Word Embeddings

Representation Learning for Words

Zhao Rui

Vector Semantic

Word Representation

- How to represent word in a vector space

apple	[0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0]
orange	[0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 ... 0 0 0 0 0 0]
car	[0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0]

One-hot Vector

- Can we use the above one-hot vector for words?

- Can not capture the semantics of the corresponding words

Vector Semantics

- Words are characterized by the words that occur with them.
- Words are close to each other in the vector space if they are semantically closer to each other.
- It is also called **distributional semantics**.

Motivations

- "**You shall know a word by the company it keeps**" - by Firth (1957)

- Example from Nida (1975); Lin(1998); Jurafsky (2015)

What is Tesgüino?

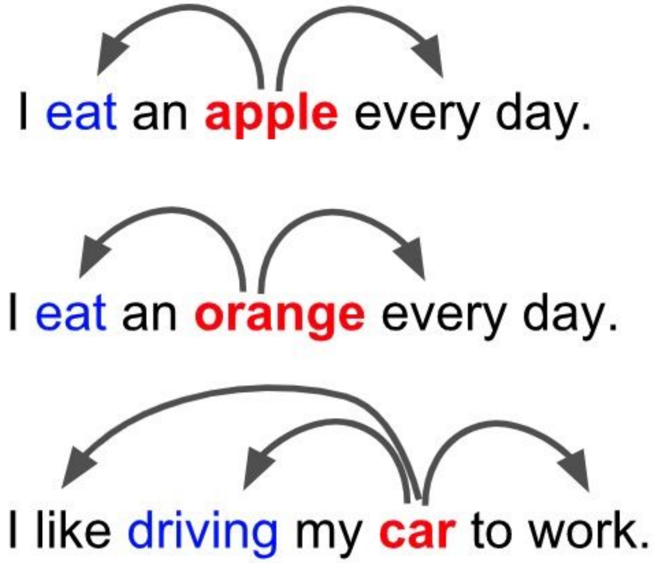
A bottle of Tesgüino is on the table.
Everybody likes tesgüino
Tesgüino makes you drunk
We make Tesgüino out of corn

- From context words, the meaning behind the word can be inferred as:
  - An alcoholic beverage like beer.

Distributional Semantics

- Words are represented by their context.
- Two words are similar if they have similar word contexts.

Context: Nearby Words



Bag-of-words

- We represent how often a word occurs in a document
    - Sklearn countvectorizer
    - It is called document-term matrix
- If we look at the column instead of the row

I eat an apple every day

I eat an orange every day

I like driving my car to work

	an	apple	car	day	driving	eat	every	like	my	orange	to	work
0	1	1	0	1	0	1	1	0	0	0	0	0
1	1	0	0	1	0	1	1	0	0	1	0	0
2	0	0	1	0	1	0	0	1	1	0	1	1

Term-Document Matrix

- Each document is a count vector in a vector space whose dimension is D
    - D is the number of documents
    - The shape of the matrix is |V| * D
    - Each row is the vector for the word
    - **Two words are similar if their vectors are similar**

	0	1	2
an	1	1	0
apple	1	0	0
car	0	0	1
day	1	1	0
driving	0	0	1
eat	1	1	0
every	1	1	0
like	0	0	1
my	0	0	1
orange	0	1	0

Word-word matrix

- Document is a kind of "context". However, it is too abstract.

- Smaller context will be better:
    - ***Window of k nearby words***, here k can be 2,3,4,..

- Instead of term-document matrix, we are going to have word-word matrix
    - Each word vector's dimension will be |V|
    - The matrix will be the shape of |V| * |V|

- To build the word-word matrix:
    - Co-occurrence: For a given corpus, the co-occurrence of a pair of words say w1 and w2 is the number of times they have appeared together in a Context Window .
    - Context Window: Context window is specified by a number and the direction (usually set to be left and right).

For example

- For context window: the window size is 2 and the direction is set to be right and left.

I eat an apple **every** day

I eat an orange **every** day

I like driving my car to work

	Target words	Context words	Co-occurrence Number
The co-occurrence:	(every,	an):	2
	(every,	apple):	1
	(every,	day):	2

Word-word matrix

- Size will be V * V.
- High-dimensional and very sparse
- Symmetry

Contexts

Targets

	like	an	to	my	driving	apple	orange	work	every	car	I	eat	day
like	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
an	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	2.0	0.0	2.0	2.0	0.0
to	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
my	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
driving	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
apple	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0
orange	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0
work	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
every	0.0	2.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	2.0
car	0.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
I	1.0	2.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0
eat	0.0	2.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	2.0	0.0	0.0
day	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	2.0	0.0	0.0	0.0	0.0

I eat an apple every day

I eat an orange every day

I like driving my car to work

```
vec_apple = mat[vocab.index('apple')].reshape(1, -1)
vec_orange = mat[vocab.index('orange')].reshape(1, -1)
vec_car = mat[vocab.index('car')].reshape(1, -1)
print('cosine scores between apple and orange vectors')
print(cosine_similarity(vec_apple, vec_orange))
print('cosine scores between apple and car vectors')
print(cosine_similarity(vec_apple, vec_car))
```

```
cosine scores between apple and orange vectors
[[1.]]
cosine scores between apple and car vectors
[[0.]]
```

The size of window

- Under different window sizes, we will have different word-word matrix

	like	an	to	my	driving	apple	orange	work	every	car	I	eat	day
like	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
an	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	2.0	0.0
to	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
my	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
driving	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
apple	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
orange	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
work	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
every	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	2.0
car	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
I	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0
eat	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
day	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0

Win size=1

	like	an	to	my	driving	apple	orange	work	every	car	I	eat	day
like	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
an	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	2.0	0.0	2.0	2.0	2.0
to	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
my	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0
driving	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
apple	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0
orange	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0
work	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
every	0.0	2.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	2.0	2.0
car	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
I	1.0	2.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	2.0	0.0
eat	0.0	2.0	0.0	0.0	0.0	1.0	1.0	0.0	2.0	0.0	2.0	0.0	0.0
day	0.0	2.0	0.0	0.0	0.0	1.0	1.0	0.0	2.0	0.0	0.0	0.0	0.0

Win size=3

- From Jurafasky (2015): the size of windows depends on your goals
    - The shorter the windows, the more syntactic the vector (1-3)
    - The longer the windows, the more semantic the representation (4-10)

Raw Count

- Raw word frequency is not a great measure of association between words
  - Very **skewed** distribution. For example, ***the*** and ***of*** are very frequent, but may not the most discriminative
  - Think about the following two cases: **(banana, monkey), (the, monkey)**

The measure should indicate whether is context word is **particularly informative** about the target word.

Pointwise Mutual Information (PMI)

- PMI
  - Do events x and y co-occur more than if they were independent?
  - Here, events will be words

- Usually, we adopt Positive PMI (PPMI)
  - Positive Pointwise Mutual Information (PPMI)

$$PPMI(w_1, w_2) = max(log_2 \frac{p(w_1,w_2)}{p(w_1)p(w_2)}, 0)$$

Positive PPMI: if PPMI is negative, make it zero

Penalize high-frequent words

Dense Vectors

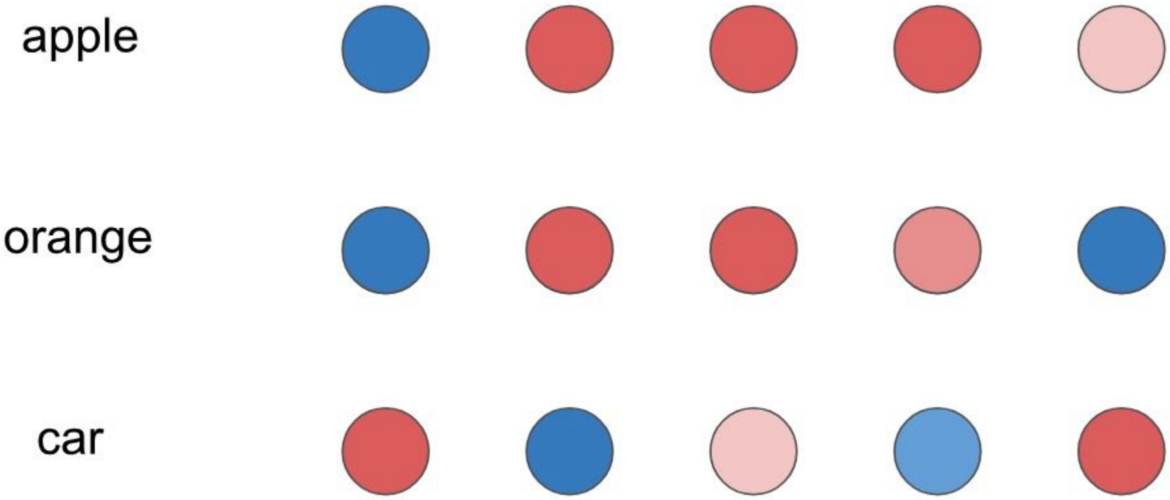
- Count-based or PPMI-based Vectors:
  - High dimensionality (|V| easily over 10,000)
  - Sparse

- Dense Vector:
  - Low dimensionality (from 50-300)
  - Dense

Dense vector: 1 Reduce overfitting (when they are used as features in downstream ML)
2 Each dimension in dense vector can contain more semantic information (like "topic")

Distributed Representation

- Words should be encoded into a **low-dimensional** and **dense** vector



From Sparse Vectors to Dense Ones

- Matrix decomposition can be applied on the word-word matrix.
- Singular Value Decomposition (SVD) is one of the classic methods.
    - Change the dimensions such that they are orthogonal to each other.
    - The new vector space will keep the first k dimensions that explain the largest amount of variance in the data.
    - Each new dimension is a linear combination of previous dimensions, given by the project matrix learned from SVD)

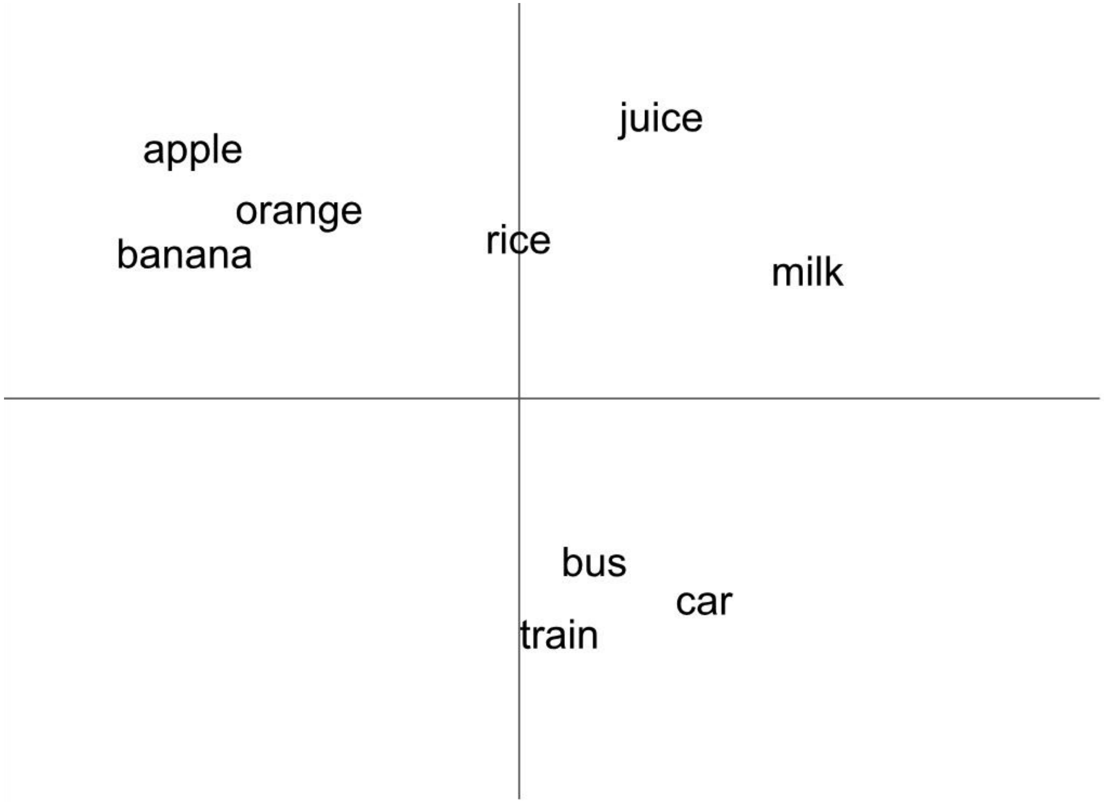
	dim0	dim1	dim2	dim3	dim4
like	-0.563652	1.497387	-0.423378	0.119592	-0.713812
an	-3.590090	-0.398018	-1.152087	1.707900	0.249389
to	-0.225862	1.595626	0.189236	-0.058551	-0.175054
my	-0.609500	2.030081	-0.494240	0.142318	0.746323
driving	-0.590864	1.794945	-0.501255	0.146783	-0.461930
apple	-2.011314	-0.236121	-0.338984	0.457370	-0.054726
orange	-2.011314	-0.236121	-0.338984	0.457370	-0.054726
work	-0.154674	1.260900	-0.017046	0.009616	-0.839717
every	-3.026532	-0.659660	1.762966	0.750429	-0.232352
car	-0.293771	1.874864	0.363469	-0.114090	0.852037
I	-2.620567	0.832339	2.084192	-0.552119	0.073469
eat	-3.081751	-0.211562	-1.373831	-1.499121	-0.042218
day	-2.363795	-0.593157	-0.158542	-1.849308	0.044472

When k is set to be 5 in our toy example.

Word Vectors

Project word vectors in a two-dimensional space. And visualize them!

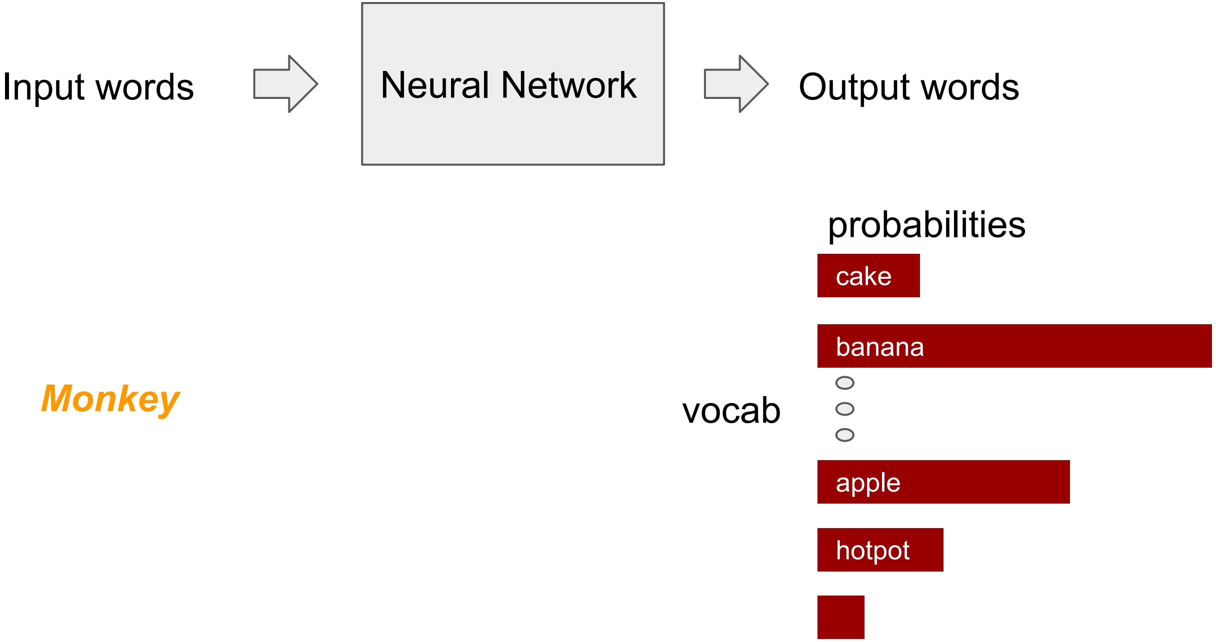
Similar words are close to each other.



Neural Word Embeddings

- Another approach is prediction based methods instead of matrix methods.
- We would like to build a machine learning model for the task that given target words, can we predict their context words? Or Given context words, can we predict their target words?
- Symmetric Matrix and Symmetric Tasks
- What is the most powerful supervised prediction model **given enough data**?
  - Neural network
- It is the Word2Vec model: a neural network based word embedding model.

Neural Network Solution



Input word and output words should be sampled from the same context

Another **self-supervised** learning example

Applications of Word Embeddings

Word Embeddings

- Word2vec, Glove, Fastext, and other open-source nlp methods can learn dense and low-dimensional vectors for words

- We can solve lots of word-level NLP problem.

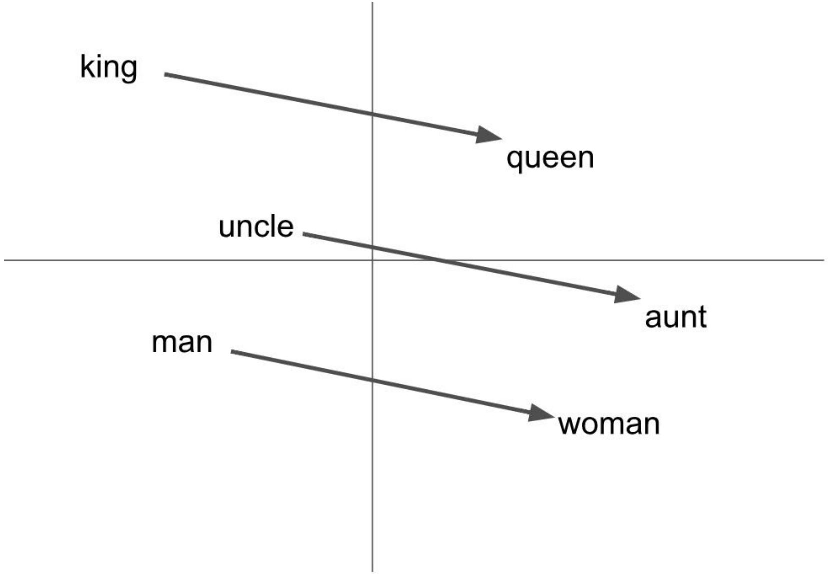
- Starting from word embeddings, we can learn vectors for higher-level natural language units such as sentences and documents.

Word Analogy

Man: Woman : : King :??

Find w to minimize:

$||V_{man} - V_{woman} + V_{king} - V_{w}||_2$



Mikolov & Chen et al. 2013
Mikolov & Sutskever et al. 2013

Expanding Knowledge Base

Discover “new” words in a category:

beijing
shanghai
tokyo
seoul
pyeongyang

(b) asian_city (15 words)

Given the list

(j) c = asian_city

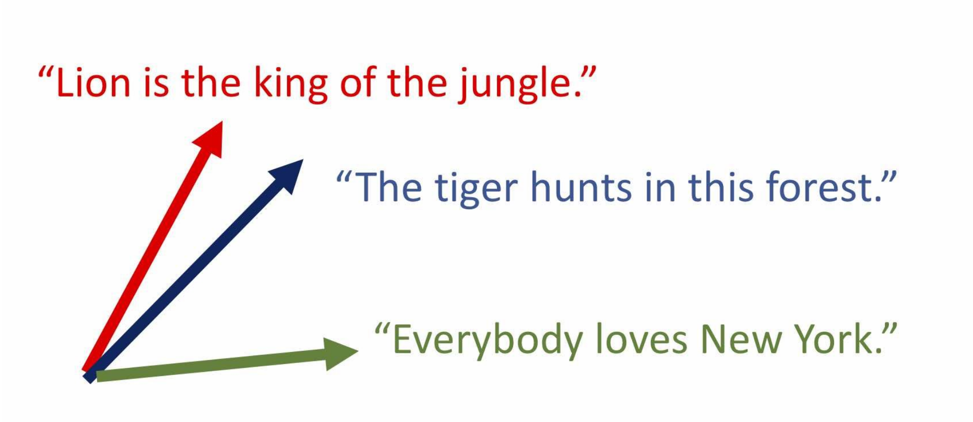
word	projection
taipei	0.837
taichung	0.819
kaohsiung	0.818
osaka	0.806
tianjin	0.765

Generate more examples

https://arxiv.org/pdf/1511.06961.pdf

Word Embedding Solution: Estimate “the best line” to capture the semantics behind the given words (rank 1 SVD on the embeddings), find other words whose embeddings are close to this line.

Sentence Embeddings

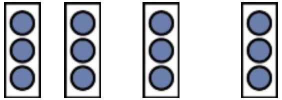


https://prakhartechviz.blogspot.com/2019/05/baseline-sentence-embeddings.html

Sequence of Words

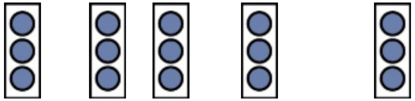
- Each sentence or document can be regarded as a sequence of vectors.

I hate this movie



4 by d

This is my favorite movie.



5 by d

- The shape of matrix depends on the length of sequence. However, the majority of ML systems need fixed-length feature vectors.
- One simple solution: average the sequence of vectors, just like bag-of-words (abandon order information).

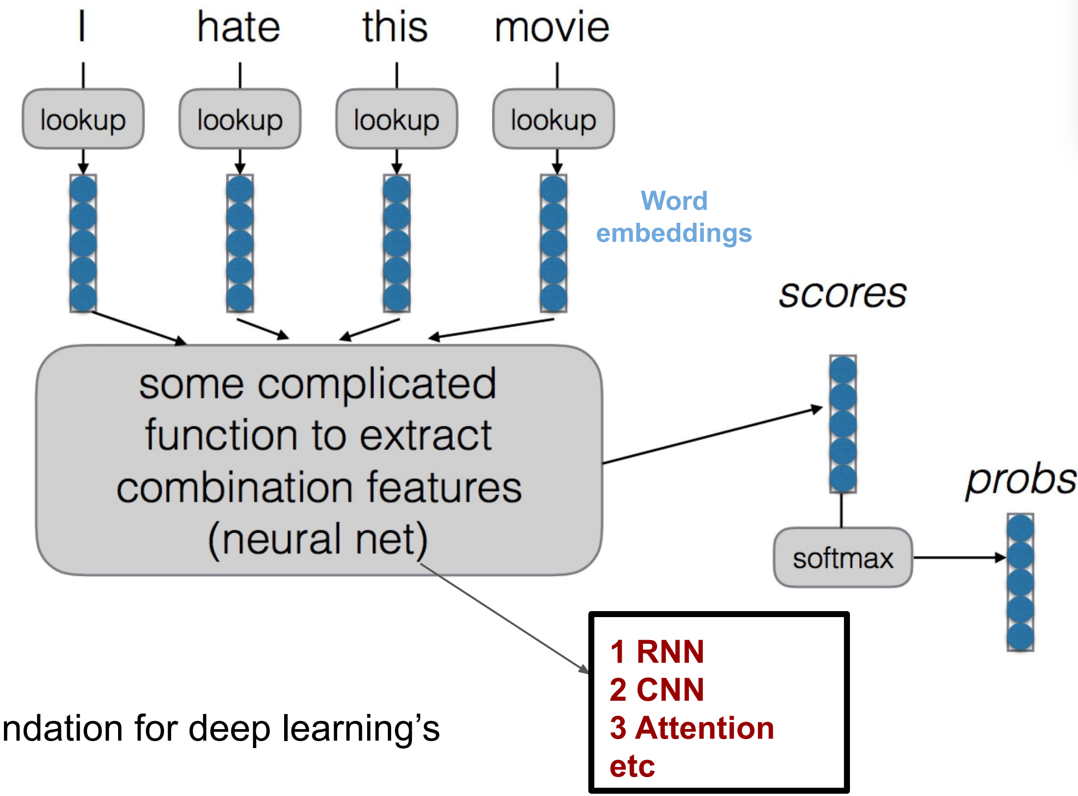
Complex Semantic

1. **Input Text:** a sequence of words;

2. **Through Word Embedding Look-up:** a sequence of word vectors;

3. Neural networks is applied upon the vector sequences to learn semantic **composition** for final prediction;

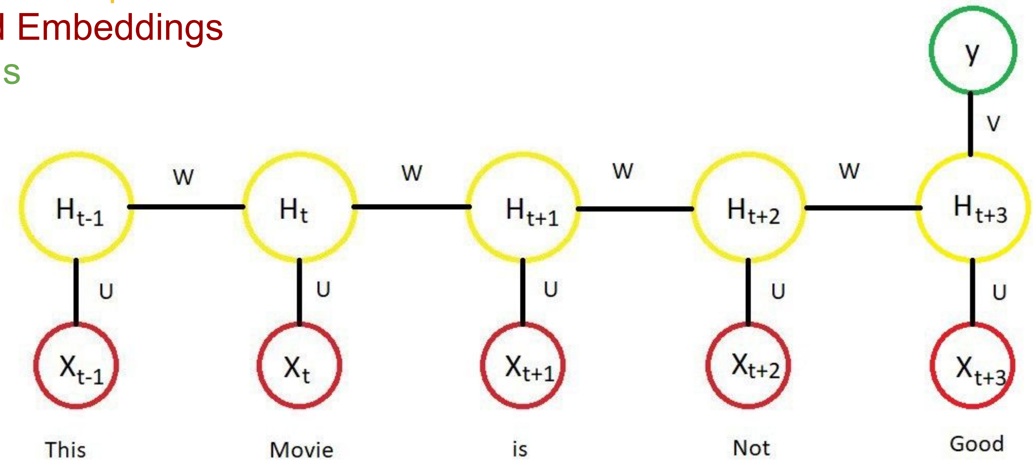
➔ Human understand the word meaning firstly, then get the whole sentence meaning by composing these words' meaning together.



Word Embeddings is the foundation for deep learning's applications on NLP

Recurrent Neural Network for NLP

U, W, V: RNN's parameters
H: Hidden Outputs
X: Word Embeddings
y: Labels



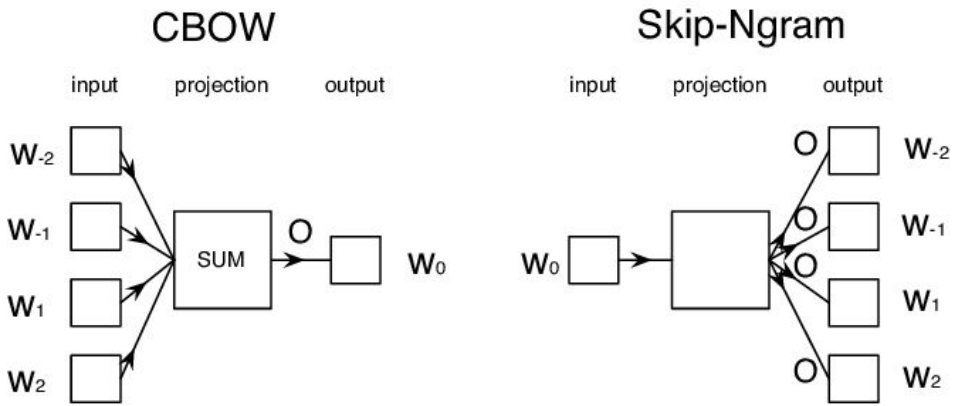
What is Word2Vec?

A Good Visualization for Word2Vec

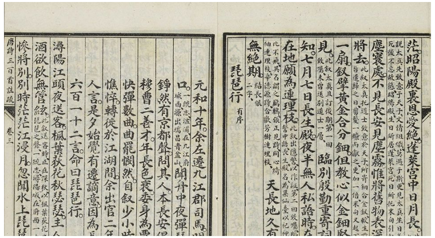
https://ronxin.github.io/wevi/

Word2Vec

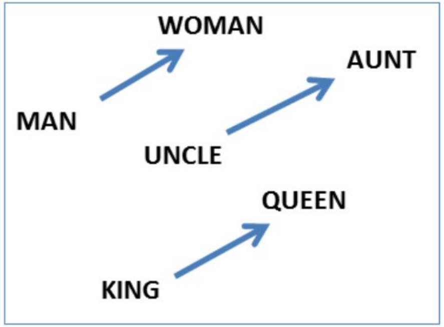
- A method of computing vector representation of words developed by Google.
- Open-source version of Word2Vec hosted by Google (in C)
- Train a simple neural network with a single hidden layer to perform word prediction tasks
- Two structures proposed **Continuous Bag of Words (cbow)** vs **skip-gram**:



Word2Vec as BlackBox



input, target

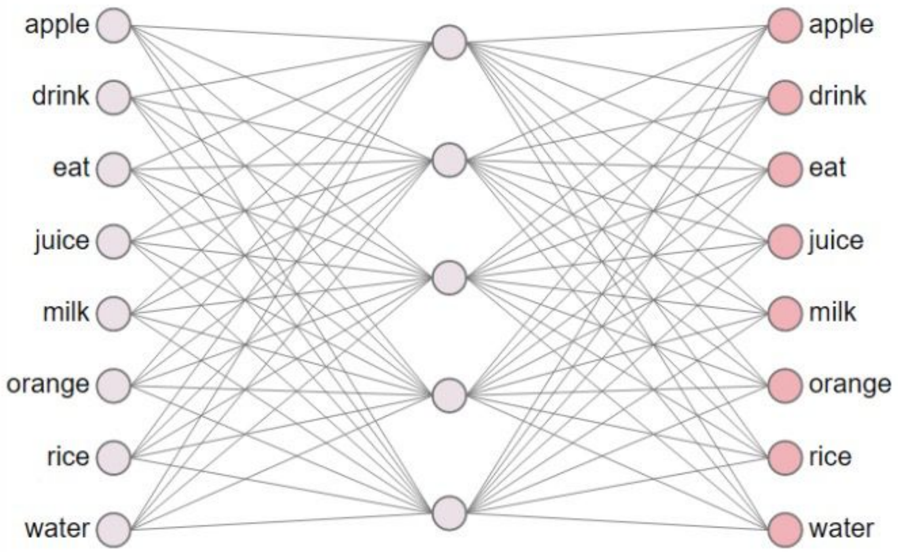


Corpus

Word2Vec Tool

Word Embeddings

Model Architecture



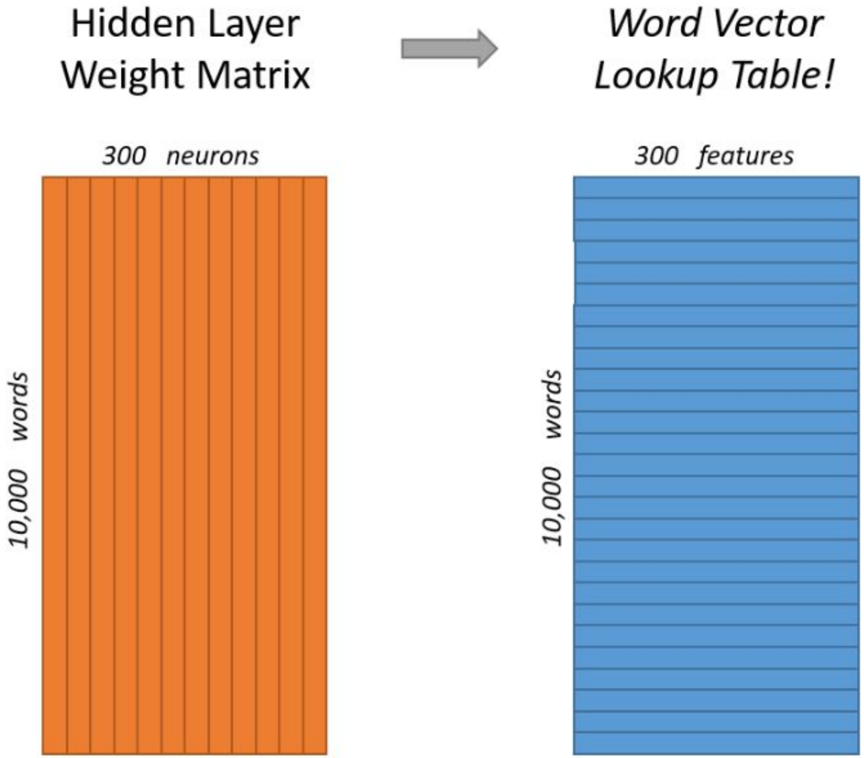
Structure Highlights:

- input layer
  - one-hot vector
- hidden layer
  - linear (identity)
- output layer
  - softmax

From Xin Rong 2016

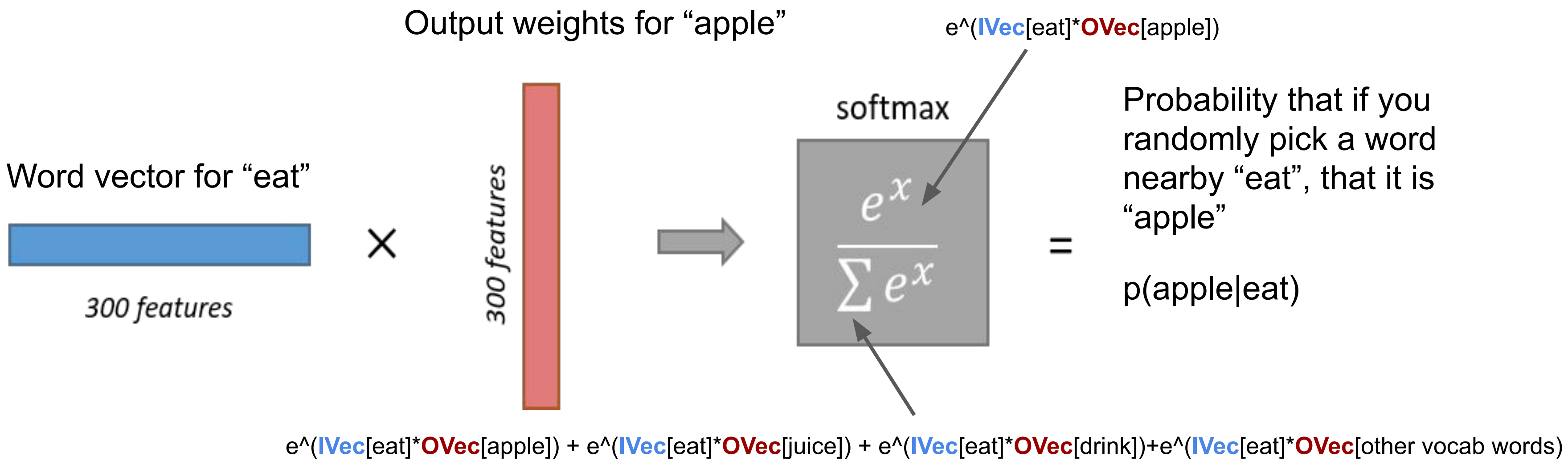
Hidden Layer

- linear-activation function here
- 300 neurons are the word vec. dimensions
- This layer is operating as a 'lookup' table
- Input word matrix denoted as **IVec**

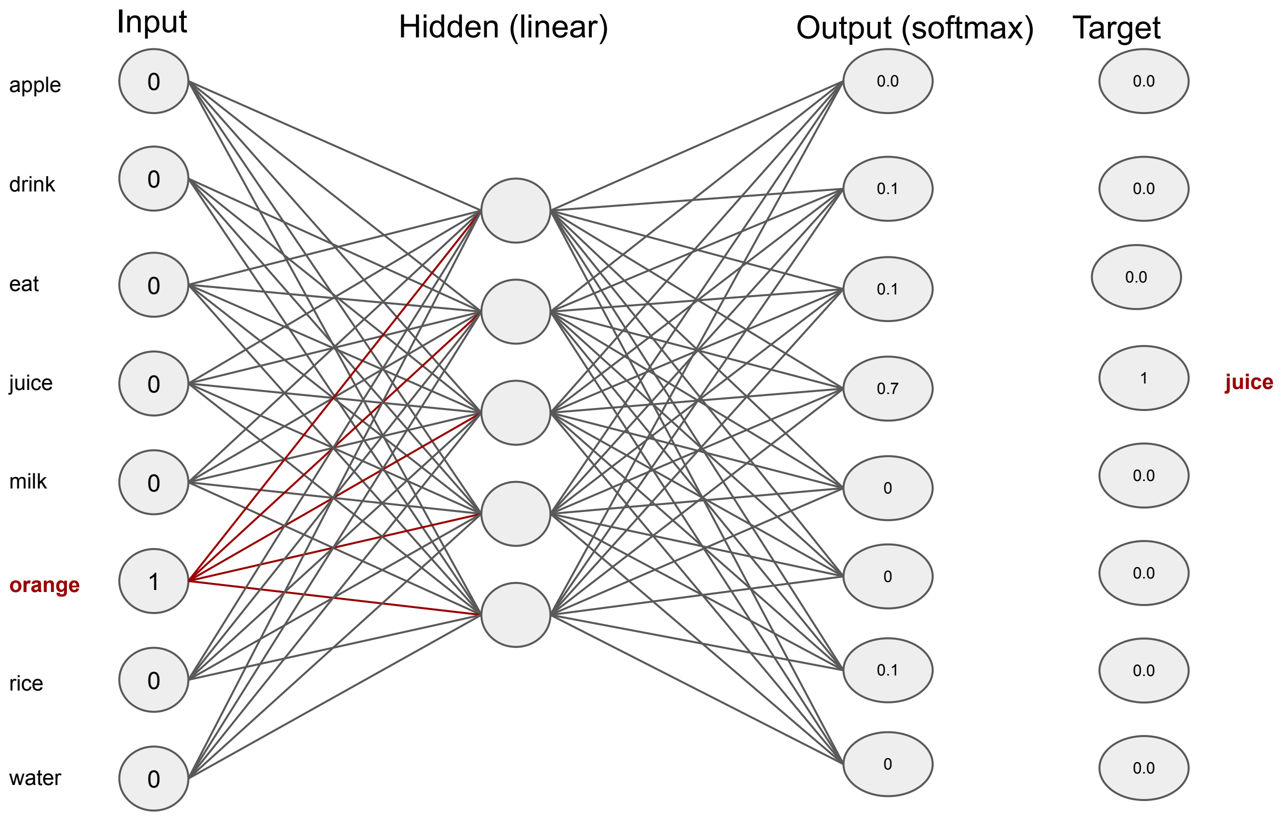


Output Layer

- Softmax classifier
- Output word matrix denoted as **OVec**

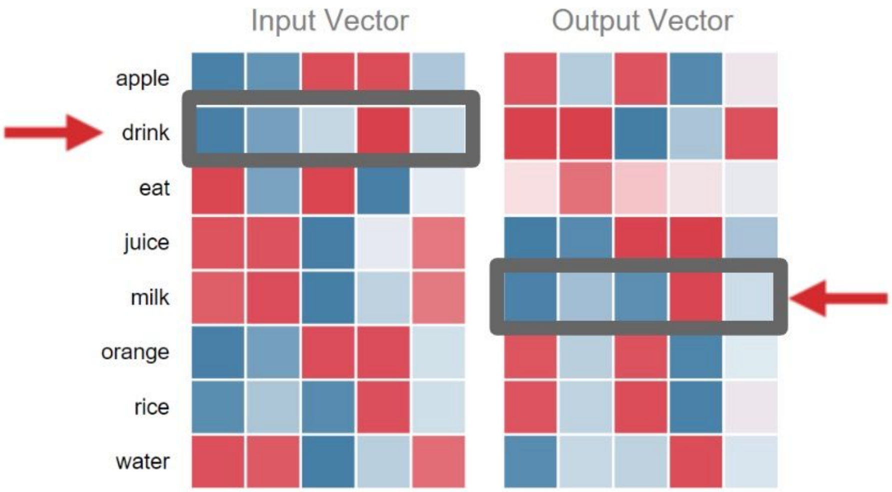


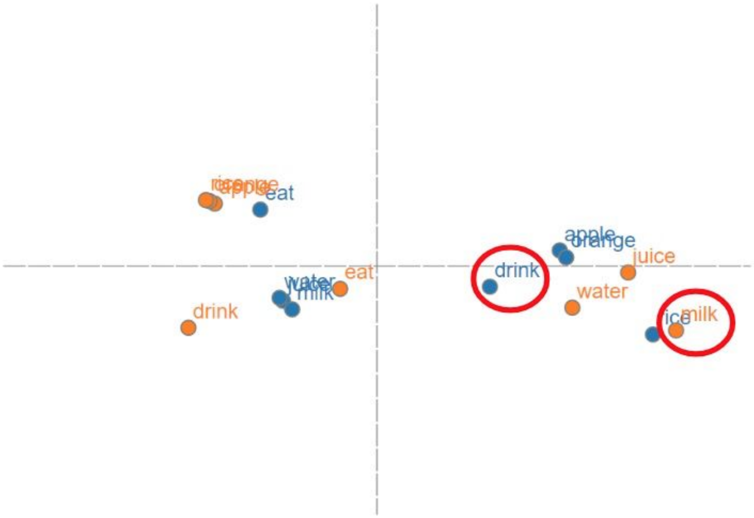
Word2Vec Network



Then, we can compute the loss and call gradient descent to update model parameters.

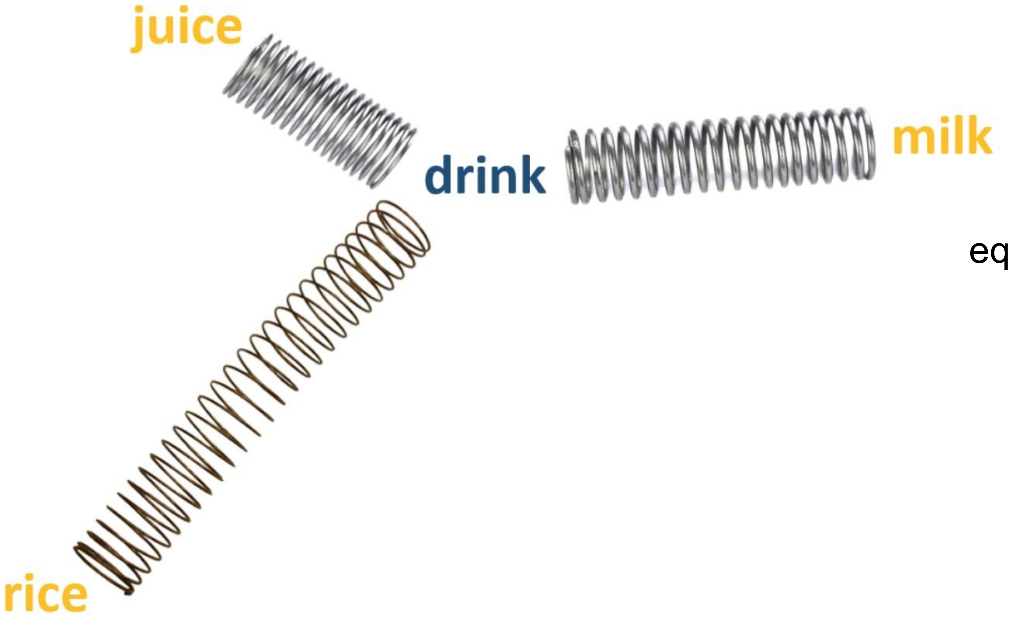
Updating Word Vectors





From Xin Rong 2016

A force-directed graph



equilibrium length is ?

Idea behind Word2Vec

- Feature vector assigned to a word will be adjusted if it can not be used for accurate prediction of that word’s context.

- Each word’s context in the corpus is the teacher sending error signals back to modify the feature vector.

- It means that words with similar **context** will be assigned similar **vectors**!

Distributional Semantics

Input vs Output Word Vectors

- Inputs: semantics encoder from one-hot/word index to semantics
- Outputs: semantics decoder from semantics to probability distributions over words.
- **In most cases, input word vectors are used.** Some have observed that combinations of these two vectors may perform better.

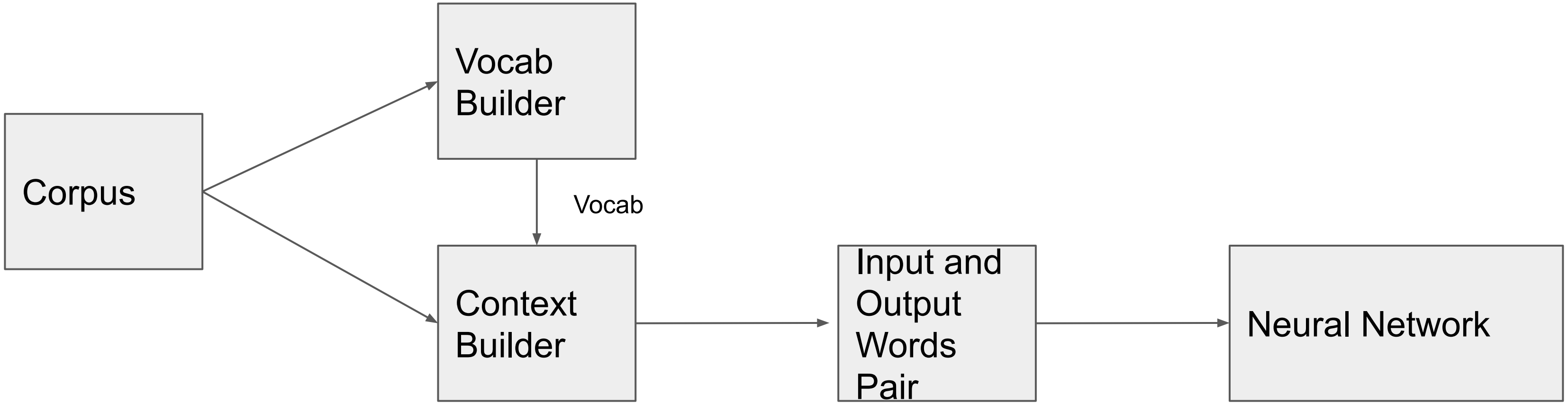
	Vector size	Overall	Semantic	Syntactic
DVRS	300	0.41	0.59	0.26
DVRS	1024	0.43	0.62	0.28
SG	300	**0.64**	**0.69**	**0.60**
SG	1024	0.57	0.60	0.55
Add 300-DVRS, 300-SG	300	0.64	0.72	0.58
Concatenate 300-DVRS, 300-SG	600	**0.67**	**0.74**	**0.60**
Add 1024-DVRS, 1024-SG	1024	0.60	0.66	0.55
Concatenate 1024-DVRS, 1024-SG	2048	0.61	0.68	0.55
Concatenate DVRS-1024, SG-300	1324	0.66	0.73	**0.60**
Oracle DVRS-1024, SG-300	1024/300	0.70	0.79	0.62

Table 2: Performance on word analogy problems with vectors trained against the first 10^9 bytes of Wikipedia.

Garten, 2014

Input and Output Words

- How to select them from corpus
- Skip-gram and CBoW differ here.



Skip-Gram

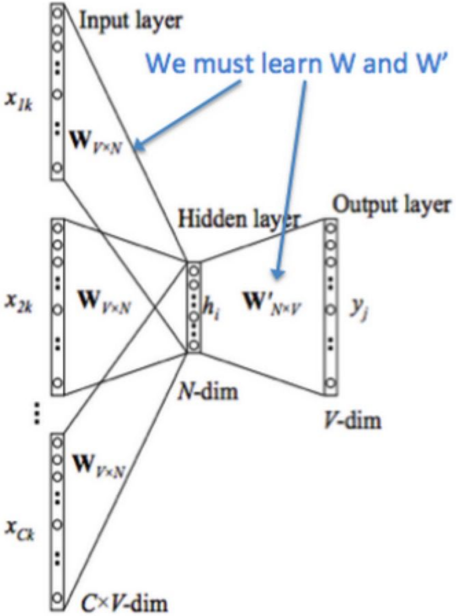
- Task Definition: given a specific word, predict its nearby word (probability output)
- Model input: source word, Model output: nearby word
- Input is one word, output is one word
- The output can be interpreted as prob. scores, which are regarded as how likely it is find each vocabulary word can be nearby your input word.

give a talk at the

Input x	Target y
talk	give
talk	a
talk	at
talk	the

CBoW

- Task Definition: given context, predict its target word
- Model input: context (several words), Model output: center word
- Input is several words, output is one word
- Core Trick: **average** these context vectors for prob score computing



give a **talk** at the

Input x
(give,a,at,the)

Target y
talk

Skip-Gram Vs CBoW

- **CBoW: learning to predict the word by the context**
- **Skip-gram: learning to predict the context by the center word**

  - CBoW: several times faster to train the skip-gram
  - Skip-gram: works well with small amount of the training data, represents well even rare words or phrases.

Context Selection

- In count-based or predict-based methods, context has a large effect.
- Small context window: more syntax-based embeddings
- Large context window: more semantics-based, topical embeddings
- **Engineering practice: window size is randomly sampled between 1 and maximum window size**

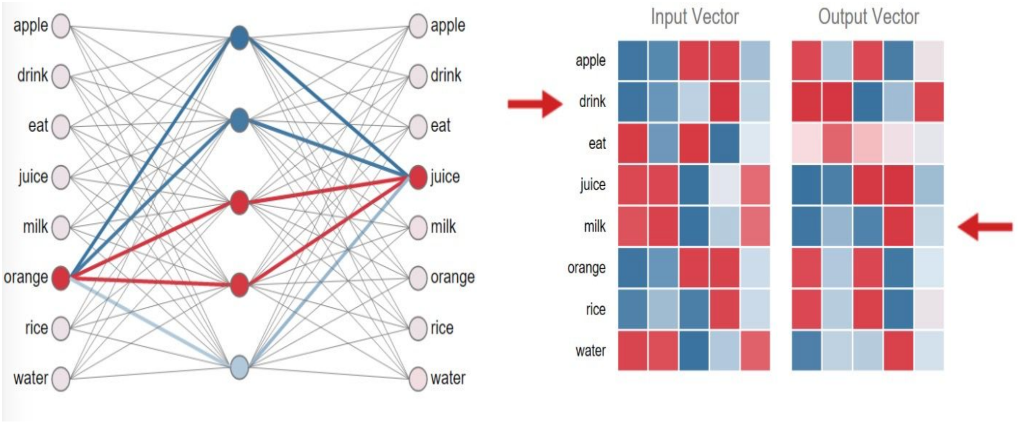
Huge Number of Parameters

- Vocab size is huge

- The Sum of operation in softmax layer is very expensive, i.e., O(v).

e^(IVec[eat]*OVec[apple]) + e^(IVec[eat]*OVec[juice]) + e^(IVec[eat]*OVec[drink])+e^(IVec[eat]*OVec[other vocab words])

- Two solutions: **Hierarchical softmax and negative sampling**



NN-based vs Matrix-based

Neural Word Embedding as Implicit Matrix Factorization

Omer Levy
Department of Computer Science
Bar-Ilan University
omerlevy@gmail.com

Yoav Goldberg
Department of Computer Science
Bar-Ilan University
yoav.goldberg@gmail.com

Abstract

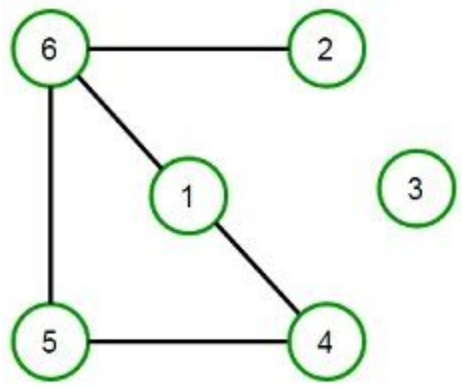
We analyze skip-gram with negative-sampling (SGNS), a word embedding method introduced by Mikolov et al., and show that it is implicitly factorizing a word-context matrix, whose cells are the pointwise mutual information (PMI) of the respective word and context pairs, shifted by a global constant. We find that another embedding method, NCE, is implicitly factorizing a similar matrix, where each cell is the (shifted) log conditional probability of a word given its context. We show that using a sparse *Shifted Positive PMI* word-context matrix to represent words improves results on two word similarity tasks and one of two analogy tasks. When dense low-dimensional vectors are preferred, exact factorization with SVD can achieve solutions that are at least as good as SGNS's solutions for word similarity tasks. On analogy questions SGNS remains superior to SVD. We conjecture that this stems from the weighted nature of SGNS's factorization.

what is important for word embeddings is that how to select **hyperparameters** and the utilization of appropriate **pre-processing** and **post-processing** steps.

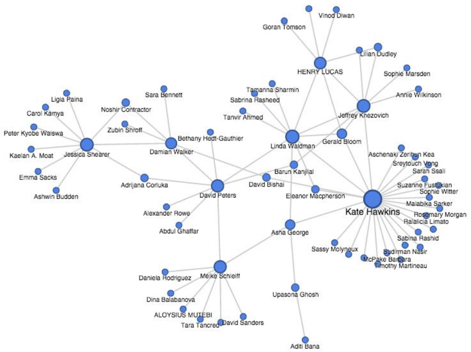
Graph Embedding

Graph Data

- Graph is an ordered pair G=(V, E).
- V is the set of nodes
- E is the collection of pairs of nodes which are called edges

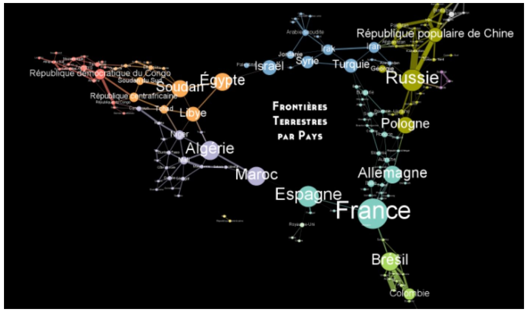


Graph Are Everywhere

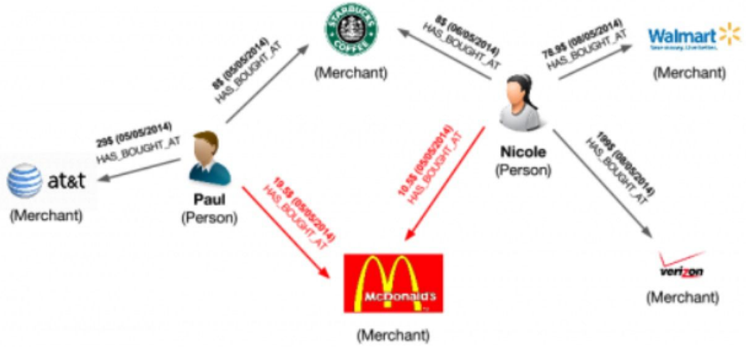


Social Network Analysis

Recommendation System



Logistics and Transportation



Fraud Detection

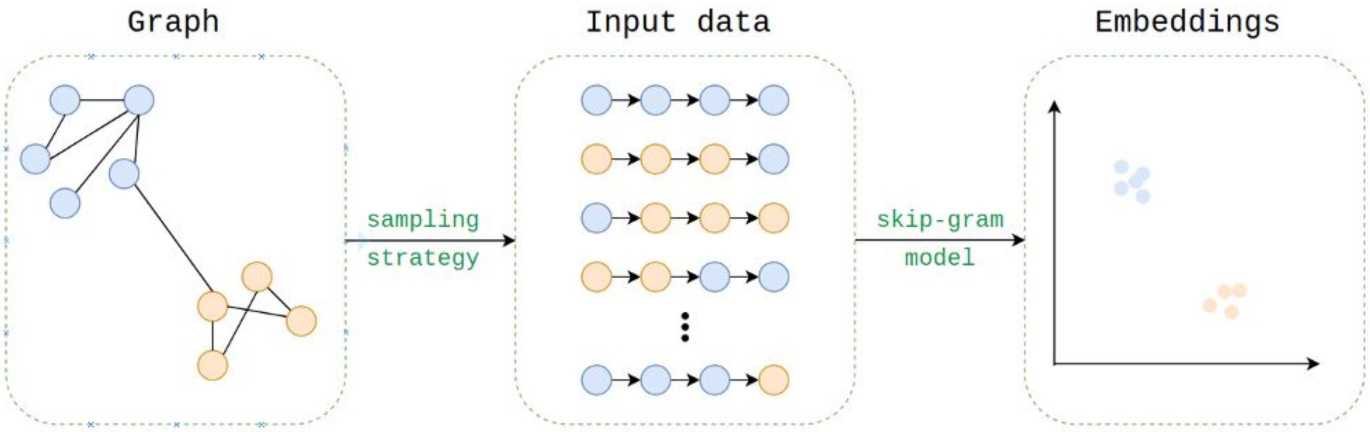
Graph Data

- Based on your tasks, define your nodes and Edges

- Apply graph mining algorithms:
    - Graph Pattern Mining
    - Graph Classification
    - Graph Compression
    - Graph Clustering
    - Etc

Embedding for Graph Data

- Embeddings can be extended beyond NLP domain
- Embeddings can be learned for any nodes in a graph



- Nodes can be items, web pages and so on in user clicked stream data
- Embeddings can be learned for any group of discrete and co-occurring states.