FIT3181 Deep Learning

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

Lecturer: Lim Chern Hong

Email: lim.chernhong@monash.edu

GROUP OF EIGHT AUSTRALIA

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

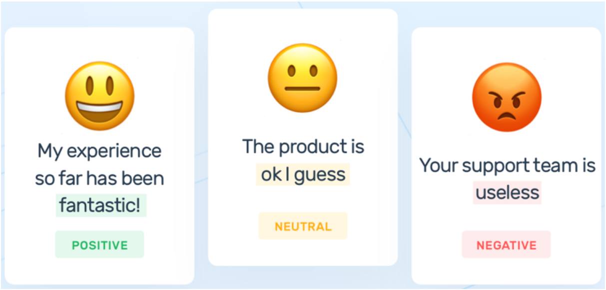
Express

Text Data

Text corpus

Movie reviews

Service reviews on FB



Sentiment analysis

Symbolic representation

Numeric representation

ML algorithm

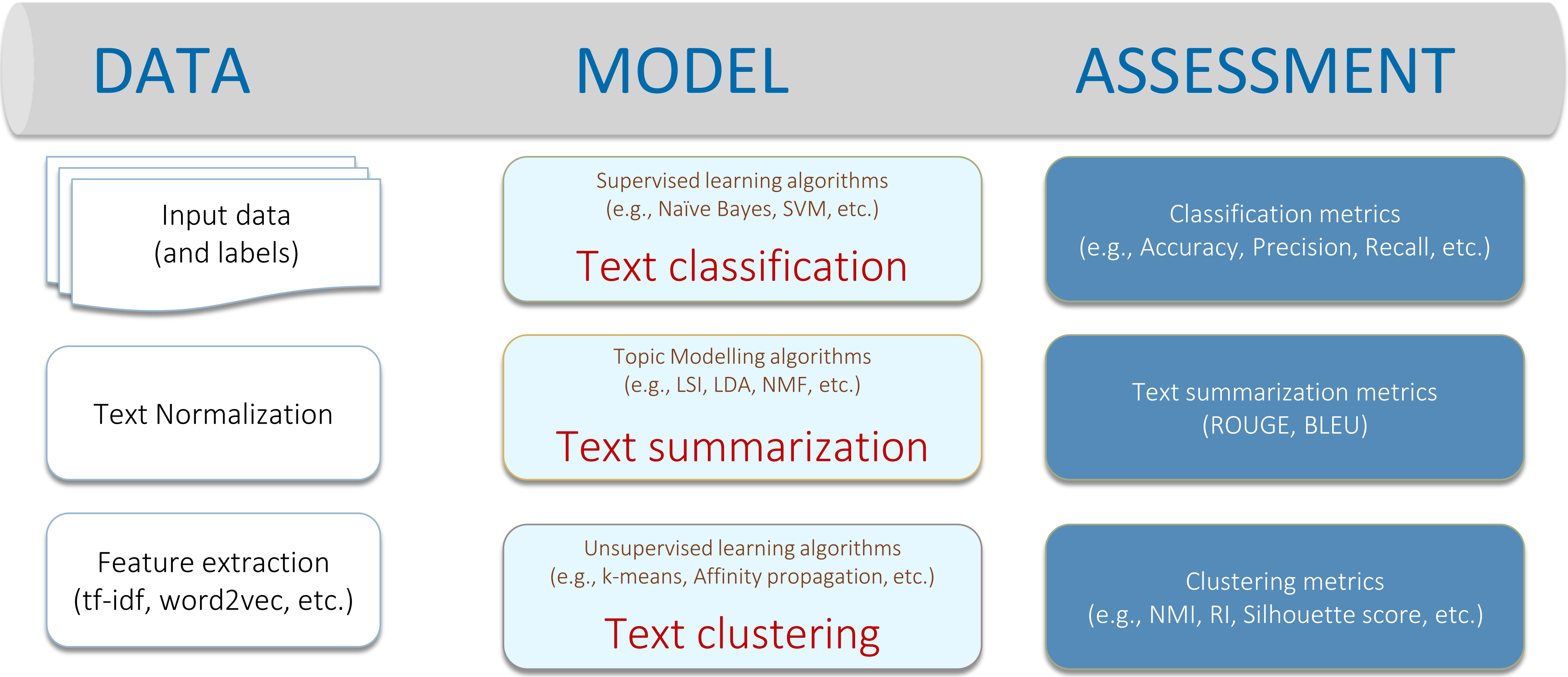
Text Analytics and Language Models

- 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

- One-hot vector encoding

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

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

Text Analytics and Language Models

- One-hot vector encoding

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

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

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

document

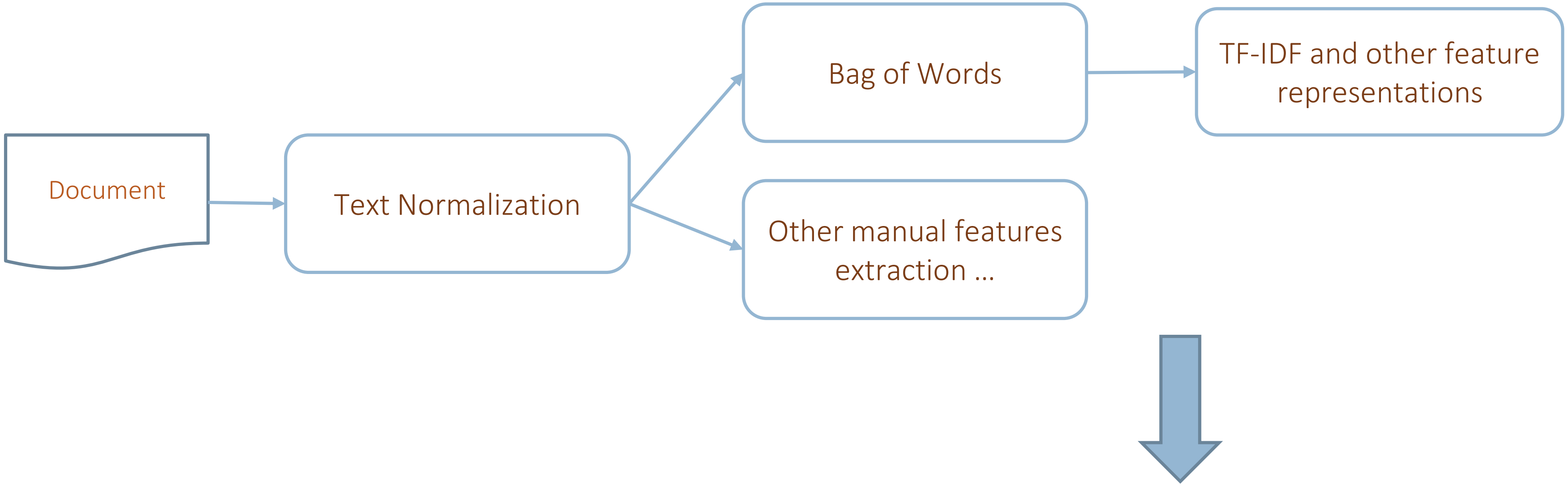
term

$$A = \left[\begin{array}{ccc} 1 & 0 & \cdots \\ 1 & 2 & \cdots \\ 2 & 2 & \cdots \\ 1 & 0 & \cdots \\ 0 & 1 & \cdots \end{array}\right] \begin{array}{l} \text{goal} \\ \text{data} \\ \text{information} \\ \\ \text{you} \end{array}$$

term-by-document matrix

Text Analytics and Language Models

- Feature extraction



Each word and document become a point in a vector space!

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

TF-IDF example

term frequency (tf)

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

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

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

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

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

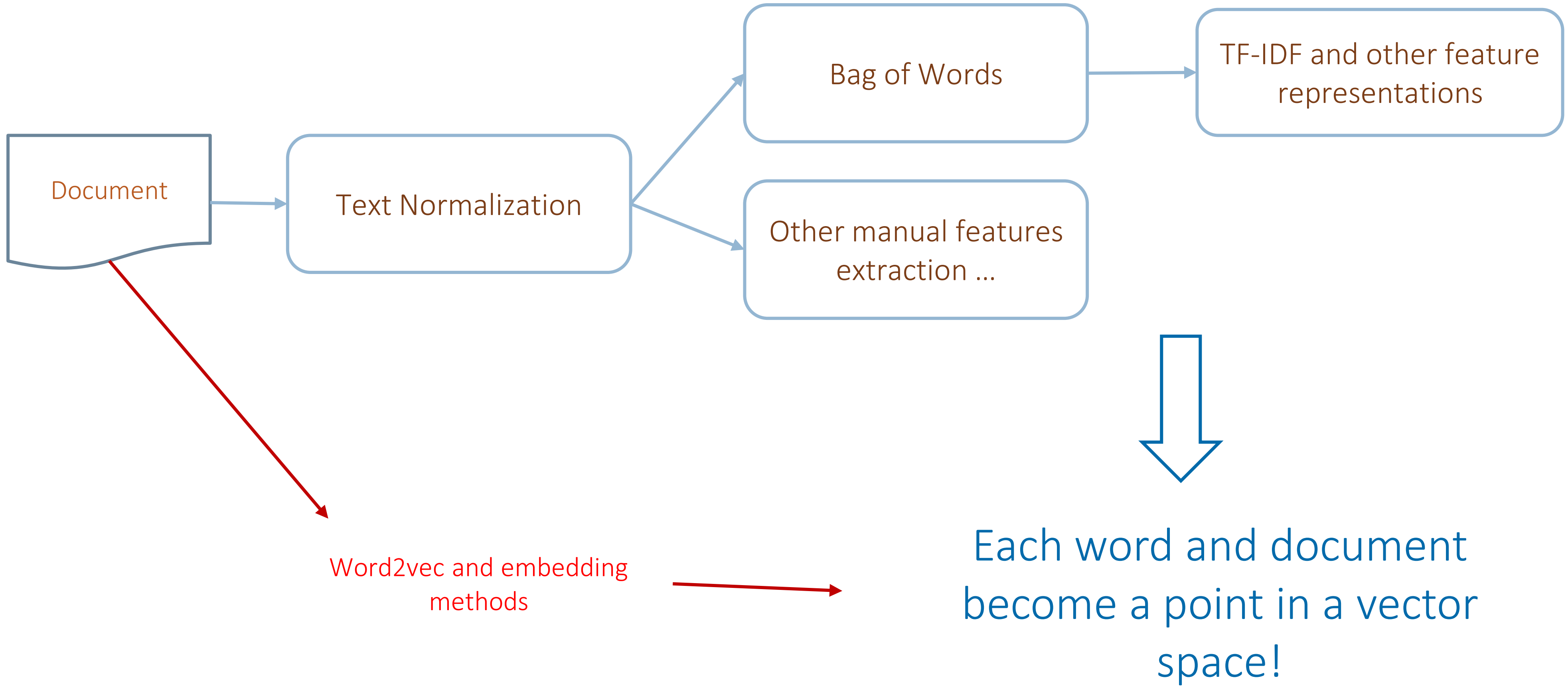
$$\frac{0.69}{\sqrt{0.69^2 + 0.69^2}}$$

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)

Input

Feature extraction (SIFT SURF, HOG)
Hand-crafted features

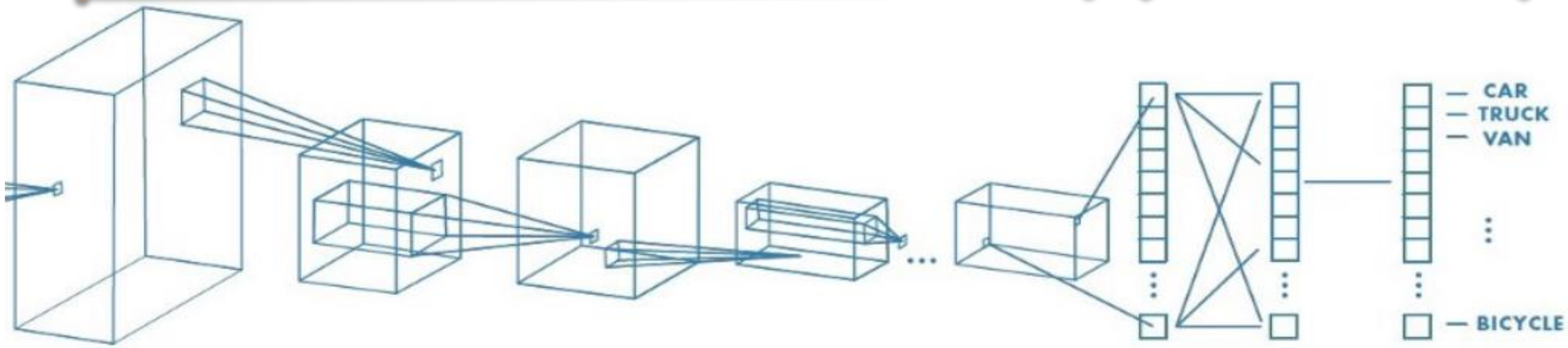
Learning algorithms (SVM, XGBoost)

- **Deep learning approach** (automatic feature learning)

Input

Feature extraction
Automatic features

Classification



Hand-crafted and automatic feature extractor

Visual data

- **Traditional approach** (hand-crafted feature learning)

Learning algorithm and feature extractor are disconnected !

Input

Feature extraction (SIFT SURF, HOG)
Hand-crafted features

Learning algorithms (SVM, XGBoost)

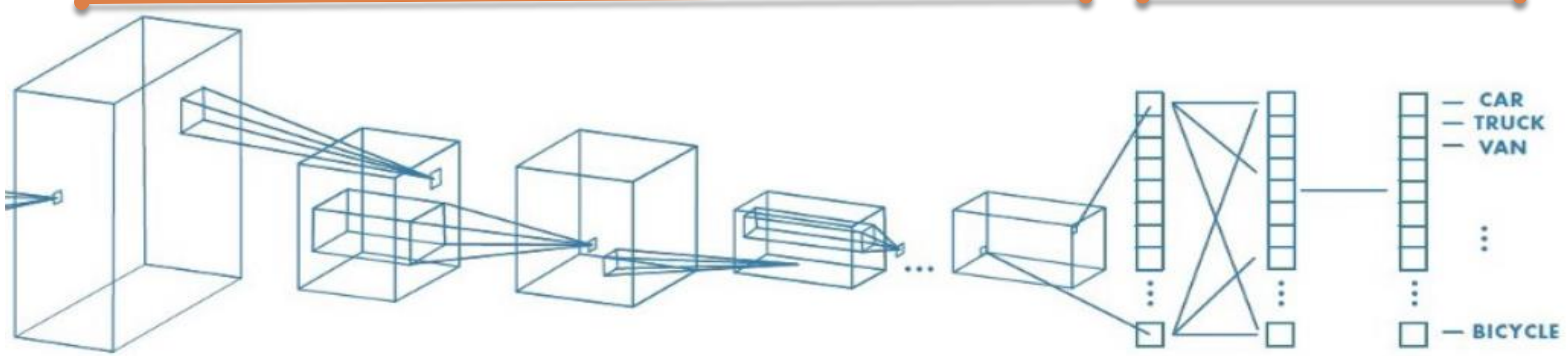
- **Deep learning approach** (automatic feature learning)

Classification and feature extractor are connected !

Input

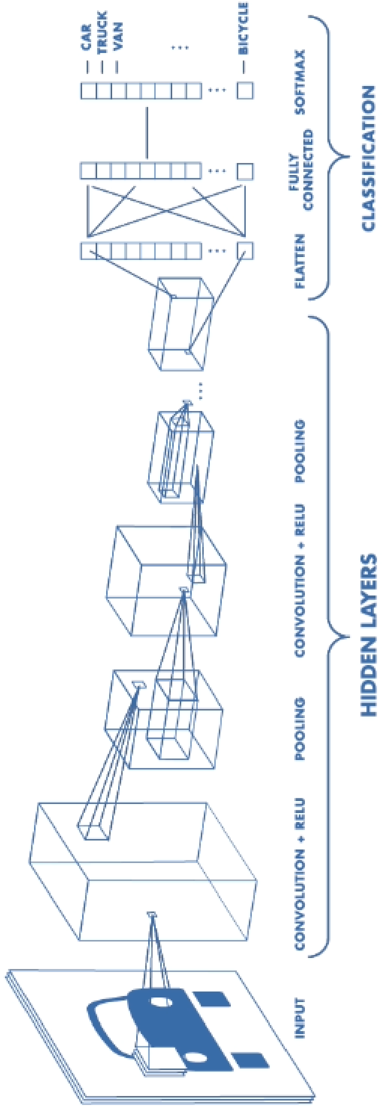
Feature extraction
Automatic features

Classification



Automatic feature extractor

Deep learning for visual data

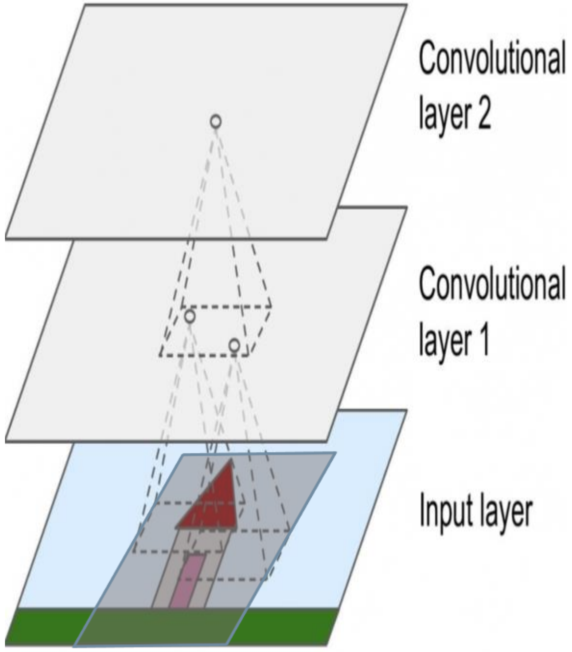


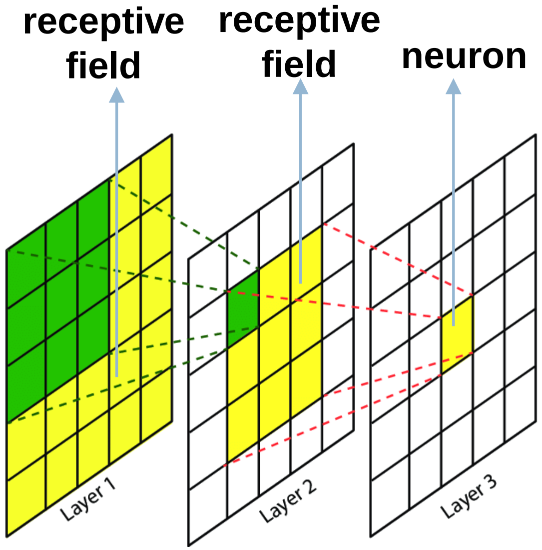
High-level feature/representation
- objects

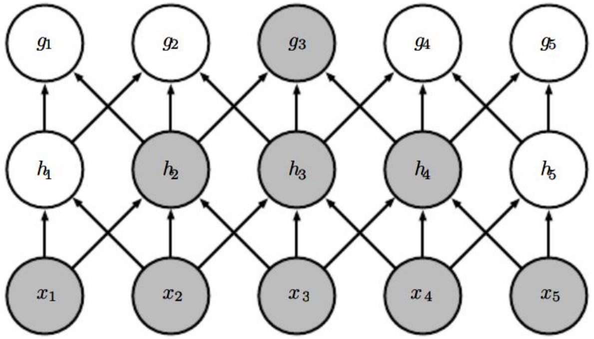
Mid-level feature/representation
- circles, triangles, boxes

Low-level feature/representation
- edges, pixels, corners

Neurons on **higher layers** have **larger receptive fields** (patches) on **input images**







Hand-crafted and automatic feature extractor

Sequential data

- **Traditional approach** (hand-crafted feature learning)

Input

I love deep learning

Feature extraction (TF-IDF)
Hand-crafted features

Learning algorithms (SVM, XGBoost)

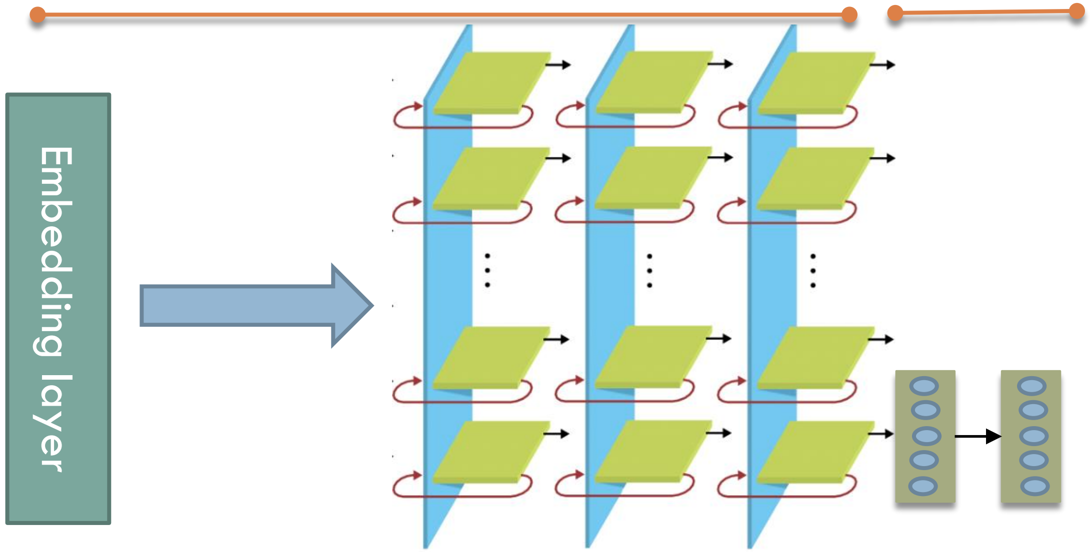
- **Deep learning approach** (automatic feature learning)

Input

I love deep learning

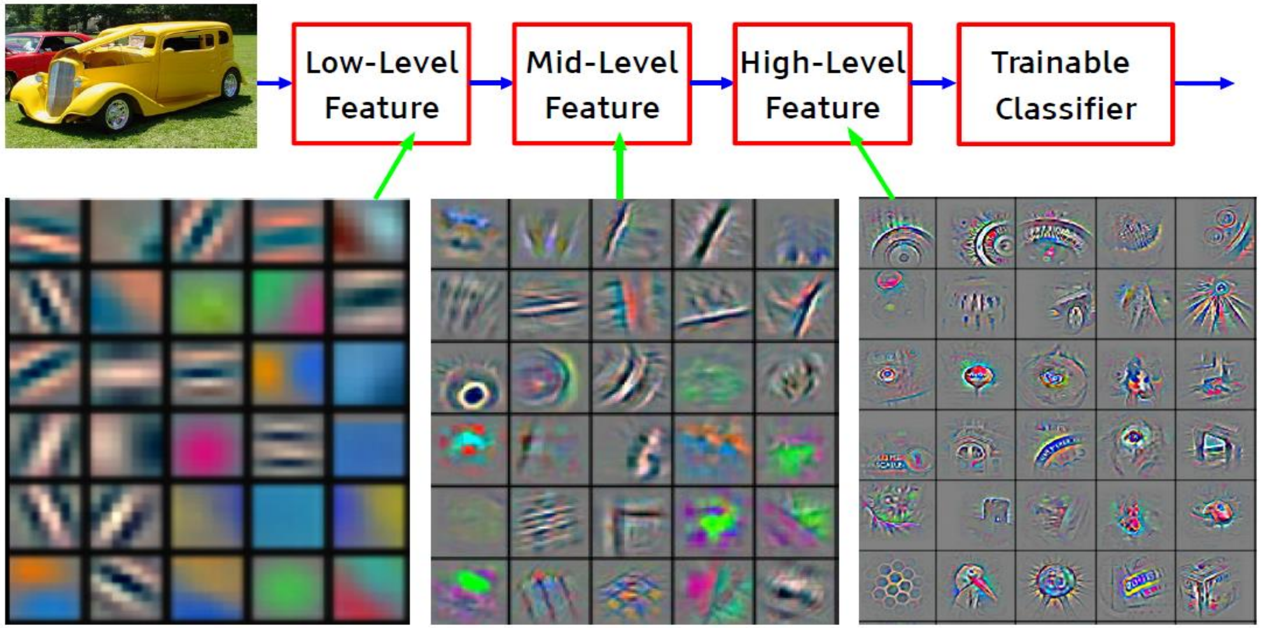
Feature extraction
Automatic features

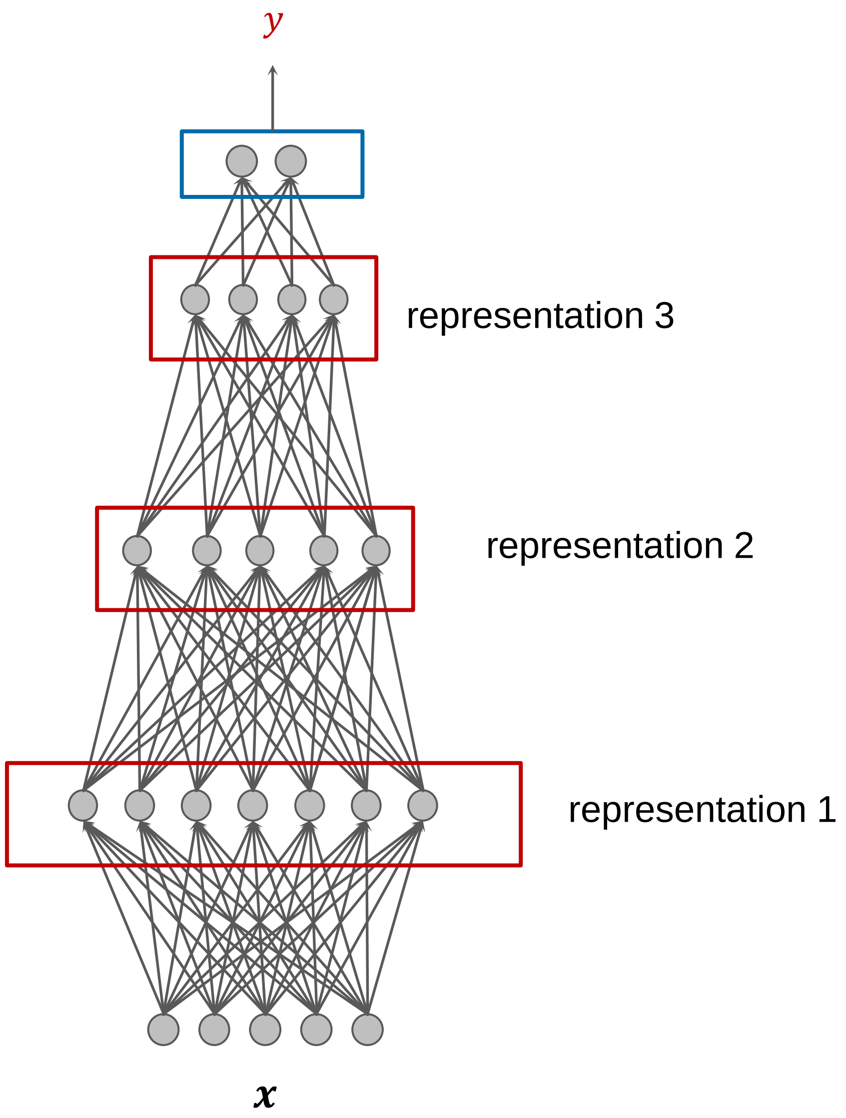
Classification



Learning representation in deep learning

"Deep Learning: machine learning algorithms based on learning multiple levels of representation and abstraction"
Yoshua Bengio





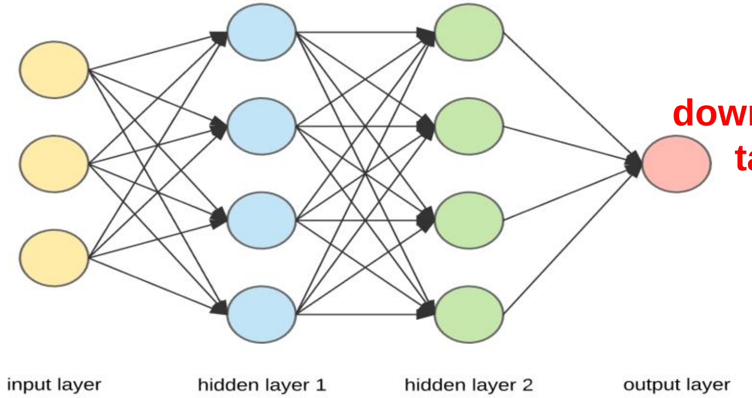
Learning representation in Deep Learning

Supervised representation learning

- **Supervised representation learning**

Raw data with labels for downstream task

representation



downstream tasks

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

- Learning the representation that fits a specific downstream task.

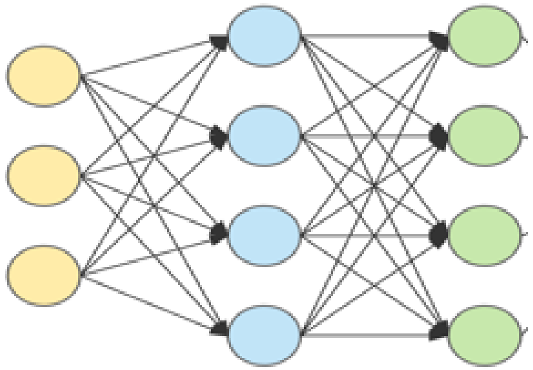
Learning representation in Deep Learning

Unsupervised representation learning

- **Unsupervised representation learning**

Raw data only

representation



general task

Unseen task 1

Unseen task 2

Unseen task 3

….

….

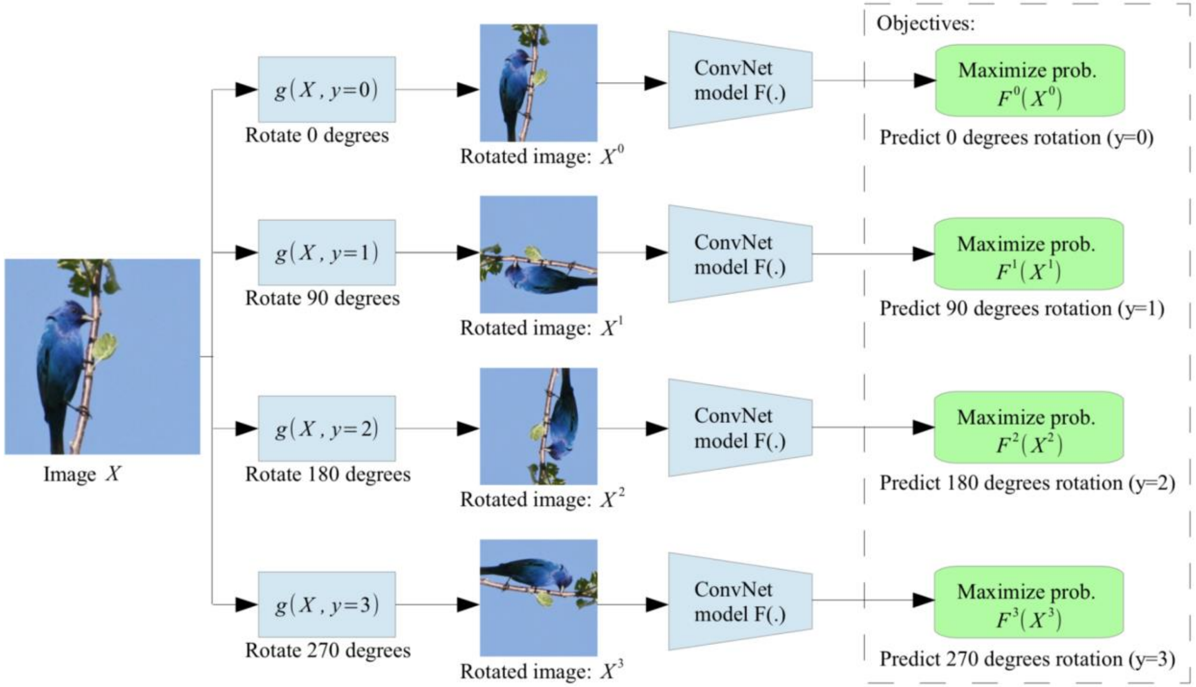
….

- Learning the representation that fits many tasks.

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

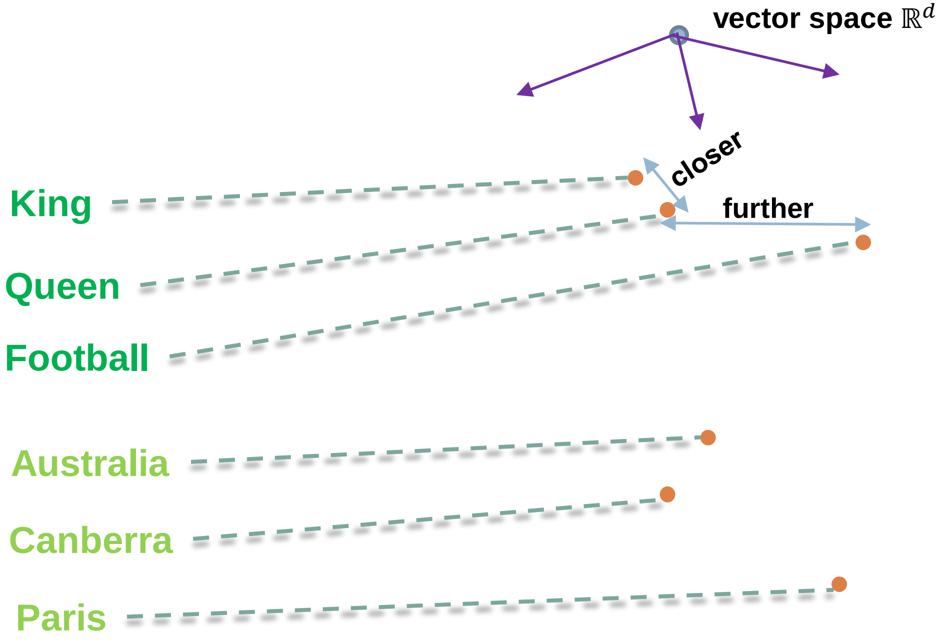
The **British royal family** comprises Queen Elizabeth II and her close relations. There is no strict legal or formal definition of who is or is not a member of the British royal family.

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.[1]

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.[2]

We desire…



Canberra : Australia = Paris : ???

$$argmin_v \|v_{Canberra} - v_{Australia} + v_{Paris} - v\| = v_{France}$$

- **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.


Q&A
When? Why? How? WHERE? WHEN? What? How? Why? WHAT? Why? When?
WHAT? HOW? WHEN? When? WHERE?
When? WHERE? WHAT? Why? When? WHAT?
WHO? WHERE? WHEN? What? HOW?
Why? HOW? Where? Where? WHERE? When? What? What! WHERE? WHERE? When? When?
Why? WHEN? WHEN? HOW? When? When? HOW? WHAT? Why?

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.	**Original sentence**

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 $\xrightarrow{predict}$ context words**

Continuous Bag of Words (CBOW)

- **Contexts words $\xrightarrow{predict}$ 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** $\xrightarrow{predict}$ **context words**

(brown, the), (brown, quick), (brown, for), (brown, jumps)

(fox, quick), (fox, brown), (fox, jumps), (fox, overs)

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

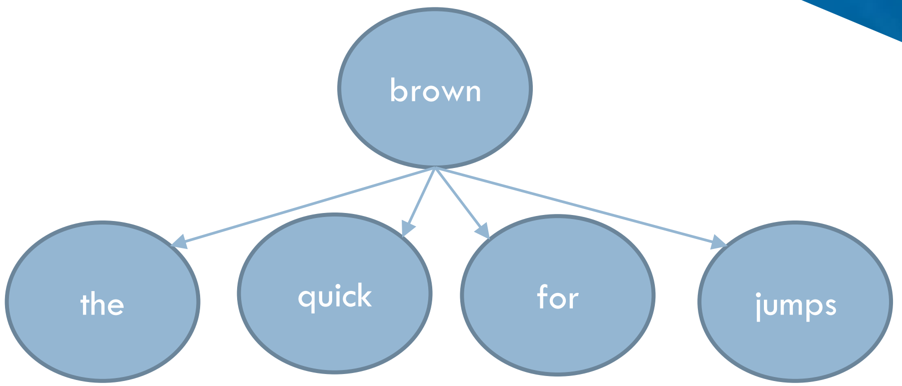
(over, fox), (over, jumps), (over, the), (over, lazy)

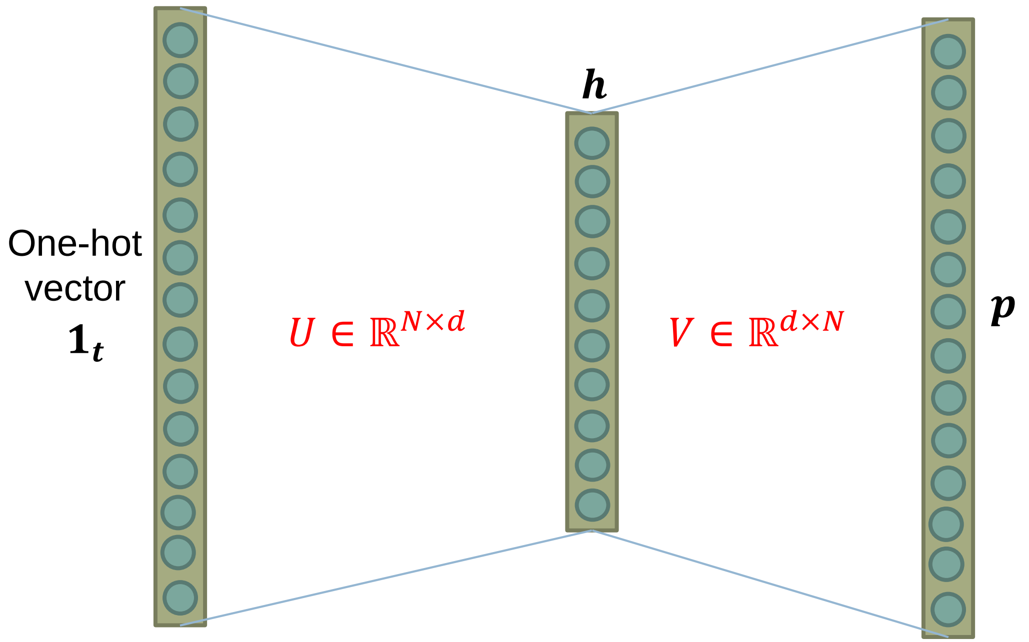
(the, jumps), (the, over), (the, lazy), (the, dogs)

Skip-gram

Modelling

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

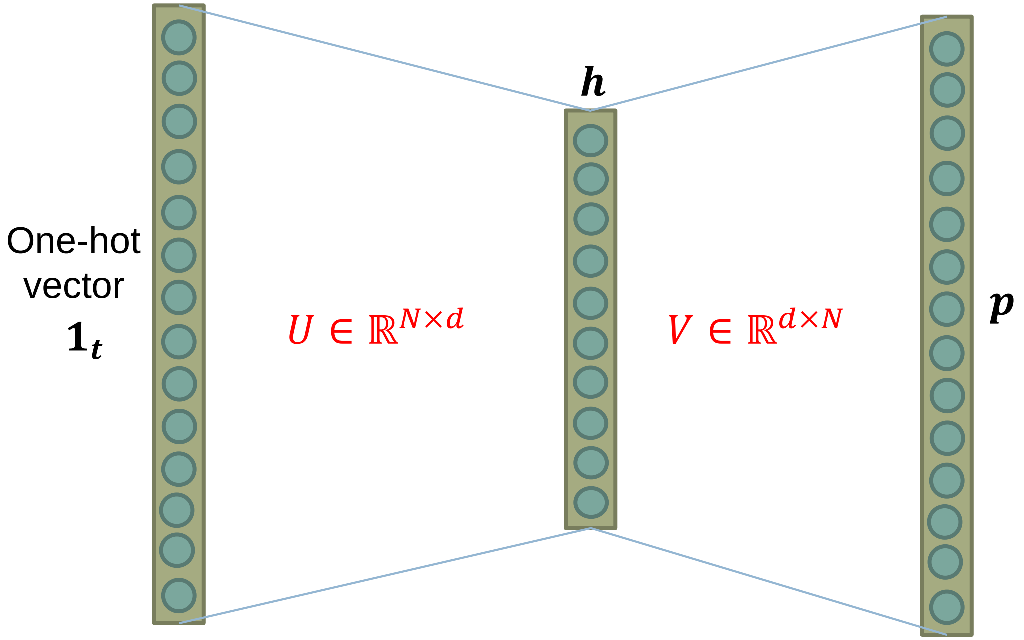




Skip-gram

Drawback

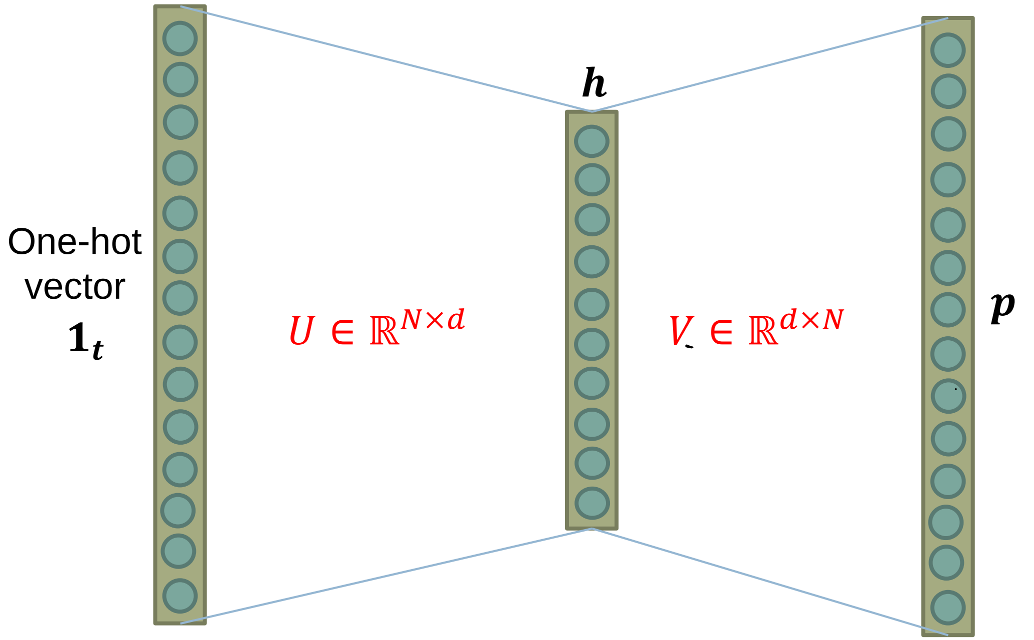
- **Two matrices**
  - $\boldsymbol{U} \in \mathbb{R}^{\boldsymbol{N} \times \boldsymbol{d}}$ (N is vocabulary size, d is embedding size)
  - $\boldsymbol{V} \in \mathbb{R}^{\boldsymbol{d} \times \boldsymbol{N}}$
- Assume that indices of tw and cw are $1 \leq t, c \leq N$ respectively. The forward propagation is as follows:
  - $h = 1_t^T U = U_t^r \in \mathbb{R}^{1 \times d}, o = hV \in \mathbb{R}^{1 \times N}, p = softmax(o)$
  - $P(\,cw = the \mid tw = brown\,) = p_c$
  - $\log P(\,cw = the \mid tw = brown\,) = \log p_c = U_t^r V_c^c - \log(\sum_{k=1}^{N} \exp(U_t^r V_k^c))$
- **Some drawbacks**
  - $p = softmax(o)$ is **computationally expensive**
  - $p \in \mathbb{R}^{1 \times N}$ is a distribution over the **vocabulary with size $\boldsymbol{N}$** (usually very big) → the values of p_i are **very tiny** → **hard to train**
    - Hierarchical SoftMax
    - Negative sampling (more popular and efficient)



Skip-gram

Negative sampling

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

(the | quick | for | jumps, brown)

(quick | brown | jumps | over, fox)

(brown | fox | over | the, jumps)

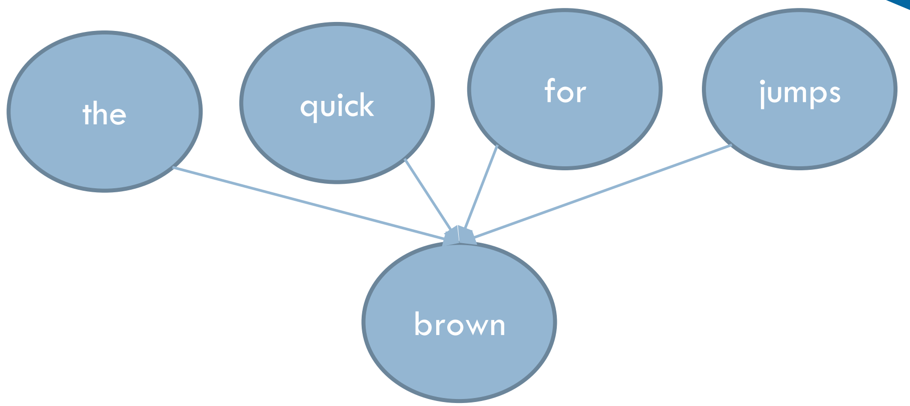
(for | jumps | the | lazy, over)

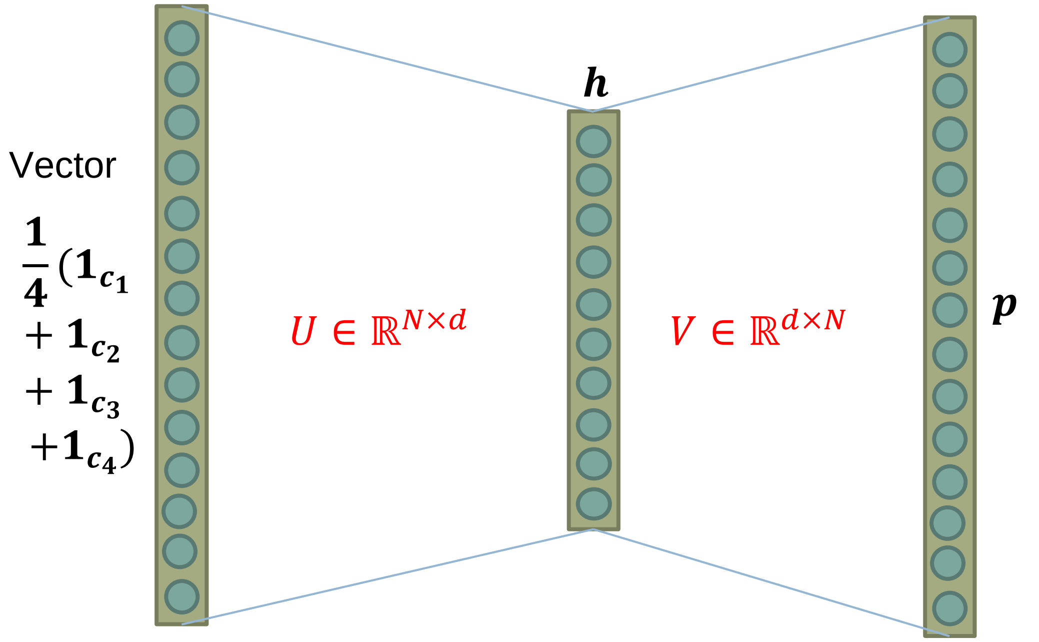
(jumps | over | lazy| dog, the)

Continuous Bag of Words (CBOW)

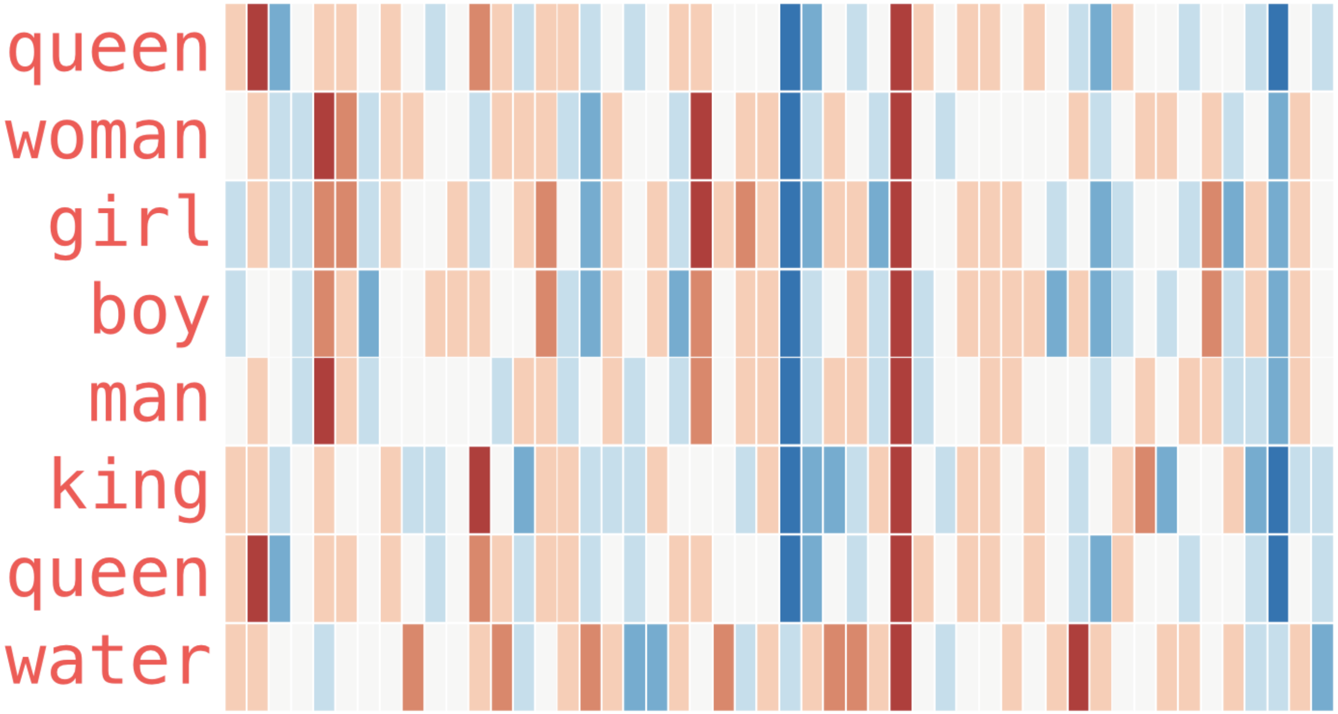
Modelling

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



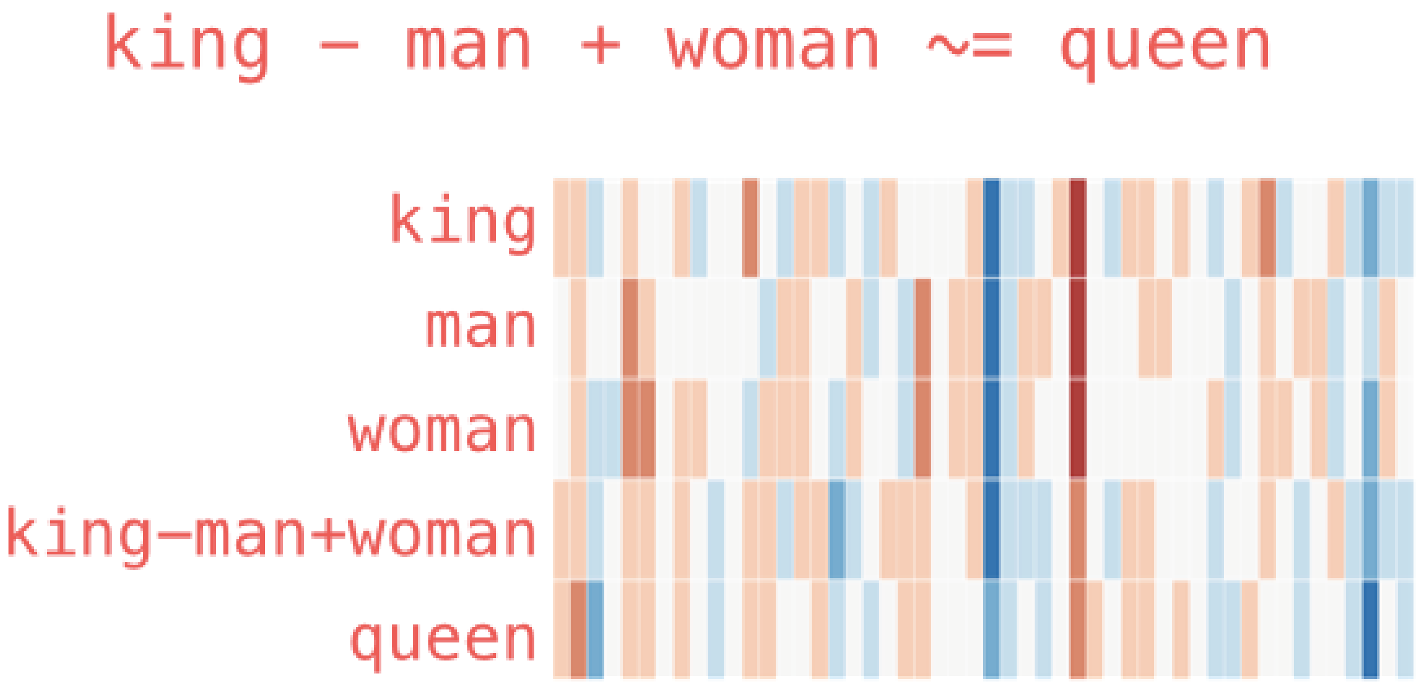


Visualization of Word2Vec representations



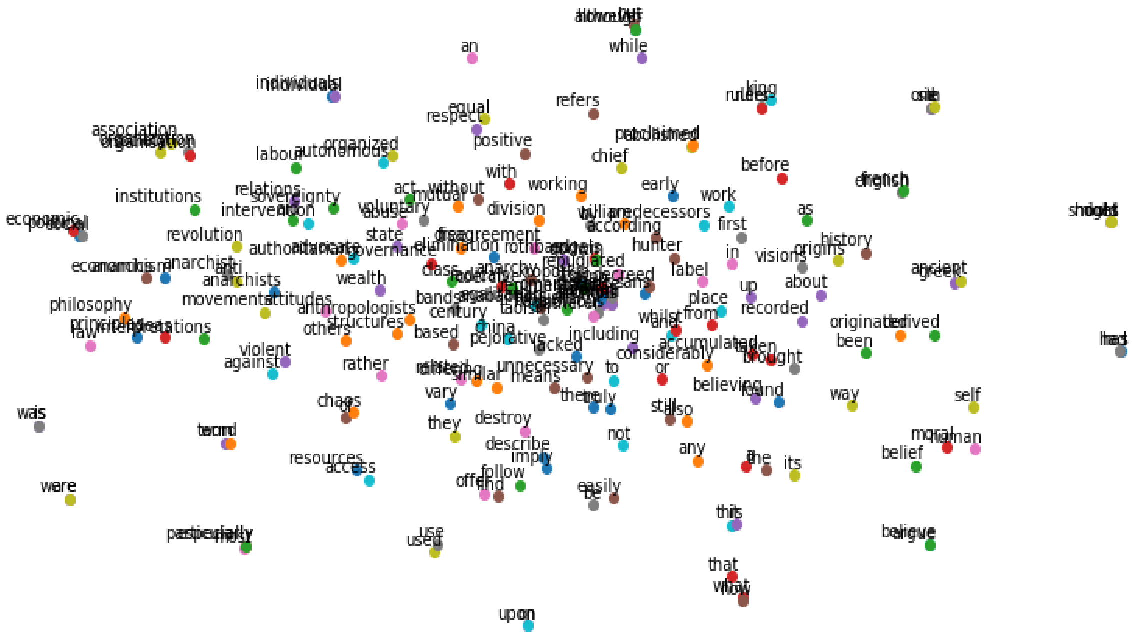
(Source: http://jalammar.github.io/)

Visualization of Word2Vec representations

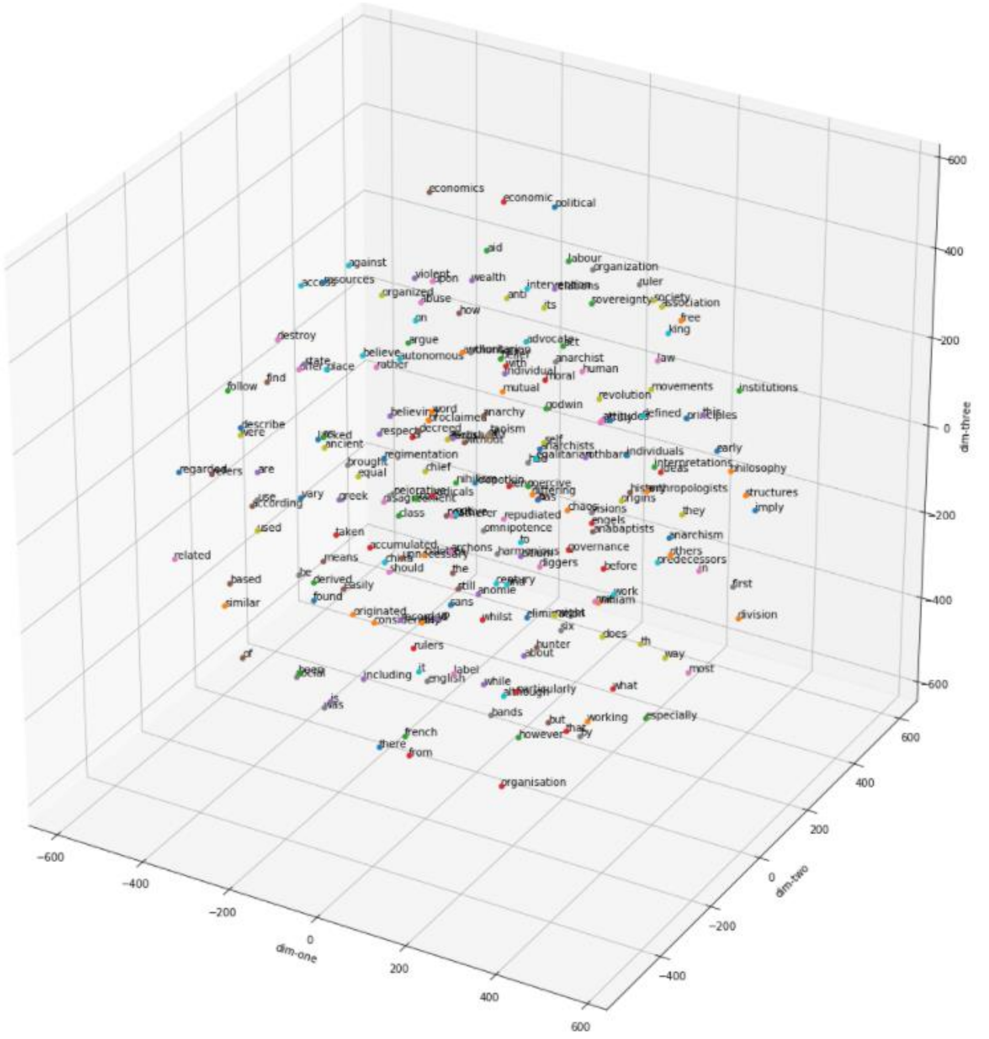


(Source: http://jalammar.github.io/)

Visualization of Word2Vec representations



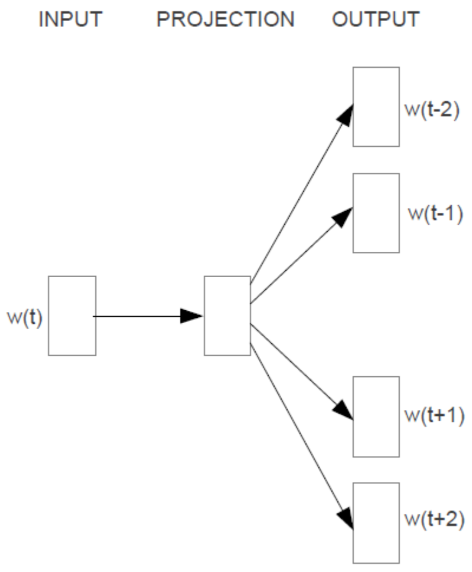
2D t-SNE plot



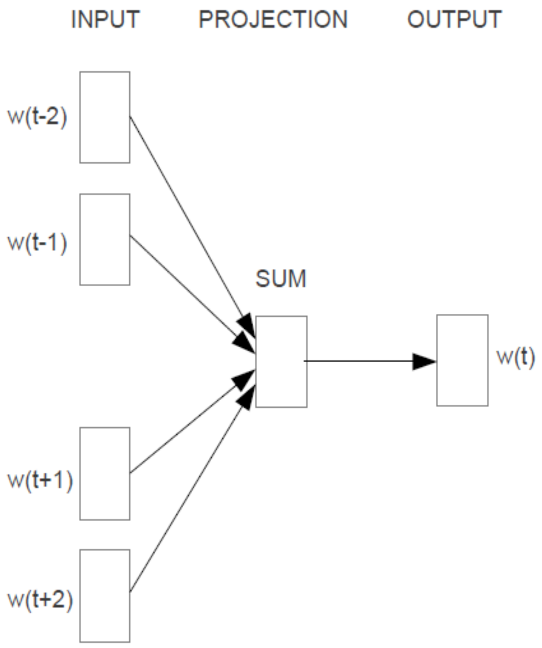
3D t-SNE plot

Word2Vec: Summary

- Skip-gram: predict the **context words** based on the **target word**.
- Continuous-bag-of-word (CBOW): predict the **target word** based on the **context words**.
- Negative sampling: speed up training by sub-sampling (e.g., frequent words)
- Hierarchical softmax: deal with large vocabulary: $O(V)$ to $O(\log V)$.



Skip-gram



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} {:s}".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

Word2Vec for initializing embedding matrix

```python
class RNN_Spam_Detection:
    def __init__(self, run_mode="scratch", embed_model= "glove-wiki-gigaword-300", embed_size = 128, data_manager= None):
        self.embed_model = embed_model
        self.embed_size = embed_size
        if run_mode != 'scratch':
            self.embed_size = int(self.embed_model.split("-")[-1])
        self.data_manager = data_manager
        self.vocab_size = self.data_manager.vocab_size +1
        self.word2idx = self.data_manager.word2idx
        self.embed_matrix = np.zeros((self.vocab_size, self.embed_size))
        self.run_mode = run_mode
        self.model = None

    def build_embedding_matrix(self):
        if os.path.exists("E.npy"):  #file existed
            self.embed_matrix = np.load("E.npy")           #Load the file for embedding matrix if existed
        else: #file not existed or first-time run
            self.word2vect = api.load(self.embed_model)   #Load embedding model
            for word, idx in self.word2idx.items():
                try:
                    self.embed_matrix[idx] = self.word2vect.word_vec(word)    #assign weight for the corresponding word and index
                except KeyError: #word cannot be found
                    pass
            np.save("E.npy", self.embed_matrix)

    def build(self):
        inputs = tf.keras.layers.Input(shape=[None])
        if self.run_mode == "scratch":
            self.embedding_layer = tf.keras.layers.Embedding(self.vocab_size, self.embed_size, mask_zero= True, trainable= True)
        else: #fine-tuned
            self.build_embedding_matrix()
            self.embedding_layer = tf.keras.layers.Embedding(self.vocab_size, self.embed_size,
                                                              weights= [self.embed_matrix], trainable= True)
        h = self.embedding_layer(inputs)
        h = tf.keras.layers.GRU(256, return_sequences=True)(h)
        h = tf.keras.layers.GRU(128)(h)
        h = tf.keras.layers.Dense(1, activation="sigmoid")(h)
        self.model = tf.keras.Model(inputs= inputs, outputs=h)
```

Word2Vec for initializing embedding matrix

```
rnn1.build()
```

```
rnn1.compile_model(optimizer="sgd", loss="binary_crossentropy", metrics=["accuracy"])
```

```
dm.train_valid_test_split()
```

```
rnn1.model.fit(dm.train_set.batch(64), epochs=5, validation_data= dm.valid_set.batch(64))
```

```
Epoch 1/5
70/70 [==============================] - 85s 1s/step - loss: 0.5273 - accuracy: 0.8607 - val_loss: 0.4353 - val_accuracy: 0.8698
Epoch 2/5
70/70 [==============================] - 83s 1s/step - loss: 0.4181 - accuracy: 0.8663 - val_loss: 0.3895 - val_accuracy: 0.8770
Epoch 3/5
70/70 [==============================] - 86s 1s/step - loss: 0.4048 - accuracy: 0.8659 - val_loss: 0.3884 - val_accuracy: 0.8743
Epoch 4/5
70/70 [==============================] - 87s 1s/step - loss: 0.4010 - accuracy: 0.8668 - val_loss: 0.3708 - val_accuracy: 0.8833
Epoch 5/5
70/70 [==============================] - 88s 1s/step - loss: 0.4048 - accuracy: 0.8643 - val_loss: 0.4040 - val_accuracy: 0.8645
<tensorflow.python.keras.callbacks.History at 0x176b1634548>
```

```
rnn1.evaluate(dm.test_set.batch(64))
```

```
9/9 [==============================] - 2s 256ms/step - loss: 0.3849 - accuracy: 0.8746
```

Training embedding matrix from scratch

```
rnn2 = RNN_Spam_Detection(data_manager=dm, run_mode= "init-fine-tune")
```

```
rnn2.build()
```

```
rnn2.compile_model(optimizer="sgd", loss="binary_crossentropy", metrics=["accuracy"])
```

```
rnn2.model.fit(dm.train_set.batch(64), epochs=5, validation_data= dm.valid_set.batch(64))
```

```
Epoch 1/5
70/70 [==============================] - 76s 1s/step - loss: 0.5131 - accuracy: 0.8665 - val_loss: 0.4116 - val_accuracy: 0.8761
Epoch 2/5
70/70 [==============================] - 76s 1s/step - loss: 0.4061 - accuracy: 0.8656 - val_loss: 0.4022 - val_accuracy: 0.8627
Epoch 3/5
70/70 [==============================] - 77s 1s/step - loss: 0.3953 - accuracy: 0.8659 - val_loss: 0.3855 - val_accuracy: 0.8707
Epoch 4/5
70/70 [==============================] - 78s 1s/step - loss: 0.3977 - accuracy: 0.8641 - val_loss: 0.3951 - val_accuracy: 0.8654
Epoch 5/5
70/70 [==============================] - 79s 1s/step - loss: 0.3990 - accuracy: 0.8634 - val_loss: 0.3869 - val_accuracy: 0.8698
<tensorflow.python.keras.callbacks.History at 0x176c375c708>
```

```
rnn2.evaluate(dm.test_set.batch(64))
```

```
9/9 [==============================] - 2s 247ms/step - loss: 0.3583 - accuracy: 0.8853
```

Using Word2Vec to initialize embedding matrix and fine tune

Something to Vector

Recent methods on learning embedding

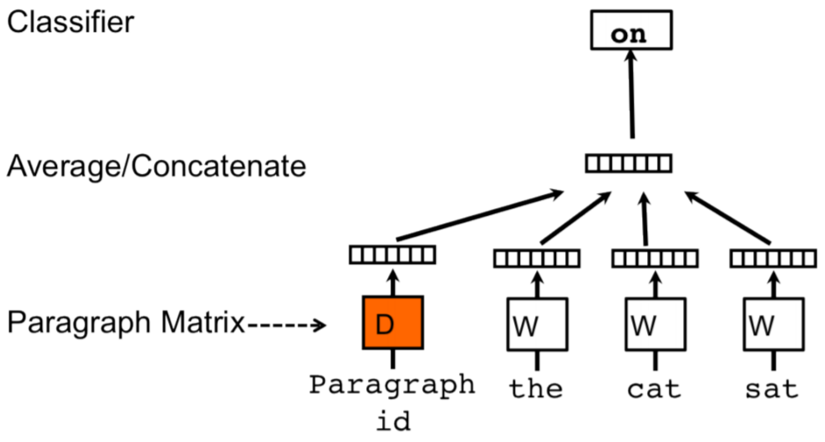
Something to Vector

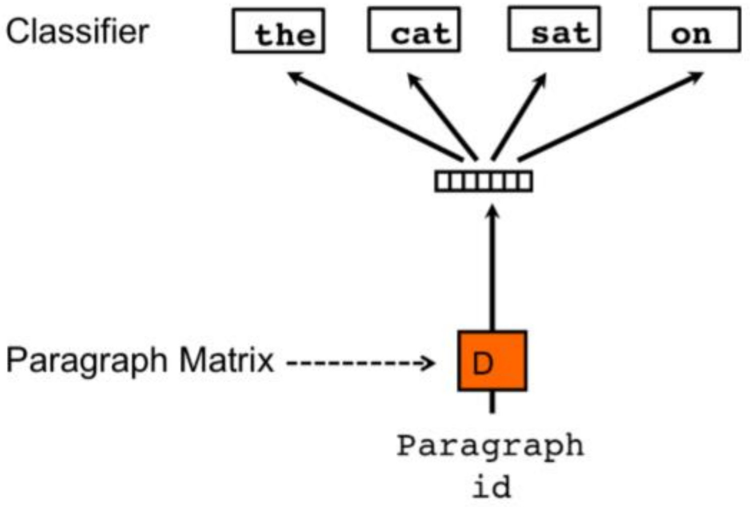
Recent Methods	
word2vec (Mikolov, et al. 2013)	distributed representation for words
doc2vec (Le an Mikolov, 2014)	distributed representation of sentences and documents
topic2vec (Niu and Dai, 2015)	distributed representation for topics
item2vec (Barkan and Koenigstein, 2016)	distributed representation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

Documents to Vectors

doc2vec (Le an Mikolov, 2014)

- 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.

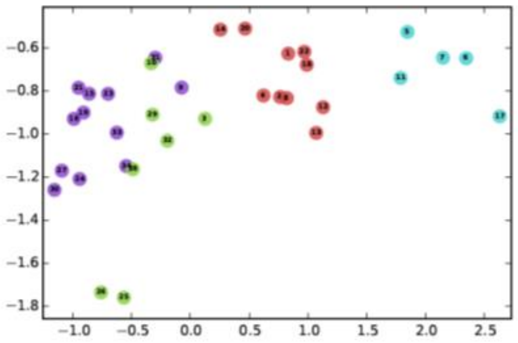




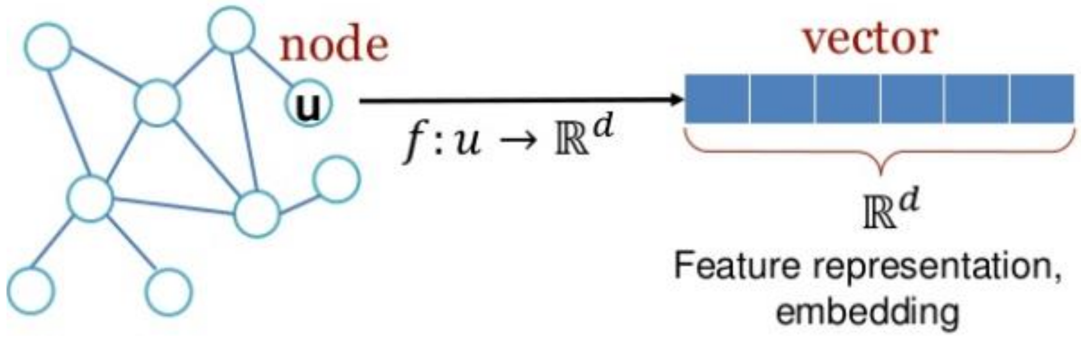
Nodes to Vectors

node2vec (Grover and Leskovec, 2016)


Input



Output



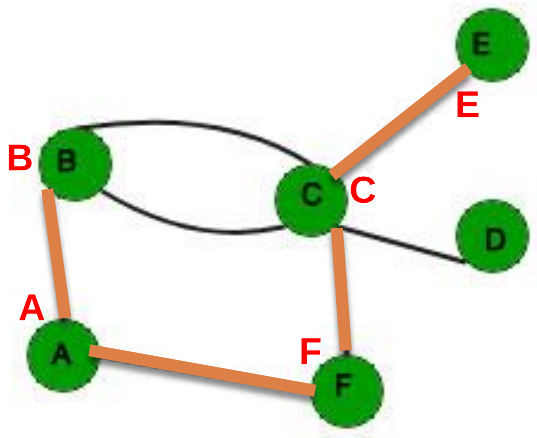
(Source: snap.stanford.edu/proj/embeddings-www, WWW 2018)

- **Motivation**:
  - Find embedding of nodes to d-dimensions so that "similar" nodes in the graph have embeddings that are close together.

Node2Vec

Idea

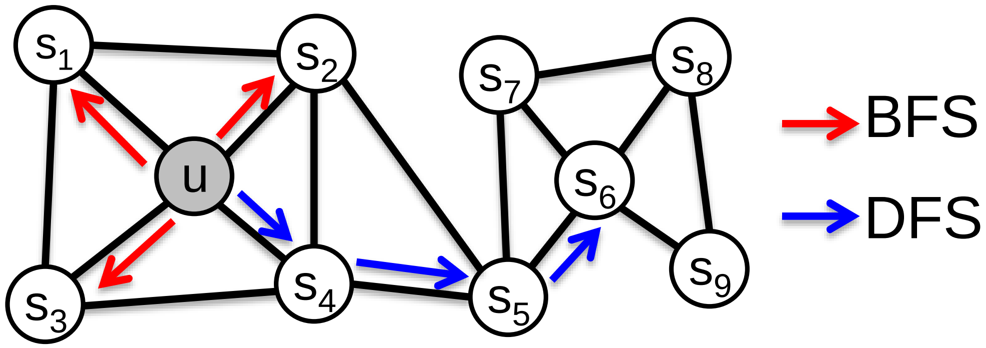
- Do **random walks** on the graph
  - B, A, F, C, E → each node is a word → $F \xrightarrow{predict} B, A, C, E$
- How to find **good random walks**?
  - Cover all important paths in a graph
  - Balance between microscopic and macroscopic views



Node2Vec

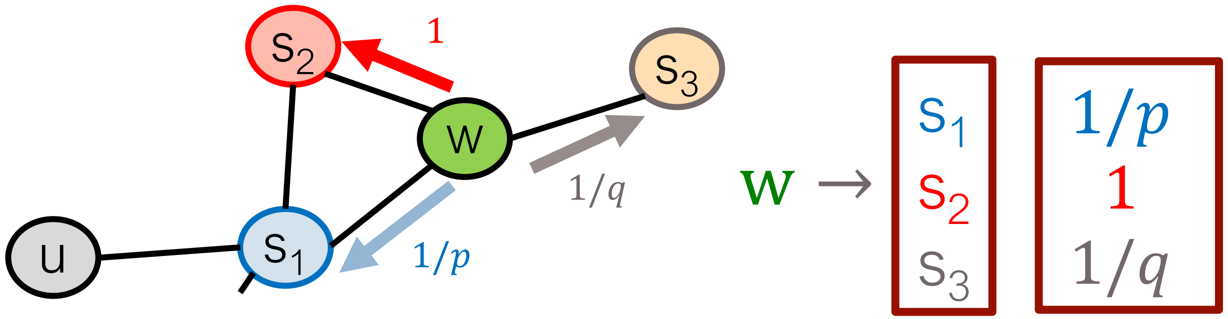
DFS vs BFS

- $N_R(u)$ 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, 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
- **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

Appendix

Motivation

- All relevant text analytics/NLP tasks
  - e.g., text classification into: politics, medical, business, science, sports
- Understand the meaning of words in the document.
  - A lot of them you never seen. In fact the ones that you rarely see tend to be the most important ones.

```
model.most_similar('insomnia')
```

```
[(u'sleeplessness', 0.6954464316368103),
 (u'chronic_insomnia', 0.6937799453735352),
 (u'sleep_disorders', 0.6682900190353394),
 (u'excessive_sleepiness', 0.6534832119941711),
 (u'migraine', 0.6464337706565857),
 (u'anxiety_insomnia', 0.6439672112464905),
 (u'constipation', 0.6267993450164795),
 (u'nighttime_heartburn', 0.6188271045684814),
 (u'migraines', 0.6186375617980957),
 (u'migraine_headaches', 0.6150763034820557)]
```

What does 'insomnia' mean?

Motivation

- Estimate the similarity between two words
- Why symbolic/one-hot vector representation will fail?
  - can’t capture ‘context’ around – word is presented independently!

```
import numpy as np
from numpy import dot
from numpy.linalg import norm

# compute cosine similarity for two vector u & v
def cosine_sim(u, v):
    return dot(u, v)/(norm(u)*norm(v))

vocabulary = ['king', 'man', 'queen', 'woman']
tokens = {w:i for i,w in enumerate(vocabulary)}

N = len(vocabulary)
W = np.zeros((N, N))
np.fill_diagonal(W, 1)


print ("cosine_similarity('king','woman'): {}".format(cosine_sim(W[tokens['king']], W[tokens['woman']])))
print ("cosine_similarity('man','woman'): {}".format(cosine_sim(W[tokens['man']], W[tokens['woman']])))
print ("cosine_similarity('queen','woman'): {}".format(cosine_sim(W[tokens['queen']], W[tokens['woman']])))
```

```
cosine_similarity('king','woman'): 0.0
cosine_similarity('man','woman'): 0.0
cosine_similarity('queen','woman'): 0.0
```

THIS ….

Motivation

- Estimate the similarity between two words
- Why symbolic/one-hot vector representation will fail?
  - can't capture 'context' around – word is presented independently!

```
from gensim.models import Word2Vec

# load pre-trained GoogleNews model
model_file = 'model/GoogleNews_small'
W = Word2Vec.load(model_file)

print ("cosine_similarity('king','woman'): {}".format(cosine_sim(W['king'], W['woman'])))
print ("cosine_similarity('man','woman'): {}".format(cosine_sim(W['man'], W['woman'])))
print ("cosine_similarity('queen','woman'): {}".format(cosine_sim(W['queen'], W['woman'])))
```

```
cosine_similarity('king','woman'): 0.128479748964
cosine_similarity('man','woman'): 0.766401290894
cosine_similarity('queen','woman'): 0.316181391478
```

VERSUS THIS ….

Toy example

- Corpus: **the quick brown fox jumps over the lazy dog**
  - Tokens: {'brown': 0, 'lazy': 1, 'over': 2, 'fox': 3, 'dog': 4, 'quick': 5, 'the': 6, 'jumps': 7}
  - Number of tokens N = 8
  - Context (window) size C = 1
  - Size of embedded vectors d = 3
  - U&V: collections of input & output vectors

quick

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

one-hot encoding

Skip-gram, forward propagation

N*1	N*d	d*1	d*N	N*1	N*1
quick		$h = (1_t^T \cdot \mathrm{U})^T$		$(h^T \mathrm{V})^T$	$p(o\|i)$

$$\begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix} \quad \begin{bmatrix}.5 & .0 & .3\\.3 & .7 & .9\\.3 & .5 & .2\\.4 & .7 & .6\\.8 & .8 & .1\\.6 & .7 & .7\\.8 & .1 & .5\\.9 & .8 & .9\end{bmatrix} \Rightarrow \begin{bmatrix}.6\\.7\\.7\end{bmatrix} \quad \begin{bmatrix}.4 & .3 & .0\\.3 & .5 & .0\\.2 & .3 & .8\\.1 & .3 & .3\\.3 & .7 & .0\\.1 & .9 & .9\\.0 & .4 & .7\\.3 & .2 & .7\end{bmatrix}^T \quad \begin{bmatrix}0.45\\0.53\\0.89\\0.48\\0.67\\1.32\\0.77\\0.81\end{bmatrix}^T \xrightarrow{\text{softmax}} \begin{bmatrix}.09\\.10\\.14\\.09\\.11\\.21\\.12\\.13\end{bmatrix}$$

$w_t = 1_t$ **U** **V**

center

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

brown

w_{t+1}

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

the

w_{t-1}

context

CBOW, forward propagation

brown

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

w_{t+1}

the

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

w_{t-1}

context

N*d

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

U

d*1

$$h = \left((w_{i-1} + w_{i+1})^T \cdot \mathrm{U}\right)^T$$

$$\begin{bmatrix}1.3\\0.1\\0.8\end{bmatrix}$$

d*N

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

V

N*1

$$\left(\mathrm{h}^{\mathrm{T}}.\mathrm{U}\right)^T$$

$$\begin{bmatrix}0.55\\0.44\\0.93\\0.40\\0.46\\0.94\\0.60\\0.97\end{bmatrix} \xrightarrow{\text{softmax}}$$

N*1

$$p(o|i)$$

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

N*1

quick

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

w_t

center

Learning word2vec: skip-gram

- For each word *t* = 1 … *T*, predict surrounding words in a window of "radius" *C* of every word.
- **Objective function**: Maximize the probability of any context word given the current centre word:

$$J(\theta) = \prod_{t=1}^{T} \prod_{-C \le j \le C, j \ne 0} p(w_{t+j} | w_t; \theta)$$

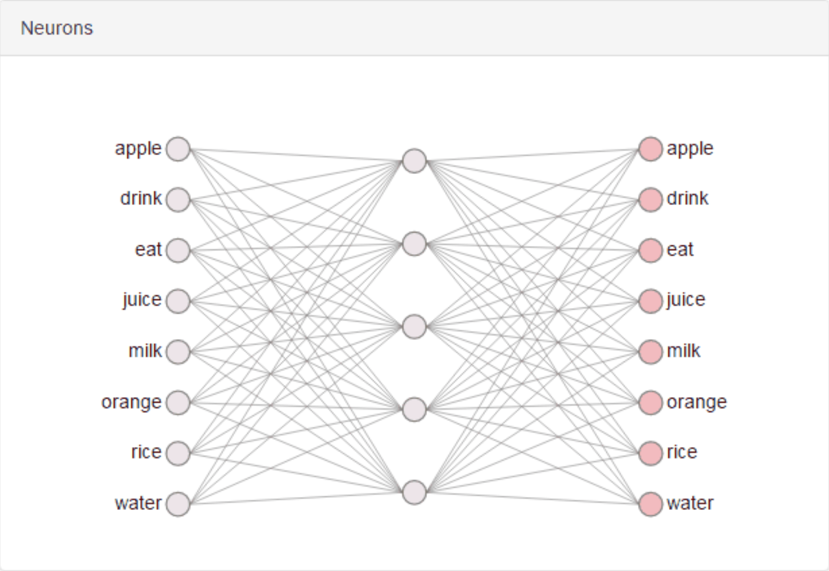
- Or, equivalently minimize the negative log likelihood:

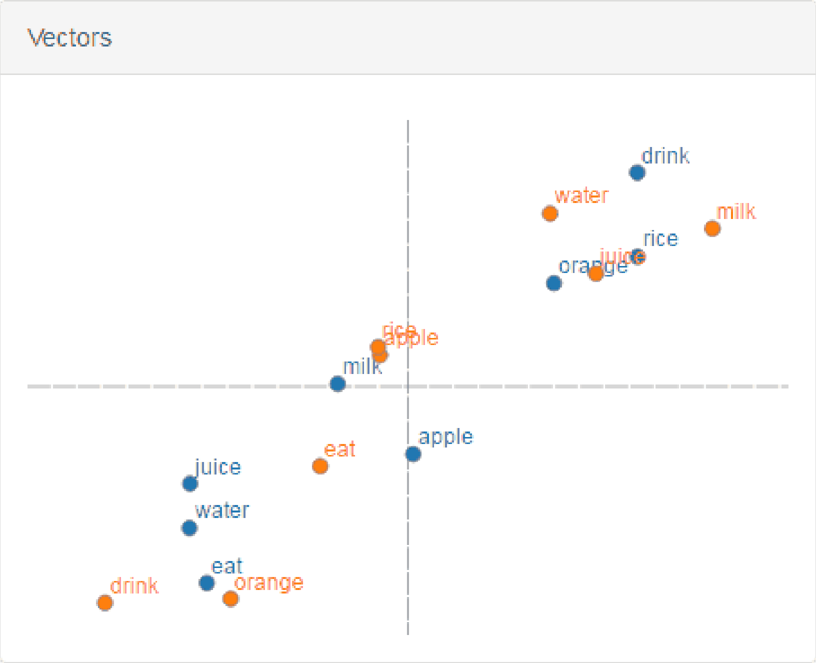
$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-C \le j \le C, j \ne 0} \log\left(p\left(w_{t+j} | w_t; \theta\right)\right)$$

where $p(o|i) = \frac{e^{u_o^T v_i}}{\sum_{w=1}^{W} e^{u_w^T v_i}}$, the *softmax* function, converting scores all classes into a probability distribution

Word embedding

- Representing meaning of words
  - Important for all NLP tasks.
  - WordNet
  - Discrete/one-hot representation
  - Methods: LSA, MDS, LDA
- **Word2vec** (Mikolov et al. 2013)
  - Distributed representation for word
  - What does it mean? Learn a real-valued vector for each word.
  - Small distances induce similar words
  - Compositionality induces real-valued vector representation for phrases/sentences.
  - A simple application of NN with exactly 2 layers!





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