

# Word2Vec

Kwabena Amponsah-Kaakyire, Jesujoba Alabi, Daria  
Pylypenko, Lisa Kluge, Morgan Wixted

# Table of Contents

- 1. Introduction**
2. Recap on Neural Networks
3. Previous Work
4. Word2Vec
5. Applications
6. Summary

# Introduction

- The goal of NLP is to be able to design algorithms to allow computers to “understand” natural language in order to perform some tasks e.g. spell checking, machine translation, etc.
- But how do we represent words in the computer?
- Much of the earlier NLP methods treat words as atomic symbols
- And to perform well in most NLP tasks we need to have some notion of similarity and a difference between words.

# One-hot encoding

- A vector with  $D \times 1$  dimension where all elements are 0 except the index where the word occurs which is equal to 1.
- Doesn't capture relationships between words.
  - E.g. Kid is another word for child.

$$\text{Kid} = [1 \ 0 \ 0 \ 0 \ 0 \ 0]^T$$

$$\text{Child} = [0 \ 0 \ 0 \ 0 \ 0 \ 1]^T$$

$$\text{Kid}^T \times \text{Child} = 0$$

# Why vector representations?

- Mapping words or phrases from a vocabulary to real number vectors.
- Previous taxonomies like WordNet used hypernym relationships and synonym sets
- Incompleteness, missing new words
- Missing nuances
- Subjective
- Difficult to measure word similarity
- Requires human intervention

# Word Vectors

- There are an estimate of over 13 million tokens in English, but are they all completely unrelated? Motel and hotel? NO!!!
- Thus, the need to encode word tokens each into some vector that represents a point in some sort of “word space” keeping the semantic, tense, gender and count.
- One-hot vector representation - words are represented as vectors with all 0s and one 1 at the index of the sorted vocabulary.

$$W^{\text{hotel}} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{pmatrix}$$

$$W^{\text{motel}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \cdot \\ \cdot \\ 1 \\ 0 \end{pmatrix}$$

$$\text{Where } (w^{\text{hotel}})^T w^{\text{motel}} = (w^{\text{hotel}})^T w^{\text{catl}} = 0$$

# SVD Based Method

For this class of methods to find word embeddings,

1. Word-Document matrix
2. window- based co-occurrence matrix

	Document 1	Document 2	Document 3	Document 4	Document 5	Document 6	Document 7	Document 8
Term(s) 1	10	0	1	0	0	0	0	2
Term(s) 2	0	2	0	0	0	18	0	2
Term(s) 3	0	0	0	0	0	0	0	2
Term(s) 4	6	0	0	4	6	0	0	0
Term(s) 5	0	0	0	0	0	0	0	2
Term(s) 6	0	0	1	0	0	1	0	0
Term(s) 7	0	1	8	0	0	0	0	0
Term(s) 8	0	0	0	0	0	3	0	0

Word Vector (Passage Vector) →

Document Vector →

	<i>I</i>	<i>like</i>	<i>enjoy</i>	<i>deep</i>	<i>learning</i>	<i>NLP</i>	<i>flying</i>	<i>.</i>
<i>I</i>	0	2	1	0	0	0	0	0
<i>like</i>	2	0	0	1	0	1	0	0
<i>enjoy</i>	1	0	0	0	0	0	1	0
<i>deep</i>	0	1	0	0	1	0	0	0
<i>learning</i>	0	0	0	1	0	0	0	1
<i>NLP</i>	0	1	0	0	0	0	0	1
<i>flying</i>	0	0	1	0	0	0	0	1
<i>.</i>	0	0	0	0	1	1	1	0

The methods were able to give words vector sufficient to encode semantic and syntactic information.

# SVD Based Method: Issues

- It takes quadratic time to train (i.e. to perform SVD) - the computational cost for a  $m \times n$  matrix is  $O(mn^2)$
- The matrix is very high dimensional in general (approximately  $10^6 \times 10^6$ )
- The dimensions of the matrix change very often (new words are added frequently and corpus changes in size)
- The matrix is extremely sparse since most words do not co-occur



# Machine Learning to the Rescue

- Training of more complex data on larger data sets.
- Better performance than simple models.
- Probably, the most successful is to use distributed representation of words.  
Eg. neural net. based lang. models perform significantly better than n-gram models.

# Table of Contents

1. Introduction
- 2. Recap on Neural Networks**
3. Previous Work
4. Word2Vec
5. Applications
6. Summary

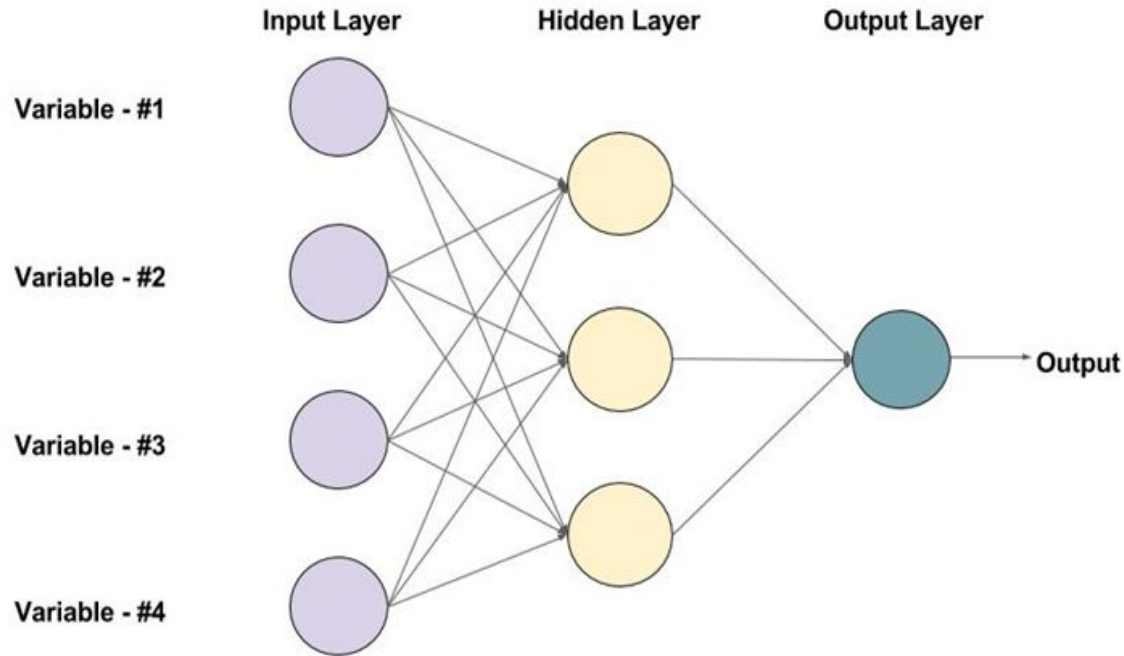
# Neural Networks

1. Feed forward neural networks
2. Recurrent neural networks
3. Others

# Feed Forward Neural Networks

- Simplest neural network
- No internal state
- No cycles in the network
- Information moves in one direction only (forward)

# Feed forward NN: Structure



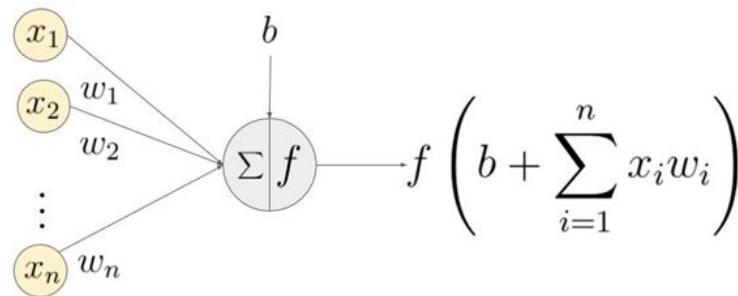
An example of a Feed-forward Neural Network with one hidden layer ( with 3 neurons )

# Neuron

Neuron: basic unit of a neural network

Calculates weighted sum of input, adds a bias and applies an activation function to produce an output.

E.g. of activation functions: sigmoid (0 – 1), Tanh (-1 – 1), ReLU (  $\geq 0$  )



An example of a neuron showing the input (  $x_1 - x_n$  ), their corresponding weights (  $w_1 - w_n$  ), a bias (  $b$  ) and the activation function  $f$  applied to the weighted sum of the inputs.

Image from Vikas Gupta

## Input Layer

First layer of the neural network

Provides input data or feature vectors to the network

## Hidden Layer

Allows for computation of complex function by combining simpler functions.

## Output Layer

Provides the output

Activation functions depend on the type of problem.

# Recurrent NNLM

Computational complexity  $Q=H \times H + H \times V$

Dominating term  $H \times V$

Can be reduced by hierarchical softmax  $H \times \log_2(V)$

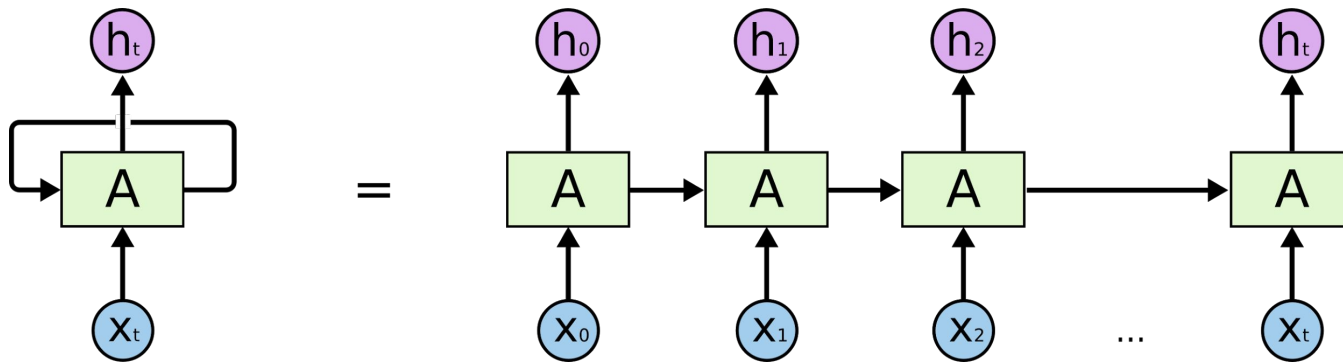
Most of the complexity will then come from  $H \times H$



# Recurrent Neural Network

Output is fed back into the network.

Keeps an internal state.



# How neural networks learn?

- Error measured by comparing output with actual value.
- Error used to update weights to reduce the error (backpropagation)
- Applying backpropagation after each training example is called stochastic gradient descent (SGD). Applying backpropagation after computing on all training examples is called batch gradient descent (BGD).
- Minibatch gradient descent interpolates between SGD and BGD.
- Learning rate determines the amount by which weights are updated.
- Each complete computation on the training set is called an epoch

# Efficient Estimation of Word Vectors

# Goals

- Two architectures for efficiently computing continuous vector representation of words from very large data sets.
- Use recently proposed techniques to measure resulting vector representation, with the expectation that similar words tend to be close to each other and also words can have multiple degrees of similarity.
- To maximise accuracy of these vector operations by developing new model architectures that preserve the linear regularities among words.

# Advantages

Goes beyond simple syntactic regularities. Algebraic operations can be performed on the word vectors.

# Table of Contents

1. Introduction
2. Recap on Neural Networks
- 3. Previous Work**
4. Word2Vec
5. Applications
6. Summary

# Previous Work

Feed forward neural network with a linear projection layer and a non-linear hidden layer to jointly learn the word vector representation and a statistical language model.

An NNLM was presented by Mikolov et al., where the word vectors are first learned using neural network with a single hidden layer. The word vectors are then used to train the NNLM.

Computationally expensive.

# Feed-forward NNLM

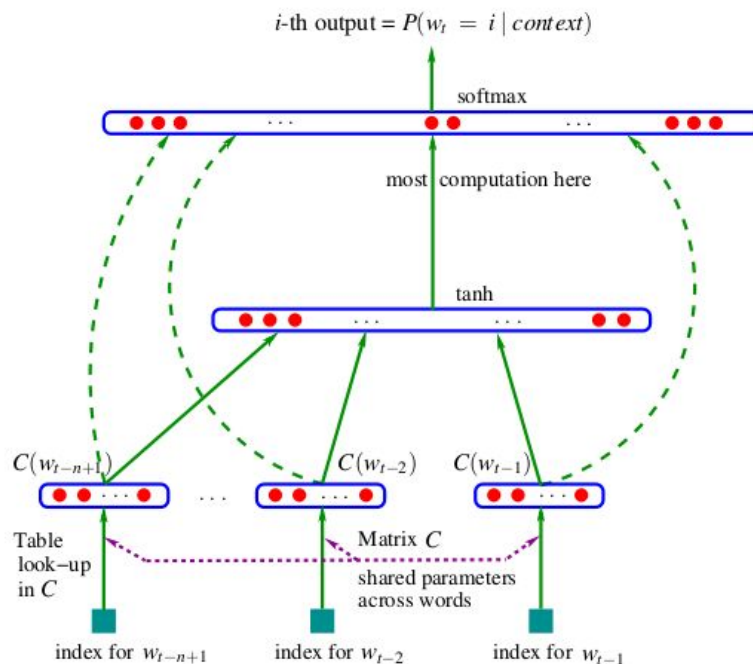
Computational complexity  $Q = N \times D + N \times D \times H + H \times V$

Dominating term  $H \times V$

Can be reduced by:

- Not normalizing during training
- Hierarchical softmax

Most of the complexity will then come from  $N \times D \times H \times \log_2(V)$





# Table of Contents

1. Introduction
2. Recap on Neural Networks
3. Previous Work
- 4. Word2Vec**
5. Applications
6. Summary

# Word2vec - An Iteration Based Method

The Idea: design a model whose parameters are the word vectors. Then train the model on a certain objective.

This idea date back to 1986

Word2vec comes in two forms.

1. Continuous Bag-of-word
2. Skip gram

Algorithmically, these two models are similar.

# New Log-Linear Models (Proposed Methods)

- Complexity in previous architectures caused by non-linear hidden layers.
- Proposed architecture capitalises on realisation that:
- Continuous word vectors can be learned using a simple model
- N-gram NNLM can be trained on top of these learned word vectors.
- A simpler model can therefore be used for word vector representation as this part is not concerned with training n-gram NNLM.

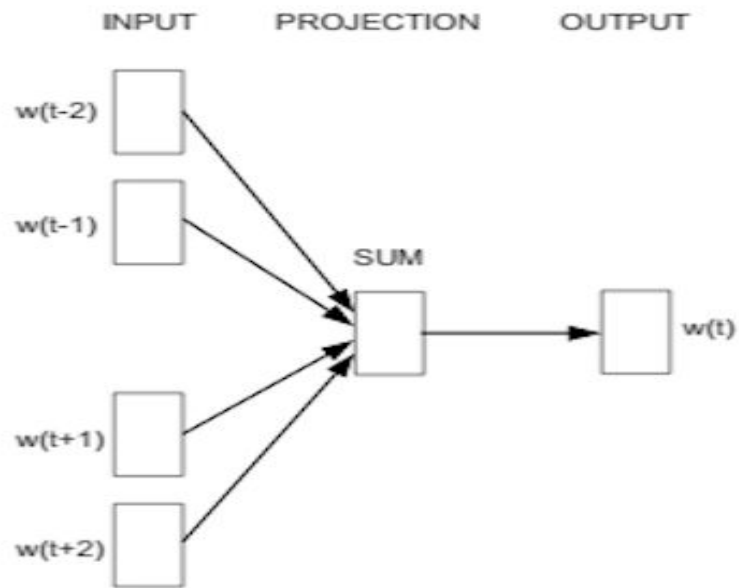
# CBOW & Skip-gram

CBOW predicts target words from the surrounding context words. E.g.

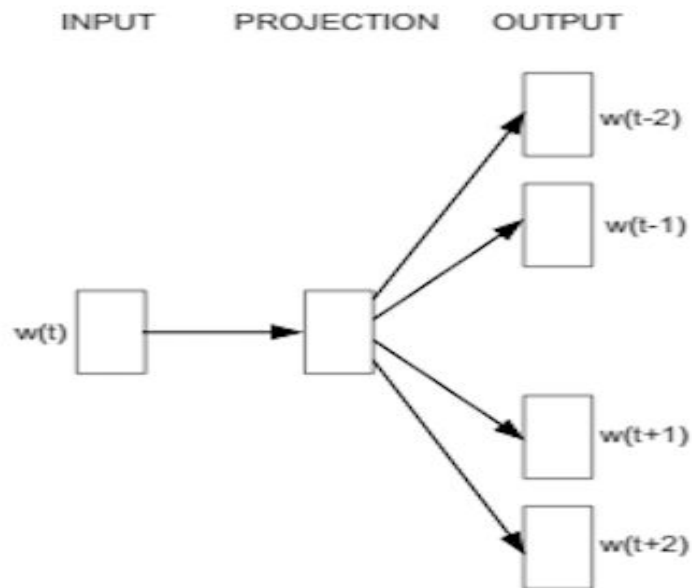
(‘the cat sit s on the .....’). Statistically, it smoothes over a lot of distributional information(therebye making an entire context as one observation). It turns out to be useful for smaller dataset.

Skip-gram predicts surrounding context words from the target words(an inverse of CBOW). Statistically, it treat each context-target pair as a new observation. Unlike CBOW, it tends to be better when we have large datasets.

# CBOW & Skip-gram



**CBOW**



**Skip-gram**

# Continuous Bag-of-Word Model (CBOW)

Similar model to feedforward NNLM

Non-linear hidden layer is removed

Projection layer is shared for all words

The training task for this model is to predict the current word given  $n$  (4 in the paper) history words and  $n$  future words as input.

A log-linear classifier is used.

Training complexity:  $Q = N \times D + D \times \log_2(V)$

# Continuous Skip-gram Model

Similar to Cbow, algorithmically, but does the inverse.

Tries to maximize classification of a word based on another word in the same sentence i.e. predict context (n previous and n next words) based on given word.

Increasing the range improves quality of the resulting word vectors as well as the computational complexity.

Gives less weight to the distant words by sampling less from those words (smoothing)

Training complexity:  $Q = C \times (D + D \times \log_2(V))$

# Parallel training of models

This uses a distributed framework called DistBelief, including feedforward NNLM and the new models proposed in this paper to train models on huge data sets in parallel using multiple CPU cores. Gradient updates are synchronised through a centralised server that keeps all the parameters.



# Relationship Between Words

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

# Accuracy

Table 2: 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.

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

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

# Accuracy

Table 4: *Comparison of publicly available word vectors on the Semantic-Syntactic Word Relationship test set, and word vectors from our models. Full vocabularies are used.*

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	<b>64.5</b>	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	<b>50.0</b>	55.9	<b>53.3</b>

# Accuracy

Table 5: *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.*

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

# Large scale parallel training of models

Table 6: *Comparison of models trained using the DistBelief distributed framework. Note that training of NNLM with 1000-dimensional vectors would take too long to complete.*

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

Table 7: *Comparison and combination of models on the Microsoft Sentence Completion Challenge.*

Architecture	Accuracy [%]
4-gram [32]	39
Average LSA similarity [32]	49
Log-bilinear model [24]	54.8
RNNLMs [19]	55.4
Skip-gram	48.0
Skip-gram + RNNLMs	<b>58.9</b>



# Learned Relationships

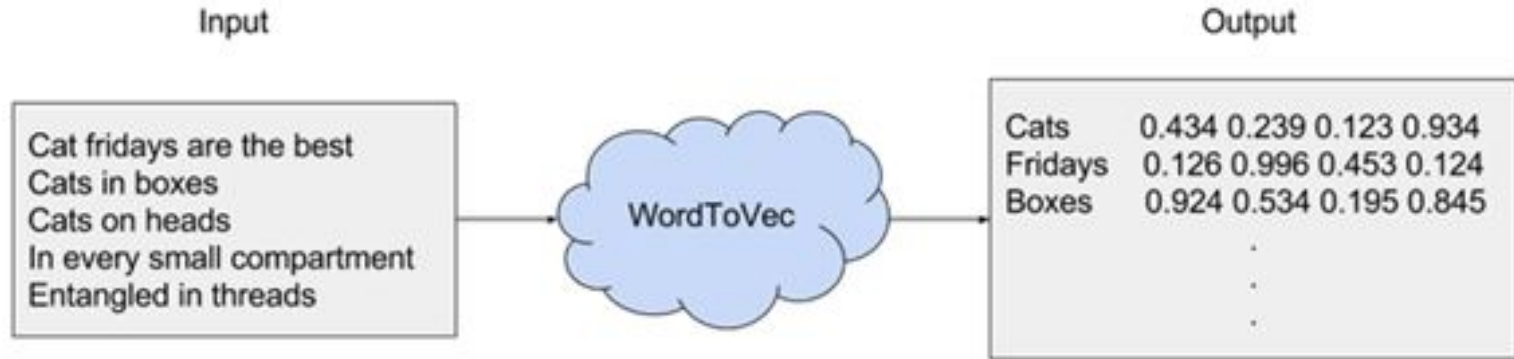
Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

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

# Word2vec Parameter learning explained

# Word2vec Parameter learning explained

Word2Vec (W2V) is an algorithm that accepts text corpus as an input and outputs a vector representation for each word, as shown in the diagram below:



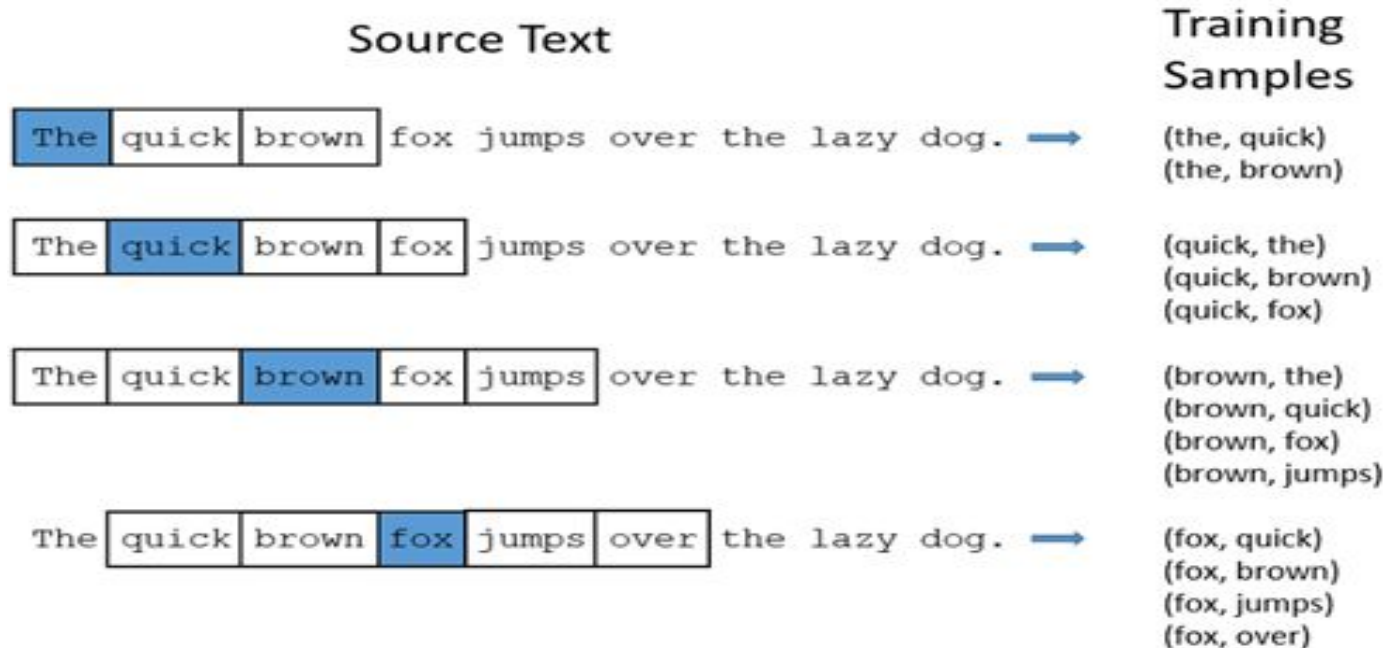
Based on the distributional hypothesis in linguistics:

1. words that are used and occur in the same contexts tend to purport similar meanings.
2. That's is, "a word is characterized by the company it keeps"



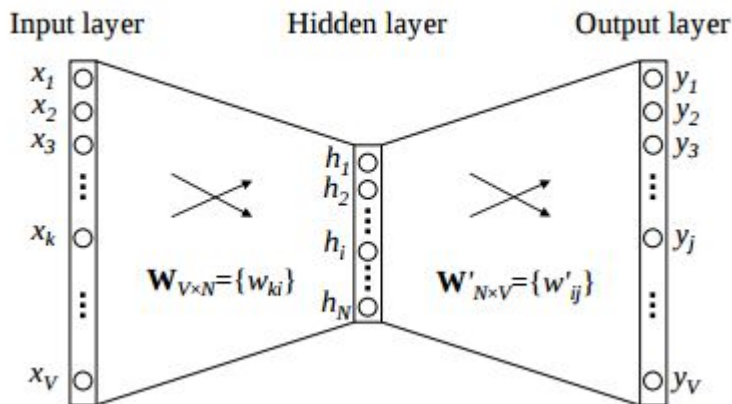
# Word2vec Parameter learning explained

To limit the number of words in each context, a parameter called “window size” is used.



# Word2vec Parameter learning explained

## One-word Context CBOW Example



$$Loss = \frac{1}{T} \sum_{i=1}^T \max \log y_j$$

From the input layer to the hidden layer, there is a Weight Matrix  $\mathbf{W}$  which is  $V \times N$

$$\mathbf{h} = \mathbf{x}^T \mathbf{W} = \mathbf{W}_{(k, \cdot)} := \mathbf{v}_{w_I}$$

From the hidden layer to the output layer, there is another weight matrix  $\mathbf{W}'$  which has  $N \times V$  dimensions

$$u_j = \mathbf{v}'_{w_j}{}^T \cdot \mathbf{h} \quad \text{Score for word at index } j \text{ in the vocabulary}$$

Now you can use a softmax (log linear) classification model to obtain the posterior distribution of words

$$p(w_j | w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^V \exp(u_{j'})}$$

Note that the score  $u_j$  is a measure of the match between the context and the next word

# Word2vec Parameter learning explained

By substituting the earlier shown equations, we can have the posterior distribution of words as

$$p(w_j|w_I) = \frac{\exp(\mathbf{v}'_{w_o}{}^T \mathbf{v}_{w_I})}{\sum_{j'=1}^V \exp(\mathbf{v}'_{w'_j}{}^T \mathbf{v}_{w_I})}$$

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i$$

Update the weight parameters

$$\mathbf{v}'_{w_j}{}^{(\text{new})} = \mathbf{v}'_{w_j}{}^{(\text{old})} - \eta \cdot e_j \cdot \mathbf{h}$$

$$\mathbf{v}'_{w_I}{}^{(\text{new})} = \mathbf{v}'_{w_I}{}^{(\text{old})} - \eta \cdot E\mathbf{H}$$

$j^*$  is the index of the actual output word (in the output layer)

$$\max p(w_o|w_I) = \max y_{j^*}$$

$$= u_{j^*} - \log \sum_{j'=1}^V \exp(u_{j'}) := -E$$

$$E = -\log p(w_o|w_I)$$

Both the input vector  $\mathbf{x}$  and the output  $\mathbf{y}$  are one-hot encoded

$v_w$  and  $v'_w$  are two representations of the input word  $w$

$v_w$  comes from the rows of  $\mathbf{W}$

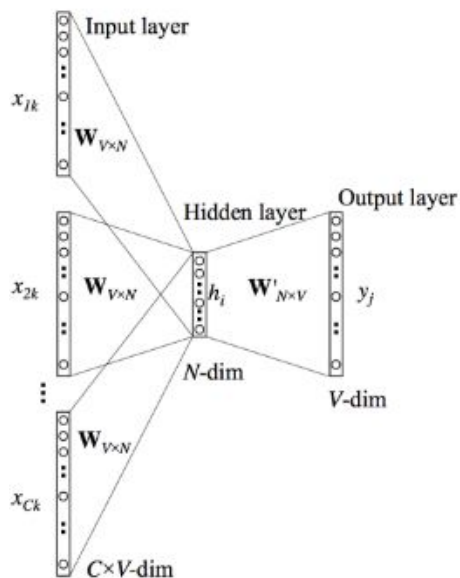
$v'_w$  comes from the columns of  $\mathbf{W}'$

$v_w$  is usually called the *input vector*

$v'_w$  is usually called the *output vector*

# Word2vec Parameter learning explained

## Multi-word Context CBOW Example



From the input layer to the hidden layer, there is a Weight Matrix  $W$  which is  $V \times N$

$$h = \frac{1}{C} W^T (x_1 + x_2 + \dots + x_C)$$

From the hidden layer to the output layer, there is another weight matrix  $W'$  which has  $N \times V$  dimensions

$$u_j = \mathbf{v}_{w_j}'^T \cdot \mathbf{h}$$

Score for word at index  $j$  in the vocabulary

Now you can use a softmax (log linear) classification model to obtain the posterior distribution of words

$$p(w_j | w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^V \exp(u_{j'})}$$

# Word2vec Parameter learning explained

## Multi-word Context CBOW Example

Input: Current word, Output: context words

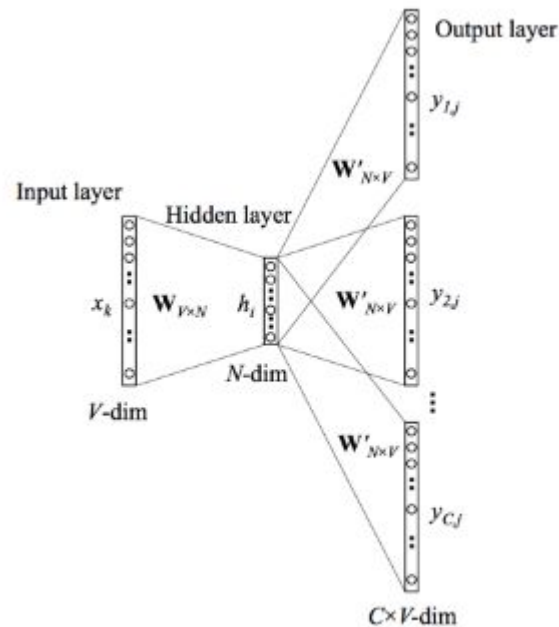
Given a sequence of training words,  $w_1, w_2, w_3 \dots w_T$

$$\text{maximize } J = \frac{1}{T} \sum_{t=1}^T \log P(w_{t-c}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+c} | w_t)$$

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

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

$$\frac{1}{T} \sum_{t=1}^T \sum_{c \in C} \log P(w_c | w_t)$$



$O(C \times m \times V)$

# Word2vec Parameter learning explained

## Skip-Gram Example

Recall that

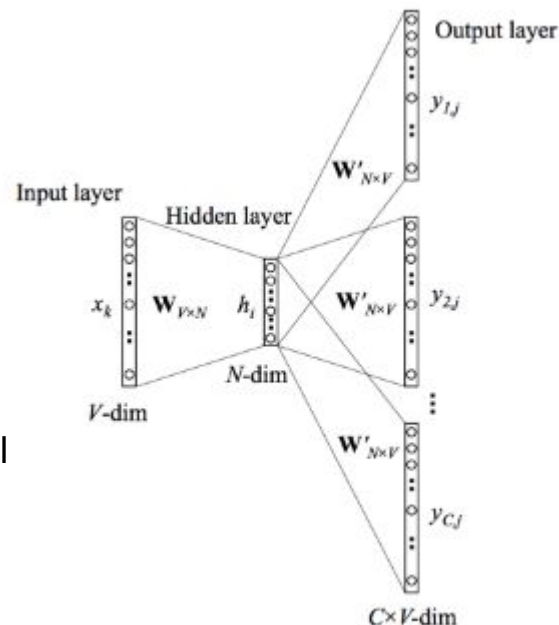
$$\mathbf{h} = \mathbf{W}_{(k, \cdot)} := \mathbf{v}_{w_l}$$

at the output layer, instead of outputting one multinomial distribution, we output  $C$  multinomial distributions using the same hidden-output layer weights

$$p(w_{c,j} = w_{o,c} | w_l) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^V \exp(u_{c,j'})}$$

$w_{c,j}$  is the  $j$ -th word on the  $c$ -th panel of the output layer.  $w_{o,c}$  is the actual  $c$ -th word in the output context

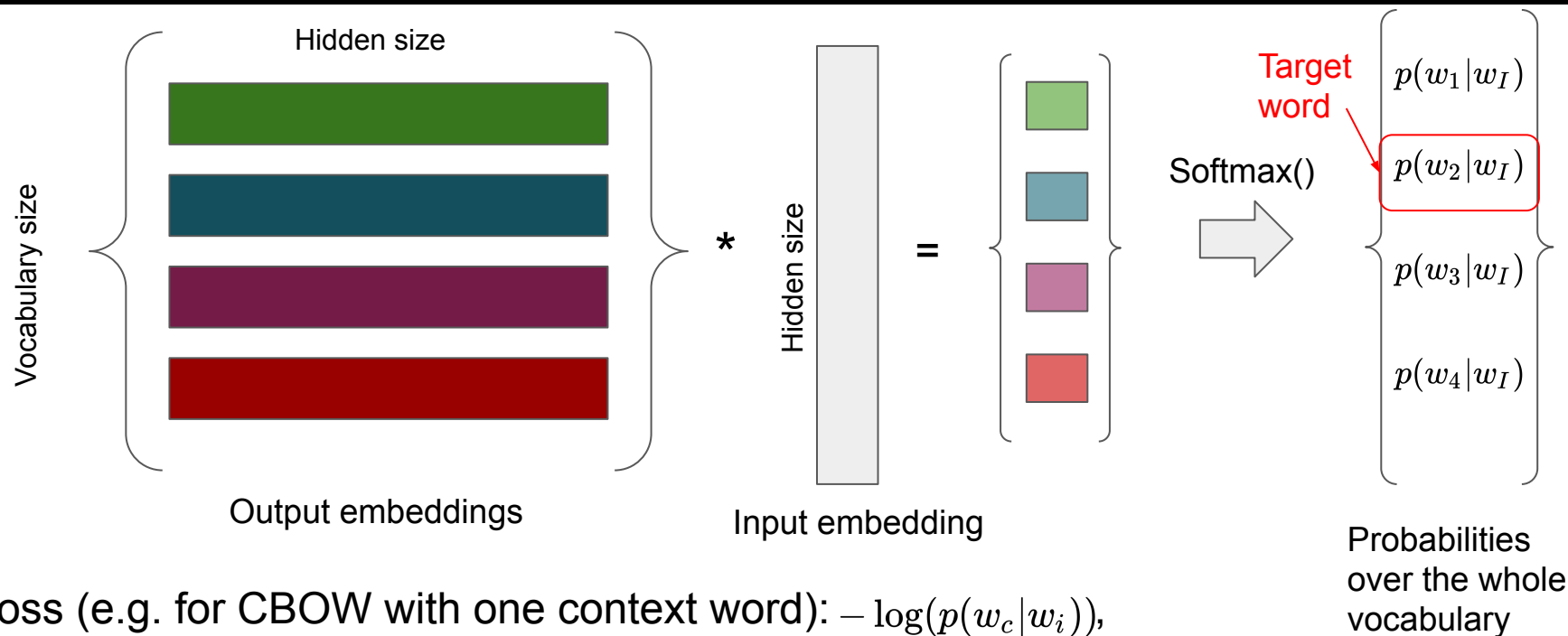
$$u_{c,j} = u_c = \mathbf{v}'_{w_j} \cdot \mathbf{h}, \text{ for } c = 1, 2, \dots, C$$



# Word2vec Parameter learning explained

As it can be seen from our derivations, when the vocabulary size is extremely large, calculating the denominator by going through all the words for every single sample is computationally impractical. The demand for more efficient conditional probability estimation leads to the new methods like hierarchical softmax, Negative Sampling, etc.

# Regular output layer



Loss (e.g. for CBOW with one context word):  $-\log(p(w_c | w_i))$ ,  
where  $w_c$  is the correct output,  $w_I$  is the input word.



Problem:

We need to calculate the dot product for each vocabulary item in order to compute a single probability value, thus complexity of the output layer is:  $H \times V$

We also need to update all the output vectors in the vocabulary after each training instance. The vocabulary can be very large.

Solutions:

1. Hierarchical softmax
2. Negative sampling

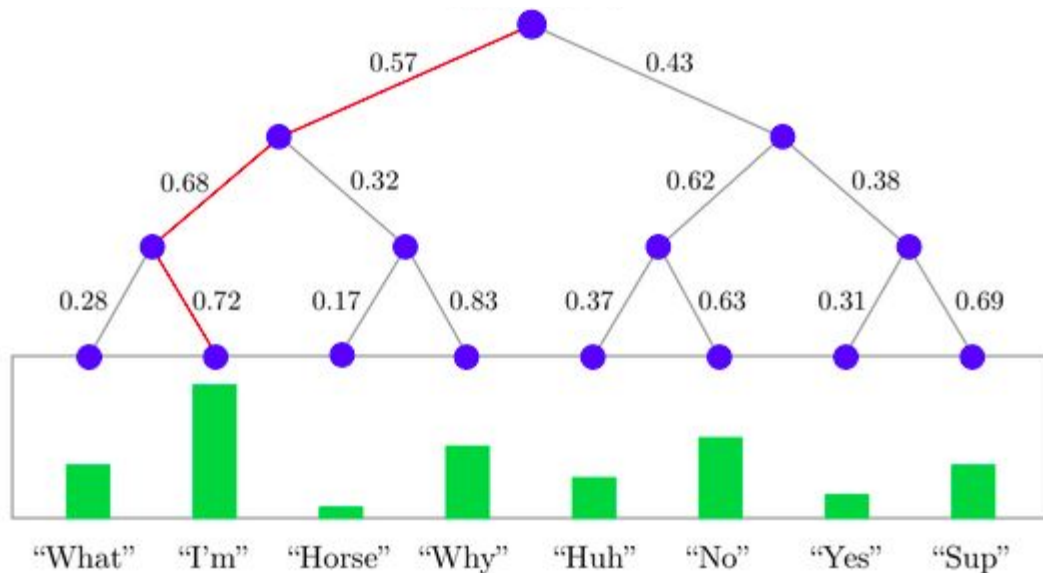
# Hierarchical softmax

Represent the vocabulary as a binary tree.

At each node the children probabilities sum to 1.

To compute a probability of the word we multiply the values of all nodes on the path to the word:  $\log_2(V)$  nodes

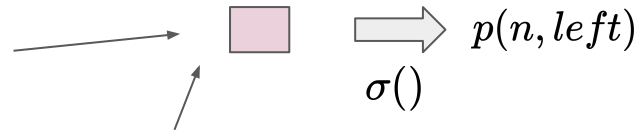
Probabilities of all words sum to 1



$$p(I'm|w_I) = 0.57 * 0.68 * 0.72 = 0.28$$

# Hierarchical softmax

input embedding



$$\sigma(x) = \frac{1}{1+e^{-x}}$$

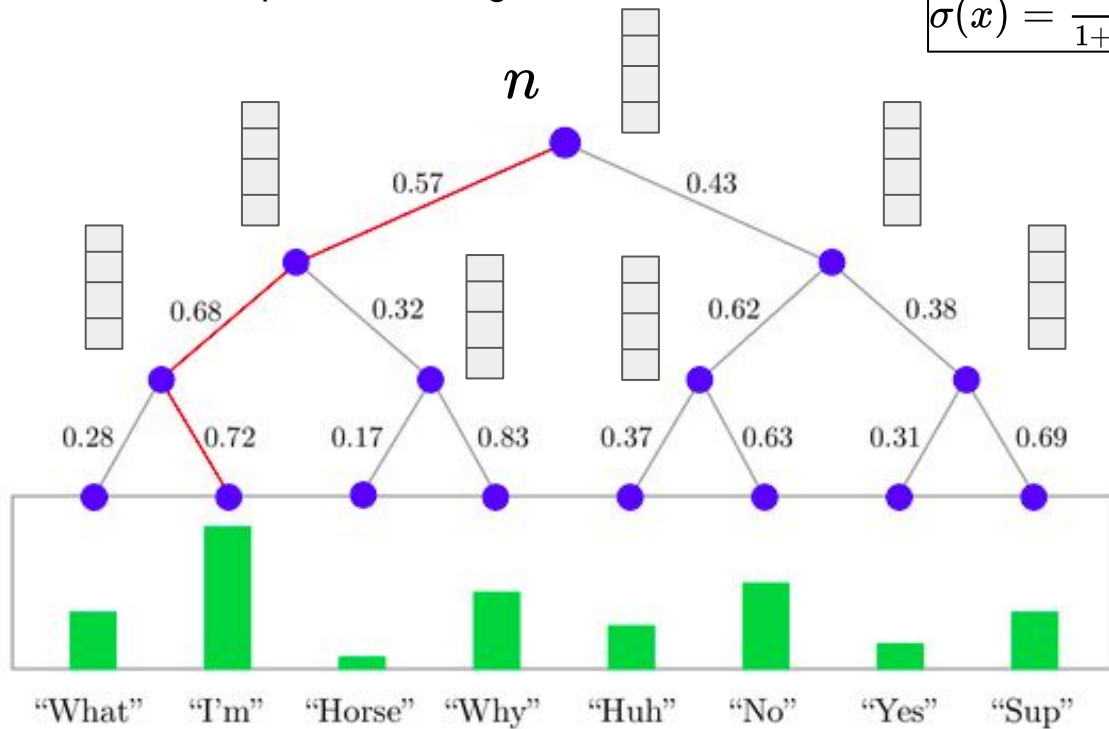
There is a vector representation for each inner node, together V-1 vectors.

To obtain children score at each node we calculate the dot product between the input embedding and the node vector.

The value is multiplied by -1 if we go to the right.

The sigmoid  $\sigma()$  function returns a probability:

$$\sigma(x) + \sigma(-x) = 1$$



# Hierarchical softmax

The loss function is then:  $-\sum_{j=2}^L \log p(\text{node}_j)$

where  $L$  is the length of the path and  $\text{node}_j$ ,  $j \in 2, \dots, L$  belongs to the path that leads to the target word.

We update only the representations of the nodes on the path.

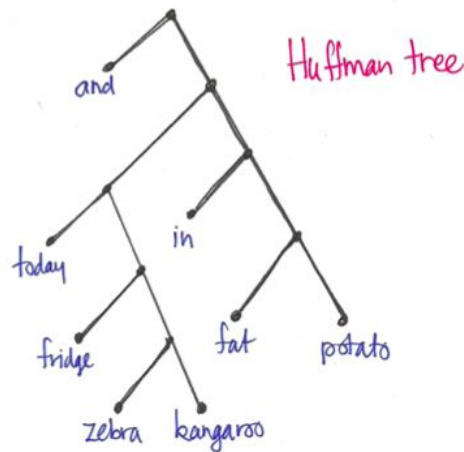
Thus the computational complexity for each training instance is hidden size by length of the path:  $H \times \log_2(V)$

In the test time we still need to compute probabilities for each leaf.

# Further speedup. Huffman tree

Assigns short paths to the frequent words. Average length of the path is reduced to unigram entropy of the corpus:  $-\sum_w f(w) \log_2 f(w)$

word	count
fat	3
fridge	2
zebra	1
potato	3
and	14
in	7
today	4
kangaroo	2



# Noise Contrastive Estimation

Turn the multiclass prediction problem into a binary classification problem: differentiate the true data from noise.

For each training sample the classifier is fed a training pair with a correct output and a number of training pairs with wrong outputs.

Instead of predicting the word given context or context given word, the model is trained to predict whether the training pair (word,context) is good or bad.

apple -> juice    apple -> house    apple -> foot

# Negative sampling

**Update only a sample of output vectors.**

Original loss: 
$$-\log(p(w_c|w_i)) = -\log\left(\frac{\exp(u_i)}{\sum_j \exp(u_j)}\right) = -u_i + \log \sum_j \exp(u_j)$$

$u_i$  is the dot product of the input embedding with the correct output embedding and  $j$  is an index over the vocabulary

Negative sampling: 
$$-\log \sigma(u_i) - \sum_{j \in W_{neg}} \log \sigma(-u_j)$$

$W_{neg}$  is the set of indices corresponding to  $K$  words that are sampled from the vocabulary (negative samples).

# Distributed representations of Words and Phrases and their Compositionality



# Content of that paper

“Distributed representations of Words and Phrases and their Compositionality”

# Content of that paper

## “**Distributed representations of Words** and Phrases and their Compositionality”

- Word2Vec (in this paper: Skip-gram)
  - Extensions: Hierarchical Softmax, Negative Sampling, Subsampling
  - Comparing performances

# Content of that paper

“Distributed representations of Words and **Phrases** and their Compositionality”

- Word2Vec (in this paper: Skip-gram)
  - Extensions: Hierarchical Softmax, Negative Sampling, Subsampling
  - Comparing performances
- Applications of Skip-gram
  - Learning Phrases

# Content of that paper

“Distributed representations of Words and Phrases and their **Compositionality**”


- Word2Vec (in this paper: Skip-gram)
  - Extensions: Hierarchical Softmax, Negative Sampling, Subsampling
  - Comparing performances
- Applications of Skip-gram
  - Learning Phrases
  - Compositionality


# Content of that paper

“Distributed representations of Words and Phrases and their **Compositionality**”

- Word2Vec (in this paper: Skip-gram)
  - Extensions: Hierarchical Softmax, Negative Sampling, Subsampling
  - Comparing performances
- Applications of Skip-gram
  - Learning Phrases
  - Compositionality
- Comparison to other models

# Subsampling

- Idea: different amount of information carried by frequent vs. infrequent words
  - Influence on co-occurrences
  - “The” + “France”  “Paris” + “France”

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

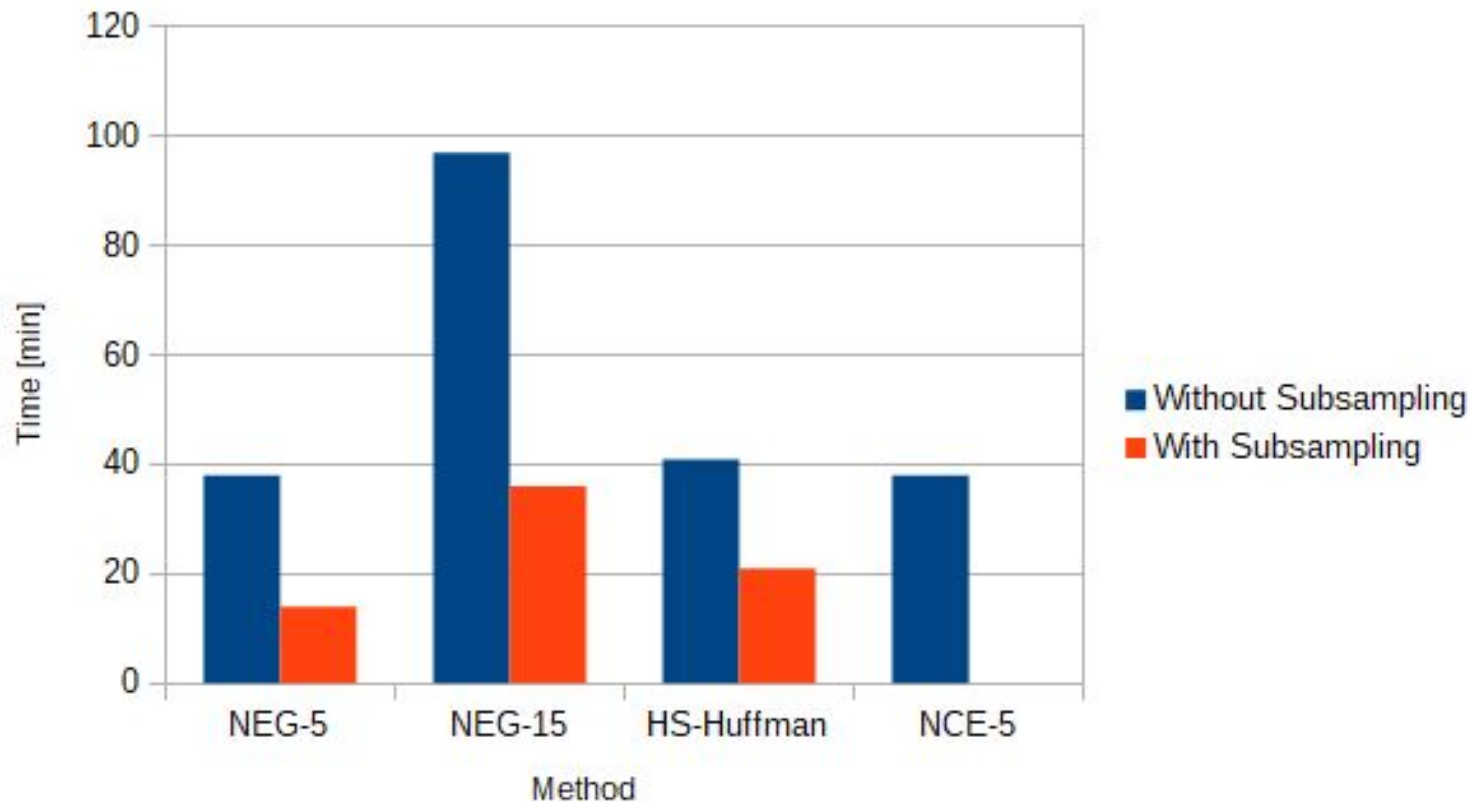
t: threshold, around  $10^{-5}$

- Frequent words have *steady* vector representations
  - they don't change after training on millions of examples

# Subsampling

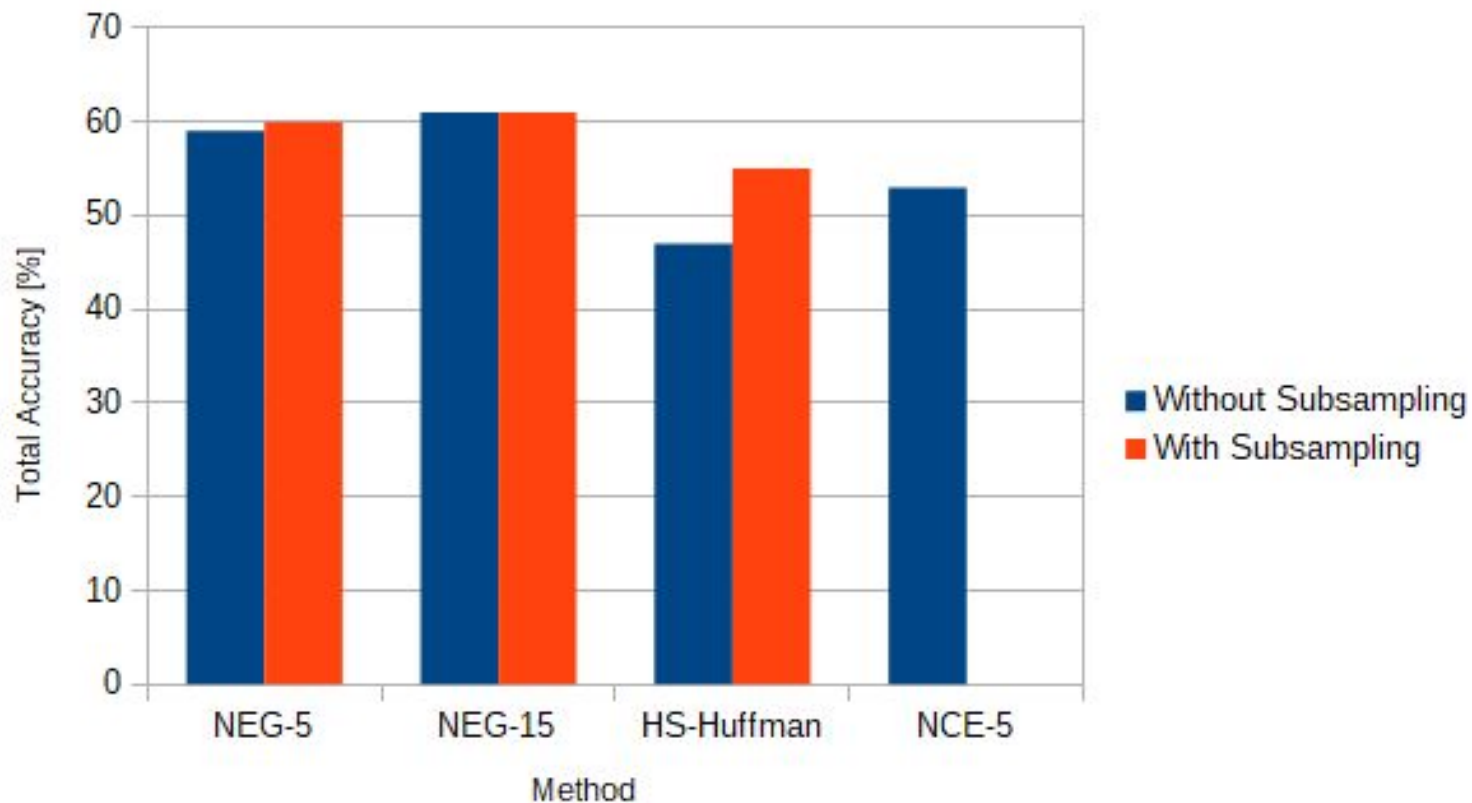
- Promise of subsampling:
  - Reducing time of learning
  - Improving accuracy of learned vectors

# Subsampling - Reducing time of learning?

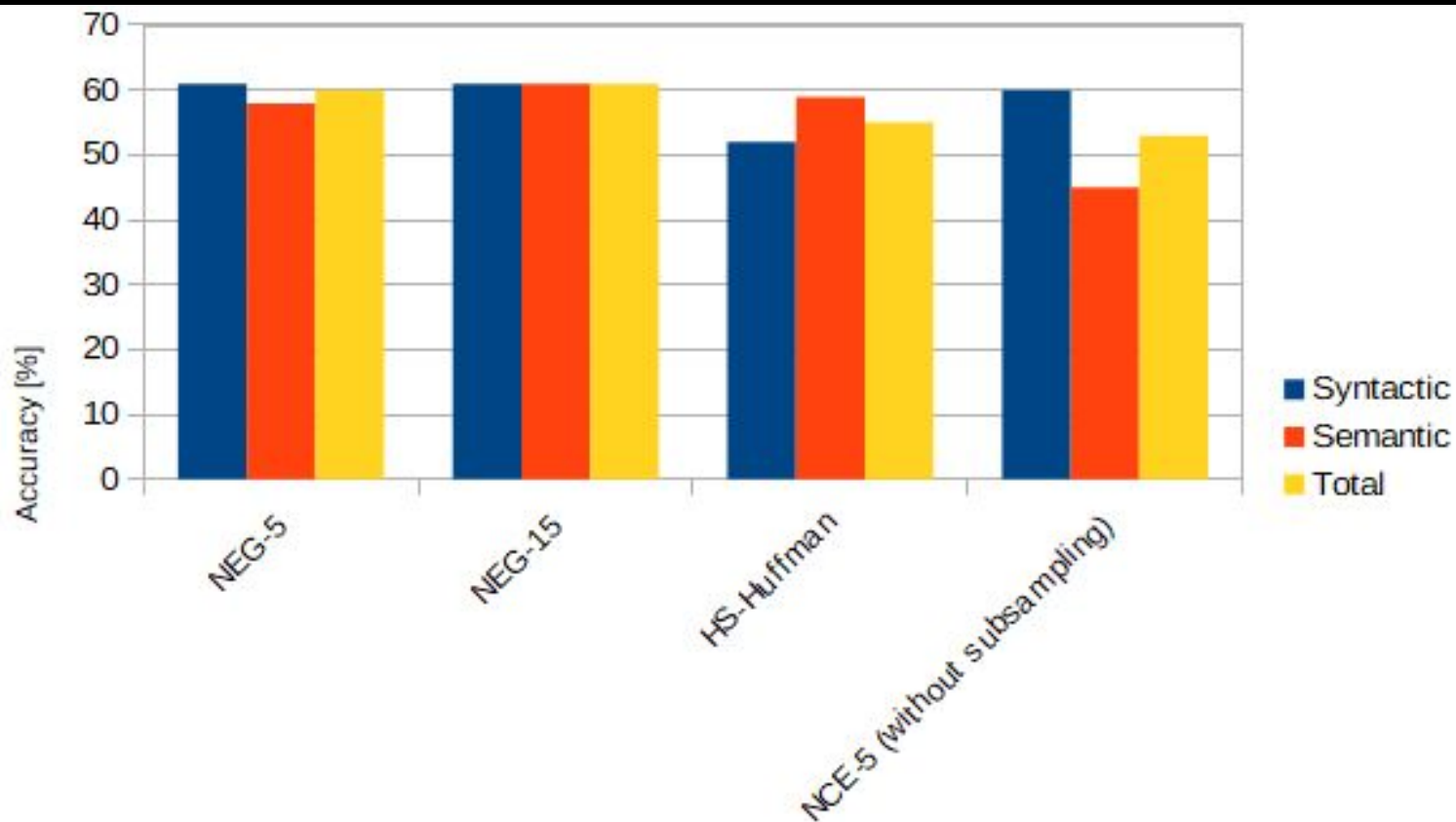




# Subsampling - Improving accuracy?



# General Accuracy Comparison



# Table of Contents

1. Introduction
2. Previous Work
3. Recap on Neural Networks
4. Word2Vec
- 5. Applications**
6. Summary

# Useful applications of Skip-Gram

- Learning Phrases
  - “Toronto Maple Leafs” = ....?




# Useful applications of Skip-Gram

- Learning Phrases
  - “Toronto Maple Leafs” = ....?



# Useful applications of Skip-Gram

- Learning Phrases
  - Finding them with data-driven approach
  - Unigram and Bigram Counts

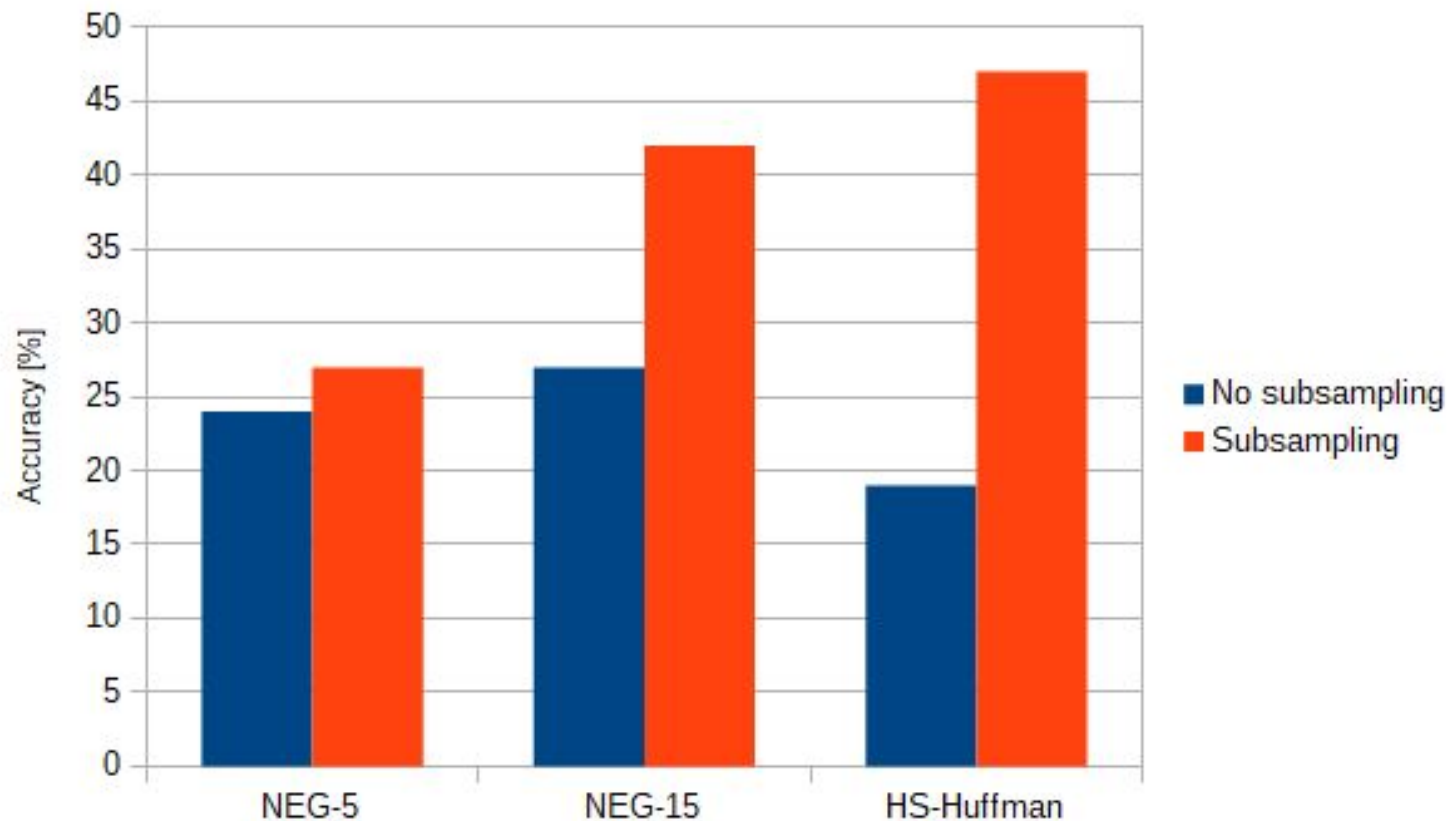

$$\text{score}(w_i, w_j) = \frac{\text{count}(w_i w_j) - \delta}{\text{count}(w_i) \times \text{count}(w_j)}$$

- Analogical reasoning task to test
  - *What word is in the same relationship to “big”, what “smaller” is to “small”? => Ideally, “bigger”.*

Newspapers			
New York San Jose	New York Times San Jose Mercury News	Baltimore Cincinnati	Baltimore Sun Cincinnati Enquirer
NHL Teams			
Boston Phoenix	Boston Bruins Phoenix Coyotes	Montreal Nashville	Montreal Canadiens Nashville Predators
NBA Teams			
Detroit Oakland	Detroit Pistons Golden State Warriors	Toronto Memphis	Toronto Raptors Memphis Grizzlies
Airlines			
Austria Belgium	Austrian Airlines Brussels Airlines	Spain Greece	Spainair Aegean Airlines
Company executives			
Steve Ballmer Samuel J. Palmisano	Microsoft IBM	Larry Page Werner Vogels	Google Amazon

Table 2: Examples of the analogical reasoning task for phrases (the full test set has 3218 examples). The goal is to compute the fourth phrase using the first three.

# Performance Comparison for Phrases





# Useful applications of Skip-Gram

- Learning Phrases
  - Increasing amount of training data has significant effect:
    - 1 billion words, 300 dimensions, context size = 5
      - 47% accuracy
    - 33 billion words, 1000 dimensions, context = whole sentence
      - 72% accuracy

# Useful applications of Skip-Gram

- Additive Compositionality
  - $\text{vector}(\text{"king"}) - \text{vector}(\text{"man"}) + \text{vector}(\text{"woman"})$ 
    - $\text{vector}(\text{"queen"})$
  - $\text{vector}(\text{"Russian"}) + \text{vector}(\text{"river"})$ 
    - $\text{vector}(\text{"Volga River"})$



*Combining* words by adding their vector representations

# Vector Compositionality- is it possible?

## Czech + currency

Koruna  
Czech crown  
Polish zolty  
CTK

## German + airlines

airline Lufthansa  
carrier Lufthansa  
flag carrier Lufthansa  
Lufthansa

## Russian + River

Moscow  
Volga River  
upriver  
Russia

- Reasonable results

# Useful applications of Skip-Gram

- Additive Compositionality
  - $\text{vector}(\text{"Russian"}) + \text{vector}(\text{"river"})$ 
    - $\text{vector}(\text{"Volga River"})$
  - Combining words by adding their vector representations
  - Word vectors represent distributions of **context**
  - Logarithmic connection to probabilities in output layer
    - Going from a product  $v_1 * v_2$   
to a sum  $\log(p_1) + \log(p_2)$

# Comparing Performance on Phrases

	Collobert (50d) (2 months)	Turian (200d) (few weeks)	Mnih (100d) (7 days)	<b>Skip-Phrase (1000d, 1 day)</b>
<b>Redmond</b>	conyers lubbock keene	McCarthy Alston Cousins	Podhurst Harlang Agarwal	<b>Redmond Wash. Redmond Washington Microsoft</b>
<b>graffiti</b>	cheesecake gossip dioramas	Gunfire Emotion impunity	anaesthetics monkeys Jews	<b>spray paint grafitti taggers</b>
<b>capitulate</b>	abdicate accede rearm	- - -	Mavericks planning hesitated	<b>capitulation capitulated capitulating</b>

# Exploiting Similarities among Languages for Machine Translation

# Main Idea

- To develop a method that can automate the process of generating and extending dictionaries and phrase tables.
- Distributed representations of words and phrases.
- Two main steps to the proposed method:
  - Building monolingual models of languages using large amounts of text.
  - Using a small bilingual dictionary to learn a linear projection between the languages.

# Dictionaries

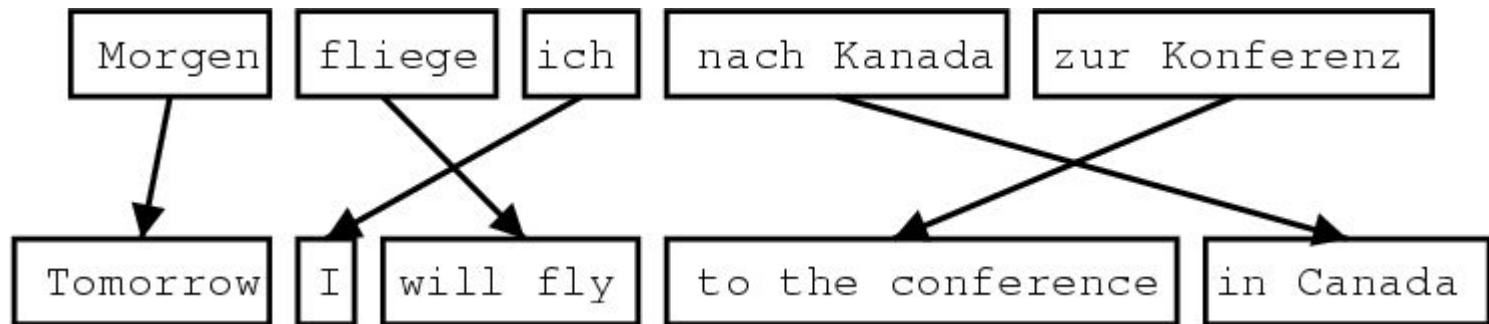
A



あ



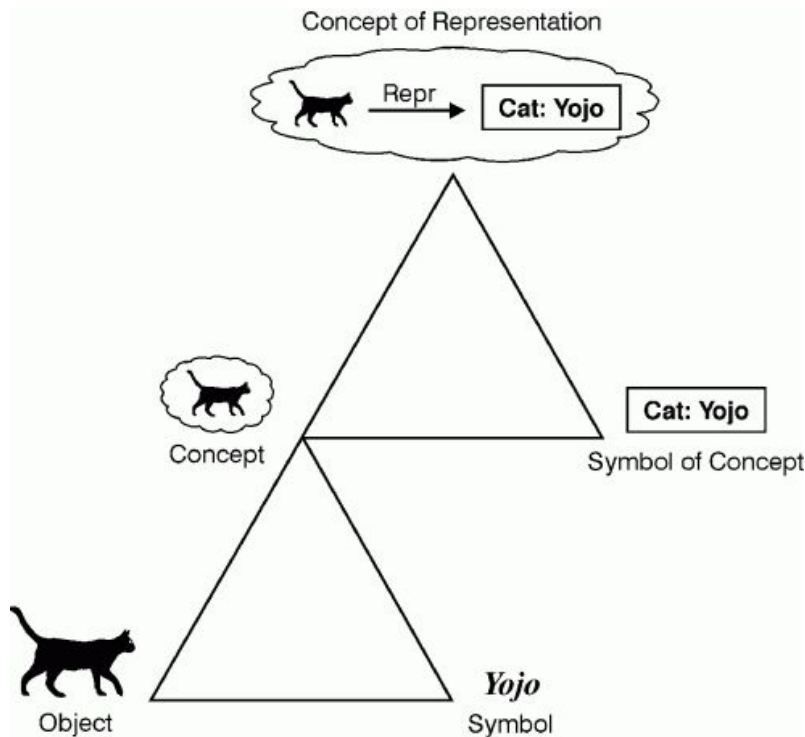
# Phrase Tables



# Method

- Skip-gram and continuous bag-of-words (CBOW)
- Train models to learn the linear projection of vector spaces between each language.
- Linear projections are learned using a bilingual dictionary after building monolingual models.
- If the word was seen in the monolingual corpora, it can be translated by projecting its vector representation from the source language space to the target language space.
- The translation is the most similar word vector.

# Why does this method work well?



- Common languages share concepts that are grounded in the real world.
- This means that there is strong similarity between the vector spaces, which means the geometric arrangements are similar.

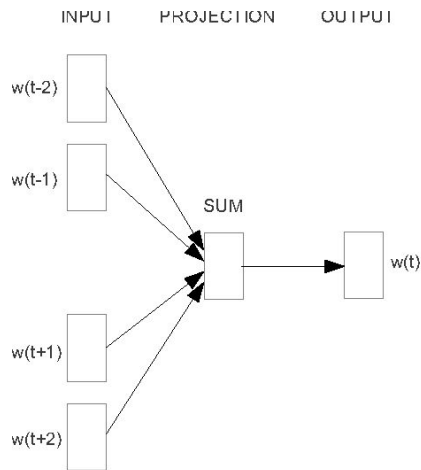
# Other Methods

- Extraction of morphological features.
- Exact context matches to infer the possible translations.
- Works well for similar languages (English and Spanish), but not for substantially different languages (English to Chinese).

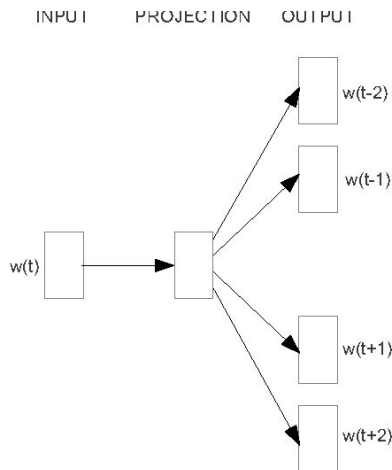
# Advantages of the Proposed Method

- Translation score for every word pair.
  - Can be used to improve the phrase tables.
    - Add better translations.
    - Filter out the errors.

# CBOW and Skip-gram Models



**CBOW**



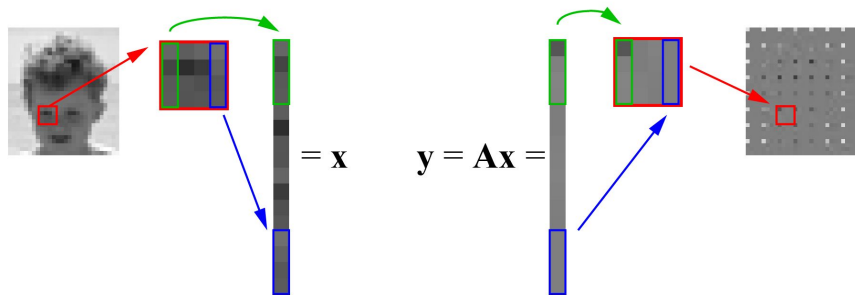
**Skip-gram**

- CBOW- to combine the representations of surrounding words to predict the word in the middle.
  - Better for larger data sets.
  - Faster and more reliable.
- Skip-gram- to learn word vector representations that are good at predicting its context in the same sentence.
  - Better for smaller data sets.
  - Better word representations.
- Both learn very similar representations for languages.

# CBOW and Skip-gram Models

- Both models are typically trained using the stochastic gradient descent.
  - The gradient is calculated using the backpropagation rule.
- Capture semantic information: school and university, lake and river, etc.
- Also capture relationships between concepts:  $\text{vector}(\text{France}) - \text{vector}(\text{Paris})$ ,  $\text{vector}(\text{Italy}) - \text{vector}(\text{Rome})$ ,

# Linear Relationships Between Languages



- Linear mapping can capture the relationship between vector spaces.
- If we know the translation for one word, we can learn the other translations to form a transformation matrix.



# Translation Matrix

- Distributed representation of word  $i$  in the source language  $x_i$
- Vector representation of its translation  $z_i$
- Goal: find a transformation matrix  $W$  such that  $W_{x_i}$  approximates  $z_i$

# Translation Matrix

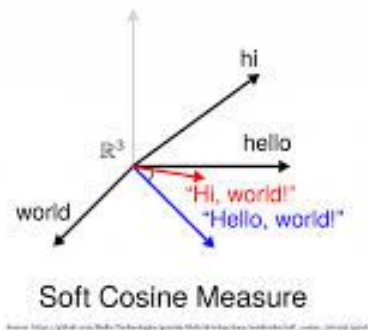
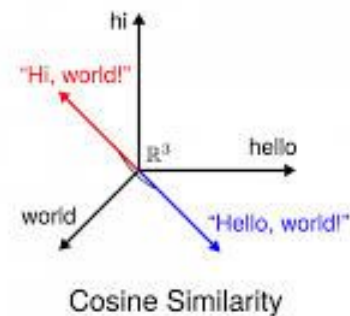
To learn  $W$ , we can use this optimization problem

$$\frac{\min}{w} \sum_{i=1}^n ||W_{x_i} - z_i||^2$$

# Translation Matrix

- Use equation to calculate  $W$ .
- For prediction, for any given new word and its continuous vector representation  $x$ , we can compute  $z = Wx$
- Find the word closest to  $z$  in the target language space using cosine similarity.

## Translation Matrix



- Cosine similarity- how similar the documents are irrespective of their size.
- Measure the cosine of the angle between two vectors projected in a multi-dimensional space.

# Experiments

- WMT11 Datasets
- Baseline techniques (similar to previously described experiments)
- English, Spanish, and Czech data
- Idea is to test if the method can provide non-obvious translations of words.
- Most frequent words and Google Translate.

# WMT11 Experiment

- Most frequent words that occurred at least five times in the corpus as training data.
- Most frequent words that occurred at least once and translations as test set.
- Top 5 accuracy and top 1 accuracy.
  - Top 1 accuracy is only counted if there is an exact match.

# WMT11 Experiment Continued

- Two baselines:
  - Edit distance- morphological structure of words.
    - Method to quantify how dissimilar two words are by counting the minimum number of operations needed to translate the words.
  - Similarity of word co-occurrences- similar to the neural network method proposed in the paper.
    - Form count-based word vectors equal to the size of the dictionary.
    - Map the word count vectors from source to target.
    - For each test word, search for the most similar vector in the target language.

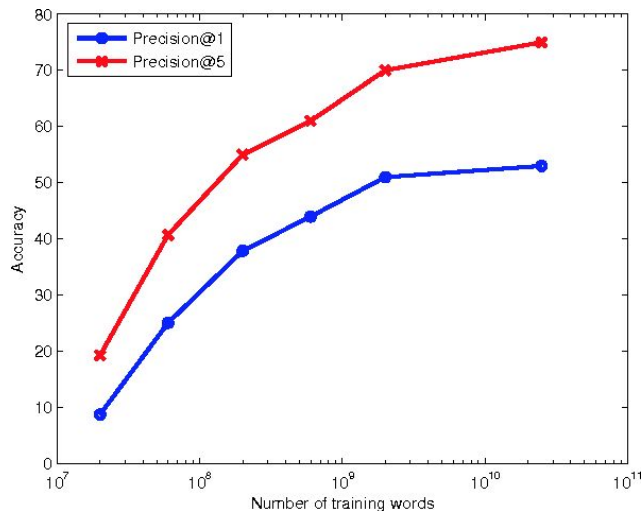
# Results of the WM11 Experiment

Translation	Edit Distance		Word Co-occurrence		Translation Matrix		ED + TM		Coverage
	P@1	P@5	P@1	P@5	P@1	P@5	P@1	P@5	
En → Sp	13%	24%	19%	30%	33%	51%	43%	60%	92.9%
Sp → En	18%	27%	20%	30%	35%	52%	44%	62%	92.9%
En → Cz	5%	9%	9%	17%	27%	47%	29%	50%	90.5%
Cz → En	7%	11%	11%	20%	23%	42%	25%	45%	90.5%

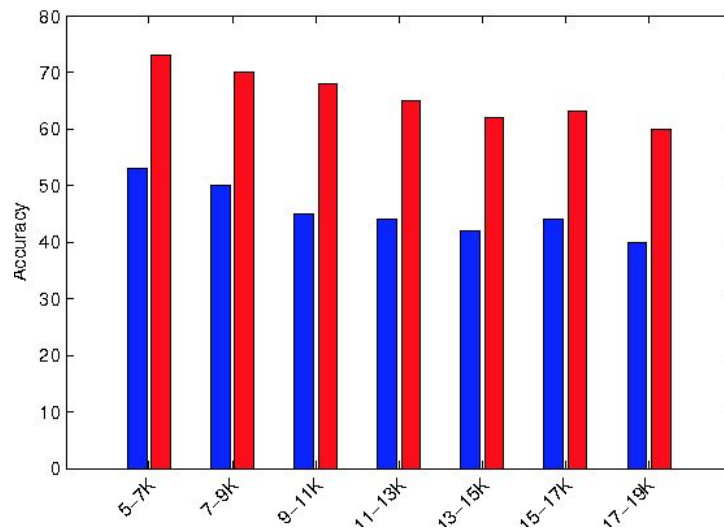


# Results from Larger Scale Experiments

- Larger corpus- billions of words.
- Accuracy increases as more words are available to be trained.



- The approach is also successful for infrequent words.



# Using Distances as Confidence Measure

Threshold	Coverage	P@1	P@5
0.0	92.5%	53%	75%
0.5	78.4%	59%	82%
0.6	54.0%	71%	90%
0.7	17.0%	78%	91%

Threshold	Coverage	P@1	P@5
0.0	92.5%	58%	77%
0.4	77.6%	66%	84%
0.5	55.0%	75%	91%
0.6	25.3%	85%	93%

- Higher accuracy but lower coverage.
- Distance between computed vector and the closest word vector as confidence measure.
- Confidence score:  $\max_{i \in V} \cos(Wx, z_i)$
- If confidence score is lower than a threshold, the translation is skipped.
- Edit distance helps accuracy.

# More Examples

- Spanish to English
  - Translation matrix alone.
  - Translations mostly correct, with some errors.
    - Emociones -> emotions = correct.
    - Protegida -> wetland, undevelopable = incorrect.
  - Errors tend to be somewhat meaningful and are semantically related.
- English to Spanish
  - High confidence score ( $> 0.5$ )
  - Edit distance and translation matrix.
    - Unacceptable -> inaceptable = correct.
    - Beneficial -> beneficioso = correct.

# Other Uses

- Correction of dictionary errors.
  - Distance between the original dictionary entry and system output is large -> most likely an error.
    - Said -> said -> listed = incorrect.
    - Will -> can -> testament = incorrect.
  - These examples were manually selected, so it might be possible to automate this.
- Translation between non-similar languages.
  - English to Vietnamese.
  - Decent results, En-> Vn: 87.8% coverage, P@1: 10%, P@5: 30%
  - Vn-> En: 87.8% coverage, P@1: 24%, P@5: 40%
  - Edit distance didn't significantly improve results.

# Table of Contents

1. Introduction
2. Previous Work
3. Recap on Neural Networks
4. Word2Vec
5. Applications
6. **Summary**

# Summary

- Word2vec is a relatively simple but can be used to learn word vectors, capturing syntax and semantics.

# Literary Sources

- Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *Proceedings of the Workshop at International Conference on Learning Representations (ICLR)*. Pages 1-12.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *Advances in Neural Information Processing Systems 26*. Pages 3111-3119.
- Xin Rong. 2014. word2vec Parameter Learning Explained. *arXiv*
- Tomas Mikolov, Quoc V. Le and Ilya Sutskever. 2014. Exploiting Similarities among Languages for Machine Translation. *CoRR*, *abs/1309.4168*.

# Extra Material

<http://ruder.io/word-embeddings-1/>

<http://ruder.io/word-embeddings-softmax/index.html>

<http://ruder.io/secret-word2vec/index.html>

<https://arxiv.org/pdf/1410.8251.pdf>

<http://onlinehub.stanford.edu/cs224> (L01, L02)

<http://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes01-wordvecs1.pdf>



# Image sources

- <https://previews.123rf.com/images/vvoennyy/vvoennyy1411/vvoennyy141100036/33528381-branch-with-yellow-and-orange-autumn-maple-leaves-isolated-on-white-background.jpg>
- <https://file.videopolis.com/F/1/64b57b57-c31b-4bdf-ac2d-bbdb25fd147d/101119.12849.toronto.intercontinental-toronto-centre.hero-iVqYJ5JD-45574-1280x720.jpeg>
- <https://s3951.pcdn.co/wp-content/uploads/2015/09/Toronto-Maple-Leafs-logo-2016-17.png>
- [https://en.wikipedia.org/wiki/Dictionary-based\\_machine\\_translation#/media/File:Translation\\_arrow.svg](https://en.wikipedia.org/wiki/Dictionary-based_machine_translation#/media/File:Translation_arrow.svg)
- <https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwiDguPiy6jiAhVRDewKHTDXB6kQjRx6BAgBEAU&url=https%3A%2F%2Fwww.statmt.org%2Fwpt05%2Fmt-shared-task%2F&psig=AOvVaw3Us4FkH2S8P2C775RBTlyb&ust=1558389592201203>
- <https://www.learnopencv.com/understanding-feedforward-neural-networks/>
- [http://immersivemath.com/ila/ch09\\_linear\\_mappings/compression\\_crop.png](http://immersivemath.com/ila/ch09_linear_mappings/compression_crop.png)
- <https://www.machinelearningplus.com/wp-content/uploads/2018/10/soft-cosine.png>