of output vector is the probability of i-th word from vocabulary to be the next word. Main feature of this model is that it learns matrix P as well as the joint word probability distribution. Inspired by this architecture, Mikolov with different teams upgraded it by separating the process of learning word vectors and learning distributions into the different neural networks.

In the CBOW Mikolov *et al*. took idea of learning word vectors separately to the whole different level. Three key differences between NNLM and proposed architecture are the absence of hidden layer, consideration of not only previous, but future words also, and the upgrade of softmax to hierarchical one. While the first difference is simply dropping hidden layer, which reduces computational complexity, and second is self-explanatory, hierarchical softmax is a new concept. In simple words: instead of treating vocabulary as an array, let's look at it as a binary tree. If we place all words from vocabulary in the leaves of the tree, we

could find any word with $\log_2 V$ complexity instead of V in array view on vocabulary. To conclude, CBOW solves the problem of predicting not next but middle word given the sequence of those. Without extra hidden layer and with hierarchical softmax computational complexity is significantly lower than that of the NNLM.

As for Skip-gram, it can be viewed as CBOW vice versa. What I mean by this is that Skip-gram solves the opposite task compared to CBOW – predicting surrounding words given one. Treating C as a hyperparameter, the model predicts C words around the given one in each direction.

The greatest part of this paper is its results. Proposed architectures allowed models to learn word vectors that capture not only syntactic but semantic similarity. Surprisingly for Mikolov *et al*., such vectors even allowed mathematical operations such as addition and subtraction. For example, if we subtract vector of the word "Russia" from vector of "Putin" and add vector of "USA", we will get vector, very similar to "Biden". Such semantic connection between vectors could be very helpful in a lot of NLP applications.

To measure quality of word vectors, Mikolov *et al.* came up with Semantic-Syntactic Word Relationship test set consisting of 5 types of semantic and 9 types of syntactic test, 8869 semantic and 10675 syntactic questions total. Mikolov *et al.* empirically found out that using increasing training dataset along with incrementing dimensionality of word vectors would drastically improve accuracy of CBOW and Skip-gram on constructed test set. Among the previously proposed models, the newest NNLM by authors produced the best accuracy of 64.5% on the syntactic part of the test set, while keeping great performance (34.2% accuracy) on the semantic part, which yields the total of solid 50.8%. As for CBOW, it's lacking in either of two parts – 15.5% on semantic and great but less than NNLM 53.1% on syntactic, leading to the total of 36.1%. Skip-gram on the over hand succeeded a lot on the semantic part, showing the best accuracy of 50%, while having 55.9% on syntactic and 53.3% total. The most important part of that comparison is that CBOW and Skip-gram were trained on a 7.6 times smaller dataset than authors NNLM, and training time for CBOW was equal to one day, for Skip-gram – three days.

Previous performances were shown under constraints, here is the table for full on power for training:

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

As you can see in the table, the new proposed architectures not only train faster and require smaller resources but perform better.

Back to the Microsoft Research Sentence Completion Challenge. The combination of Skip-gram and RNNLMs was able to outperform RNNLMs alone, leading to new state of art (at the time) result 58.9% accuracy on benchmark from Microsoft. That's what I call success!

P.S. So irresponsible of authors to drop such bombs at the end of the article in small 6-line paragraph, but these maniacs were not satisfied with the results they initially got (I guess), so they trained the model on the 100B words and produced more than 1.4 mil word vectors. Even more than that, authors published all vectors along with single-machine multi-threaded C++ code for computing the word vectors, using both the continuous bag-of-words and skip-gram architectures. In simple words: anyone who has resources can play with their implementations and test them out; anyone can use those beautiful word vectors for any NLP task. Such things are just mind-blowing...

Practice

Part 1

I really like borsch, it reminds me of Izhevsk.

Aspic is ok, hard to forget Izhevsk. Wish to forget.

Sometimes I visit Octoberfest, reminds of how hard to get things done.

This course is lame. Some borsch write this aspic lectures. Sometimes I'm thinking about Izhevsk.

Part 2

['I', 'really', 'like', 'borsch', 'it', 'reminds', 'me', 'of', 'Izhevsk']

['Aspic', 'is', 'ok', 'hard', 'to', 'forget', 'Izhevsk', 'Wish', 'to', 'forget']

['Sometimes', 'I', 'visit', 'Octoberfest', 'reminds', 'of', 'how', 'hard', 'to', 'get', 'things', 'done']

['This', 'course', 'is', 'lame', 'Some', 'borsch', 'write', 'this', 'aspic', 'lectures', 'Sometimes', 'i', 'thinking', 'about', 'Izhevsk']

Part 3

Aspic	Izhevsk	Octoberfest	Some	Sometimes	This	Wish	about	aspic	borsch
0	1	0	0	0	0	0	0	0	1
1	1	0	0	0	0	1	0	0	0
0	0	1	0	1	0	0	0	0	0
0	1	0	1	1	1	0	1	1	1

course	done	forget	get	hard	how	is	it	lame	lectures	like	me	of
0	0	0	0	0	0	0	1	0	0	1	1	1

0	0	2	0	1	0	1	0	0	0	0	0	0
0	1	0	1	1	1	0	0	0	0	0	0	1
1	0	0	0	0	0	1	0	1	1	0	0	0

ok	**really**	**reminds**	**things**	**thinking**	**this**	**to**	**visit**	**write**	**I**
0	1	1	0	0	0	0	0	0	1
1	0	0	0	0	0	2	0	0	0
0	0	1	1	0	0	1	1	0	1
0	0	0	0	1	1	0	0	1	1

Part 4

I will find cosine similarity of third and fourth sentences.

Vector of 3rd sentence: 00101000000101110000001001100110

Vector of 4th sentence: 01011101111000001011000000011001

Dot product of two vectors:

0*0+0*1+1*0+0*1+1*1+0*1+0*0+0*1+0*1+0*1+0*1+1*0+0*0+1*0+1*0+1*0+0*1+0*0+
0*1+0*1+0*0+0*0+1*0+0*0+0*0+0*0+1*0+1*0+0*1+0*1+1*0+1*0+0*1 = 1

Length of the first vector:
sqrt(0*0+0*0+1*1+0*0+1*1+0*0+0*0+0*0+0*0+0*0+0*0+1*1+0*0+1*1+1*1+1*1+0*0+0
*0+0*0+0*0+0*0+0*0+1*1+0*0+0*0+1*1+1*1+0*0+0*0+1*1+1*1+0*0) =
3.3166247903554

Length of the second vector:
sqrt(0*0+1*1+0*0+1*1+1*1+1*1+0*0+1*1+1*1+1*1+1*1+0*0+0*0+0*0+0*0+0*0+1*1+0
*0+1*1+1*1+0*0+0*0+0*0+0*0+0*0+0*0+0*0+0*0+1*1+1*1+0*0+0*0+1*1) =
3.7416573867739413

Cosine similarity: 2 / (3.4641016151377544 * 4.123105625617661) = 0.0805822964