Word Vectors

SCHOOL OF INFOCOMM

Lexical Semantics

- **Linguistic study of word meanings and relation between them**

- **Key terms and concepts**
  - **Lemma** - is the word you find in the dictionary. *e.g. Sing is the lemma for sing, sang, sung*
  - **Word senses** – the different meanings of a lemma. *e.g. mouse (device or rodent?)*
  - **Synonyms** – words that have identical meaning and can be easily substituted *E.g. car & automobile*
  - **Similarit**y – words that are not synonyms but have similar features *e.g. muscle, bone*
  - **Word relationship** – words that are not similar but associated with each other by co-participating in the same event e.g. toast and coffee
  - **Semantic frames** – words that denote participation and role in the same event. *e.g. buyer and seller*
  - **Connotation** – words with affective meanings related to emotions, sentiments, state *e.g. Happy and sad*

Example

Connotation or affective meanings of words varied around 3 dimensions

Valence *– the pleasantness of the emotions invoked by the word*

Arousal *– the intensity of emotion invoked by the word*

Dominance *- the degree of control exerted by the word*

	Valence	Arousal	Dominance
Courageous	8.05	5.5	7.38
Music	7.67	5.57	4`.5
Brave	8.68	5.59	5.89

The word “music” can be represented by the vector [7.67 , 5.57, 6.5]
"Courageous" and "Brave" are closer in the vector space

Distribution Hypothesis

Image: wikipedia

- Challenging to build a formal theory of the meaning of words based on different aspect of word meanings
- Philosopher Ludwid Wittgenstein suggested that “meaning of a word is its use in the language”
- Linguists popularized the idea to "You shall know a word by the company it keeps“
- The distributional hypothesis suggests that the more semantically similar two words are, the more distributionally similar they will be in turn, and thus the more that they will tend to occur in similar linguistic contexts.

Example: Inferring meaning from words distribution

*Mary is a stubborn girl. Her **pervicacious** nature always gets her in trouble whenever she is out running an errand*

What is the meaning of ***pervicacious***

Ongchoi is delicious sauteed with garlic.
Ongchoi is superb over rice.
...**ongchoi** leaves with salty sauces...
spinach sauteed with garlic over rice...
...chard stems and leaves are delicious...
:...collard greens and other salty leafy greens

What is the meaning of ***Ongchoi***

Example from : Speech and Language Processing: https://web.stanford.edu/~jurafsky/slp3/6.pdf

Introducing Vector Semantics

- The meaning of a word is **computed** from the distribution of words around it
- The word is then defined as vector, and each element in the vector derive from counts / and occurrences of neighbouring words (embedding)
- Words are represented as points in multi-dimensional semantic space
- Semantics similar words are located in distinct parts of the semantic space

Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for sentiment analysis. Simplified from Li et al. (2015).

Image: https://web.stanford.edu/~jurafsky/slp3/6.pdf

Representing Word Vectors

Vector or distributional models of meaning are based on co-occurrence matrix

Counting methods
- Word-word matrix

Embedding methods
- Word2Vec

Word-Word Matrix

Using the surrounding words and create a word-word matrix

Columns and rows are the words

Each cell records the number of times the word in the row co-occurs with the word in the column in some context

Example

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened

well suited to programming on the **digital** computer. The data on both PC and Mac computers.

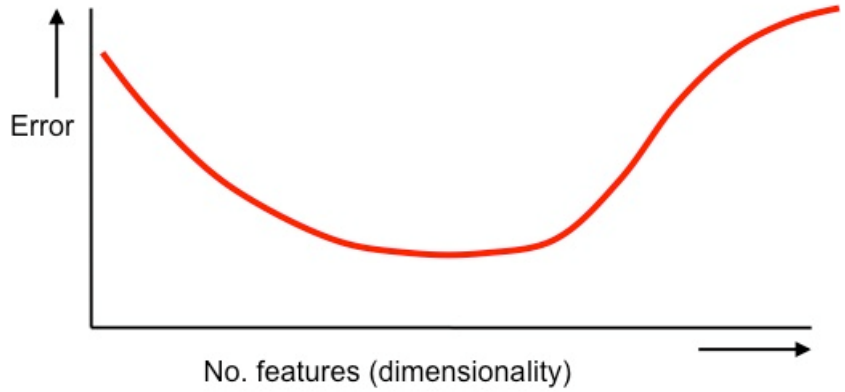
Take every occurrence of the word "digital" and 'apricot' and count the context words (nouns, adjectives etc) around it ...

	computer	enjoyment	jam	Mac	PC	perserve	pinch	programming	tablespoon
apricot	0	1	1	0	0	1	1	0	1
digital	1	0	0	1	1	0	0	1	0

Problem with counting

Problem: Counting creates far too large sparse matrix depending on the number of words and documents

- Can we fix the number of dimensions regardless of document size and words



Problem: Word-Word matrix neglects word order

- What can we do the maintain some information on sequences of words

  → We need higher weightage for 'closer' words

Introducing Word2Vec

Word2vec is a group of related models that are used to generate **distributed representation** of words

First created & published by Mikolov of Google in 2013

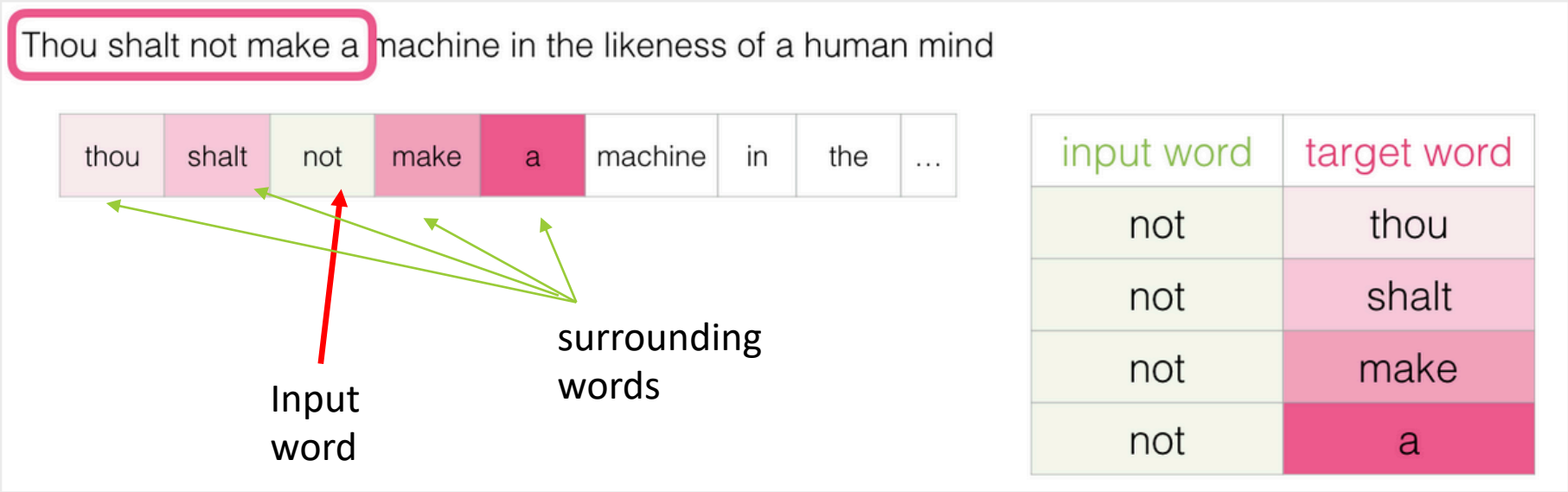
A form of **unsupervised learning** with implied labelling

Allows meaning of words to be encoded in a **fixed number** of dimensions

Word2Vec - Key Ideas

- A word2vec model is a simple neural network model with a single hidden layer
- The model predicts the nearby words for each and every word in the corpus
- The goal is to get weights learned by the hidden layer of the model once the model is trained
- These weights can then be used as the word vector

Training Word2Vec: Getting label data

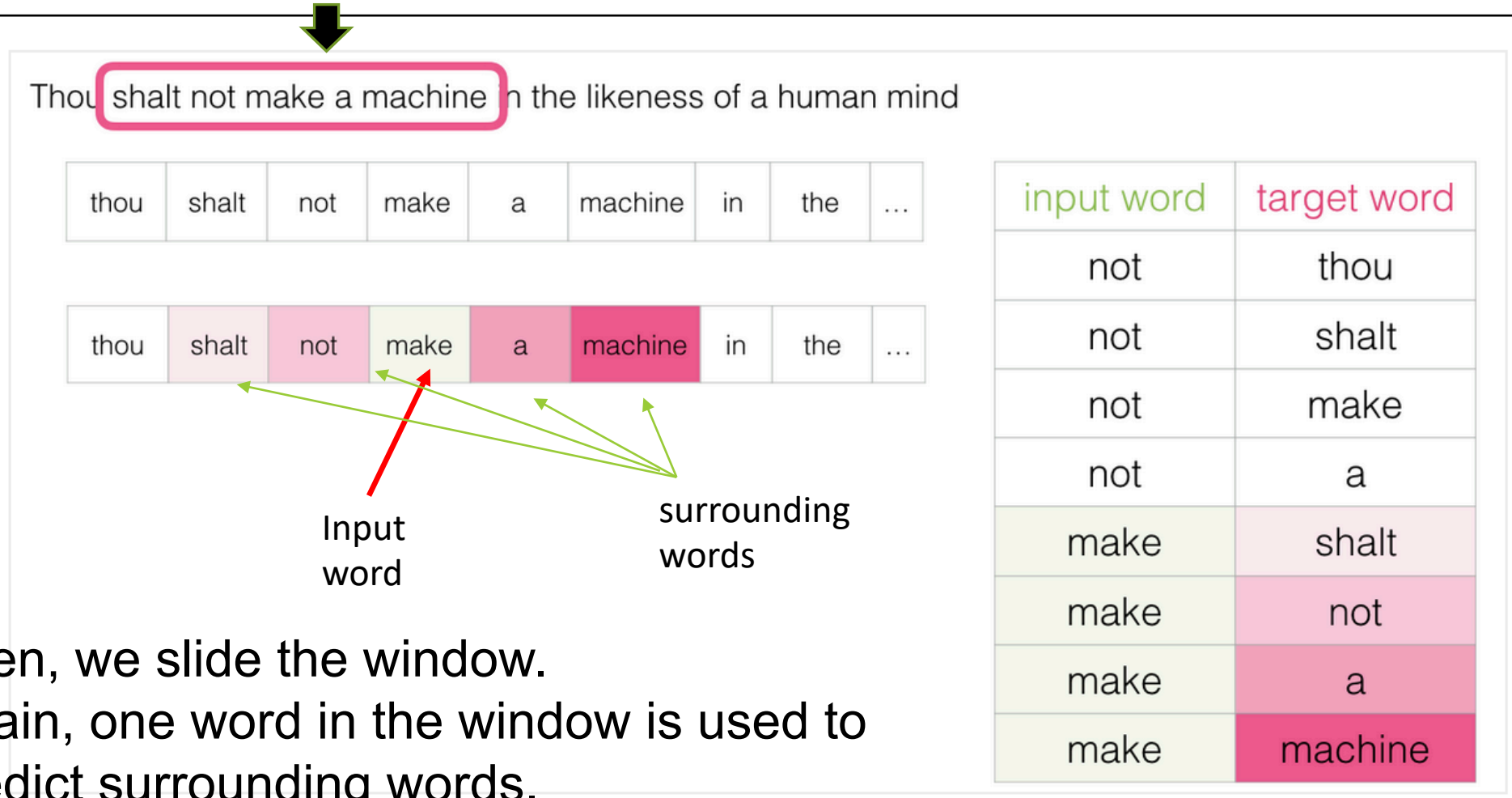


input word	target word
not	thou
not	shalt
not	make
not	a

First, from an un-labelled text, we need to find labelled data.
We define a context window.
The middle word in the window is used to predict surrounding words.
In this example, the word 'not' predicts 4 words (thou, shalt, make, a).

Illustration from: http://jalammar.github.io/illustrated-word2vec/

Training Word2Vec: Getting label data



input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

Then, we slide the window.
Again, one word in the window is used to predict surrounding words.
In this example, the word 'make' predicts 4 word (shalt, not, a , machine).

Illustration from: http://jalammar.github.io/illustrated-word2vec/

Training Word2Vec: Getting label data

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	…

thou	shalt	not	make	a	machine	in	the	…

thou	shalt	not	make	a	machine	in	the	…

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	
a	
a	
a	

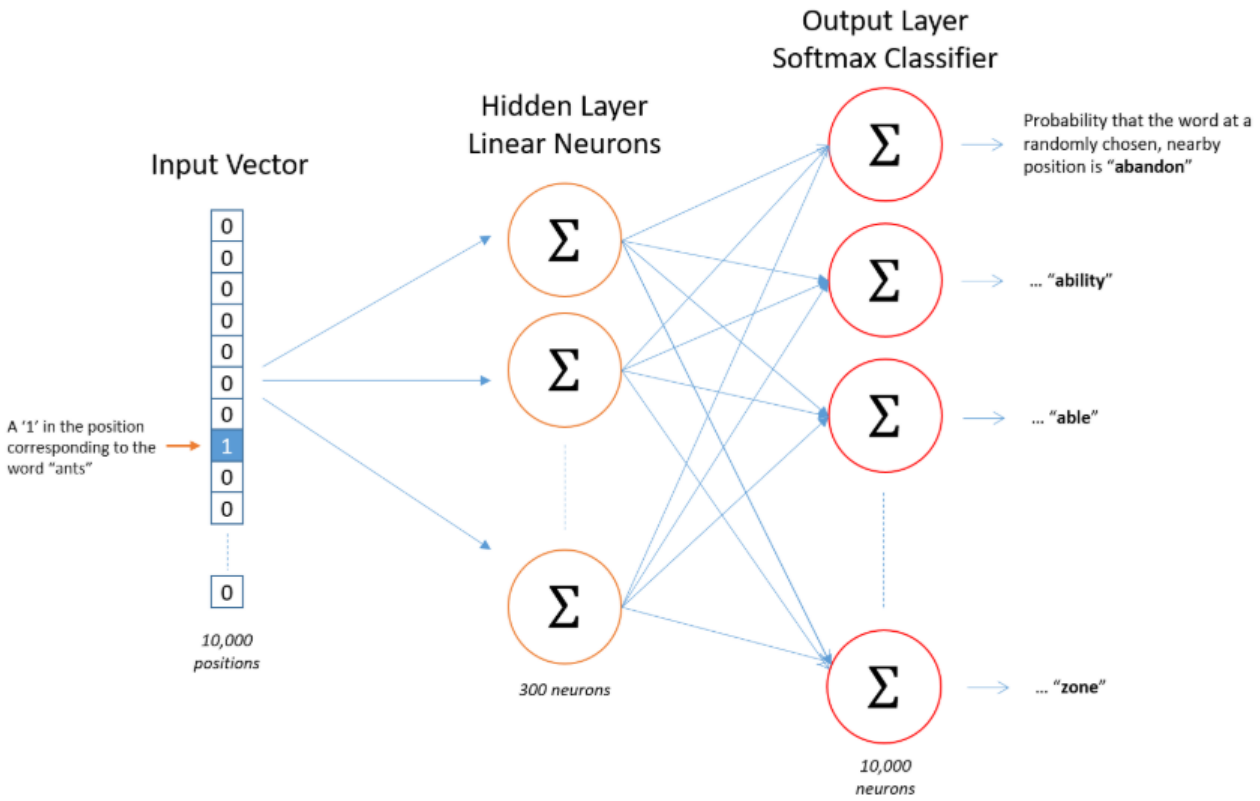
????

Then, we slide the window again.

Test your understanding:
In this window, which one word is used to predict which 4 words?

Illustration from: http://jalammar.github.io/illustrated-word2vec/

Training Word2Vec : Predicting



Let us say that we have a vocabulary size of 10,000. Then each input word is represented using one-hot encoding in a 10,000 components vector.

The neuron network will be trained predict the target word.

The target word is also represented using one-hot encoding using a 10,000 components vector, but the output layer of the network is a probability distribution.

There is a hidden layer of neurons where the weights are adjusted as the network learns.

McCormick, C. (2016, April 19). *Word2Vec Tutorial - The Skip-Gram Model*. Retrieved from http://www.mccormickml.com

Training Word2Vec: Predicting (make → not)

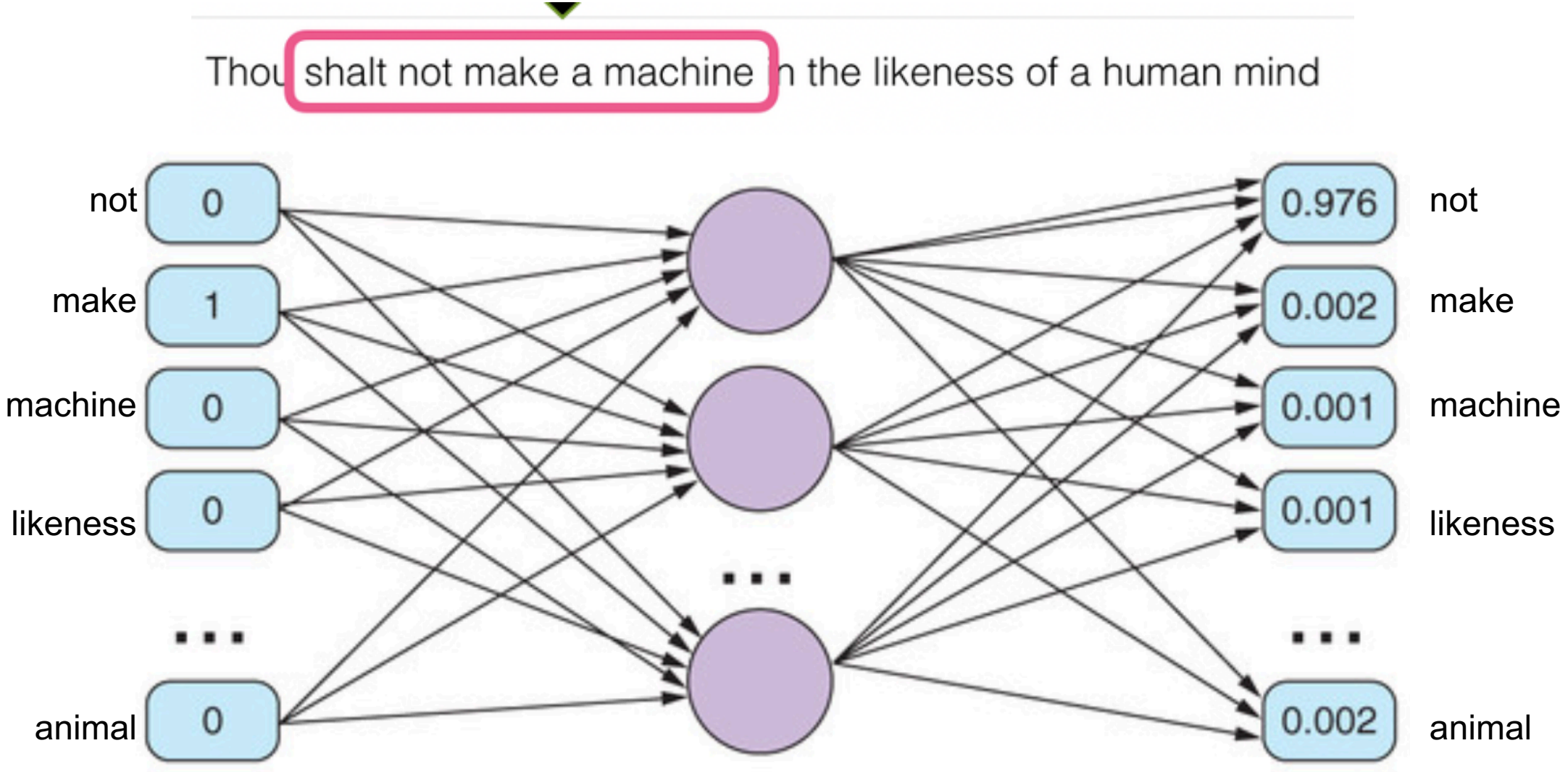


Illustration from: Natural Language Processing In Action (Manning Publications)

Training Goal : Predicting (make → machine)

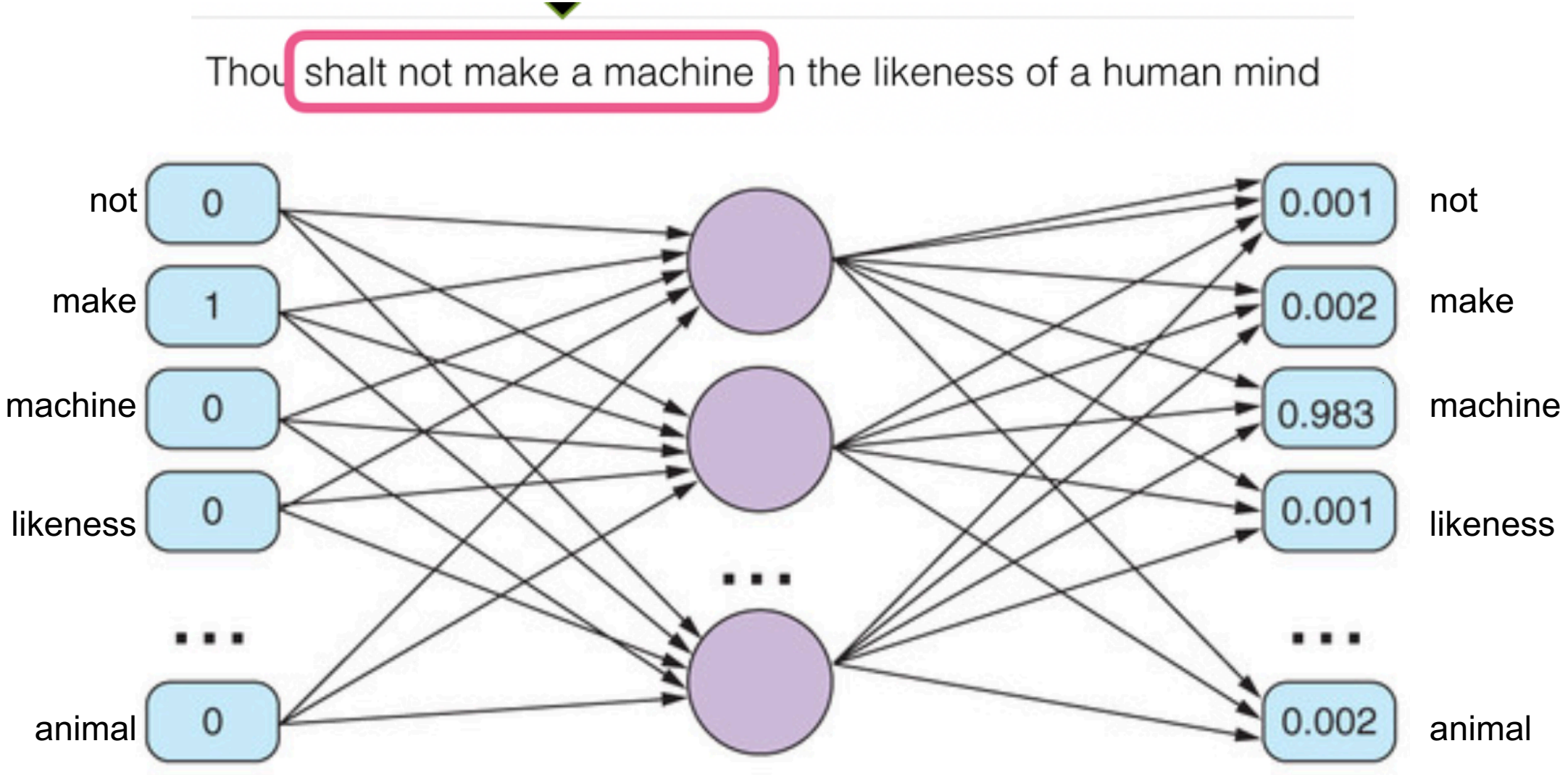


Illustration from: Natural Language Processing In Action (Manning Publications)

Deriving the Word Vector

Suppose that after training, the weights of the hidden layer is represented in a weighted matrix

If you multiply a one-hot vector by a the same matrix, it will effectively just *select* the matrix row corresponding to the “1”

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

One hot encoding

Weight matrix

So, the row in the hidden weighted matrix becomes the vector of the word with 3 learned features or dimensions

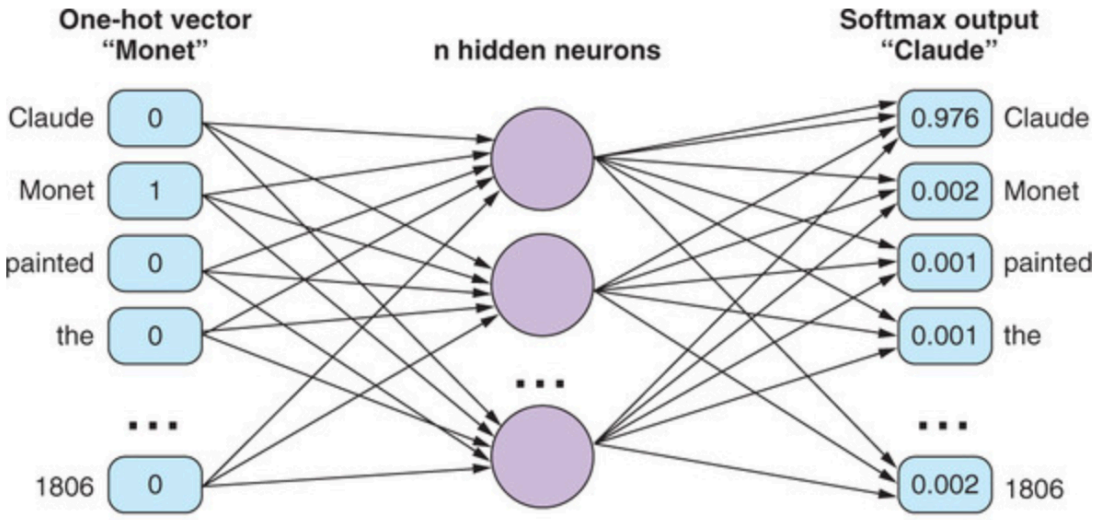
Illustration from: http://jalammar.github.io/illustrated-word2vec/

MAKE =

```
[-0.11328125 -0.03686523  0.09423828  0.00799561  0.02490234 -0.16699219  0.03662109  0.07324219  0.21386719
 -0.03857422 -0.09863281 -0.09033203 -0.06884766  0.34765625  0.05151367 -0.03369141  0.15332031  0.10595703
  0.09765625  0.05078125 -0.03173828  0.01855469  0.24804688  0.18359375  0.00363159 -0.1484375  -0.16113281
 -0.00836182 -0.09082031  0.04736328  0.0012207   0.12597656 -0.13964844  0.04418945 -0.05395508 -0.05639648
  0.07275391  0.14648438  0.19433594  0.15820312  0.17871094 -0.07568359  0.15625    -0.1171875   0.04125977
 -0.03039551 -0.04785156 -0.00136566  0.03149414  0.02624512  0.04370117  0.08984375  0.05859375 -0.21191406
 -0.12011719  0.16503906 -0.06884766 -0.13476562 -0.05517578 -0.03442383  0.02209473 -0.02722168 -0.09082031
 -0.01647949 -0.07275391  0.09667969  0.00915527  0.24902344 -0.06884766  0.06835938  0.09667969  0.11669922
  0.0378418  -0.05200195 -0.16601562 -0.16699219  0.11962891  0.16503906 -0.02929688  0.04882812  0.00328064
 -0.23632812 -0.03881836  0.01647949  0.04516602 -0.10400391  0.06030273  0.05175781  0.01495361  0.09765625
  0.140625   -0.00138855  0.04248047 -0.08398438 -0.03027344 -0.375      -0.01269531  0.12353516  0.11962891
  0.02734375 -0.00411987 -0.08105469  0.01672363  0.14550781 -0.11474609 -0.15625     0.07470703 -0.13476562
 -0.04663086  0.01385498 -0.125      -0.09619141 -0.06689453  0.02832031  0.08935547 -0.02685547  0.19335938
 -0.03686523 -0.06176758 -0.02819824  0.10400391  0.12158203 -0.01177979  0.37890625  0.01257324 -0.07275391
  0.05029297 -0.09277344  0.03271484 -0.01794434 -0.08300781 -0.18066406 -0.05297852  0.22167969  0.08447266
  0.00592041 -0.04980469  0.06396484 -0.04711914 -0.14941406  0.16113281 -0.14746094 -0.07128906  0.05908203
  0.07421875  0.1640625   0.0559082  -0.19433594 -0.02819824  0.08496094 -0.09912109 -0.05957031 -0.02111816
 -0.01525879 -0.03491211 -0.10498047  0.08447266  0.00271606 -0.10546875 -0.13867188 -0.00358582 -0.02661133
 -0.11962891 -0.15136719  0.1640625  -0.12890625  0.04663086 -0.01275635  0.17773438 -0.04663086 -0.17773438
 -0.13476562 -0.07714844 -0.16210938 -0.13476562 -0.10058594  0.02160645 -0.10058594 -0.0534668   0.21972656
  0.00332642 -0.23339844  0.10693359  0.10302734 -0.25       -0.16699219 -0.17089844 -0.15527344  0.01623535
 -0.10888672  0.14160156 -0.06933594  0.02319336 -0.13867188 -0.15234375 -0.06005859 -0.00787354 -0.24609375
 -0.3515625  -0.125       0.09619141 -0.11865234 -0.10205078  0.06689453 -0.01208496 -0.14160156 -0.13769531
 -0.07080078  0.16796875 -0.13476562  0.04638672  0.16992188  0.09814453  0.20703125 -0.21972656  0.12988281
  0.20117188  0.17871094 -0.08447266 -0.03515625  0.09912109  0.10107422 -0.12158203  0.06738281  0.05053711
  0.07373047 -0.05249023 -0.0625      0.03881836  0.10595703  0.32226562 -0.02209473  0.09521484 -0.10644531
  0.22558594 -0.03735352  0.04858398 -0.171875   -0.02233887 -0.07617188 -0.12011719  0.09082031 -0.07568359
  0.1171875   0.10595703 -0.0480957  -0.07470703  0.03662109 -0.23339844 -0.07373047 -0.02661133  0.09228516
  0.15527344  0.08984375  0.24121094  0.03125     0.02429199  0.30859375 -0.19042969 -0.08886719 -0.01757812
 -0.06176758 -0.0324707  -0.02441406  0.20019531  0.01202393  0.11914062 -0.203125   -0.10693359 -0.14453125
  0.18847656  0.19042969  0.18847656  0.17285156  0.11474609 -0.06494141 -0.14941406 -0.1796875   0.00369263
 -0.12060547  0.05688477 -0.11279297 -0.03088379  0.14355469 -0.08398438  0.13085938 -0.07861328 -0.25
  0.06030273  0.02966309 -0.12792969  0.13769531 -0.12353516 -0.00891113 -0.12988281 -0.03417969  0.04125977
 -0.04858398  0.29492188 -0.20214844]
```

Architecture Variation

Skip-gram model predict the neighbours of a word
e.g. one word predicts a few



CBOW model predicts a word based on it's neighbourhood. Many words predict one.

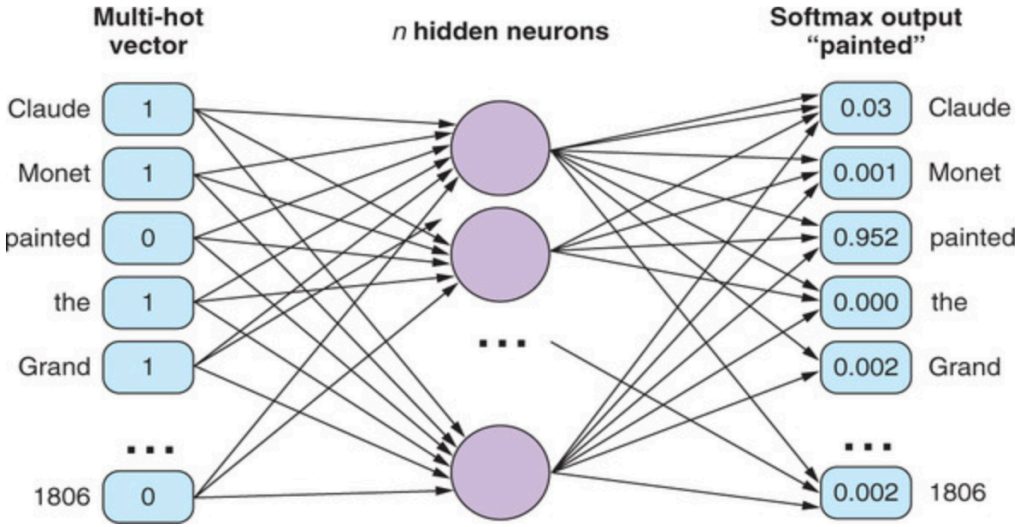


Illustration from: Natural Language Processing In Action (Manning Publications)

Skip-gram or CBOW

Skip-gram: works well with small amount of corpus, represents well even for rare words or phrases

CBOW: several times faster to train than the skip-gram, slightly better accuracy for the frequent words

Pre-Trained Word2Vec Models

Source	Description
Google News	Word vectors for a vocabulary of 3 million words and phrases that they trained on roughly 100 billion words from a Google News dataset. The vector length is 300 features. https://code.google.com/archive/p/word2vec/
Wikipedia	Wikipedia2Vec is a tool for learning embeddings of words and entities from Wikipedia for 12 languages https://wikipedia2vec.github.io/wikipedia2vec/pretrained/
中文词向量Multiple sources	Pretrained Chinese word embedding vectors https://github.com/Embedding/Chinese-Word-Vectors
Language Technology Group at the University of Oslo	Pretrained multi-language word embedding vectors from University of Oslo http://vectors.nlpl.eu/repository/

Introducing Gensim

Gensim is an open-source library for unsupervised topic modeling and natural language processing.

Design to handle large text collections using data streaming and incremental online algorithms, which differentiates it from most other machine learning software packages that target only in-memory processing.

Gensim includes streamed parallelized implementations of fastText, word2vec, doc2vec algorithms, latent semantic analysis (LSA, LSI, SVD), non-negative matrix factorization (NMF), latent Dirichlet allocation (LDA).

Installation

Run in your terminal :

```
$ pip install --upgrade gensim
```

Or

```
$ conda install -c conda-forge gensim
```

Gensim: Loading Pretrain Vector

```
pretrained_vec_file = "./data/GoogleNews-vectors-negative300.bin.gz"
google_model  = gensim.models.KeyedVectors.load_word2vec_format(pretrained_vec_file ,
                                                                binary=True)

vector1 = google_model.word_vec('make')
vector2 = google_model['make']
vector3 = google_model.get_vector('make')
```

Codes explanation:

1. Locates the location of the pre-trained file
2. Loads into the KeyedVectors structure which is essentially a mapping between words and vectors
3. Access the vector that represents the entity (string) "make"

Exercise 1

Using Gensim for Word2Vec

Refer to ex1-word2vec-basic.iypnb

This notebook will show you how to :

- load a pre-trained module using GENSIM

- Retrieve the vector for a given word

Interesting Results
Organized Concepts & Similarity Words

```
1 google_model.similar_by_word('Thailand')
```

```
[('Thai', 0.7535779476165771),
 ('Cambodia', 0.7131428718566895),
 ('Bangkok', 0.7014991044998169),
 ('Thais', 0.6784130334854126),
 ('Malaysia', 0.6679352521896362),
 ('Indonesia', 0.6669518947601318),
 ('Chiang_Mai', 0.6564016938209534),
 ('Thailands', 0.6523814797401428),
 ('Southeast_Asia', 0.621160089969635),
 ('Laos', 0.6205481290817261)]
```

```
1 google_model.similar_by_word('Lee_Kuan_Yew')
```

```
[('Lee_Kwan_Yew', 0.7458120584487915),
 ('Lee_Hsien_Loong', 0.6708043217658997),
 ('Goh_Chok_Tong', 0.6635027527809143),
 ('prime_minister_Lee_Kuan', 0.644353985786438),
 ('Goh_Keng_Swee', 0.6256564855575562),
 ('LKY', 0.6218506693840027),
 ('Singapore_Lee_Kuan', 0.6171931028366089),
 ('Dr_Mahathir', 0.6003636717796326),
 ('PAP_strongman', 0.5983213186264038),
 ('Jeyaretnam', 0.5875519514083862)]
```

```
1 google_model.similar_by_word('small')
```

```
[('large', 0.733115017414093),
 ('tiny', 0.7187926769256592),
 ('medium_sized', 0.6426597833633423),
 ('Small', 0.6270469427108765),
 ('smaller', 0.6194288730621338),
 ('minuscule', 0.5676835775375366),
 ('larger', 0.5446805953979492),
 ('mid_sized', 0.5328439474105835),
 ('midsized', 0.5198310613632202),
 ('sizable', 0.5175076723098755)]
```

```
1 google_model.similar_by_word('fever')
```

```
[('fevers', 0.6539823412895203),
 ('mania', 0.5566962957382202),
 ('feverish', 0.5537868142127991),
 ('fever_vomiting', 0.5078603625297546),
 ('sore_throat', 0.507377564907074),
 ('sore_throat_fever', 0.5043977499008179),
 ('coughing_runny_nose', 0.5000587701797485),
 ('fever_sore_throat', 0.4963510036468506),
 ('fever_aches', 0.49502936005592346),
 ('chills_sore_throat', 0.4949982166290283)]
```

Source: Distributed Representations of Words and Phrases and their Compositionality, https://arxiv.org/pdf/1310.4546.pdf

Interesting Results
Similarities + learn relationships

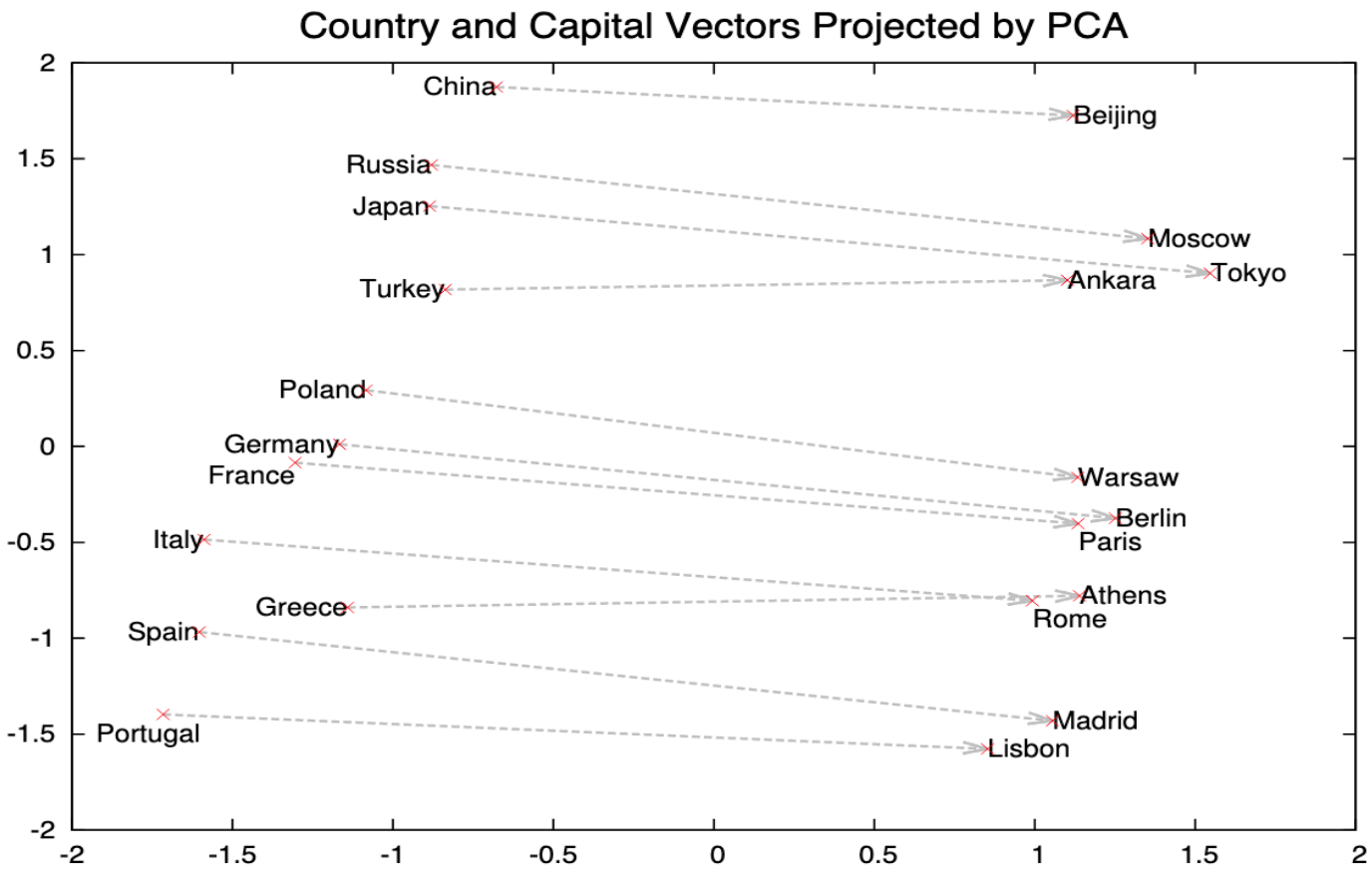


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Source: Distributed Representations of Words and Phrases and their Compositionality, https://arxiv.org/pdf/1310.4546.pdf

Interesting Results
More Learned relationship

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

Source: Distributed Representations of Words and Phrases and their Compositionality, https://arxiv.org/pdf/1310.4546.pdf

Interesting Results
Vector-oriented reasoning

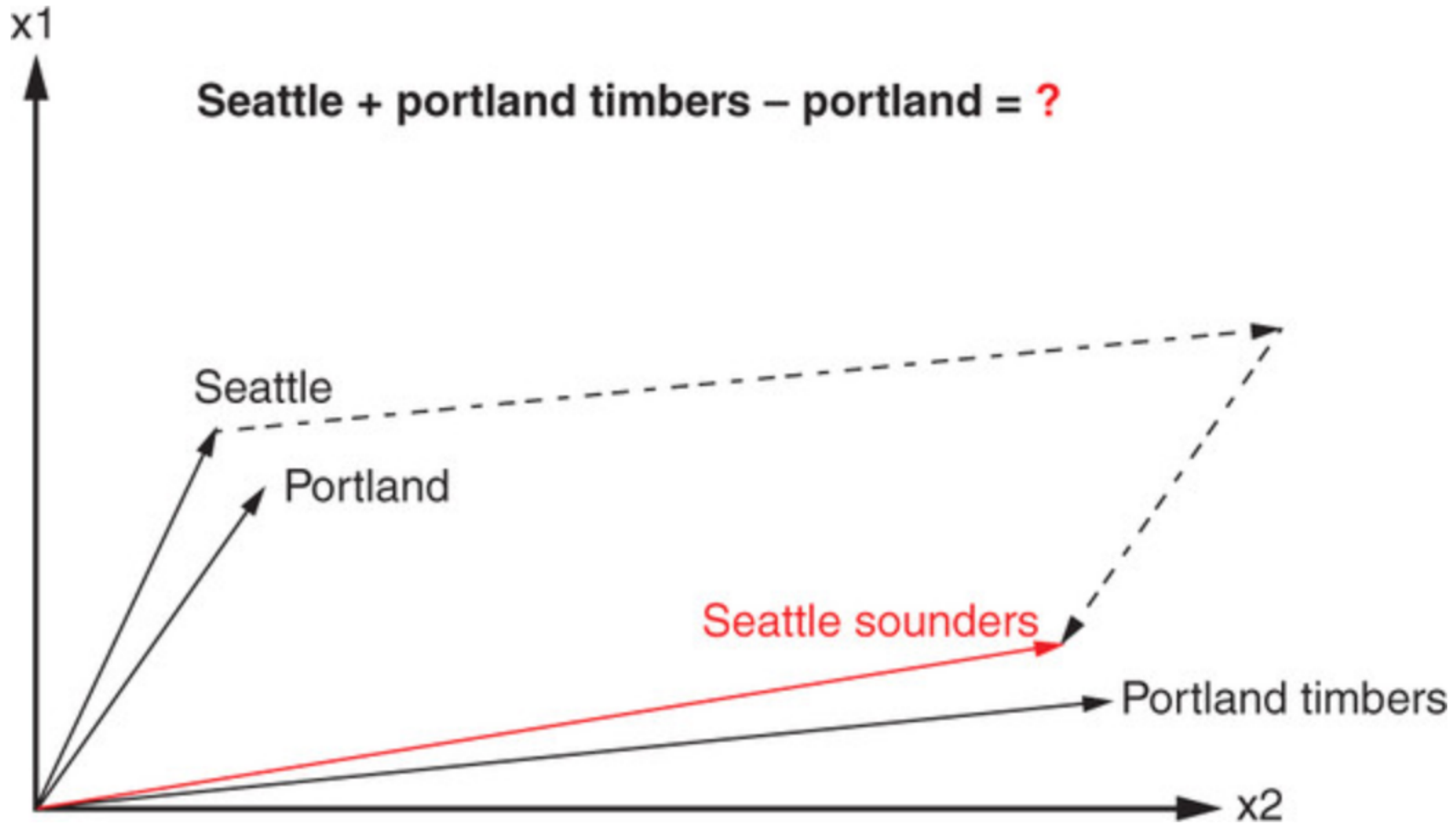


Illustration from: Natural Language Processing In Action (Manning Publications)

Exercise 2

Explore Similarity of Words in Word2Vec

Refer to ex2-word2vec-similarity.iypnb

This notebook will show you how to :

- compare the similarity between words using cosine different
- Use word2vec to find similar words

Exercise 3

Explore algebra operations with Word2Vec

Ref to ex3-word2vec-reasoning.ipynb

This notebook will show how to explore semantic properties of the word2Vec model

Domain Specific Word2Vector Models

Why Google News Word2Vec may not be enough

- Google news based on news articles before 2006
- Vocabulary may be unique to industry or domain
- Documents to analysis is used in a restricted domain
- Domain specific uses text in unique ways

Gensim: Custom Word2Vec model

```
tokens = [ ['DNA', 'order' ], [ 'sequence', 'negative] ..... ]

from gensim.models import Word2Vec

custom_model = Word2Vec(tokens, size=100, window=5, min_count=5, workers=2)

custom_model.save("custom_word2vec.model")

custom_model.wv.save('new_custom_word2vec.kv')
```

Codes explanation:

Document needs to be tokenized into into sentences. Each sentence is split into tokens.

Instantiate a Word2Vec model with the token list

Save the model into an external file so that it can be further trained in when new corpus are available

Save the Keyed Vectors into an external file. For downstream application can use the word embedding.

Gensim: Retraining Custom Word2Vec model

```
new_doc_tokens = [ ['xxx', 'yyy' ], [ 'PSLE', 'score'] ..... ]

new_model = Word2Vec.load("custom_word2vec.model")

new_model.build_vocab(new_docs_tokens, update=True)  ## prepare the model vocabulary

new_model.train(new_docs_tokens, total_examples= len(new_docs_tokens), epochs=50)
```

Codes explanation:

Prepares the next corpus. Break each documents into _____. Break each sentence into _____.
Reload the an existing model
Builds up the new vocabulary
Train the model with the new set of sentences

Exercise 4

Creating a Custom Word2Vec model

Refer to ex4-word2vec-custom-model.ipynb

This notebook will show you how to use Gensim to create a custom Word2Vec based on a new set of documents

Comparing Word Vectors

Scaled angle between the vectors

Vector length doesn't matter

Makes most sense for word vectors

Why?

- e.g. [2, 2, 2] and [4, 4, 4] should be the same vector
- It's the ratios of frequencies that define meaning

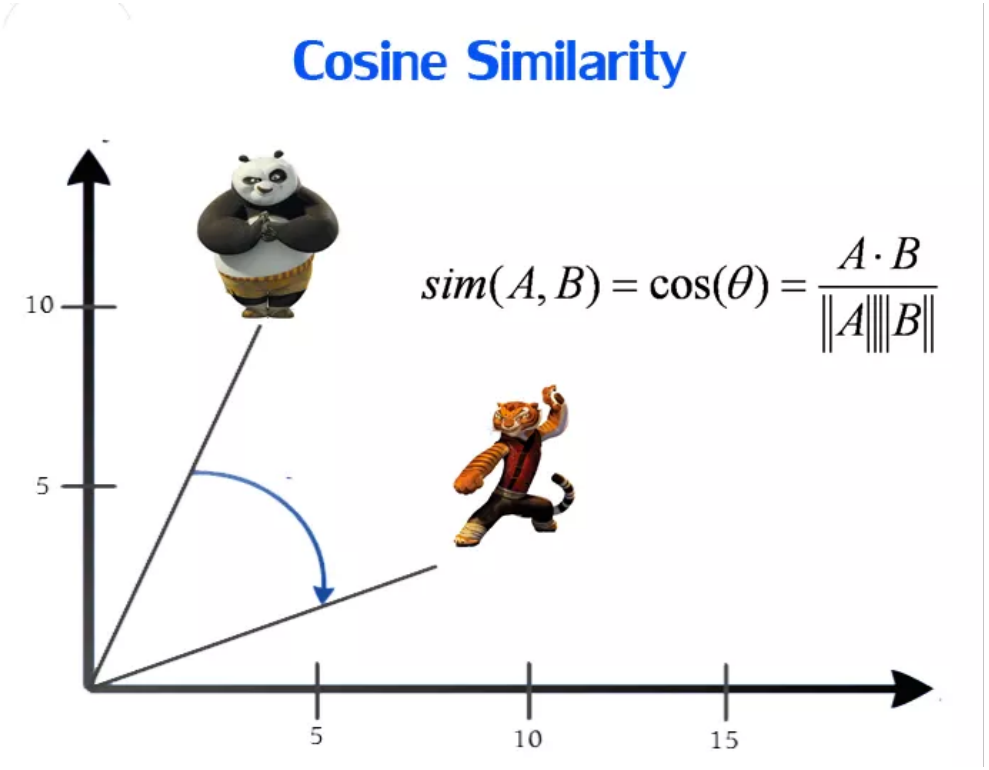


Illustration from: unknown

How to use Word2Vec

Word Vectors can be combined to create document features

- Document Vector is the mean of its word vectors
- Document Vector is sum of its word vectors

Use Document Vectors for ML on Documents

- Classification
- Recommendation

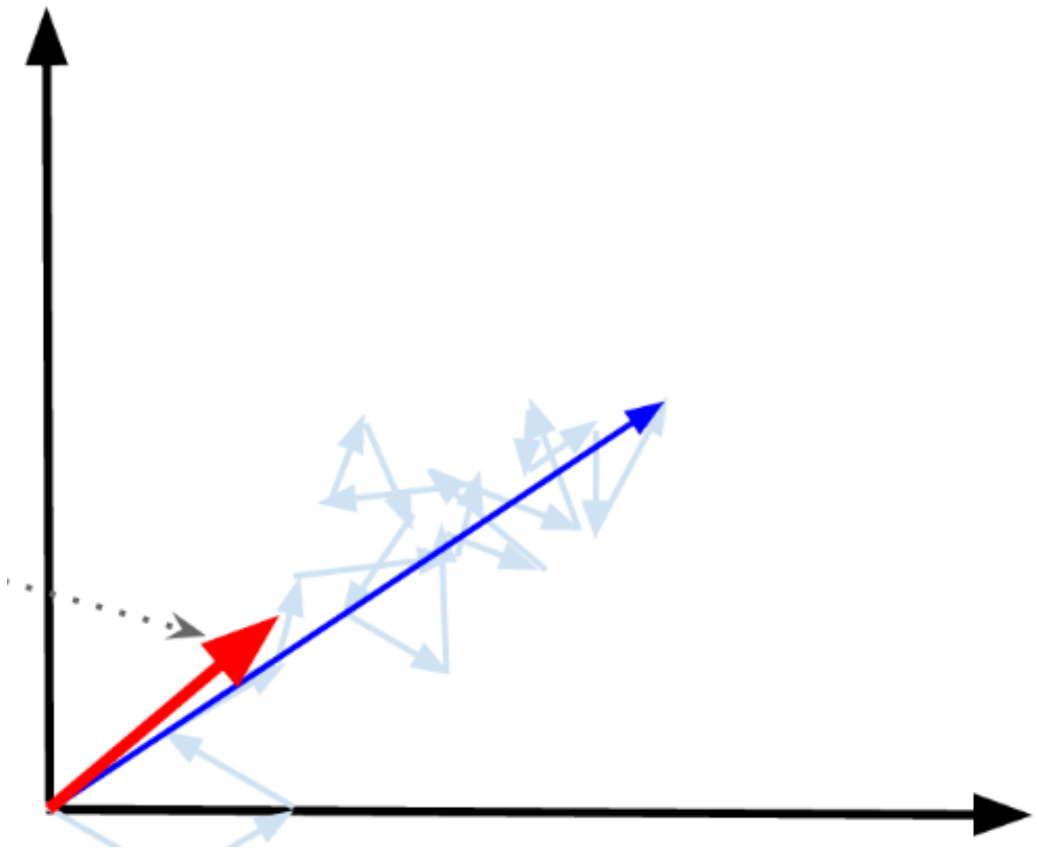


Illustration from: unknown

Gensim: Derive Document Vector

```
def get_doc_vec(words, word2vec_model):
    good_words = []
    for word in words:
         try:
            if word2vec_model[word] is not None:
                 good_words.append(word)
        except:
            continue
    if len(good_words) == 0:
        return None
    return word2vec_model[good_words].mean(axis=0)
```

The input **words** contains tokens relevant to a document.
Checks if a token is represented in the word2vec model
If exist, builds up an working list of tokems (good_words)
Use the list of tokens (good_words) to access the word vectors and return the mean of the vectors.
The mean of the vectors is taken as the document vector

Application: Text Classification

Problem: Categorizing News Articles

- Is document about Politics? Sports? Science/Tech? etc

- Approach:

  Word Vectors → Document Vectors

  Classification on Document Vectors

Application: Recommendation

- Problem: Find me news stories I care about!

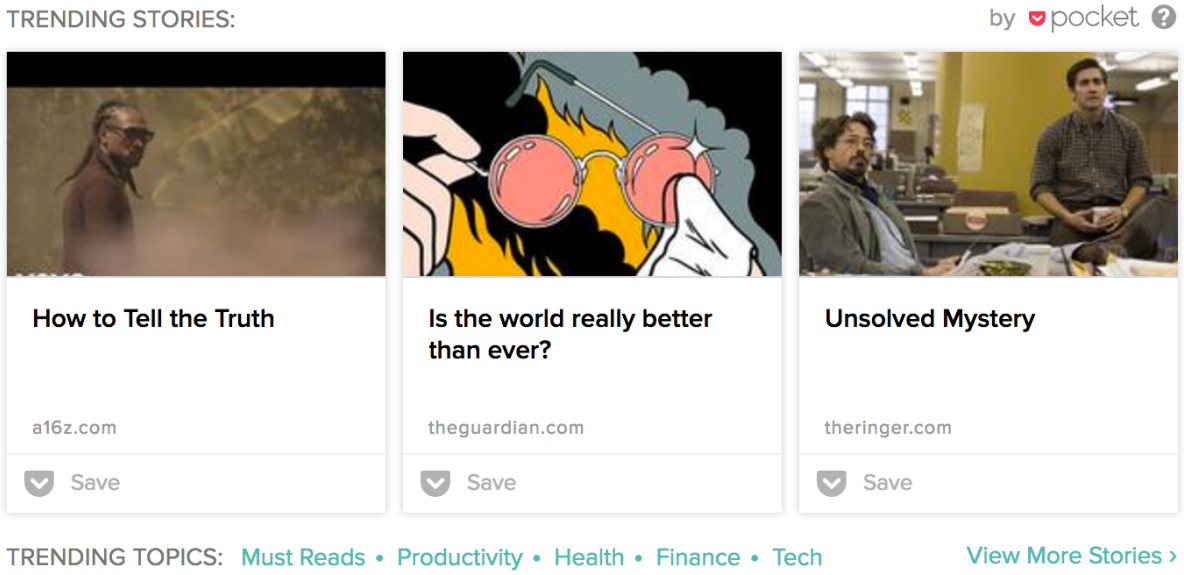
- Approach:

  Word Vectors → Document Vectors

  Suggest documents that are similar to:

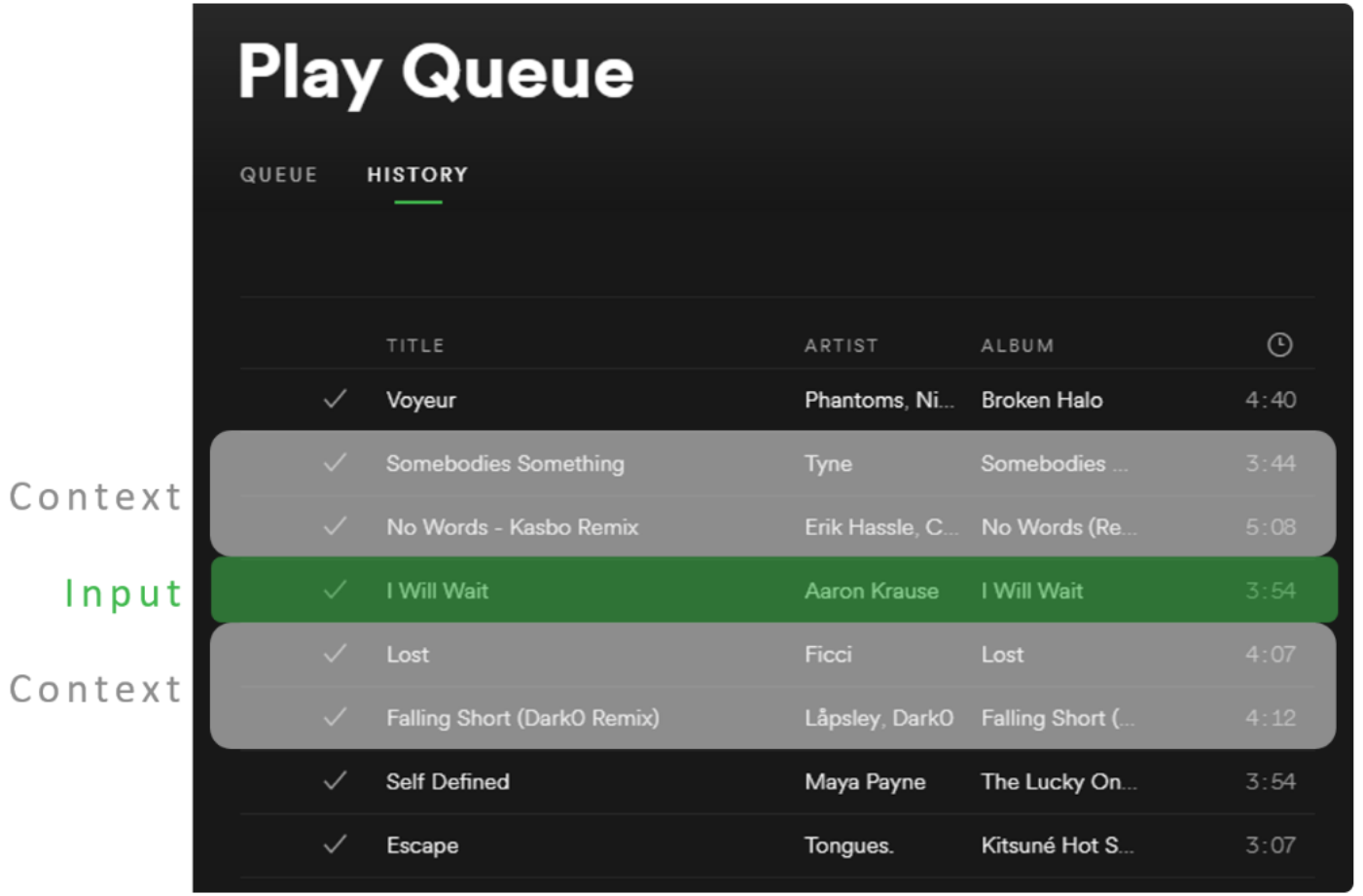
  a) User search query

  b) Similar articles that user 'favorited'



Application: Music Recommendation

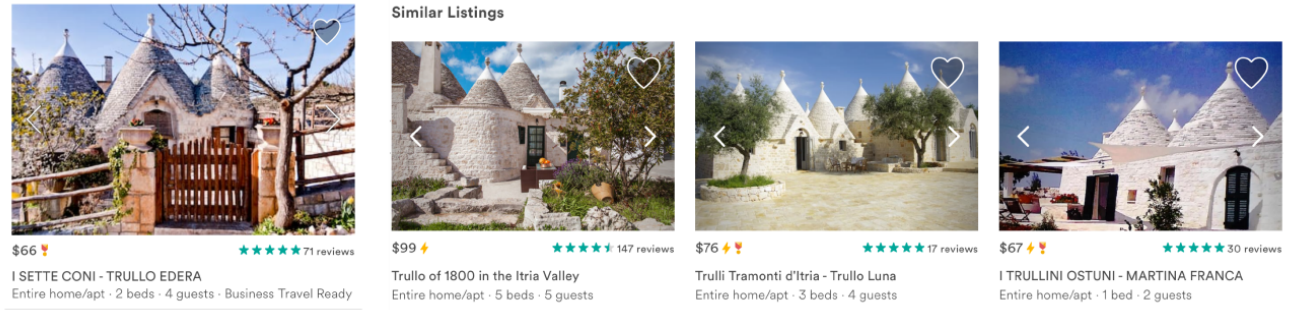
- Anghami streams over 700 million songs a month.
- Each listening queue is considered as a sentence. Each song is considered as a word
- Use word embedding to created "song vector" for each song in the context
- Find similar songs that frequently **played together**

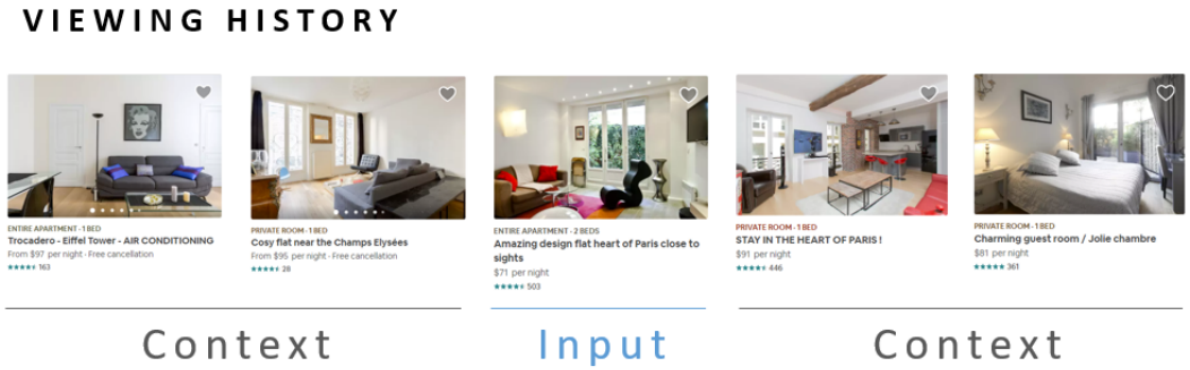


For more info: https://towardsdatascience.com/using-word2vec-for-music-recommendations-bb9649ac2484

Application: Listing Embedding

- AirBnB is a marketplace to list and match lodging, homestay and tourism accommodations
- Use very advance AI to rank and recommend similar listing to users
- Uses datasets of viewing history
- The sequence of clicks as context
- Each click contains features that can be embedded





For more info*: Listing Embeddings in Search Ranking*
https://medium.com/

Exercise 5

Using Word2Vec to solve document classification problem

Refer to ex5-word2vec-classification-problel.ipynb

This notebook will show you how to :

- Derive document vectors using Word2Vec
- Develop a classification model where the features of the document is based on vectors and not word count

References

Speech and Language Processing Chapter 6 - Vector Semantics and Embeddings, https://web.stanford.edu/~jurafsky/slp3/6.pdf

Vectoring Words (Word Embeddings) - Computerphile, https://youtu.be/gQddtTdmG_8

The Illustrated Word2vec, http://jalammar.github.io/illustrated-word2vec/

Efficient Estimation of Word Representations in Vector Space, https://arxiv.org/abs/1301.3781

Distributed Representations of Words and Phrases and their Compositionality, https://arxiv.org/abs/1310.4546

Linguistic Regularities in Continuous Space Word Representations, https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/rvecs.pdf

Gensim Word2Vec Tutorial, https://www.kaggle.com/pierremegret/gensim-word2vec-tutorial#Materials-for-more-in-depths-understanding:

Applying word2vec to Recommenders and Advertising , https://mccormickml.com/2018/06/15/applying-word2vec-to-recommenders-and-advertising/

Word Embeddings in NLP and its Applications , https://www.kdnuggets.com/2019/02/word-embeddings-nlp-applications.html