Research Paper Summary/Notes

Efficient Estimation of Word Representations in Vector Space, Google.

Laxman Singh Tomar

October 2019

Research Paper Summary/Notes

Efficient Estimation of Word Representations in Vector Space, Google.

Laxman Singh Tomar

October 2019

Contents

Dedication

I would like to dedicate this work to teachers whom I've never met in real but their valuable contribution have made this work possible.

Acknowledgement

(I encountered several resources which helped me in getting through this work)

I would like to express my sincere gratitude to:

- Tomas Mikilov et. al.
- Chris Mccormick
- Jay Alammar

Abstract

This paper introduces two architectures to compute word vectors from large datasets. Quality of the word vectors can be measured on Similarity tasks. Word Vectors provide state-of-the-art performance on measuring word similarities(semantically and syntactically).

Keywords: Word Embeddings; CBOW; Skipgram

Chapter 1

Introduction

Most of the current NLP systems treat words as atomic units without considering the notion of similarity. The advantages of this approach are simplicity and robustness. A popular example might be N-gram Models used for statistical language modeling. Although, the pitfall of this approach is that text corporas available contain approximately billions of words; and if you're looking for relevant in-domain data it's too limited. So, these basic techniques will not help us moving forward.

A better option would be to rely on Distributed Representation of Words i.e. Word Vectors which can outperform N-gram models.

1.1 Motivation

Inability of existing NLP systems to regard the notion of similarity and ineffectiveness when it comes to relevant in-domain data.

1.2 Objectives

Introduce techniques that can be used to learn high quality word-vectors from huge datasets containing billions of words, and million words in vocabulary.

Chapter 2

Methods

2.1 Model Architectures

In this paper, Computational complexity is defined in terms of number of parameters accessed during model training. It's proportional to:

$$E * T * Q$$

where, E- Number of training epochs T - Number of words in training set Q - depends on the model

2.2 Feed Forward Neural Net Language Model

These are Probabilistic models having input, projection, hidden and output layer [1]. Input layer encodes N previous word using 1-of-V encoding (V is vocabulary size). Then, input layer is projected to projection layer P with dimensionality $N*D$. Also, hidden layer (of size H) computes the probability distribution over all words.Here, the complexity per training example:

$$Q = N * D + N * D * H + H * V$$

It can reduce Q by using hierarchical softmax and Huffman binary tree (for storing vocabulary).

2.3 Recurrent Neural Net Language Model

RNNLMs are similar to NNLMs. They can also be referred as NNLMs minus the projection layer. Their complexity per training example:

$$Q = H * H + H * V$$

Hierarchical softmax and Huffman tree can be used here as well.

2.4 Log Linear Models

In Log Linear Models, non-linear hidden layer causes most of the complexity. NNLMs can be successfully trained in two steps:

1. Learn continuous word vectors using simple models.

2. N-gram NNLM trained over the word vectors.

2.4.1 Continuous Bag-of-Words Model

It is similar to feedforward NNLMs. It doesn't have any nonlinear hidden layer. Here, projection layer shared for all words and order of words does not influence projection. Log-linear classifier uses a window of words to predict the middle word. The complexity per training example:

$$Q = N * D + D * log2V$$

2.4.2 Skipgram Model

It is similar to Continuous Bag-of-Words but uses the middle world of the window to predict the remaining words in the window. Distant words are given less weight by sampling fewer distant words.The complexity per training exampleis:

$$Q = C * (D + D * log2V)$$

where C is the max distance of the word from the middle word. Given a C and a training data, a random R is chosen in range 1 to C. For each training word, R words from history (previous words) and R words from future (next words) are marked as target output and model is trained.

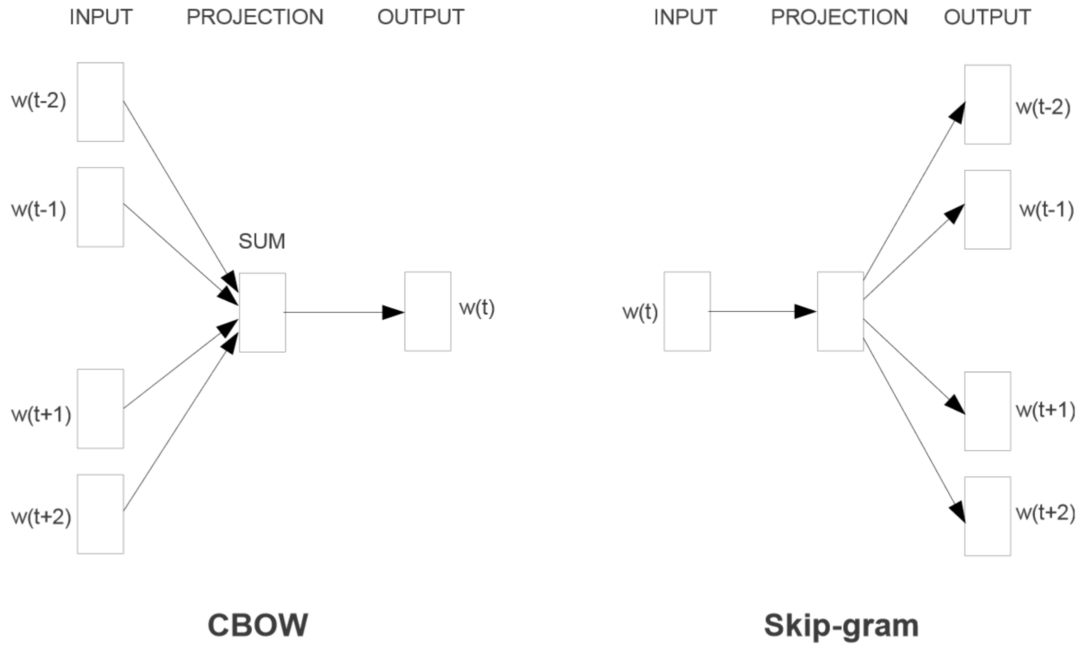


Figure 1: New Model Architecturs: CBOW architecture predicts the current word based on the context; Skipgram architecture predicts surrounding words given the current word

Chapter 3

Results

To compare quality of word vectors generated previous papers had used tables showing words and words similar to it. Here, we've subjected them into a more complex similarity task. How can we denote relationship between two pair of words? Say, for example: Big : Biggest :: Small : ?.

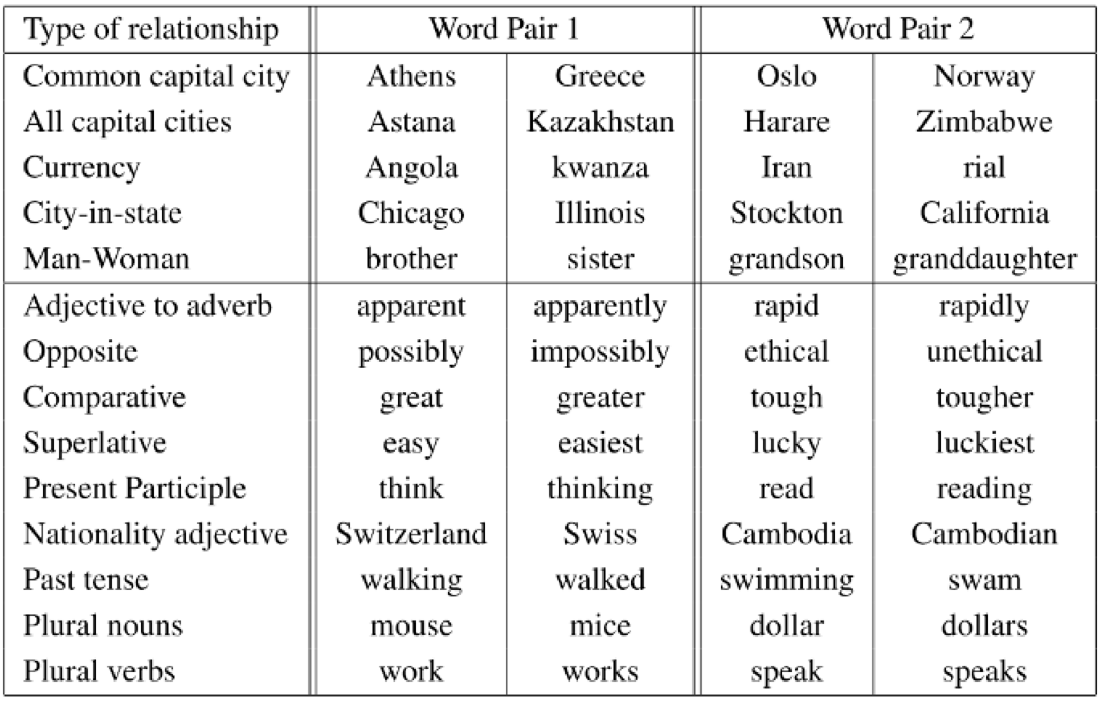
The answer to this problem is quite surprising as it can be answered with simple algebraic operations applied over vector representations of words. To find a word similar to Small, we can simply compute:

$$X(?) = vector(''biggest'') - vector(''big'') + vector(''small'')$$

Later we will search for a word similar to X in vector space using Cosine Similarity. When your vectors are well-trained it'll give correct answers on most occasions.

3.1 Task Description

Skip-gram beats all other models for Semantic Word Relationship accuracy tasks (e.g. - relating Athens with Greece). Here, answers are marked correct only if the closest word to the vector computed using methods above is exactly same as the correct word, implying synonyms are counted as mistakes.

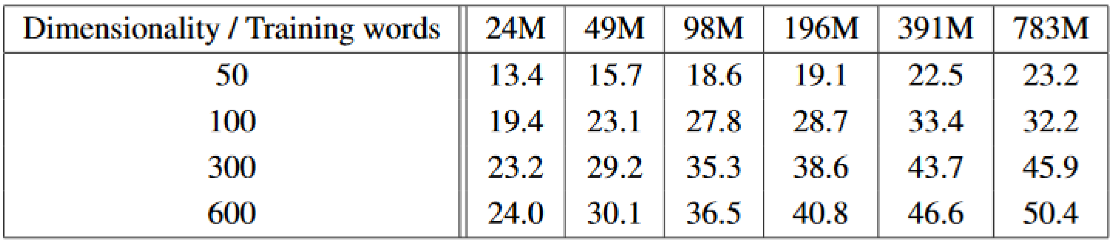


Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Figure 2: Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set

3.2 Maximization of Accuracy

Google News corpus containing 6B tokens is used. To estimate best choice of model architecture, most frequent 30k words are used. Results show that adding either more number of dimensions or adding more training data leads to diminishing improvements. So, we have to increase both vector dimensionality and the amount of training data together.

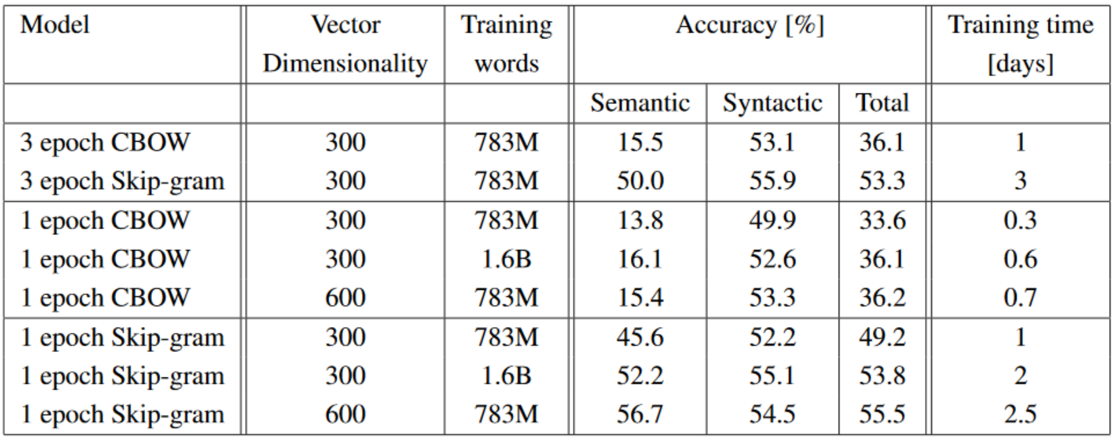


Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

Figure 3: Accuracy on subset of the Semantic-Syntactic Word Relationship test set, using word vectors from the CBOW architecture with limited vocabulary. Only questions containing words from the most frequent 30k words are used

3.3 Comparison of Model Architectures

Continuous Bag-of-Words Model outperforms other models for semantic accuracy tasks (e.g. great with greater) - with skip-gram just behind in performance. Training a model on twice as much as data using one epoch gives better results than iterating over the same data for three epochs.



Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Figure 4: Comparison of models trained for three epochs on the same data and models trained for one epoch. Accuracy is reported on the full Semantic-Syntactic data set

3.4 Microsoft Research Sentence Completion Challenge

Skip-gram architecture combined with RNNLMs outperforms RNNLMs (and other models) for Microsoft Research Sentence Completion Challenge. See Table 1.

3.5 Examples of Learned Relationships

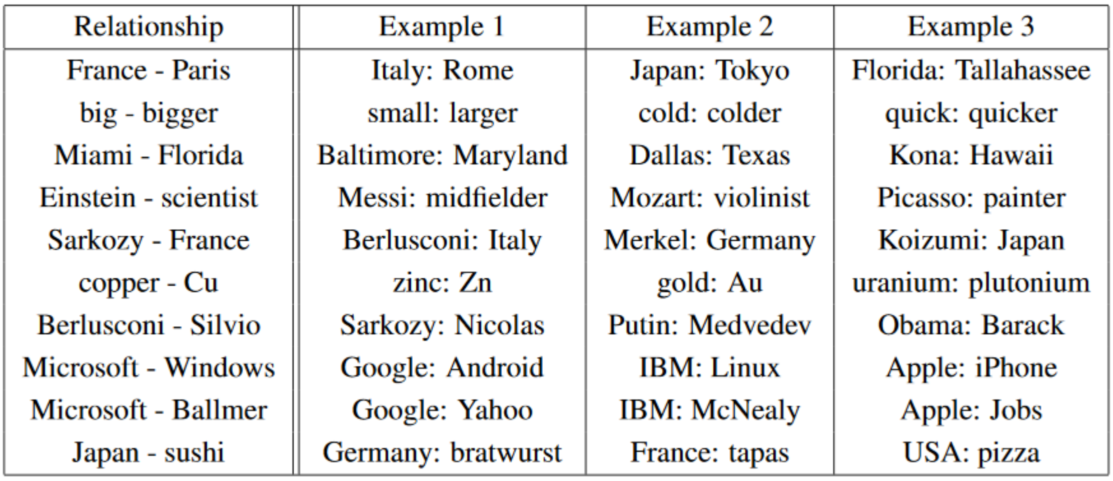
Model can learn relationships like "Queen is to King as Woman is to Man". This allows algebraic operations like:

$$Vector(''King'') - Vector(''Man'') + Vector(''Woman'')$$

Table 1: Comparison and combination of models on the Microsoft Sentence Completion Challenge

Architecture	**Accuracy (in percentage)**
4-gram	39
Average LSA similarity	49
Log-bilinear model	54.8
RNNLMs	55.4
Skip-gram	48.0
Skip-gram + RNNLMs	58.9

This approach can be useful in solving problems like: Selecting out-of-the-list words, by computing average vector for a list of words, and then finding the most distant word vector.



Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Figure 5: Examples of the word pair relationships, using the best word vectors from Skip-gram model trained on 783M words with 300 dimensionality

Chapter 4

Discussion

4.1 Discussion

In this paper, two new architectures Continuous Bag-of-Words and Skipgram were introduced and word vectors generated from both these models were evaluated on wide array of syntactic and semantic tasks.

4.2 Conclusions

We observed that it is possible to train high quality word vector representations using very simple model architectures, compared to the popular neural network models like feed-forward and RNNs. Due to lower computational complexity, we are able to compute very accurate word vector representations from a much larger dataset.

List of Figures

List of Tables

Bibliography

[1] Bengio, Y. A neural probabilistic language model. *NIPS* (2000).

Appendix A

Google's Word2Vec implementation in C

`https://code.google.com/archive/p/word2vec/`

You can also find here some pre-trained models that they have provided. Note that it's possible to load these pre-trained models into gensim if you want to work with them in Python.

Appendix B

Presentation by T. Mikolov

On December 9th, 2013 at NIPS 2013 Tomas Mikolov from Google gave a presentation on Word2Vec. I think this is mainly a re-hash of the content in the two papers(there is a follow-up paper). Seeing it presented differently may help you pull out some additional insights, though.

`https://docs.google.com/file/d/0B7XkCwpI5KDYRWRnd1RzWXQ2TWc/edit`