

**COPYRIGHT WARNING:** Copyright in these original lectures is owned by Monash University. You may transcribe, take notes, download or stream lectures for the purpose of your research and study only. If used for any other purpose, (excluding exceptions in the Copyright Act 1969 (Cth)) the University may take legal action for infringement of copyright.

Do not share, redistribute, or upload the lecture to a third party without a written permission!

## **FIT3181/5215 Deep Learning**

Week 08: Learning Representation and DL for Language:  
Word Embedding

### **Lecturers:**

Dr Trung Le

A/Prof Zongyuan Ge

Dr Arghya Pal

Email: [trunglm@monash.edu](mailto:trunglm@monash.edu)



Department of Data Science and AI  
Faculty of Information Technology, Monash University, Australia

# Outline

- Text Analytics and Language Models
- Learning representation in machine learning and deep learning
- Word embedding
  - Skip-gram
  - Continuous bag of words (CBOW)
  - Negative sampling
- Something to vector
  - Doc2Vec
  - Node2Vec

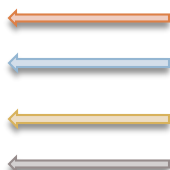
# Text Analytics and Language Models

# Text analytics

Real-world



Perceive



Observed world



Express



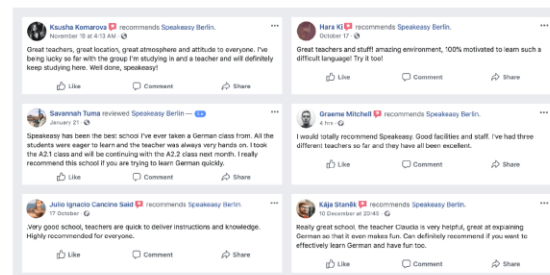
Text Data



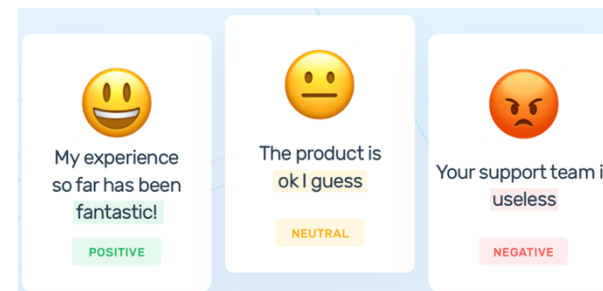
Text corpus



Movie reviews



Service reviews on FB



Sentiment analysis

Symbolic representation

Numeric representation



ML algorithm

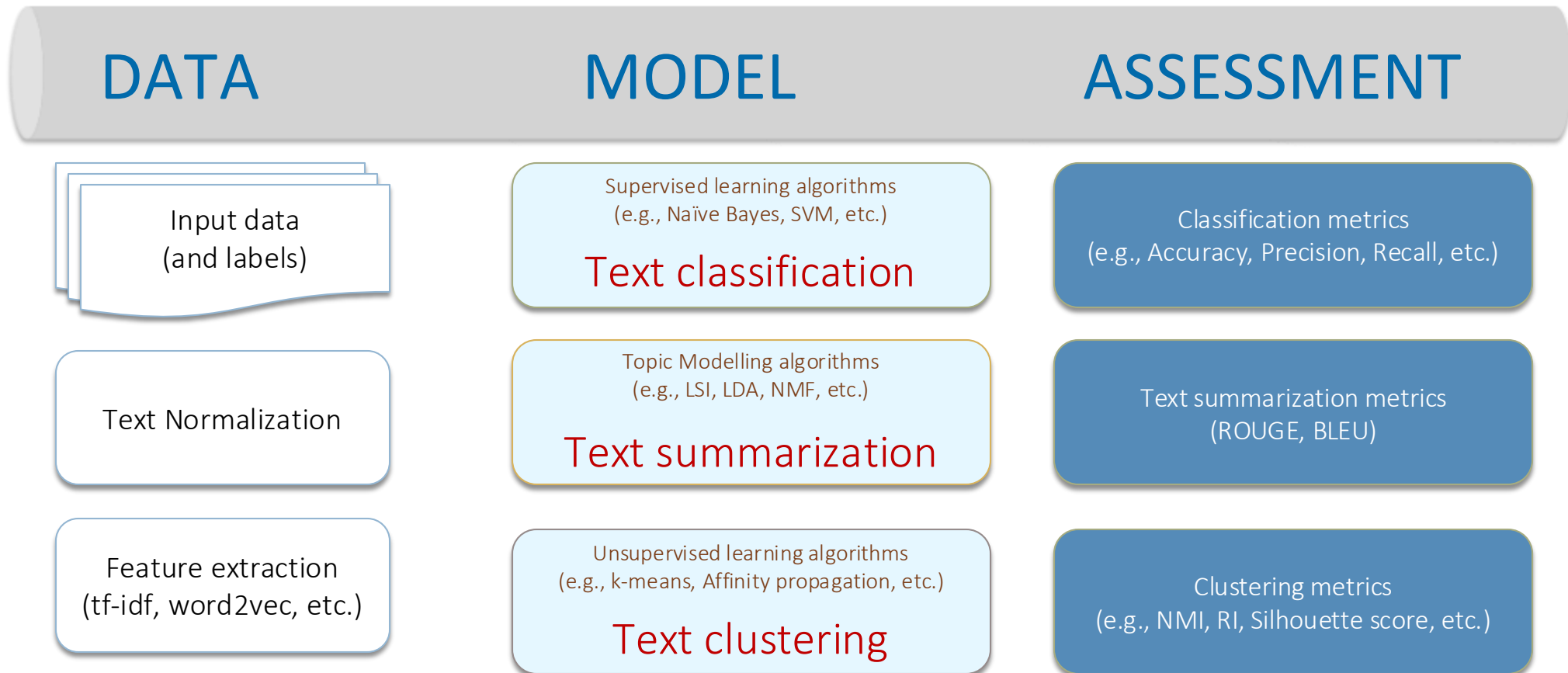


# Text Analytics and Language Models

- Text analytics' key tasks:
  - Text classification
  - Text clustering
  - Text summarization
- Some key applications
  - Spam detection
  - News articles categorization
  - Marketing and CRM (customer relationship management)
  - Recommendations

# Text Analytics and Language Models

## □ ML pipeline for Text analytics



# Text Analytics and Language Models

## □ Text normalization

### ○ Expanding contractions

- **Contractions** are shortened version of words or syllables
- e.g., isn't → is not, you're → you are
- Exist extensively and pose a problem to text analytics

### ○ Lemmatization

- removing word **affixes** to get to a **base form** of the *root* word.
- e.g. cars → car, running → run, is → be

### ○ Removing special characters and symbols

- e.g. !, .

### ○ Removing stop words

- e.g., a, and

# Text Analytics and Language Models

## □ Bag-of-word vector representation

Terms	Doc1	Doc2
goal		
data		
information		
insight		
you		

Document 1  
“The **goal** is to turn **data** into **information**, and **information** into **insight**”  
Carly Fiorina

Document 2  
“**You** can have **data** without **information**, but **you** cannot have **information** without **data**.”  
Daniel Keys Moran

$$\text{one\_hot}(\text{goal}) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\text{one\_hot}(\text{data}) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

...

# Text Analytics and Language Models

## □ Bag-of-word vector representation

Terms	Doc1	Doc2
goal	1	
data	1	
information	2	
insight	1	
you	0	

Document 1  
“The **goal** is to turn **data** into **information**, and **information** into **insight**”  
Carly Fiorina

Document 2  
“**You** can have **data** without **information**, but **you** cannot have **information** without **data**.”  
Daniel Keys Moran

`doc_1 = one_hot(goal) + one_hot(data) + one_hot(information) + one_hot(information) + one_hot(insight)`

$$= \begin{bmatrix} 1 \\ 1 \\ 2 \\ 1 \\ 0 \end{bmatrix}$$

# Text Analytics and Language Models

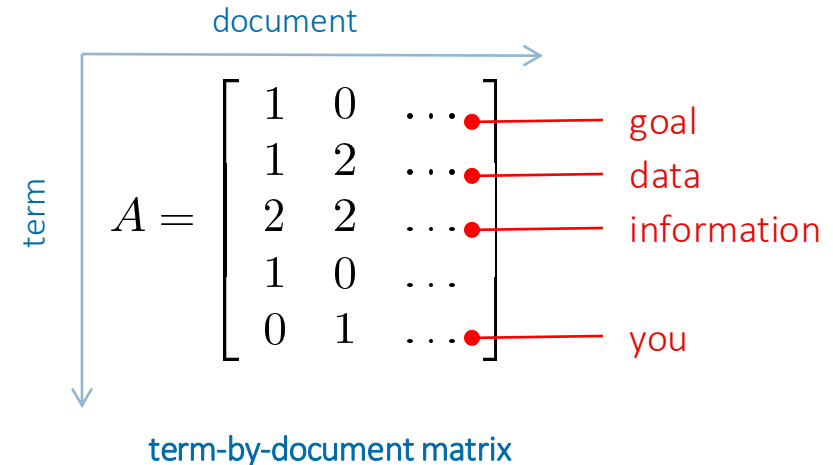
## □ Bag-of-word representation

Terms	Doc1	Doc2
goal	1	0
data	1	2
information	2	2
insight	1	0
you	0	1

$$doc_1 = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 1 \\ 0 \end{bmatrix} \quad doc_2 = \begin{bmatrix} 0 \\ 2 \\ 2 \\ 0 \\ 1 \end{bmatrix}$$

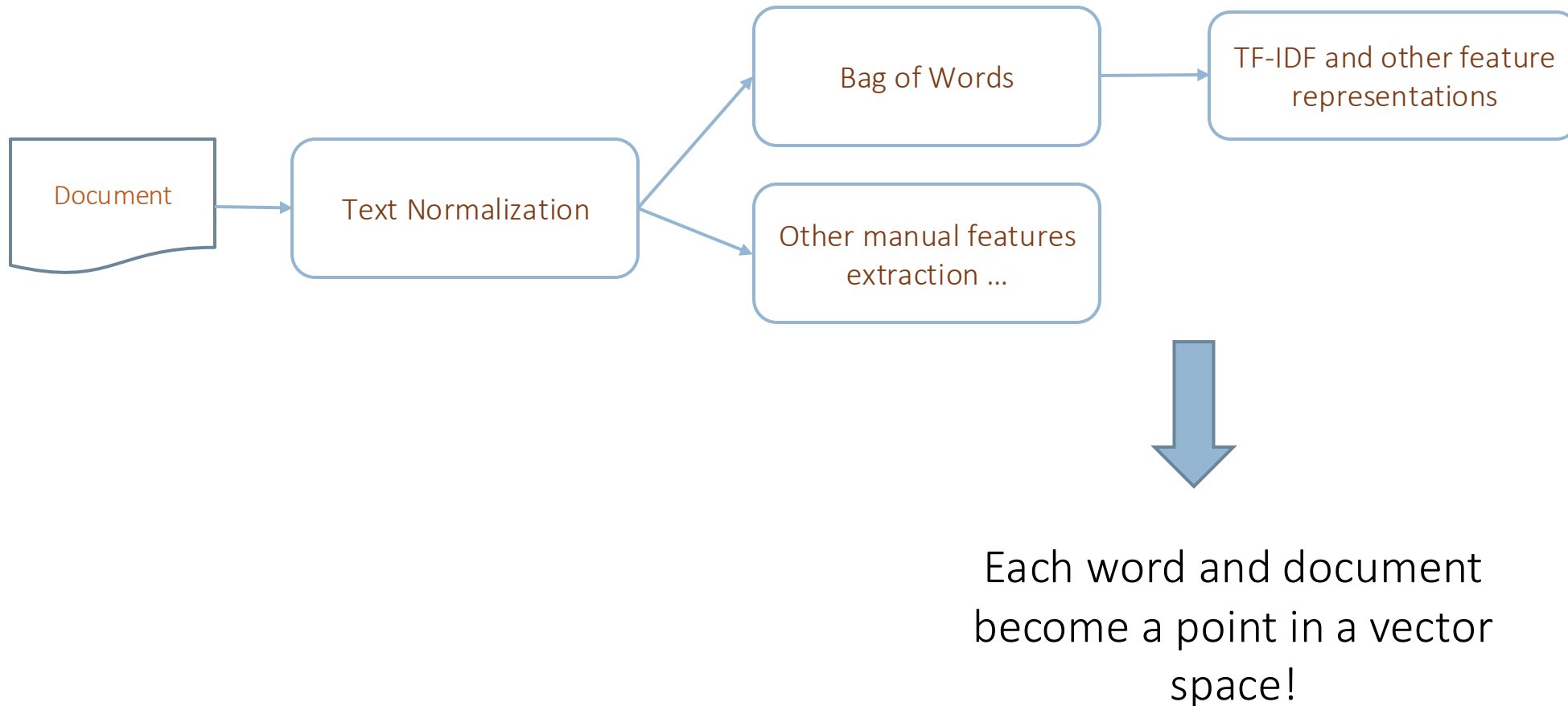
Document 1  
“The **goal** is to turn **data** into **information**, and **information** into **insight**”  
Carly Fiorina

Document 2  
“**You** can have **data** without **information**, but **you** cannot have **information** without **data**.”  
Daniel Keys Moran



# Text Analytics and Language Models

## □ Feature extraction



# Feature Extraction

## Tf-Idf weighting

- More information beyond word counts
- **TF**: term frequency - number of term occurrences in a document
- **IDF**: inverse document-frequency - how much information the term provides in corpus  $C$ .
  - $idf(t, C) = \log \frac{|C|}{|C_t|}$ , where
    - $|C|$ : the number of documents in the corpus
    - $|C_t| = |\{d \in C : t \in d\}|$ : the number of documents containing term  $t$
  - More documents contain term  $t$ , less information it provides ( $idf \rightarrow 0$ )
- **TF-IDF**:
  - $tfidf(t, d, C) = tf(t, d) \times idf(t, C)$
- Question: what happens if term  $t$  is not in the corpus, i.e.  $|C_t| = 0$ ?

# Feature Extraction

## Tf-Idf weighting

- Question: what happens if term  $t$  is not in the corpus?

- (One) solution - smoothing

- $idf(t, C) = \log \frac{1+|C|}{1+|C_t|}$

- Normalizing  $v = tfidf$

- $v_{norm} = \frac{v}{\sqrt{v_1^2 + \dots + v_n^2}}$

# TF-IDF example

term frequency (tf)

Terms	goal	data	information	insight	you
Doc1	1	1	2	1	0
Doc2	0	2	2	0	1

$DF(t)$  = Number of documents in which term  $t$  appears at least once.

document frequency (df)

Terms	goal	data	information	insight	you
df	1	2	2	1	1

inverse document frequency (idf)

Terms	goal	data	information	insight	you
idf	0.69	0	0	0.69	0.69

$$\log \frac{2}{1}$$

$$\log \frac{2}{2}$$

$$IDF(t) = \log \left( \frac{N}{DF(t)} \right)$$

Where:

- $N$  = total number of documents
- $DF(t)$  = number of documents containing the term  $t$

Document 1

“The **goal** is to turn **data** into **information**, and **information** into **insight**”  
Carly Fiorina

Document 2

“**You** can have **data** without **information**, but **you** cannot have **information** without **data**.”  
Daniel Keys Moran

Tf-idf

$$TF - IDF(t, d) = TF(t, d) \times IDF(t)$$

Terms	goal	data	information	insight	you
Doc1	0.69	0	0	0.69	0
Doc2	0	0	0	0	0.69

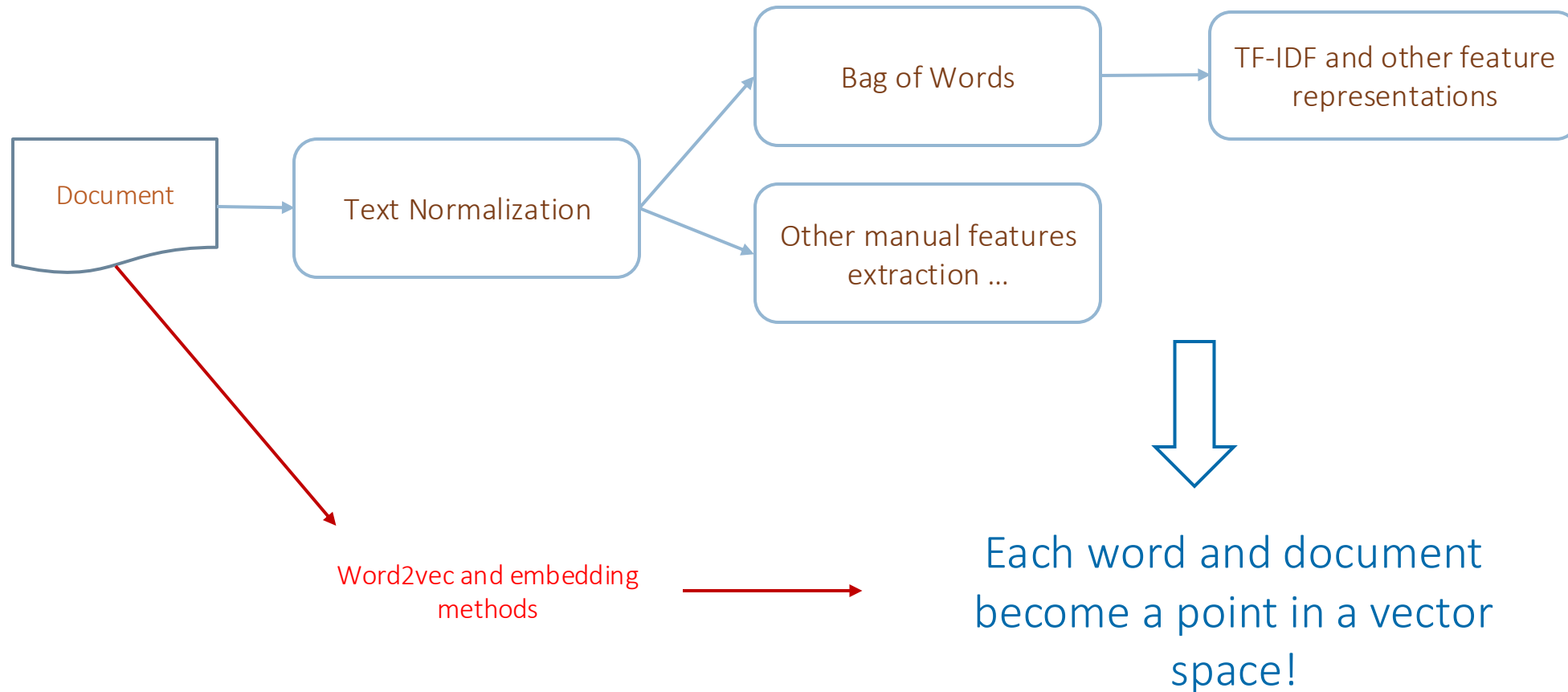
$$\frac{0.69}{\sqrt{0.69^2 + 0.69^2}}$$

tfidf (l2 normalized)

Terms	goal	data	information	insight	you
Doc1	0.71	0	0	0.71	0
Doc2	0	0	0	0	1

# Text Analytics and Language Models

## □ Feature extraction



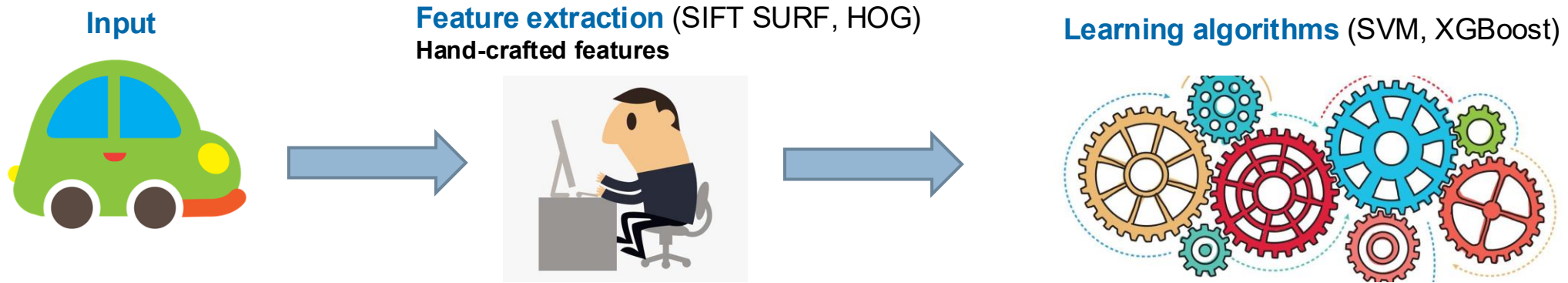


Machine Learning = Learning Representation++

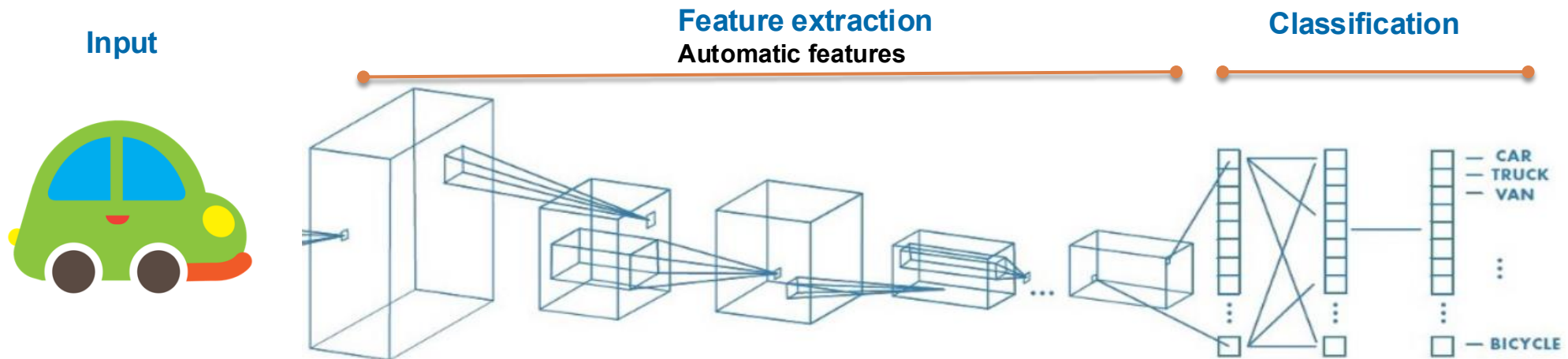
# Hand-crafted and automatic feature extractor

Visual data

## □ Traditional approach (hand-crafted feature learning)



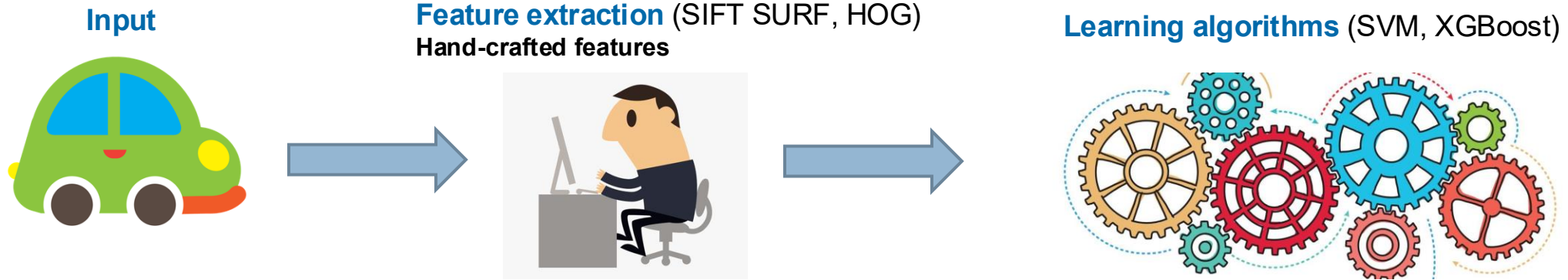
## □ Deep learning approach (automatic feature learning)



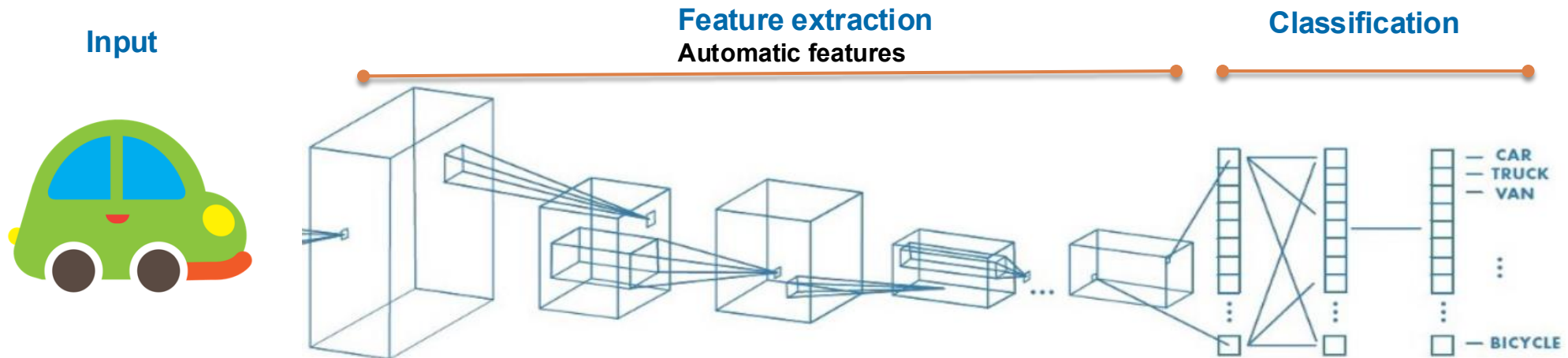
# Hand-crafted and automatic feature extractor

Visual data

## □ Traditional approach (hand-crafted feature learning)

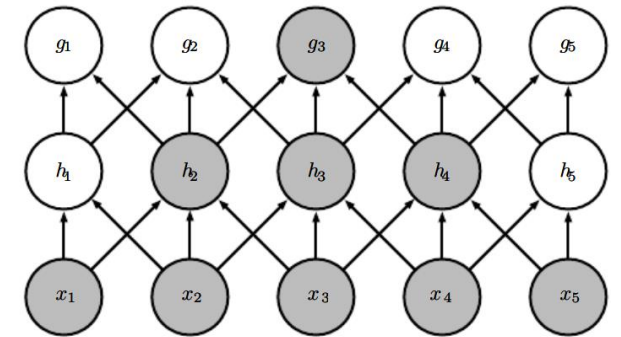
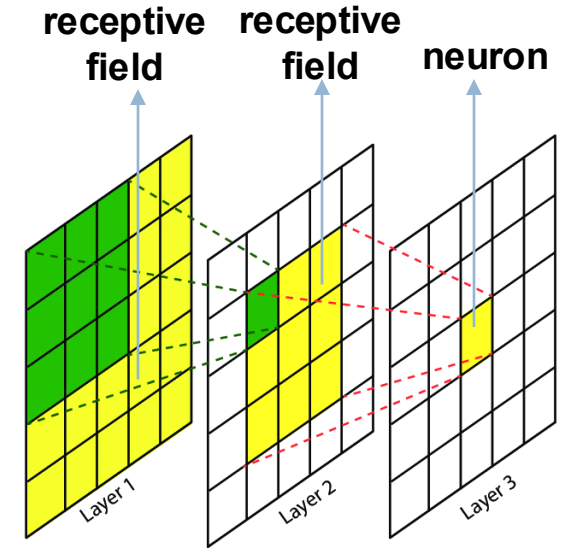
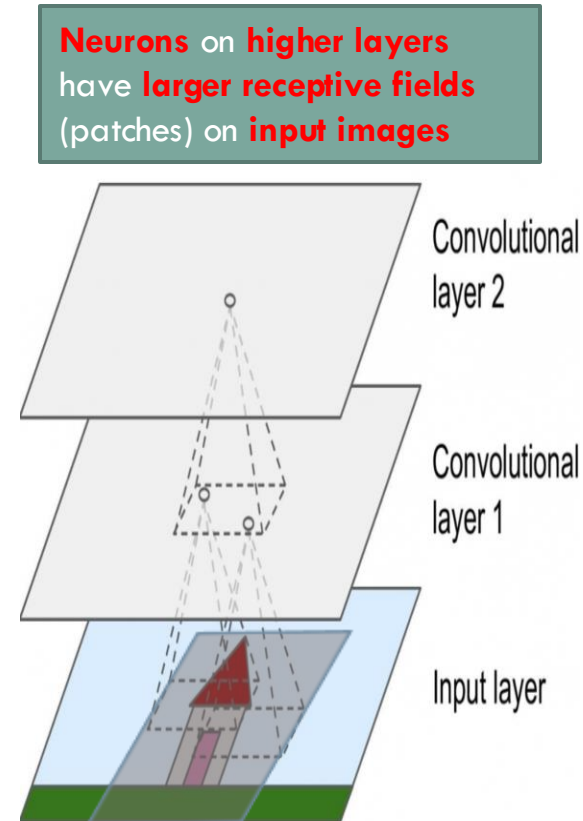
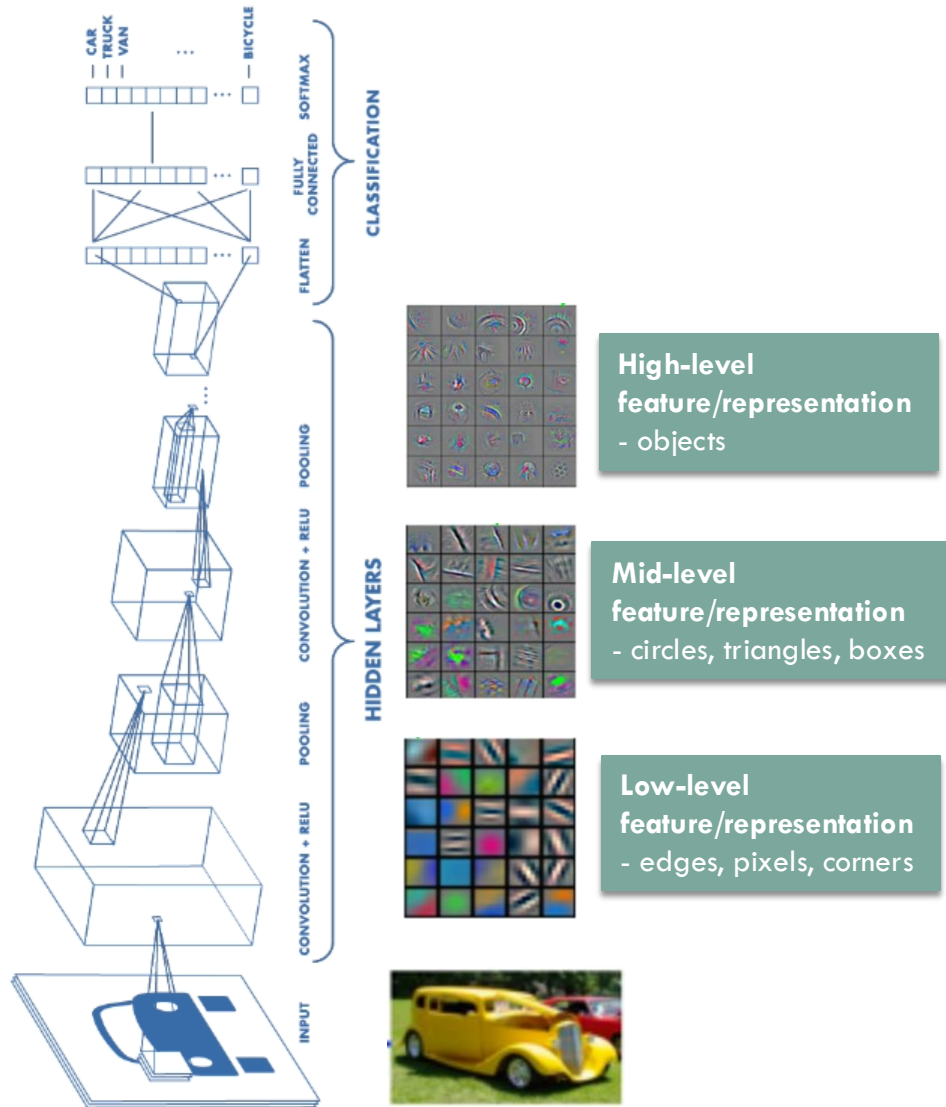


## □ Deep learning approach (automatic feature learning)



# Automatic feature extractor

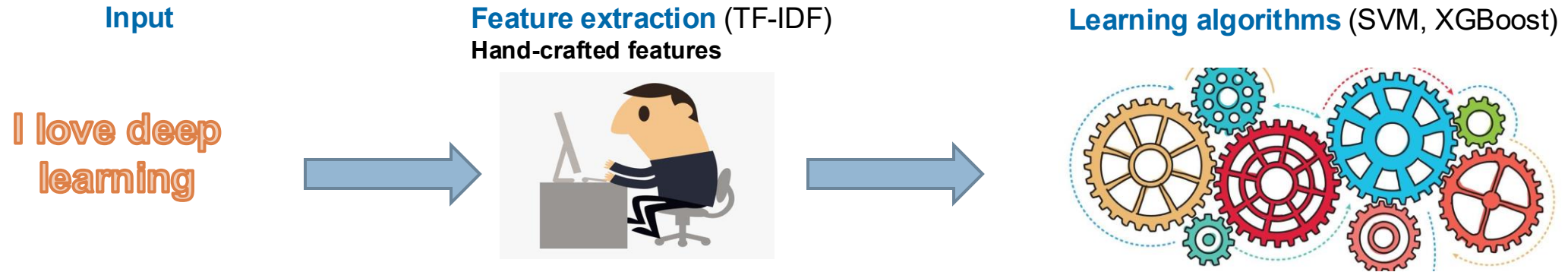
Deep learning for visual data



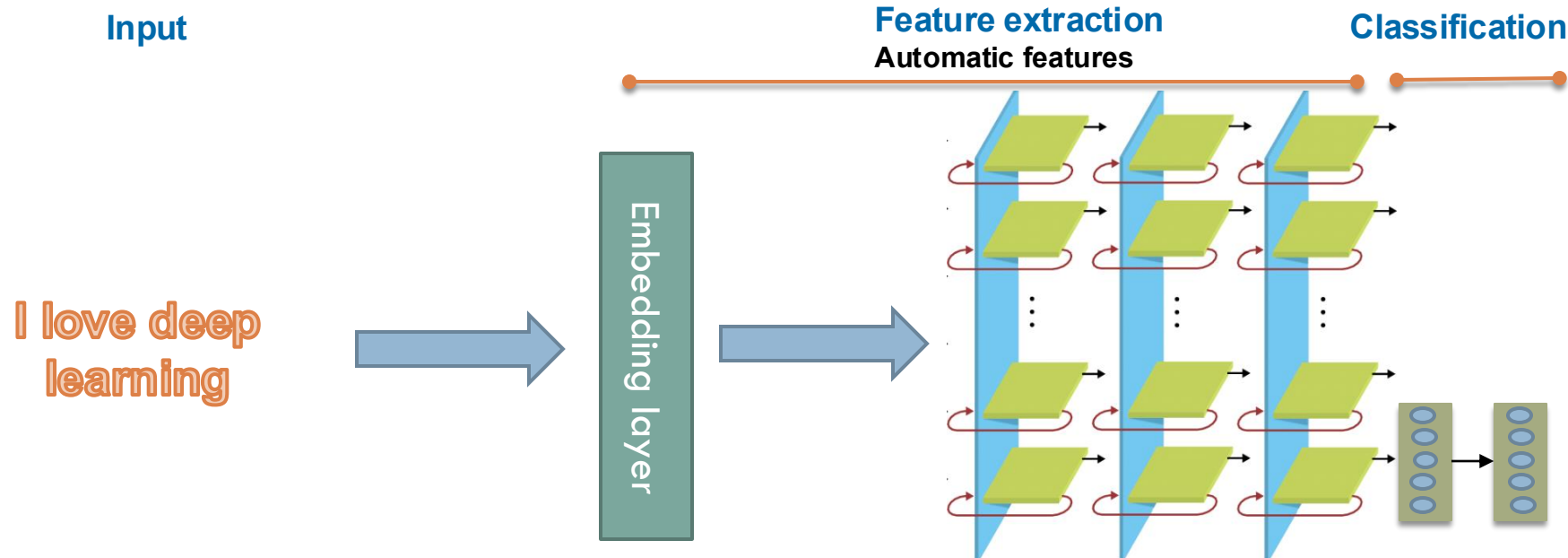
# Hand-crafted and automatic feature extractor

Sequential data

## □ Traditional approach (hand-crafted feature learning)



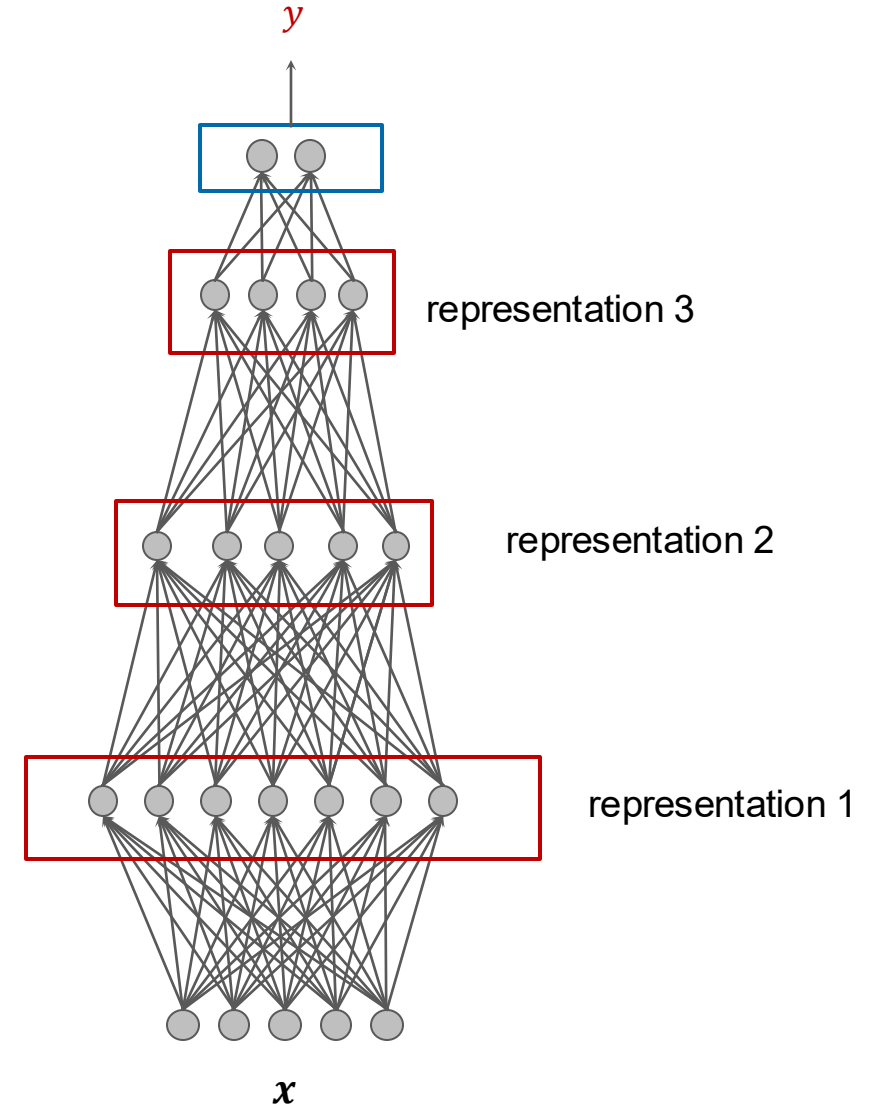
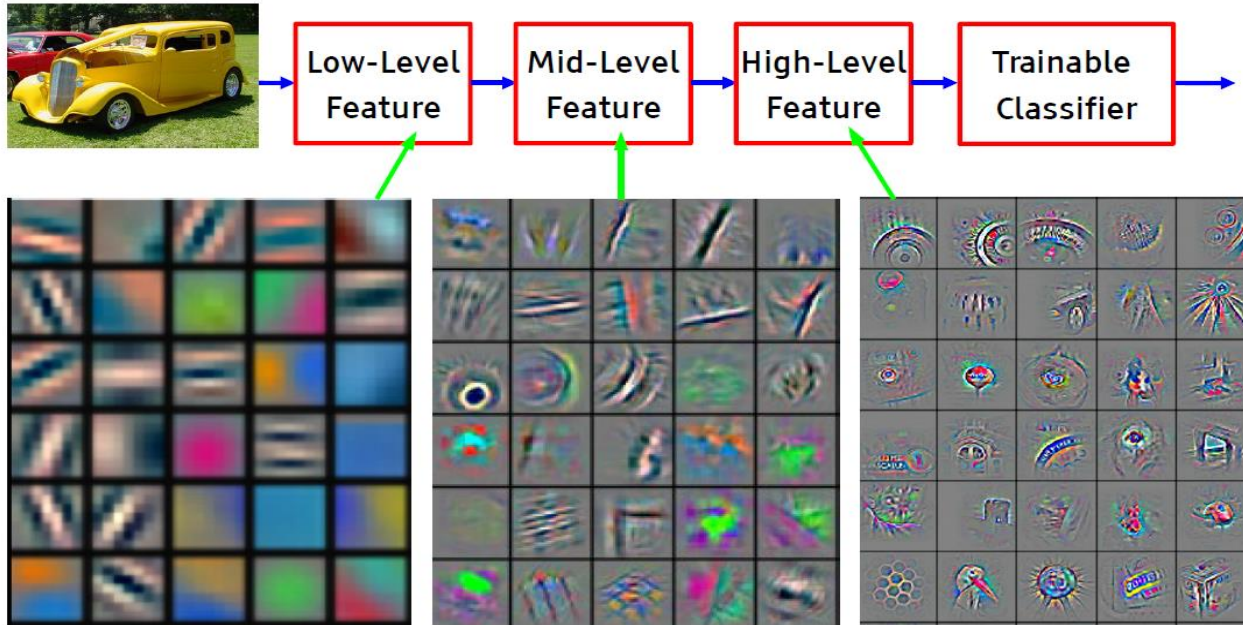
## □ Deep learning approach (automatic feature learning)



# Learning representation in deep learning

“Deep Learning: machine learning algorithms based on learning **multiple levels** of representation and abstraction”

*Yoshua Bengio*



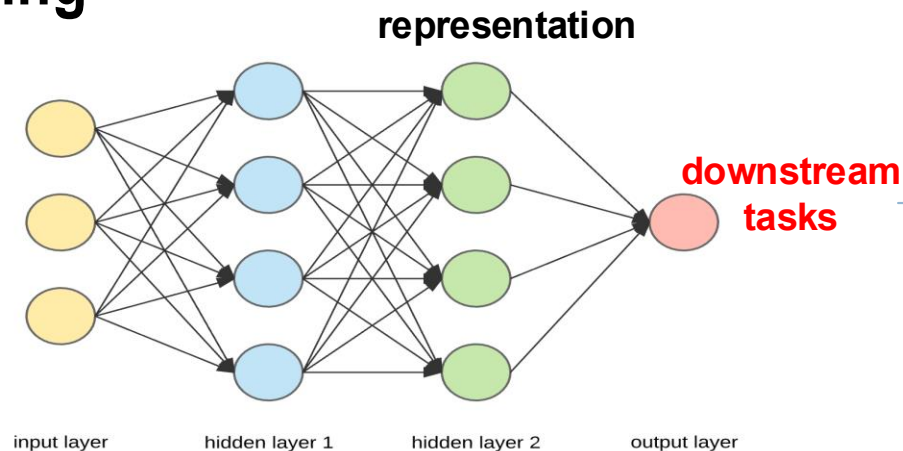
# Learning representation in Deep Learning

## Supervised representation learning

### □ Supervised representation learning



Raw data with labels for downstream task



Image, text classification  
Regression  
Object recognition  
Image segmentation  
Name entity recognition  
....

### □ Learning the representation that fits a specific downstream task.

## Unsupervised representation learning



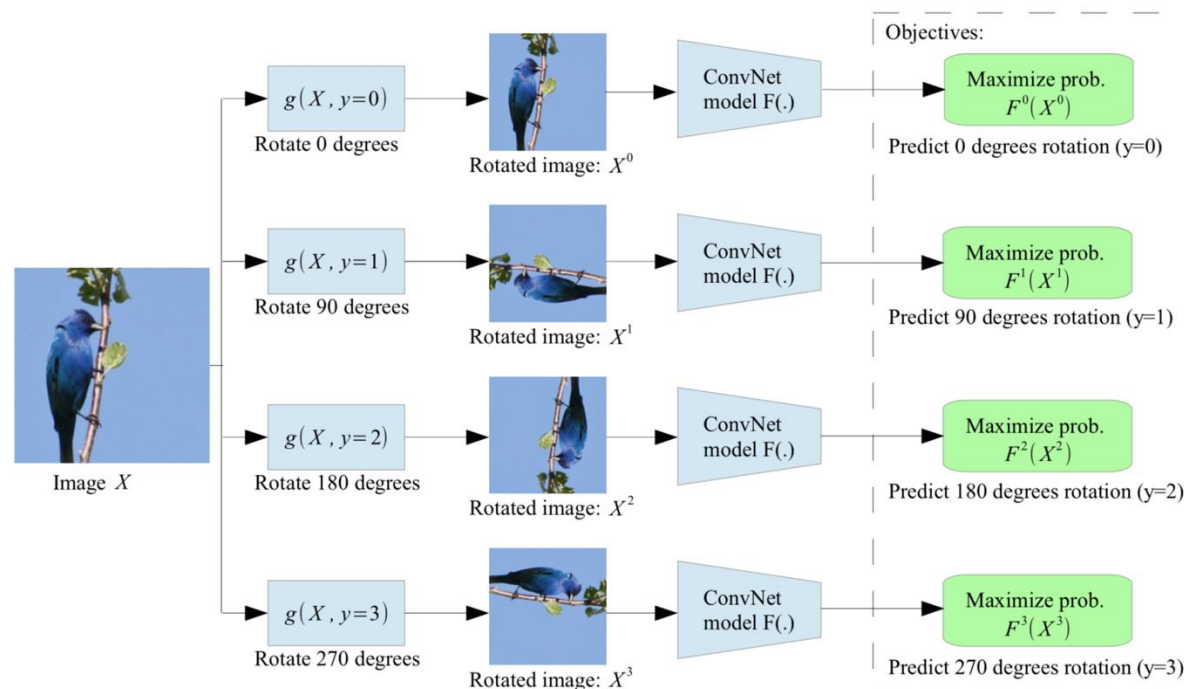
Unseen task 1  
Unseen task 2  
Unseen task 3  
....  
....  
....

- 

# Learning representation in Deep Learning

## Self-supervised learning

- A possible and efficient workaround for unsupervised representation learning.
- Deep learning is **only good** at **supervised learning** that requires **labels**
- What if we have **only raw data** (images)?
  - Devise **pretext task** to require the model to **predict something** → supervised learning.
  - The art is to devise the **good** and **meaningful pretext task**.



### Pretext task:

- Rotate images  $0^\circ, 90^\circ, 180^\circ, 270^\circ$  and try to **predict the angle**
- **4 labels** for  $0^\circ, 90^\circ, 180^\circ, 270^\circ$ .

# Word embedding (Word2Vec)

## Wikipedia text corpus

Association football

Royal family

British royal family

From Wikipedia, the free encyclopedia

*For the history of the monarchy, see [Monarchy of the United Kingdom](#)*

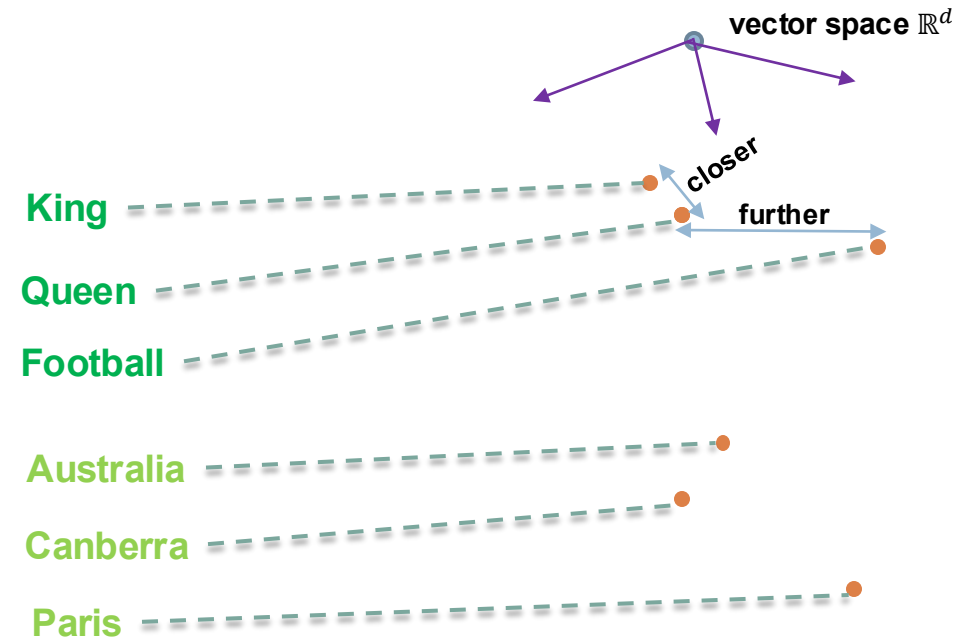
The **British royal family** comprises [Queen Elizabeth II](#) and her close relations. There is no strict legal or formal definition of who is or is not a member of the British royal family.

Those who at the time are entitled to the style [His or Her Royal Highness \(HRH\)](#), and any styled [His or Her Majesty \(HM\)](#), are normally considered members, including those so styled before the beginning of the current monarch's reign. By this criterion, a list of the current royal family will usually include the monarch, the children and male-line grandchildren of the monarch and previous monarchs, the children of the eldest son of the [Prince of Wales](#), and all of their current or widowed spouses.

Some members of the royal family have official residences named as the places from which announcements are made in the [Court Circular](#) about official engagements they have carried out. The state duties and staff of some members of the royal family are funded from a parliamentary annuity, the amount of which is fully refunded by the Queen to the Treasury.<sup>[1]</sup>

Since 1917, when [King George V](#) changed the name of the royal house from [Saxe-Coburg and Gotha](#), members of the royal family have belonged, either by birth or by marriage, to the [House of Windsor](#). Senior titled members of the royal family do not usually use a [surname](#), although since 1960 [Mountbatten-Windsor](#), incorporating [Prince Philip's](#) adopted surname of [Mountbatten](#), has been prescribed as a surname for Elizabeth II's direct descendants who do not have royal styles and titles, and it has sometimes been used when required for those who do have such titles. The royal family are regarded as British [cultural icons](#), with young adults from abroad naming the family among a group of people that they most associated with [British culture](#).<sup>[2]</sup>

We desire...



Canberra : Australia = Paris : ???

$$\operatorname{argmin}_v \|v_{\text{Canberra}} - v_{\text{Australia}} + v_{\text{Paris}} - v\| = v_{\text{France}}$$

- **We have:** Many texts in Wikipedia
- **We want:** Learn vector representations for words that preserve semantic and linguistic relationship carried in the text corpus
- **We need:** Devise pretext task to cast the learning word representation to supervised learning.



Word embedding

# Motivation of Word2Vec

“You shall know a word by  
the company it keeps”



J.R. Firth 1957

# Word2Vec: Pretext task

## Pretext task

- What is the **pretext task** of Word2Vec?

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

context word   target word   context word

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

Original sentence

### Skip-gram

- **Target word**  $\xrightarrow{\text{predict}}$  **context words**

### Continuous Bag of Words (CBOW)

- **Context words**  $\xrightarrow{\text{predict}}$  **target word**

# Skip-gram

## Pretext task

- What is the **pretext task** of Skip-gram?

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

context word   target word   context word

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

## Skip-gram

- Target word  $\xrightarrow{\text{predict}}$  context words

(brown, the), (brown, quick), (brown, fox),  
(brown, jumps)

(fox, quick), (fox, brown), (fox, jumps), (fox, over)

(jumps, brown), (jumps, fox), (jumps, over),  
(jumps, the)

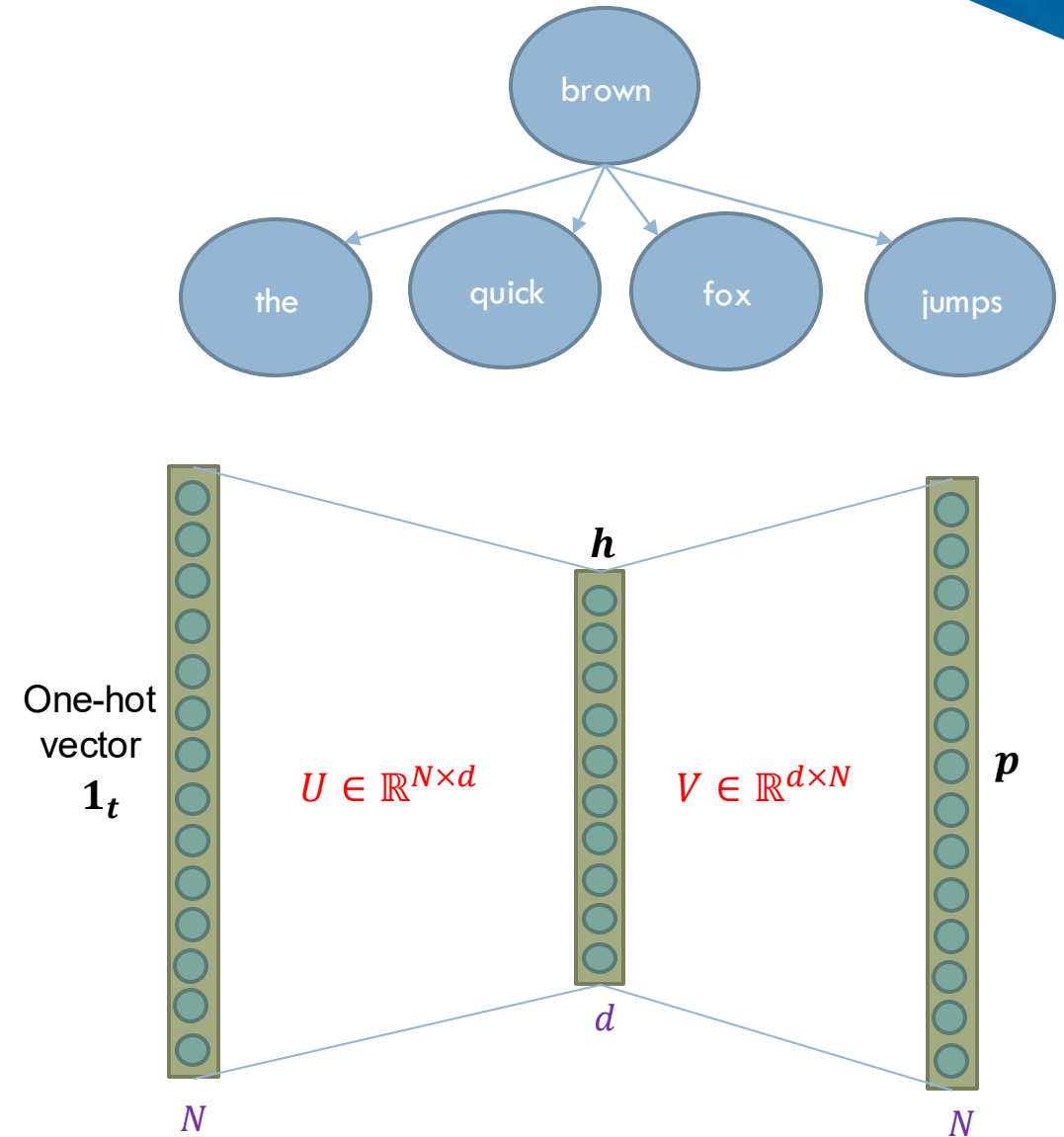
(over, fox), (over, jumps), (over, the), (over, lazy)

(the, jumps), (the, over), (the, lazy), (the, dogs)

# Skip-gram

## Modelling

- **Current window**
  - The quick **brown** fox jumps
- $P(\text{the, quick, fox, jumps} \mid \text{brown}) = P(\text{the} \mid \text{brown}) \times P(\text{quick} \mid \text{brown}) \times P(\text{fox} \mid \text{brown}) \times P(\text{jumps} \mid \text{brown})$
- $\log P(\text{the, quick, fox jumps} \mid \text{brown}) = \log P(\text{the} \mid \text{brown}) + \dots + \log P(\text{jumps} \mid \text{brown})$
- $(\text{brown}, \text{the}), (\text{brown}, \text{quick}), (\text{brown}, \text{fox}), (\text{brown}, \text{jumps})$ . Let consider  $tw(\text{target word}) = \text{brown}$  and  $cw(\text{context word}) = \text{the}$ .
- **Two matrices**
  - $U \in \mathbb{R}^{N \times d}$  ( $N$  is vocabulary size,  $d$  is embedding size)
  - $V \in \mathbb{R}^{d \times N}$
- Assume that indices of  $tw$  and  $cw$  are  $1 \leq t, c \leq N$  respectively. The forward propagation is as follows:
  - $h = \mathbf{1}_t U = U_t^r \in \mathbb{R}^{1 \times d}$ ,  $o = hV \in \mathbb{R}^{1 \times N}$ ,  $p = \text{softmax}(o) \in \mathbb{R}^{1 \times N}$
  - $P(cw = \text{the} \mid tw = \text{brown}) = p_c$
  - $\log P(cw = \text{the} \mid tw = \text{brown}) = \log p_c = U_t^r V_c^c - \log(\sum_{k=1}^N \exp(U_t^r V_k^c))$
- Train the model by **maximizing log likelihood**.



# Toy example

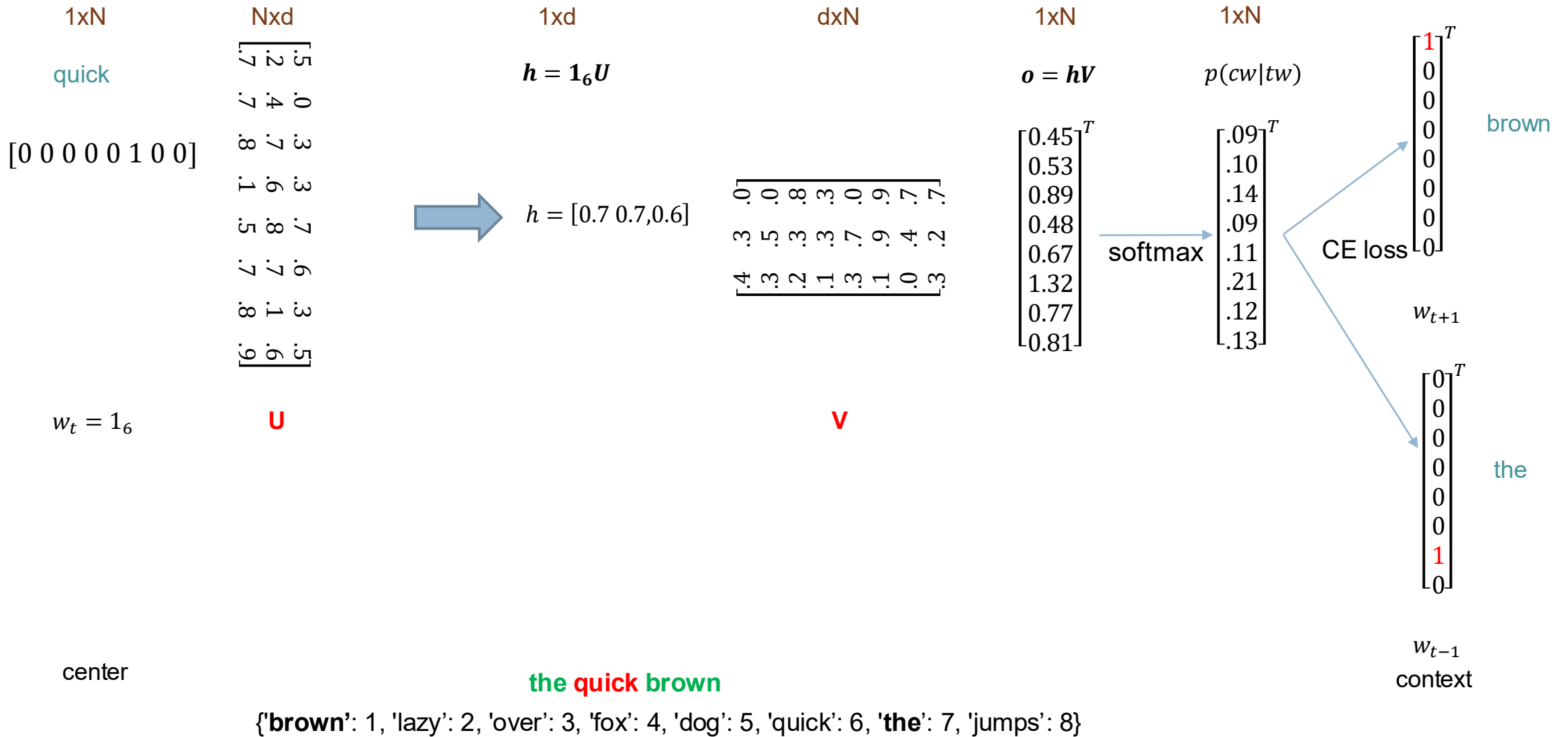
- Corpus: **the quick brown fox jumps over the lazy dog**
  - **Tokens:** {'brown': 1, 'lazy': 2, 'over': 3, 'fox': 4, 'dog': 5, 'quick': 6, 'the': 7, 'jumps': 8}
  - **Number of tokens**  $N = 8$
  - **Context (window) size**  $C = 3$
  - **Size of embedded vectors**  $d = 3$
  - **U&V:** collections of input & output vectors

quick

[0 0 0 0 0 1 0 0]

one-hot encoding

# Skip-gram, forward propagation



{**brown**: 1, 'lazy': 2, 'over': 3, 'fox': 4, 'dog': 5, 'quick': 6, **the**: 7, 'jumps': 8}

# Skip-gram

## Drawback

### Two matrices

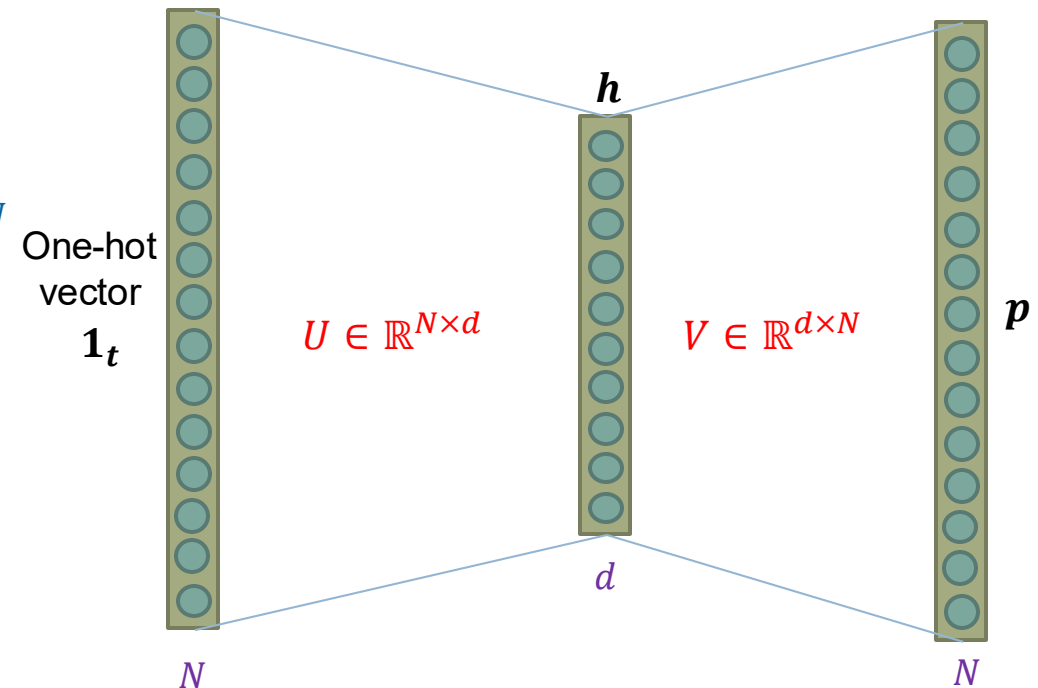
- $U \in \mathbb{R}^{N \times d}$  ( $N$  is vocabulary size,  $d$  is embedding size)
- $V \in \mathbb{R}^{d \times N}$

### Assume that indices of $tw$ and $cw$ are $1 \leq t, c \leq N$ respectively. The forward propagation is as follows:

- $h = \mathbf{1}_t U = U_t^r \in \mathbb{R}^{1 \times d}$ ,  $o = hV \in \mathbb{R}^{1 \times N}$ ,  $p = \text{softmax}(o) \in \mathbb{R}^{1 \times N}$
- $P(cw = the \mid tw = brown) = p_c$
- $\log P(cw = the \mid tw = brown) = \log p_c = U_t^r V_c^c - \log(\sum_{k=1}^N \exp(U_t^r V_k^c))$

### Some drawbacks

- $p = \text{softmax}(o)$  is **computationally expensive**
- $p \in \mathbb{R}^{1 \times N}$  is a distribution over the **vocabulary with size  $N$**  (usually very big)  $\rightarrow$  the values of  $p_i$  are **very tiny**  $\rightarrow$  **hard to train**
  - Hierarchical SoftMax
  - Negative sampling (more popular and efficient)

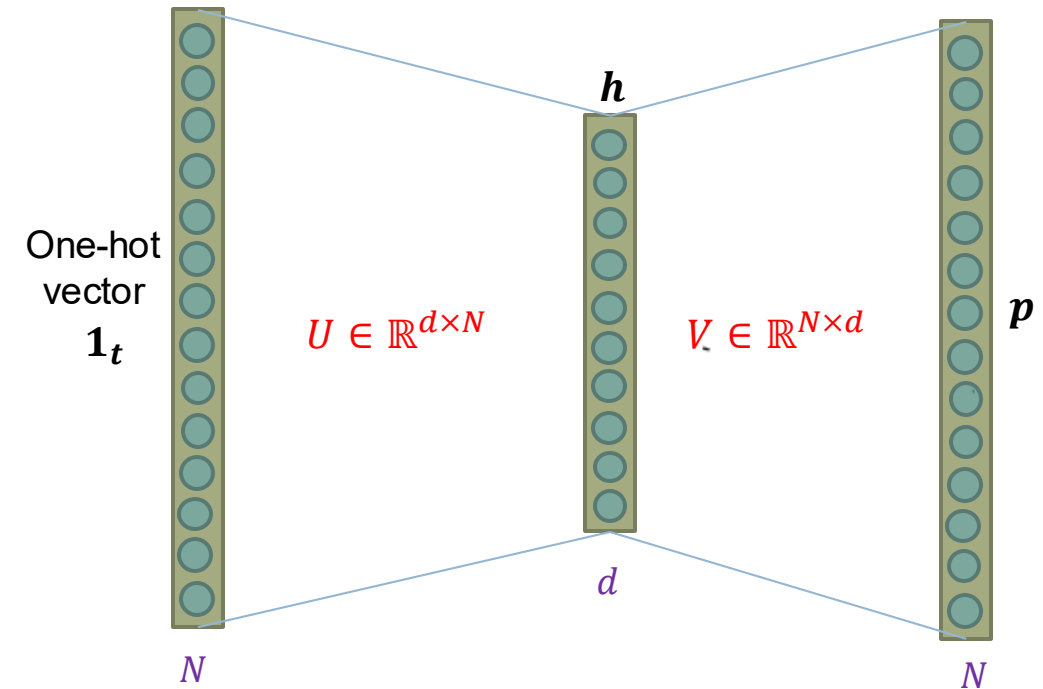


Instead of computing a full softmax over the vocabulary, you **only update the target word and a few “negative” words**.

# Skip-gram

## Negative sampling

- Transform  $N$ -class prediction to binary prediction with negative examples
- Consider a **positive (true) pair** (tw=**brown**, cw = the)
  - $[(\text{brown}, \text{the}), 1]$
  - Sample randomly some (two) words
    - $[(\text{brown}, ng_1 = \text{hello}), 0]$  and  $[(\text{brown}, ng_2 = \text{awsome}), 0]$
    - Let denote the indices of  $ng_1$  and  $ng_2$  by  $1 \leq n_1, n_2 \leq N$ .
- The forward propagation
  - $h = 1_t U = U_t^r \in \mathbb{R}^{1 \times d}$ ,  $o = hV \in \mathbb{R}^{1 \times N}$ ,  $p = \text{sigmoid}(o) \in \mathbb{R}^{1 \times N}$
  - $P(y = 1 \mid \text{tw}=\text{brown}, \text{cw}=\text{the}) = p_c$
  - $P(y = 1 \mid \text{tw}=\text{brown}, ng_1=\text{hello}) = p_{n_1}$
  - $P(y = 1 \mid \text{tw}=\text{brown}, ng_2=\text{awsome}) = p_{n_2}$
- Optimization problem
  - $\max[\log p_c - \alpha \log p_{n_1} - \alpha \log p_{n_2}]$  where  $\alpha > 0$  is a trade-off parameter.



# Continuous Bag of Words (CBOW)

## Pretext task

- What is the **pretext task** of CBOW?

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

context word		target word	context word					
The	quick	brown	fox	jumps	over	the	lazy	dogs.

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

The	quick	brown	fox	jumps	over	the	lazy	dogs.
-----	-------	-------	-----	-------	------	-----	------	-------

## Continuous Bag of Words (CBOW)

- **Context words** <sup>predict</sup> **target word**

(the | quick | fox | jumps, brown)

(quick | brown | jumps | over, fox)

(brown | fox | over | the, jumps)

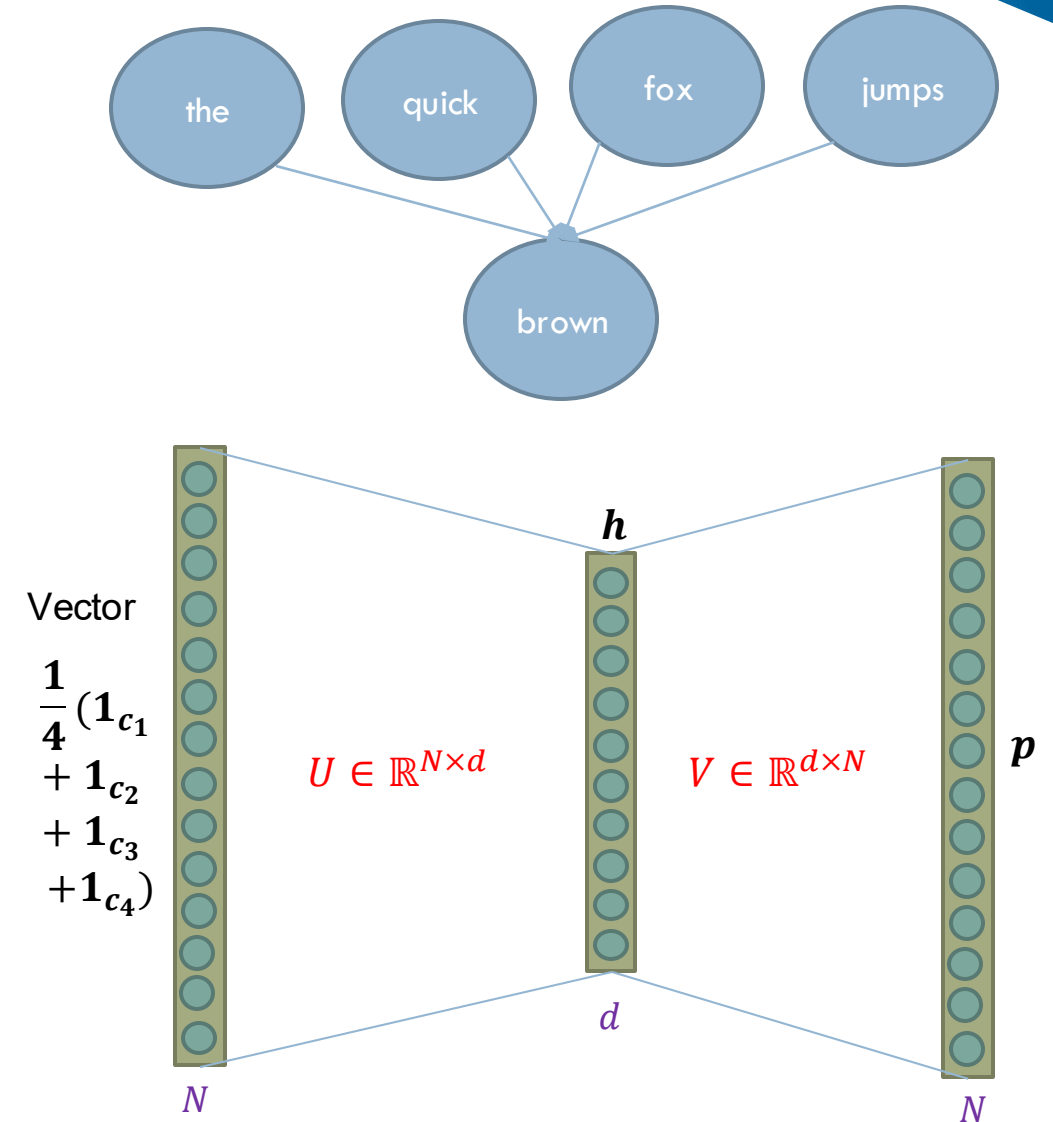
(fox | jumps | the | lazy, over)

(jumps | over | lazy | dog, the)

# Continuous Bag of Words (CBOW)

## Modelling

- **Current window**
  - The quick **brown** fox jumps
- Need to formulate:  $P(\text{brown} \mid \text{the}, \text{quick}, \text{fox}, \text{jumps})$
- $(\text{the} \mid \text{quick} \mid \text{fox} \mid \text{jumps}, \text{brown})$ . Let consider  $tw = \text{brown}$  and  $cw_1 = \text{the}$ ,  $cw_2 = \text{quick}$ ,  $cw_3 = \text{fox}$ ,  $cw_4 = \text{jumps}$ .
- **Two matrices**
  - $U \in \mathbb{R}^{N \times d}$  ( $N$  is vocabulary size,  $d$  is embedding size)
  - $V \in \mathbb{R}^{d \times N}$
- Assume that indices of  $tw$  is  $1 \leq t \leq N$  and  $cw_{1:4}$  are  $1 \leq c_{1:4} \leq N$  respectively. The forward propagation is as follows:
  - $h = \frac{1_{c_1} + \dots + 1_{c_4}}{4} U = \frac{1}{4} (U_{c_1}^r + \dots + U_{c_4}^r) = \overline{U}^r \in \mathbb{R}^{1 \times d}$ ,  $o = hV \in \mathbb{R}^{1 \times N}$ ,  $p = \text{softmax}(o) \in \mathbb{R}^{1 \times N}$
  - $P(\text{brown} \mid \text{the}, \text{quick}, \text{fox}, \text{jumps}) = p_t$
  - $\log P(\text{brown} \mid \text{the}, \text{quick}, \text{fox}, \text{jumps}) = \log p_t = \overline{U}^r V_t^c - \log(\sum_{k=1}^N \exp(\overline{U}^r V_k^c))$
- Train the model by **maximizing log likelihood**.



# CBOW: forward propagation

brown

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}^T$$

 $w_{t+1}$ 

the

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}^T$$

 $w_{t-1}$ 

context

Nx d

$$\begin{bmatrix} .5 & .0 & .3 \\ .2 & .4 & .7 \\ .7 & .7 & .8 \\ .1 & .6 & .3 \\ .5 & .8 & .7 \\ .5 & .7 & .6 \\ .8 & .1 & .3 \\ .9 & .6 & .5 \end{bmatrix}$$

**U**

1x d

$$h = \frac{1_1 + 1_7}{2} U$$

$$h = [0.75 \ 0.15 \ 0.4]$$

d x N

$$\begin{bmatrix} .4 & .3 & .2 & .1 & .3 & .0 & .3 \\ .3 & .5 & .3 & .7 & .9 & .4 & .2 \\ .0 & .8 & .3 & .0 & .9 & .7 & .7 \end{bmatrix}$$

**V**

1 x N

$$o = Vh$$

$$\begin{bmatrix} 0.55 \\ 0.44 \\ 0.93 \\ 0.40 \\ 0.46 \\ 0.94 \\ 0.60 \\ 0.97 \end{bmatrix}^T$$

softmax

1 x N

$$p(o|i)$$

$$\begin{bmatrix} .11 \\ .10 \\ .16 \\ .09 \\ .10 \\ .16 \\ .11 \\ .17 \end{bmatrix}^T$$

1 x N

quick

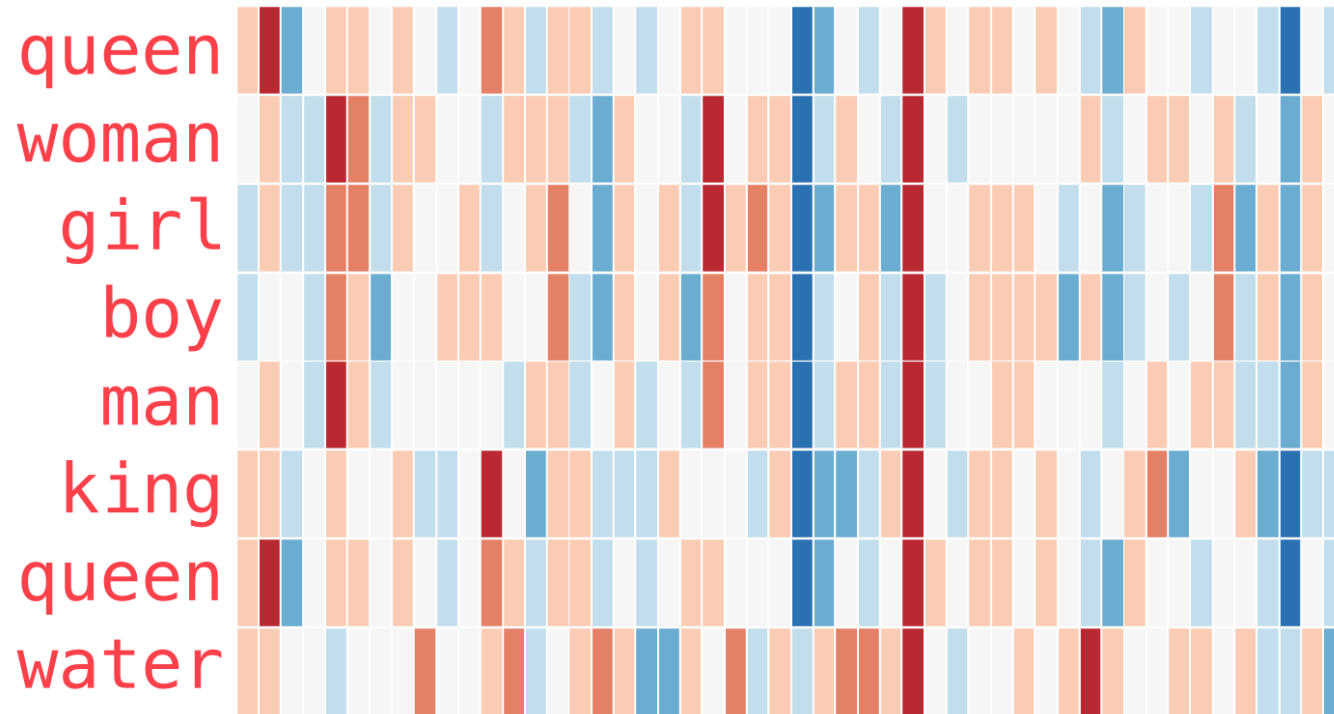
$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}^T$$

 $w_t$ 

center

CE loss

# Visualization of Word2Vec representations

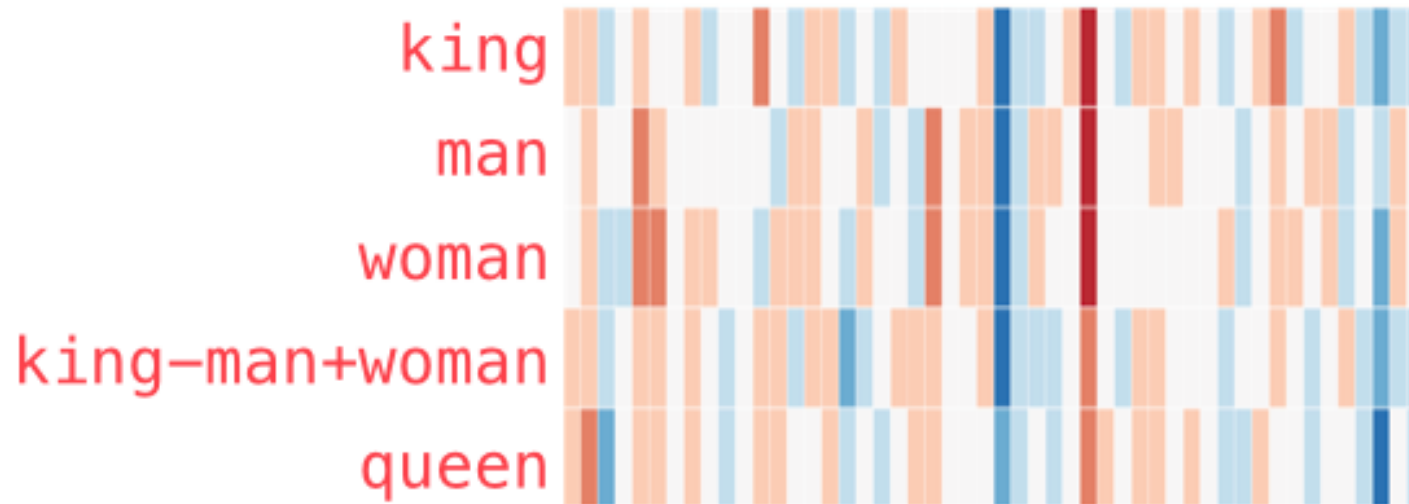


(Source: <http://jalammar.github.io/>)

It's a **heatmap** that uses colors to represent the numerical values in the **vector representations** (embeddings) for a list of words. Each row corresponds to a word, and each column represents a single dimension within the word's vector.

# Visualization of Word2Vec representations

king - man + woman  $\approx$  queen

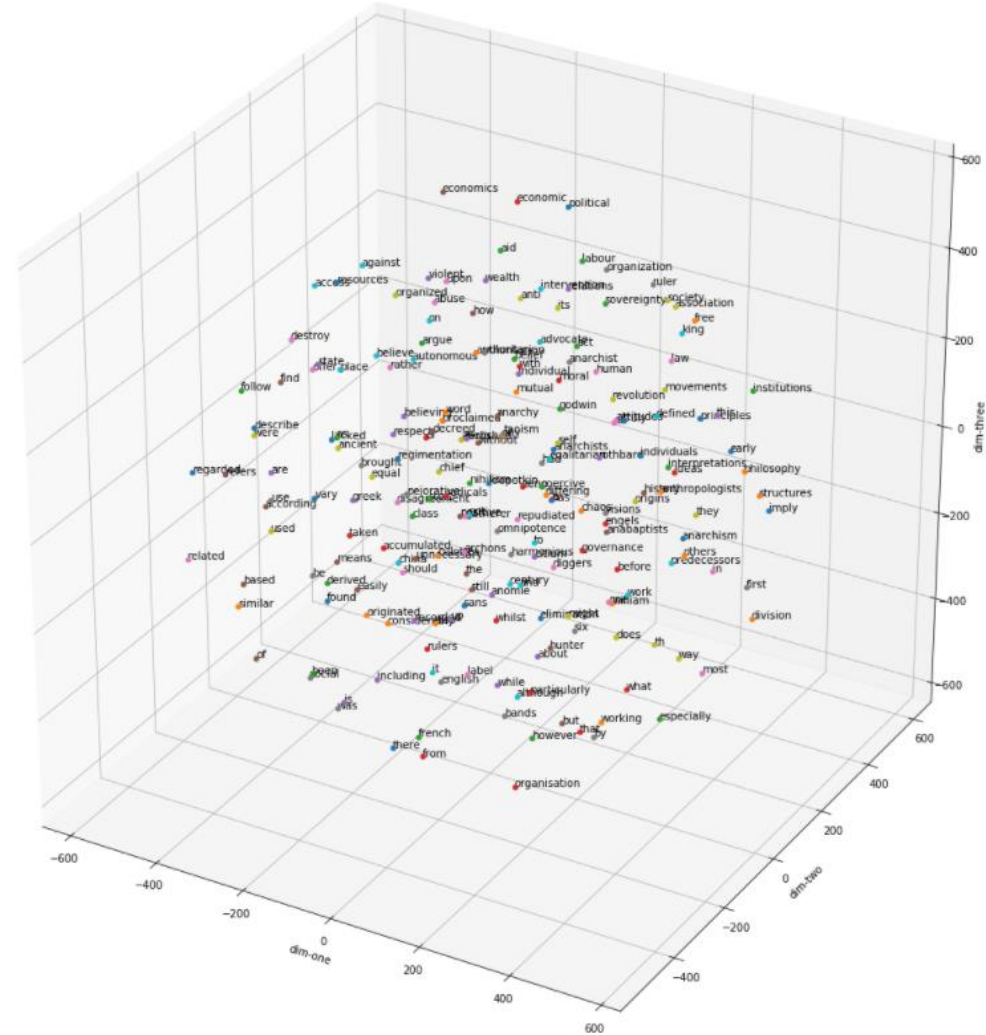


(Source: <http://jalammar.github.io/>)

# Visualization of Word2Vec representations



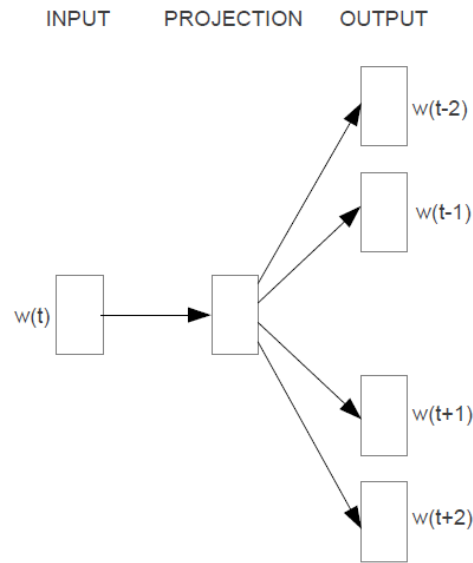
## 2D t-SNE plot



### 3D t-SNE plot

# Word2Vec: Summary

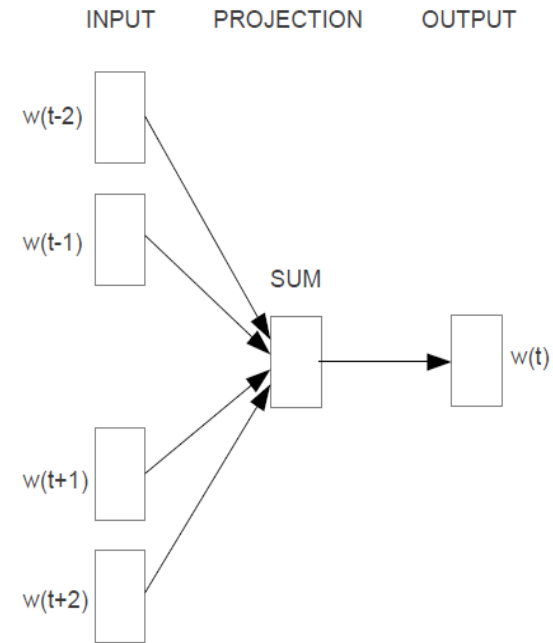
- **Skip-gram**: predict the **context words** based on the **target word**.
- **Continuous-bag-of-words (CBOW)**: predict the **target word** based on the **context words**.



**Skip-gram**

- **Negative sampling**: speed up training by sub-sampling (e.g., frequent words)
- **Hierarchical softmax**: deal with large vocabulary:  $O(V)$  to  $O(\log V)$ .

<https://medium.com/@abhishekjainindore24/hierarchical-softmax-for-word-embeddings-47e1ca398ed6>



**CBOW**



Word2Vec in use

# Train Word2Vec on a dataset

```
import gensim.downloader as api
from gensim.models import Word2Vec
```

```
dataset = api.load("text8")
model = Word2Vec(dataset)
model.save("./text8-word2vec.bin")
```

**Train Word2Vec on text8 dataset (skip-gram, window size =5)**

```
from gensim.models import KeyedVectors
model = KeyedVectors.load("./text8-word2vec.bin")
word_vectors = model.wv
```

**Load pretrained Word2Vec from a file**

```
word_vectors.get_vector('king')
```

```
array([ 1.0615952 ,  3.374791 , -2.4529877 , -0.89189297,  2.0385118 ,
        -1.0867552 , -1.2593621 ,  0.20547654, -0.70700854,  0.29547456,
        -0.9151951 ,  0.99069464,  1.7152157 ,  0.64989454,  0.33458185,
         2.499265 , -1.0269971 ,  4.957024 , -6.161608 , -0.10745641,
         0.10324214,  1.1219409 ,  0.98975873, -0.08191033, -0.7929074 ,
        -0.28150806, -1.0557121 ,  0.27056807,  0.31582335,  2.9731138 ,
        -1.4136707 ,  0.93965536,  1.1514933 ,  0.38530475, -1.5722595 ,
        -0.08922919, -1.2710185 , -0.7054481 ,  0.7354161 ,  1.4659075 ,
         1.2870685 , -1.2874846 , -1.6638854 ,  0.20794497, -0.1928033 ,
        -3.8513193 , -0.0873706 ,  0.43098506,  0.12324328, -1.6535882 ,
        -0.6248446 , -0.28294212,  1.4047468 , -0.42495435,  0.7049425 ,
        -0.26330888, -1.7225645 , -0.866658 ,  1.3149631 , -0.5719914 ,
        -1.3960481 ,  1.7349594 ,  2.8976836 ,  2.233186 ,  0.905698 ,
         0.24419262,  1.7447696 ,  2.4310687 , -0.6564301 ,  2.1977458 ,
        -0.28740513, -0.0529648 ,  1.8151288 ,  1.2035793 ,  0.51843506,
         2.2382748 , -1.7706063 , -1.7169152 , -3.8160467 ,  0.2048373 ,
         1.1777579 ,  2.9256532 ,  0.7214914 , -3.804784 , -0.3797294 ,
        -1.3870562 , -1.8468527 ,  0.96608454, -0.51972026, -1.4571909 ,
        -2.1815338 , -1.7526524 , -2.4643364 , -0.5413108 , -0.6252542 ,
         0.33478758, -0.27308032, -2.7191868 , -2.4398658 , -1.7016346 ],
        dtype=float32)
```

**Get vector representation of a word**

```
word_vectors.cosine_similarities(word_vectors.get_vector('king'), [word_vectors.get_vector('queen'), word_vectors.get_vector('australia')])

array([0.7218381 , 0.09479931], dtype=float32)
```

**Compute cosine similarity**

# Advanced operations with Word2Vec

```
def print_most_similar(word_conf_pairs, k):
    for i, (word, conf) in enumerate(word_conf_pairs):
        print("{:.3f} {}".format(conf, word))
        if i >= k-1:
            break
    if k < len(word_conf_pairs):
        print("...")
```

**Print returned results in better form**

```
print_most_similar(word_vectors.most_similar(positive=["france", "berlin"], negative=["paris"]), 1)
```

0.802 germany

**france – paris + berlin = germany**

```
print(word_vectors.doesnt_match(["hindus", "parsis", "singapore", "christians"]))
```

singapore

**Not matched word**

```
print_most_similar(word_vectors.most_similar("king"), 10)
```

0.759 prince  
0.722 queen  
0.712 vii  
0.698 emperor  
0.682 kings  
0.669 elector  
0.668 regent  
0.666 constantine  
0.663 throne  
0.663 pope

**Top 10 similar words of king**

```
print_most_similar(word_vectors.most_similar("china"), 10)
```

0.795 japan  
0.760 taiwan  
0.746 india  
0.667 thailand  
0.661 indonesia  
0.651 pakistan  
0.647 tibet  
0.645 afghanistan  
0.643 burma  
0.639 kazakhstan

**Top 10 similar words of China**

Something to Vector

# Recent methods on learning embedding

## Something to Vector

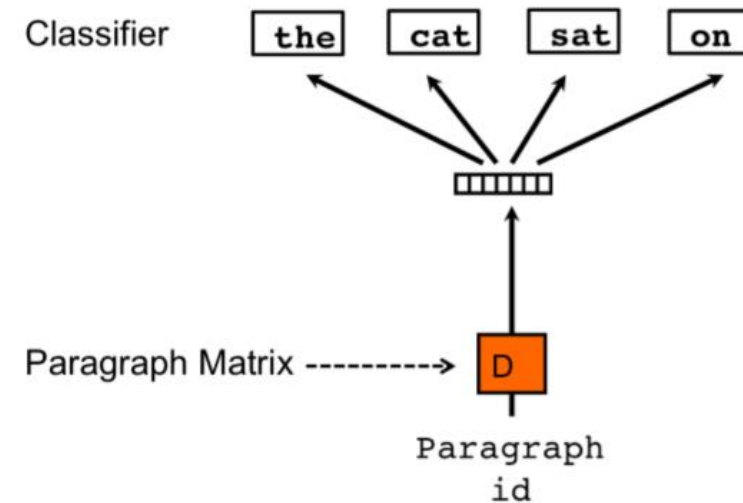
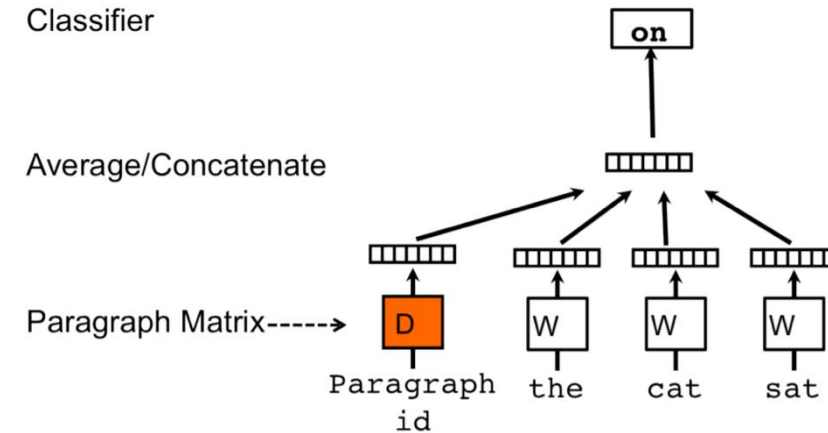
Recent Methods	
<b>word2vec</b> (Mikolov, et al. 2013)	distributed representation for <b>words</b>
<b>doc2vec</b> (Le an Mikolov, 2014)	distributed representation of <b>sentences</b> and <b>documents</b>
<b>topic2vec</b> (Niu and Dai, 2015)	distributed representation for <b>topics</b>
<b>item2vec</b> (Barkan and Koenigstein, 2016)	distributed representation of <b>items</b> in <b>recommender systems</b>
<b>med2vec</b> (Choi et al., 2016)	distributed representations of <b>ICD codes</b>
<b>node2vec</b> (Grover and Leskovec, 2016)	distributed representation for <b>nodes</b> in a network
<b>paper2vec</b> (Ganguly and Pudi, 2017)	distributed representations of textual and <b>graph-based information</b>
<b>sub2vec</b> (Adhikari et al., 2017)	distributed representation for <b>subgraphs</b>
<b>cat2vec</b> (Wen et al., 2017)	distributed representation for <b>categorical values</b>
<b>fis2vec</b> (2017)	distributed representation for <b>frequent itemsets</b>

# Documents to Vectors

## doc2vec (Le and Mikolov, 2014)

- Word2Vec gives vectors for **words**, but many NLP tasks (e.g., document classification, sentiment analysis, information retrieval) need vectors for **whole documents**.
- Doc2Vec maps each document  $D$  to a vector  $v_D$  in such a way that similar documents have similar vectors.

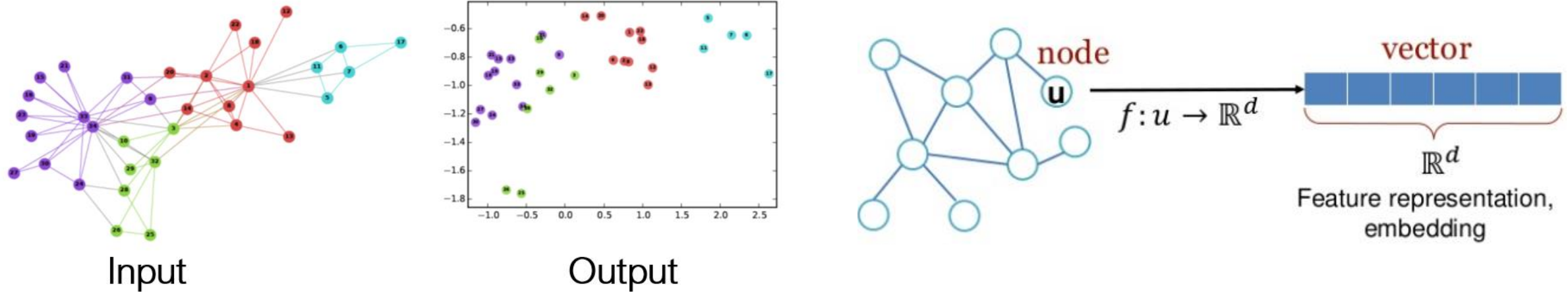
- $N$  documents (paragraphs) and  $M$  words
  - Embedding matrix for documents (paragraphs):  $D \in \mathbb{R}^{N \times p}$
  - Embedding matrix for words:  $W \in \mathbb{R}^{M \times q}$
- Embedding paragraph id and word ids to paragraph vector ( $\mathbb{R}^{1 \times p}$ ) and word vector ( $\mathbb{R}^{1 \times q}$ )
  - Take average and concatenate
- Given a document (paragraph), the task is to predict the target word from the context words or vice versa.



# Nodes to Vectors

node2vec (Grover and Leskovec, 2016)

- In graphs (social networks, citation networks, knowledge graphs, etc.), each node has connections (edges) to other nodes.
- Node2Vec converts each node into a **vector** in  $\mathbb{R}^d$  that captures **structural** and **neighborhood** information.



(Source: [snap.stanford.edu/proj/embeddings-www](http://snap.stanford.edu/proj/embeddings-www), WWW 2018)

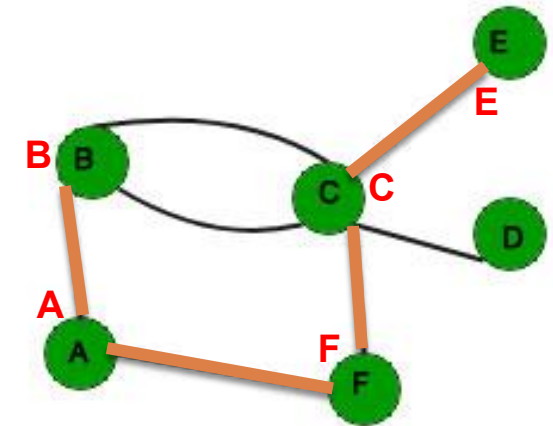
## □ Motivation:

- Find embedding of nodes to  $d$ -dimensions so that “similar” nodes in the graph have embeddings that are close together.

# Node2Vec

## Idea

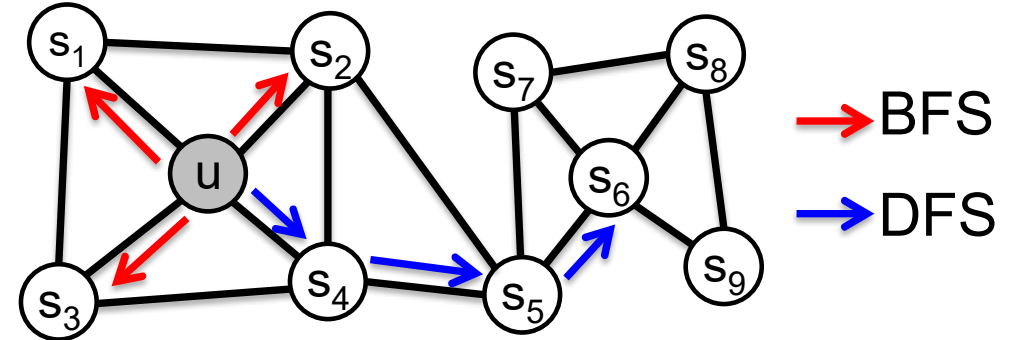
- Do **random walks** on the graph
  - B, A, F, C, E  $\rightarrow$  each node is a word  $\rightarrow F \xrightarrow{\text{predict}} B, A, C, E$
- How to find **good random walks**?
  - Cover all important paths in a graph
  - Balance between microscopic and macroscopic views



# Node2Vec

## DFS vs BFS

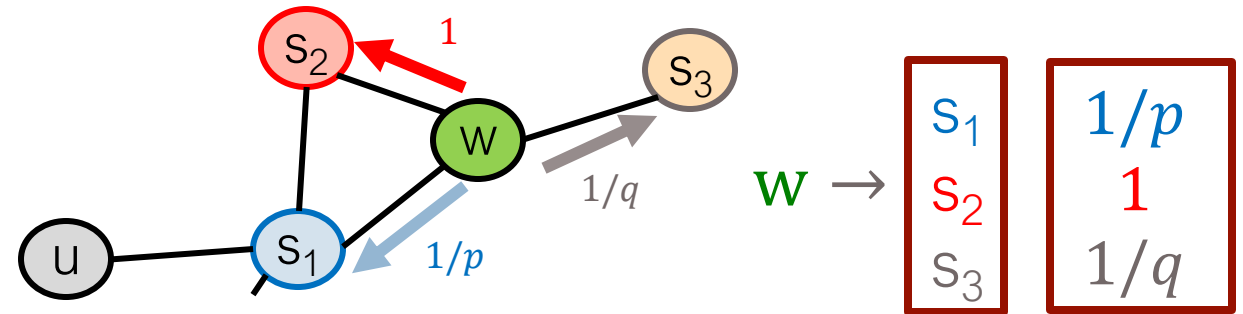
- $N_R(u)$  represents the neighborhood of a node  $u$ .
- $N_{BFS}(u) = \{s_1, s_2, s_3\}$ 
  - Local microscopic view
- $N_{DFS}(u) = \{s_4, s_5, s_6\}$ 
  - Global macroscopic view



# Node2Vec

## Random walks

- Walker is at  $w$ . Where to go next?
- $p, q$  model transition probabilities
  - $p$  ... return parameter
  - $q$  ... "walk away" parameter
- **BFS-like** walk: Low value of  $p$
- **DFS-like** walk: Low value of  $q$



(Source: [snap.stanford.edu/proj/embeddings-www](http://snap.stanford.edu/proj/embeddings-www), WWW 2018)

# Reading and references

- **Word embedding paper:** <https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>
- **Good document to explain Word2Vec:** <https://arxiv.org/pdf/1411.2738.pdf>
- **Doc2Vec paper:** [https://cs.stanford.edu/~quocle/paragraph\\_vector.pdf](https://cs.stanford.edu/~quocle/paragraph_vector.pdf)
- **Node2Vec paper:** <https://arxiv.org/pdf/1607.00653.pdf>
- **Good blog for self-supervised learning:** <https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html>

# Summary

- Text Analytics and Language Models
- Learning representation in machine learning and deep learning
- Word embedding
  - Skip-gram
  - Continuous bag of words (CBOW)
  - Negative sampling
- Something to vector
  - Doc2Vec
  - Node2Vec

Thanks for your attention!

Question time

