Google's Word2vec

A tool for computing distributed representations of word vectors

3/23/2017

Introduction

- What are word vectors?
  - We can encode words by weighting them across different dimensions
  - Ex.) In a small vocabulary: boy, girl, hen, rooster
  - Vector encoding [gender, is_chicken]
  - Boy: [1.0 , 0.0], hen: [0.0, 1.0]
  - Girl: [0.0, 0.0], rooster: [1.0, 1.0]

This allows for…

- The following calculation
  - Vec(boy) = Vec(girl) + Vec(rooster) – Vec(hen)
  - [1.0, 0.0] = [0.0, 0.0] + [1.0, 1.0] – [0.0, 1.0]

- Learning these representations can allow models to infer **syntactic** and **semantic** regularities within language

Types of Syntactic and Semantic Connections

Table 1: *Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.*

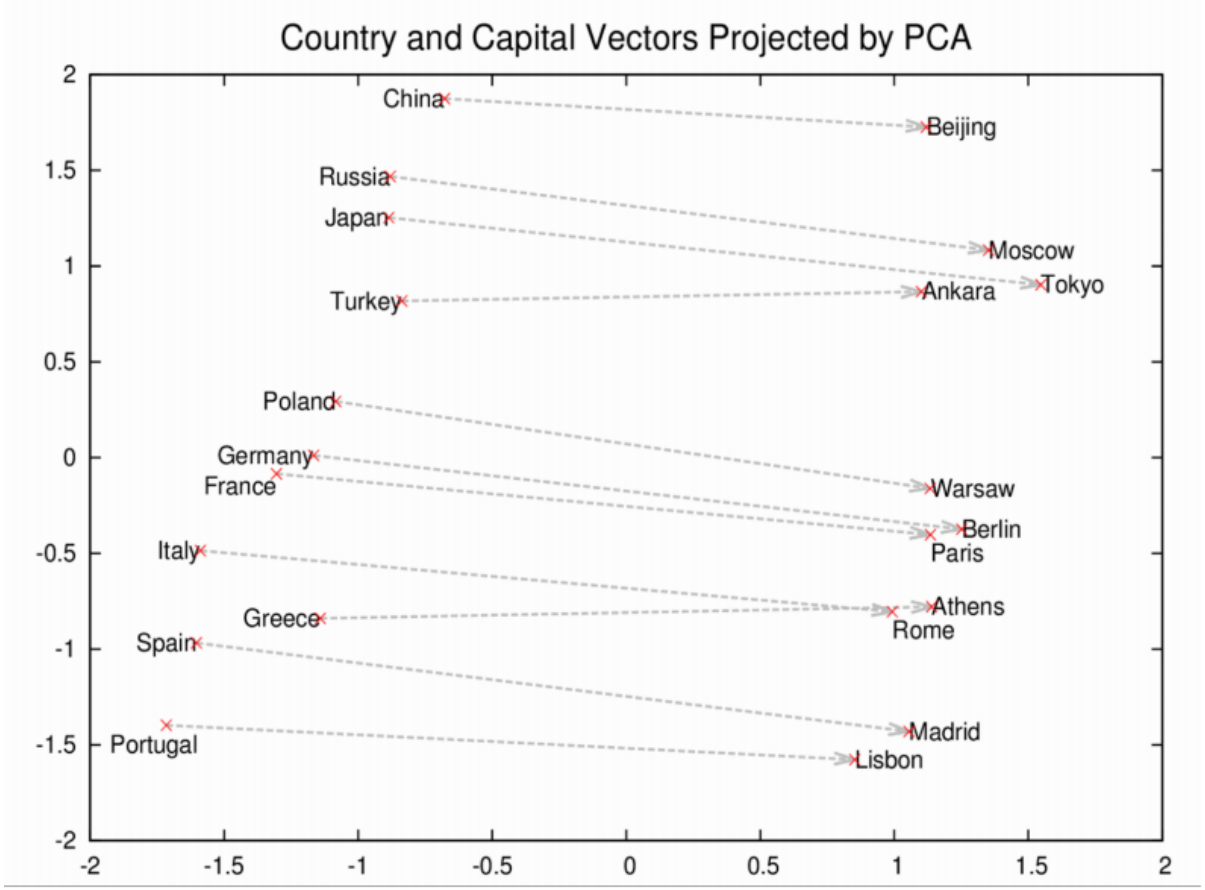
Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	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

Word2vec

- Created at Google by a team of researchers led by Tomas Mikolov
- Un-supervised learning algorithm which uses a large amount of text to create high dimensional representations of words
- Structured as a neural network with 1 hidden layer
- 2 major architectures: CBOW and skip-gram

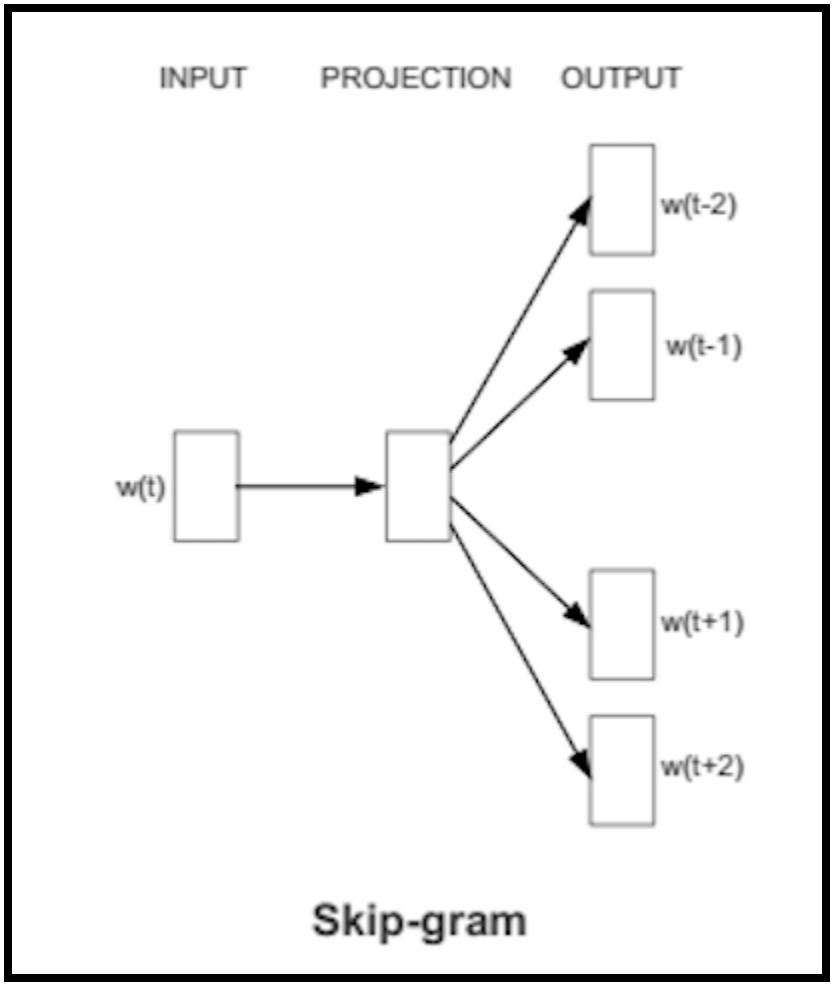
Interesting Results 1

- Vec(Rome) – Vec(Italy) + Vec(China) = Vec(Beijing)



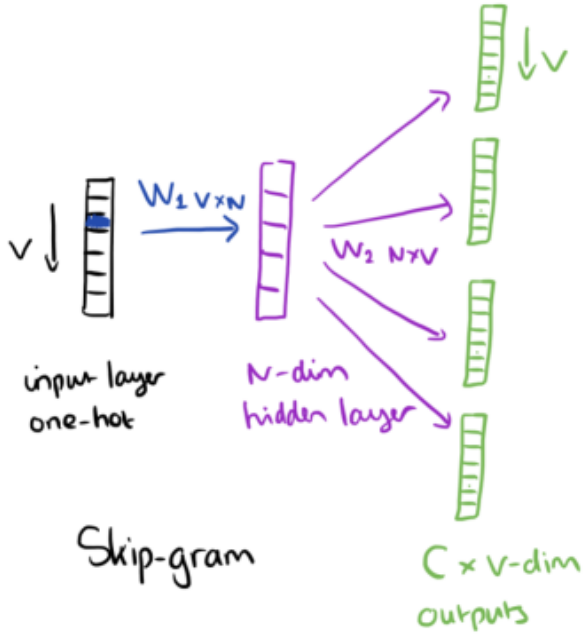
Continuous Skip-gram Model

- Model predicts context from current word
  - Ex) “ice”
  - Predicted context words: “melting”, “winter”, “cold”



Skip-gram

- If there are C context words at the output layer we output C multinomial distributions



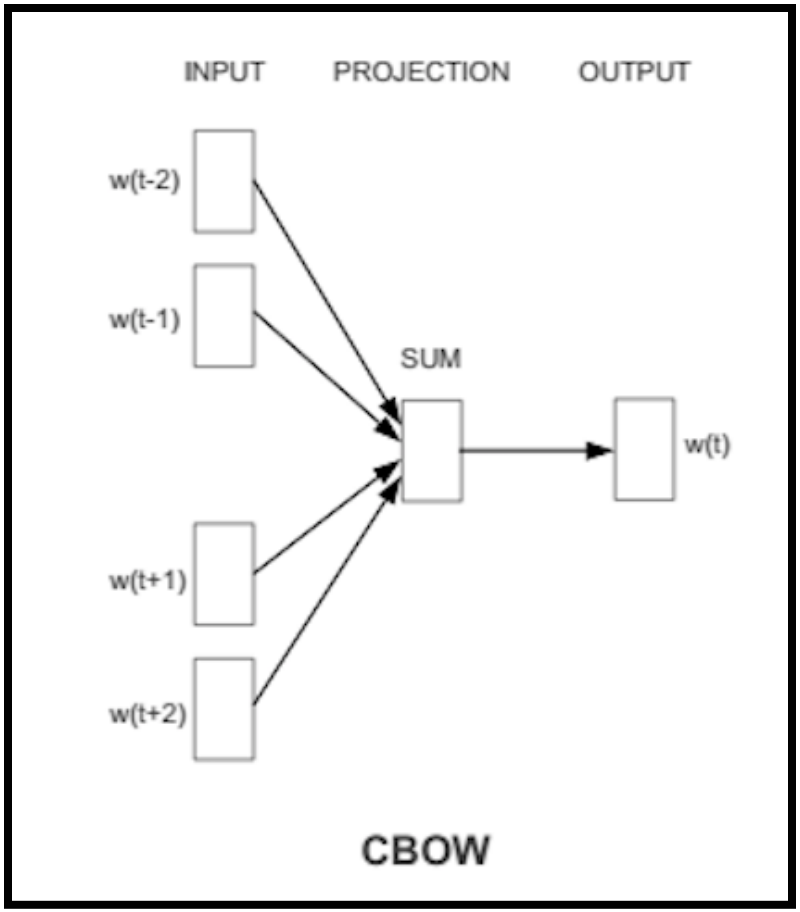
Insight

- In the skip-gram model, the hidden layer is acting as a "look-up table" for the input word vector

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

Continuous Bag of Words (CBOW)

- Model predicts the current word from context
  - Ex.) “children”, “__”, “in”, “a”, “school” ,“bus”
  - predict “ride”



CBOW

- Each input word is encoded in “one-hot” 1xV vectors if V is the size of the vocabulary
- The second weight matrix generates a score for each word, and softmax can be used to obtain the posterior distribution of words

NNLM and RNNLM

- NNLM (Feedforward neural net language model)
  - Input, projection, hidden, and output layers
  - High computational complexity
- RNNLM (Recurrent neural net language model)
  - Input, hidden and output layers
  - Hidden layer connects to itself with time-delay
  - High computational complexity

Results

- Both CBOW and Skip-gram perform well compared to NNLM and RNNLM

Table 3: *Comparison of architectures using models trained on the same data, with 640-dimensional word vectors. The accuracies are reported on our Semantic-Syntactic Word Relationship test set, and on the syntactic relationship test set of [20]*

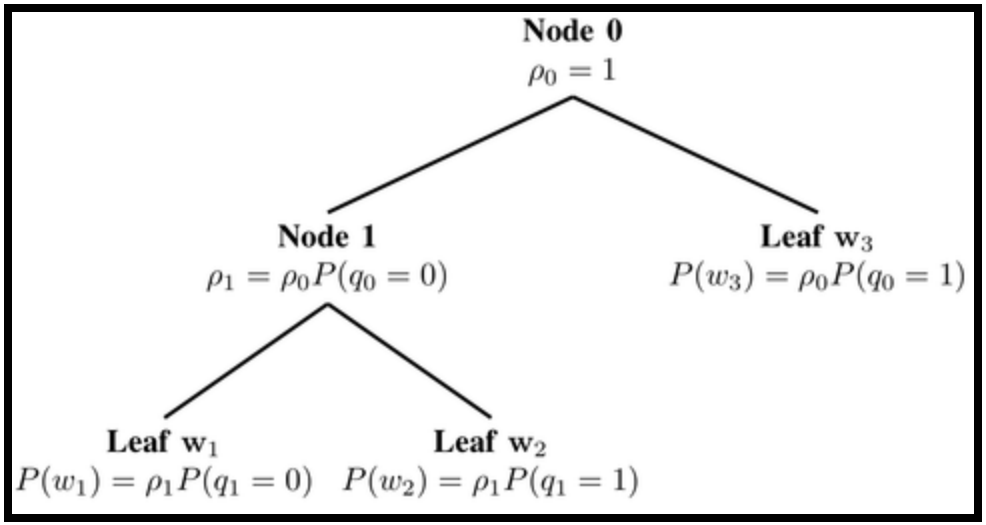
Model	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness
Architecture	Semantic Accuracy [%]	Syntactic Accuracy [%]	Test Set [20]
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

Additional Results

- architecture: skip-gram (slower, better for infrequent words) vs CBOW (fast)
- the training algorithm: hierarchical softmax (better for infrequent words) vs negative sampling (better for frequent words, better with low dimensional vectors)

Optimization w/ Hierarchical Softmax

- H-softmax replaces the flat softmax layer with a hierarchical layer that has words as leaves



H-softmax

- Think of regular softmax as a tree of depth 1, with V leaf nodes
- Computing the softmax probability of one word involves calculating the probabilities of all V
- Structuring the softmax as a binary tree we only have to follow the path from the root to the leaf, considering at most log(V) nodes

Optimization w/ Negative Sampling

- Only update a sample of output words per iteration
- Mikolov et al. uses the following procedure to counter the lesser knowledge given by words such as "in" and "the" compared to rarer words
  - Each word is discarded with the following probability, where f defines the frequency of the word

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