



# Word Embeddings

NLP II 2025

Assoc. Prof. Attapol Thamrongrattanarit



## Models of language we have so far...

- Logistic Regression with bag-of-word features
  - The logistic regression is a decent model for finding the features that are indicative of the labels.
  - The model is easy to train. It doesn't take long or a lot of computing power.
  - The bag-of-word features rely too much on exact string matches. They don't recognize synonyms or semantically-related words.
- N-gram language models
  - They can be trained on large datasets easily.
  - They also rely on exact string matches.  
 $P(w| \text{'The authority has issued'})$  vs  $P(w| \text{'The officials have declared'})$

A good model of language must  
capture the semantic relations of  
words and phrases

---

A horizontal bar with a gold segment on the left and a red segment on the right.

## How do we 'model meanings'?

- We start from the smallest unit that still conveys meaning because that seems to be the easiest thing to do. How do we model meaning of a word? This is the main focus of 'lexical semantics'
- What do linguists think when they think about meanings of words?



## Lemma and word senses

- A lemma is the base or dictionary form of a word.
  - "run" is the lemma for: "running," "ran," "runs"
  - "better" (as an adjective) has the lemma "good"
- A word sense refers to a specific meaning of a word.
- Many words are homonymous, meaning they have multiple (unrelated) senses depending on context despite having the same form.
  - "Bat" 1: "He swung the bat and hit the ball." (sports equipment)
  - "Bat" 2: "A bat flew out of the cave." (flying mammal)
  - "Bank" 1: "The client saw a loan officer at the bank" (financial institution)
  - "Bank" 2: "Over time, the river shaves off part of the bank" (sloping mound)



## Lexical Relation

- Synonym: {dirty, filthy, dusty} {clean, sterile} {big, large}
  - The dining table is dirty after the meal.
  - The dining table is dusty/filthy after the meal.
  - My life revolves around my big sister.
  - My life revolves around my large sister.
- Antonym: dirty vs clean, easy vs hard

# Antonym



king

antonym?

queen



slave





## Word Similarity

Lexical relations have their challenges. The notion of 'word similarity' is more appealing and more flexible.

word1	word2	similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3





## More lexical relations

- Hyponym and hypernym:

dog is an animal  $\longleftrightarrow$  *dog* is a hyponym of *animal* AND

*animal* is a hypernym of *dog*

cat is an animal  $\longleftrightarrow$  *cat* is a hyponym of *animal*

- Cohyponym: *dog* and *cat* are cohyponyms because they have the same hypernym.

Animal



Hypernymy

A horizontal bar with a gold segment on the left and a red segment on the right.

## What do we get from hypernymy?

- Hypernymy, hyponymy, co-hypernymy present 'similar words' e.g. coffee vs tea. But some words are related but not similar e.g. coffee vs cup because they are in the same semantic field.
- Synonymy and antonymy present 'similar words' for similar reasons. They are in the same semantic field.

```
from nltk.corpus import wordnet as wn
poses = { 'n': 'noun', 'v': 'verb', 's': 'adj (s)', 'a': 'adj', 'r': 'adv' }
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
        ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
...
adverb: well, good
adverb: thoroughly, soundly, good
```

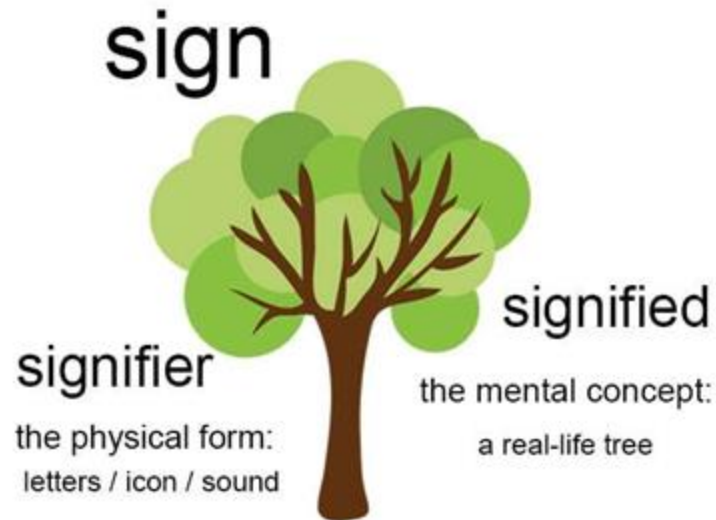
```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

WordNet example <https://en-word.net/>

## Representing a word

- Denotational semantics is an approach where we define meaning of a word by mapping to a mathematical object.
- Signifier (word)  $\rightarrow$  Signified (idea or thing)
- Solution 1: We represent the word with a one-hot vector



# Computing with meaning

- In the old days of traditional NLP, we represent a word with one-hot vectors with the size equal to the vocabulary.
  - hostel = [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
  - hotel = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
  - Bangkok = [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
  - Bangkok hostel = [1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
  - Bangkok hotel = [1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
- We compare the similarity of two vectors by computing cosine similarity
- If we search for 'Bangkok hotel,' we will get 'Bangkok hotel' and 'Bangkok' and nothing of 'hostels' because the cosine similarity is zero.

A horizontal bar with a gold segment on the left and a red segment on the right.

# Computational Lexical Semantics

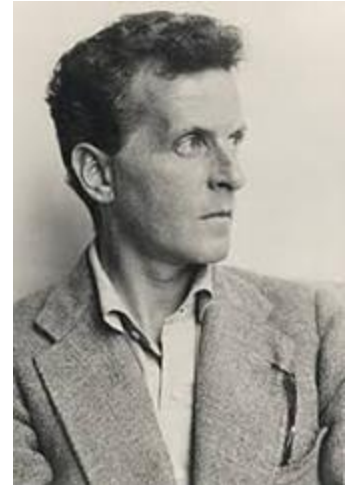
- Lexical semanticists point out that
  - The same word form might have multiple senses.
  - Word meanings can be understood in terms of their relations
  - Semantic similarity is key for understanding word meanings
- Computational linguists aim to make a computational model of word meanings.

# Philosophers have very good ideas about languages



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

*"Die Bedeutung eines Wortes ist sein Gebrauch in der Sprache"*  
*The meaning of a word is its use in the language*  
Ludwig Wittgenstein  
Philosophische Untersuchungen (1953)

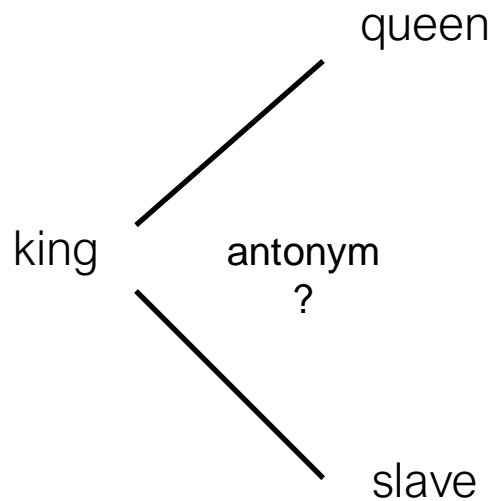




## Co-occurrences suggest meaning

	ordered	throne	he	she	killed	poor	...
king	30	20	5	8	10	3	
queen	25	15	3	12	3	2	
slave	5	3	8	6	40	25	
woman	10	5	4	9	5	10	
...							

similarity(king, queen) = 0.74



similarity(king, slave) = 0.34



# Animal



similarity(dog, cat) =  
0.57



similarity(dog, elephant) =  
0.34

## Word-context matrix

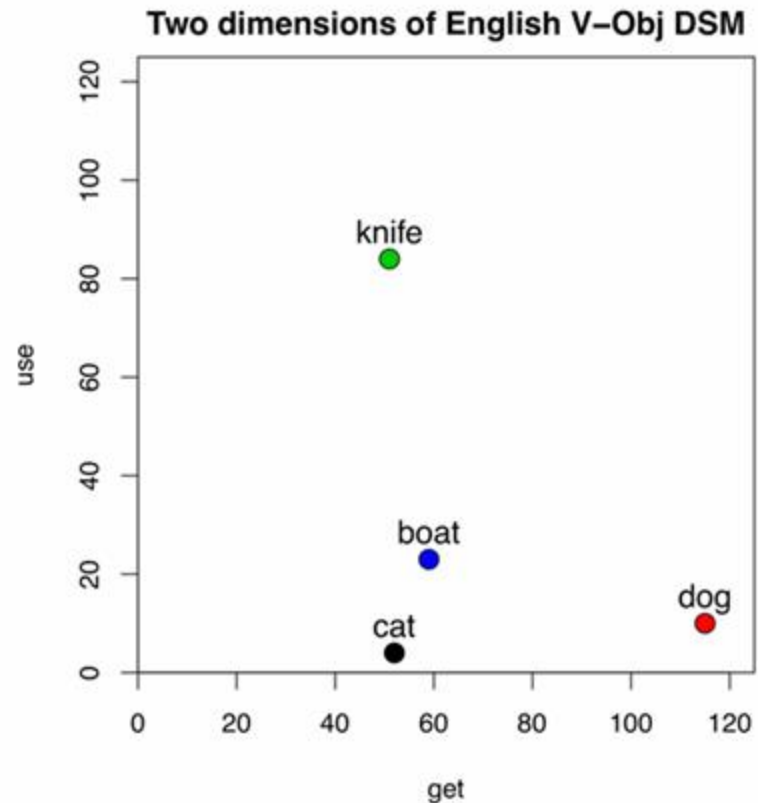
Collect what each word co-occurs with

Suppose the context is the nearest verb to the word

	get	see	use	hear	eat	kill
knife	51	20	84	0	3	0
cat	52	58	4	4	6	26
dog	115	83	10	42	33	17
boat	59	39	23	4	0	0
cup	98	14	6	2	1	0
pig	12	17	3	2	9	27
banana	11	2	2	0	18	0

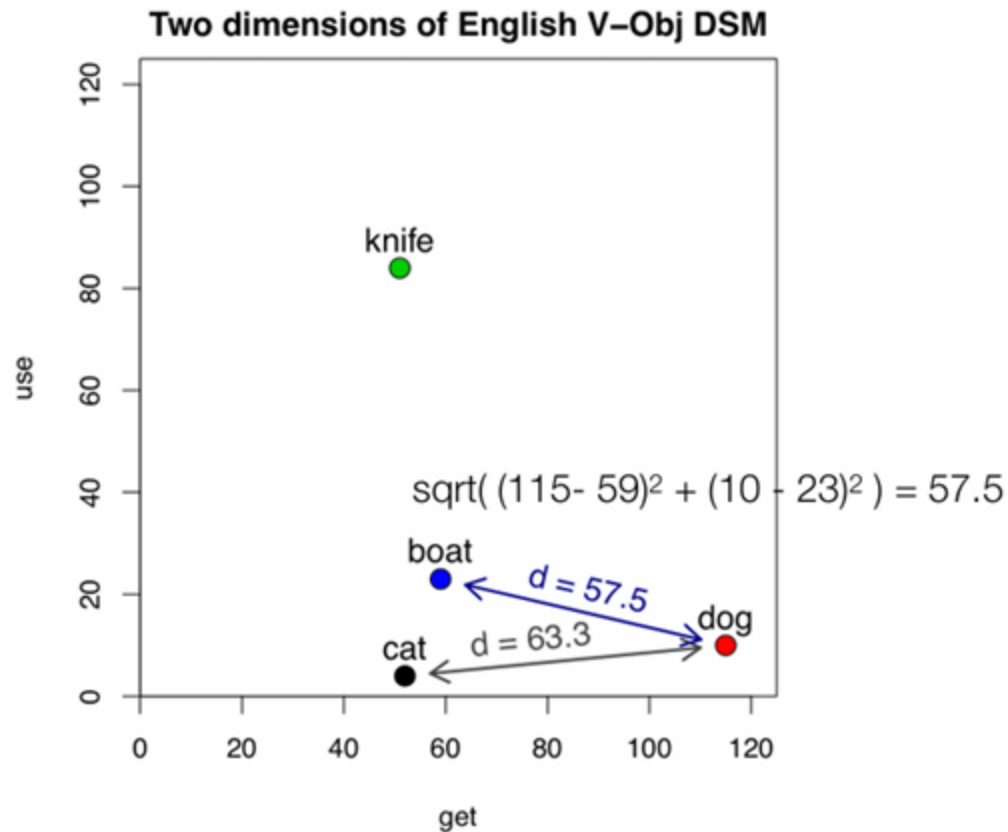
$$[x, y] = [\text{\#get}, \text{\#use}]$$

	<b>get</b>	see	<b>use</b>	hear	eat	kill
knife	<b>51</b>	20	<b>84</b>	0	3	0
cat	<b>52</b>	58	<b>4</b>	4	6	26
dog	<b>115</b>	83	<b>10</b>	42	33	17
boat	<b>59</b>	39	<b>23</b>	4	0	0
cup	<b>98</b>	14	<b>6</b>	2	1	0
pig	<b>12</b>	17	<b>3</b>	2	9	27
banana	<b>11</b>	2	<b>2</b>	0	18	0



$[x, y] = [\text{\#get}, \text{\#use}]$

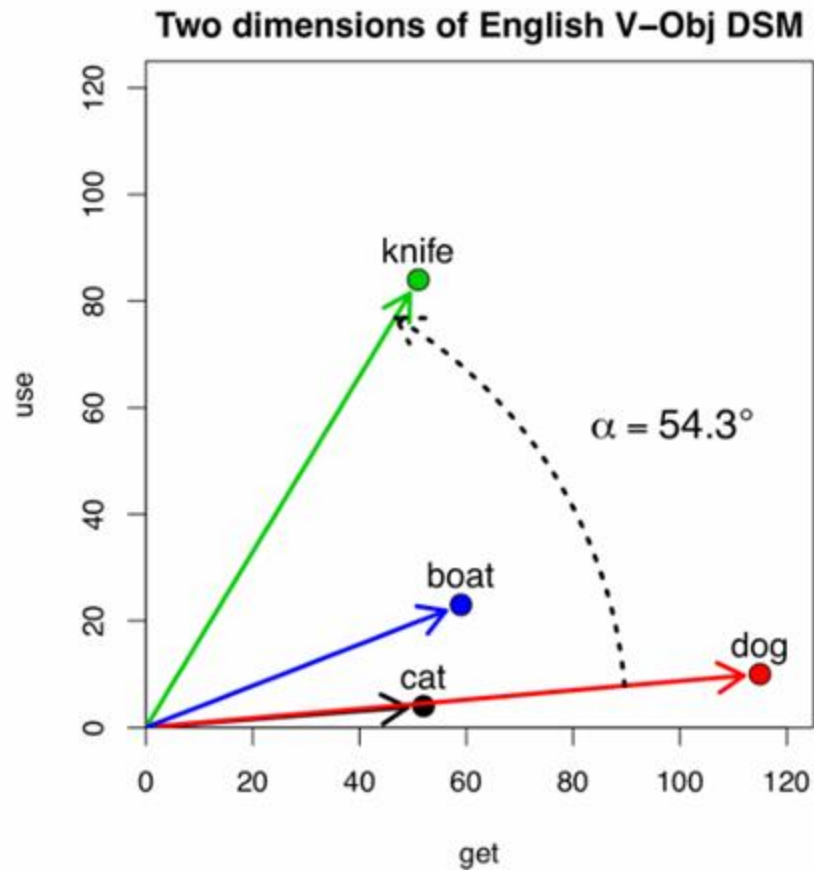
	get	use
knife	51	84
cat	52	4
dog	115	10
boat	59	23
cup	98	6
pig	12	3
banana	11	2





$$[x, y] = [\text{\#get}, \text{\#use}]$$

	get	use
knife	51	84
cat	52	4
dog	115	10
boat	59	23
cup	98	6
pig	12	3
banana	11	2



## Cosine-distance and similarity

	get	use
knife	51	84
cat	52	4
dog	115	10
boat	59	23
cup	98	6
pig	12	3
banana	11	2

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

$$\text{similarity (knife, dog)} = 0.59$$

$$\text{distance} = 1 - \text{similarity}$$

$$\text{distance (knife, dog)} = 1 - 0.59 = 0.41$$

$$\text{Similarity (cat, dog)} = 1$$

$$\text{distance (cat, dog)} = 1 - 1 = 0$$



## Word-context matrix

	get	see	use	hear	eat	kill
knife	51	20	84	0	3	0
cat	52	58	4	4	6	26
dog	115	83	10	42	33	17
boat	59	39	23	4	0	0
cup	98	14	6	2	1	0
pig	12	17	3	2	9	27
banana	11	2	2	0	18	0

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

```
>> import numpy as np
>> dog = np.array([115,83,10,42,33,17])
>> cat = np.array([52,58,4,4,6,26])
>> dog.dot(cat) / (np.linalg.norm(dog) *
np.linalg.norm(cat))
0.9229773468286717
```

## How to create word-context matrix

	get	see	use	hear	eat	kill
knife	51	20	84	0	3	0
cat	52	58	4	4	6	26
dog	115	83	10	42	33	17
boat	59	39	23	4	0	0
cup	98	14	6	2	1	0
pig	12	17	3	2	9	27
banana	11	2	2	0	18	0

- Determine what is in your vocabulary
  - every word?
  - occurrence > 10?
- Determine the context window size



## From counts to word embeddings

- Problems:
  - The size of the word embeddings is still equal to the vocabulary.
  - Most of the values in the embeddings are zero.
- Solution
  - Use math to compress the word-context matrix and reduce the dimension of each vector in the matrix



# Dense vectors

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 ,
-0.31012 , -0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -
0.55809 , 0.66421 , 0.1961 , -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -
1.0783 , -0.34354 , 0.33505 , 1.9927 , -0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 ,
0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 , -1.0137 , -0.21585 , -0.15155 ,
0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]
```





## Dense vectors

“king”



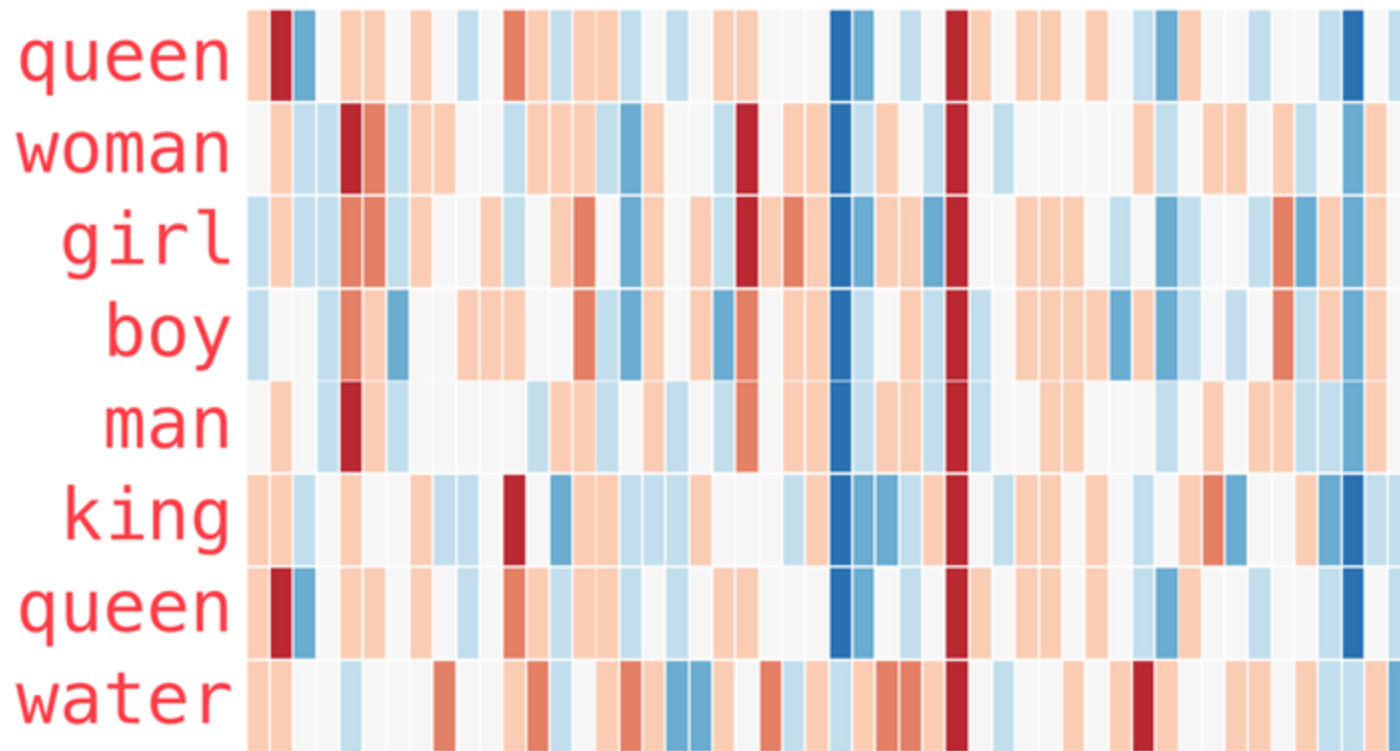
“Man”



“Woman”



Can you see patterns in word vectors?



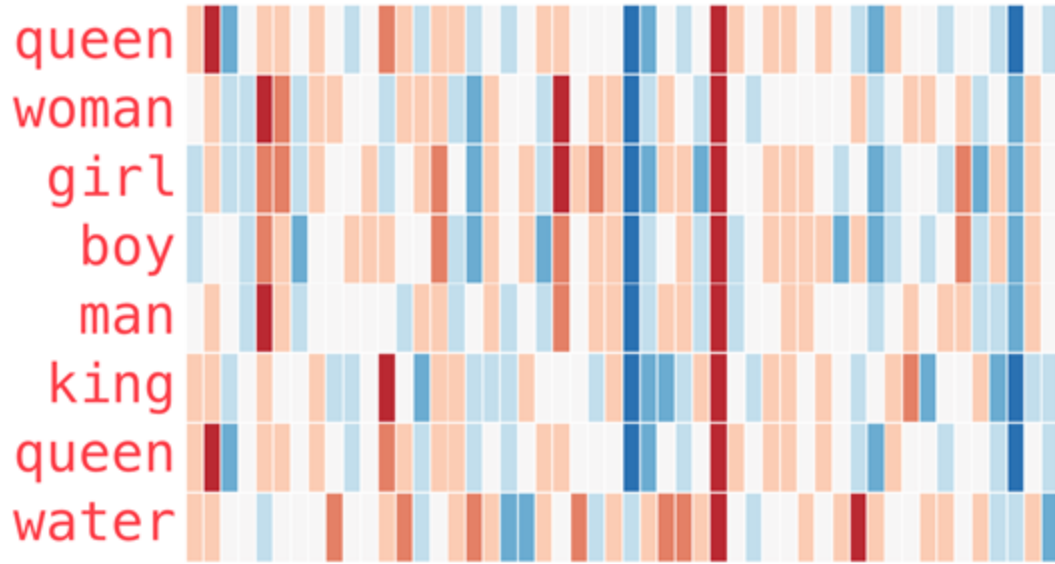
A horizontal bar with a gold segment on the left and a red segment on the right.

## Word vectors are in the same vector space

- Since they are in the (semantic) space, we can calculate the (semantic) distance between a pair of words. We use cosine similarity to compute the distance (how far apart) between two vectors.



# Compute cosine similarity



Word1	Word2	Cosine sim
girl	boy	0.932
woman	man	0.886
king	queen	0.783
queen	woman	0.6
man	water	0.386
king	water	0.235

[Vector:Glove.6B.50D](#)



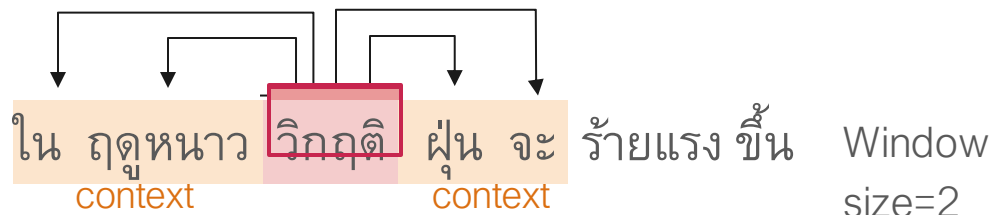
## Why do we prefer dense vectors?

- Short vectors may be easier to use as features in machine learning (fewer weights to tune)
- Dense vectors may generalize better than explicit counts
- Dense vectors may do better at capturing synonymy:
  - car and automobile are synonyms; but are distinct dimensions
  - a word with car as a neighbor and a word with automobile as a neighbor should be similar, but aren't.
- In practice, dense vectors work better.

A horizontal bar with a gold segment on the left and a red segment on the right.

## Learning word embeddings

- Word embedding embeds into each word and captures the syntactic and semantic features for the word.
- Word2vec is one of the many algorithms that learn word embeddings from word-context matrix/windows.



ใน	ฤดูหนาว	วิกฤติ	ฝุ่น	จะ	ร้ายแรง	ขึ้น
----	---------	--------	------	----	---------	------

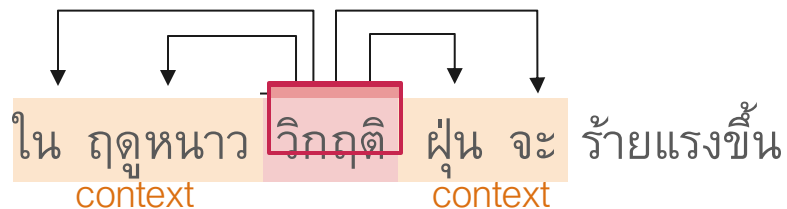
ใน	ฤดูหนาว	วิกฤติ	ฝุ่น	จะ	ร้ายแรง	ขึ้น
----	---------	--------	------	----	---------	------

ใน	ฤดูหนาว	วิกฤติ	ฝุ่น	จะ	ร้ายแรง	ขึ้น
----	---------	--------	------	----	---------	------

ใน	ฤดูหนาว	วิกฤติ	ฝุ่น	จะ	ร้ายแรง	ขึ้น
----	---------	--------	------	----	---------	------

Input word	Target word
วิกฤติ	ใน
วิกฤติ	ฤดูหนาว
วิกฤติ	ฝุ่น
วิกฤติ	จะ
ฝุ่น	ฤดูหนาว
ฝุ่น	วิกฤติ
ฝุ่น	จะ
ฝุ่น	ร้ายแรง
จะ	วิกฤติ
จะ	ฝุ่น
จะ	ร้ายแรง
จะ	ขึ้น
...	...

# Text classification problem

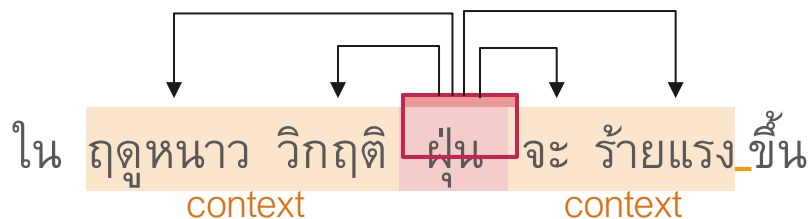


Window size=2

## Training data

Input	Label
วิกฤติ	ใน
วิกฤติ	ถูหนาว
วิกฤติ	ฝุ่น
วิกฤติ	จะ

# Text classification problem

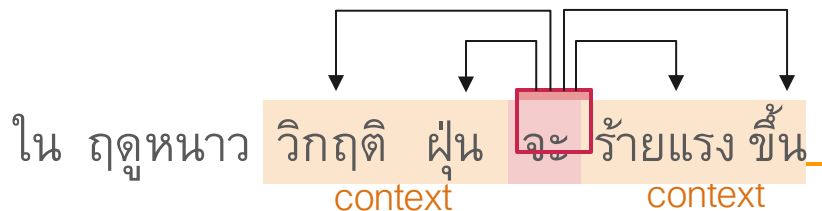


Window size=2

## Training data

Input	Label
วิกฤติ	ใน
วิกฤติ	ฤดูหนาว
วิกฤติ	ฝุ่น
วิกฤติ	จะ
ฝุ่น	ฤดูหนาว
ฝุ่น	วิกฤติ
ฝุ่น	จะ
ฝุ่น	ร้ายแรง

# Text classification problem



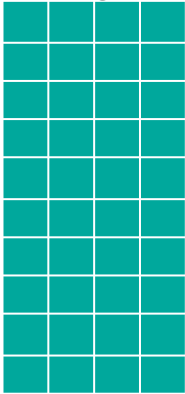


Window size=2

## Training data

Input	Label
วิกฤติ	ใน
วิกฤติ	ถูหนาว
วิกฤติ	ฝุ่น
วิกฤติ	จะ
ฝุ่น	ถูหนาว
ฝุ่น	วิกฤติ
ฝุ่น	จะ
ฝุ่น	ร้ายแรง
จะ	วิกฤติ
จะ	ฝุ่น



แปลง embedding ให้ทำนายคำรอบ ๆ      คำว่า วิกฤติ มีโอกาสเกิดร่วมกับคำใดบ้าง

Softmax (  )       $\odot$       (  )      =       )

weight      embedding       $P(c|w)$   
 $V \times k$        $k$        $V$



## Softmax function (review)

vector	$\exp(\text{vector})$	$\exp(\text{vector}) / \text{sum of exp}$	probability
0.25	1.284	1.284	0.151
-0.75	0.472	0.472	0.055
1.5	4.481	4.481	0.529
0.8	2.225	2.225	0.262
		$\div$ 8.462	$=$
sum( $\exp(\text{vector})$ ) = 8.462			sum(prob) = 1

A horizontal bar with a gold segment on the left and a red segment on the right.

## Word2Vec training process

[https://docs.google.com/spreadsheets/d/1a7XE0MwSE6bGjbnO\\_N6i89Wo3acV5Yu8IKuifl4AEU8/edit?gid=0#gid=0](https://docs.google.com/spreadsheets/d/1a7XE0MwSE6bGjbnO_N6i89Wo3acV5Yu8IKuifl4AEU8/edit?gid=0#gid=0)

# Intuition for word2vec training

- Use the current state of word embedding models to predict whether the output word should co-occur with the center word:
  - $P(\text{context}|\text{center})$  should be very high (close to 1) for all of the words in the window around the center word. Adjust the embeddings and the context matrix so that  $P(\text{context}|\text{center})$  is close to 1.
  - $P(\text{context}|\text{center})$  should be very low (close to 0) for the other words in the vocabulary. Adjust the embeddings and the context matrix so that  $P(\text{context}|\text{center})$  is close to 0.
- Shift the window to change to the new center word and repeat

Loss function

$\theta = U, V$      $D =$  a list of tokens

$$\begin{aligned} J(\theta) &= \sum_{i=1}^{\text{len}(D)} \sum_{j=-2}^2 -\log P(\text{context}=D[i+j] | \text{center}=D[i]) \\ &= \frac{\exp(u_{D[i+j]}^T \cdot v_{D[i]})}{\sum_{k \in \text{vocab}} \exp(u_k^T \cdot v_{D[i]})} \\ &= \frac{\exp(u_{D[i+j]}^T \cdot v_{D[i]})}{\text{sum}(\exp(U \cdot v_{D[i]}))} \end{aligned}$$

A horizontal bar with a gold segment on the left and a red segment on the right.

## Practically

- When we train a word embedding model, we will define a loss function and then use an optimization function to minimize it.
- Computing the probability over the vocabulary is very expensive. Why?

A horizontal bar with a gold segment on the left and a red segment on the right.

## Negative Sampling

Negative sampling is a technique used in the training of the Word2Vec model to make it computationally efficient. Word2Vec algorithm involves computing probabilities over the entire vocabulary, which can be computationally expensive. Negative sampling addresses this problem by simplifying the training objective into that of binary classification.

# Change the loss function to a binary loss function

Input word	Target word
วิกฤติ	ใน
วิกฤติ	ฤดูหนาว
วิกฤติ	ฝุ่น
วิกฤติ	จะ
ฝุ่น	ฤดูหนาว
ฝุ่น	วิกฤติ
ฝุ่น	จะ
ฝุ่น	ร้ายแรง
จะ	วิกฤติ
จะ	ฝุ่น
จะ	ร้ายแรง
จะ	ขึ้น



Input word	Output word	Target
วิกฤติ	ใน	1
วิกฤติ	ฤดูหนาว	1
วิกฤติ	ฝุ่น	1
วิกฤติ	จะ	1
ฝุ่น	ฤดูหนาว	1
ฝุ่น	วิกฤติ	1
ฝุ่น	จะ	1
ฝุ่น	ร้ายแรง	1
จะ	วิกฤติ	1
จะ	ฝุ่น	1
จะ	ร้ายแรง	1
จะ	ขึ้น	1

## Sample words that are not in the window

Input word	Output word	Target
วิกฤติ	ใน	1
วิกฤติ	กต	0
วิกฤติ	นกยูง	0
วิกฤติ	ประชาชาติ	0
วิกฤติ	ฝุ่น	1

Pick randomly from vocabulary  
(random sampling)

Word

กต

กตিকা

กรรม

นกยูง

นัด

.

ประชาชาติ

...



# Embedding matrix V and context matrix U

## Embedding (V)

				กต
				กตিকা
				...
				ฝุ่น
				...
				วิกฤติ
				...
				...
				...
				ไฮโล

## Context (U)

				กต
				กตিকা
				...
				นกยูง
				...
				...
				ฝุ่น
				...
				...
				ไฮโล

Look up  
embeddings



วิกฤติ



กต



นกยูง









ฝุ่น



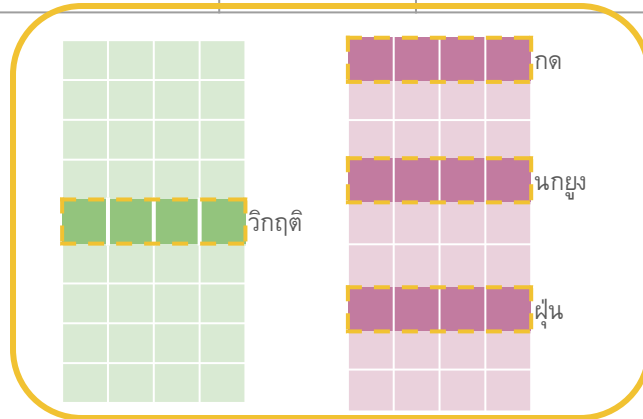
# Sigmoid function

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

Input word	Output word	Target	Input · Output	sigmoid( )
วิกฤติ 	ฝุ่น 	1	0.2	0.55
วิกฤติ 	กด 	0	-1.11	0.25
วิกฤติ 	นกยูง 	0	0.74	0.68

# Learning from positive and negative signal

Input word	Output word	Target	Input · Output	sigmoid( )	Error
วิกฤติ	ฝุ่น	1	0.2	0.55	0.45
วิกฤติ	กด	0	-1.11	0.25	-0.25
วิกฤติ	นกยูง	0	0.74	0.68	-0.68



Update  
Model  
Parameters

The loss function for negative sampling

$\theta = U, V$      $D =$  a list of tokens

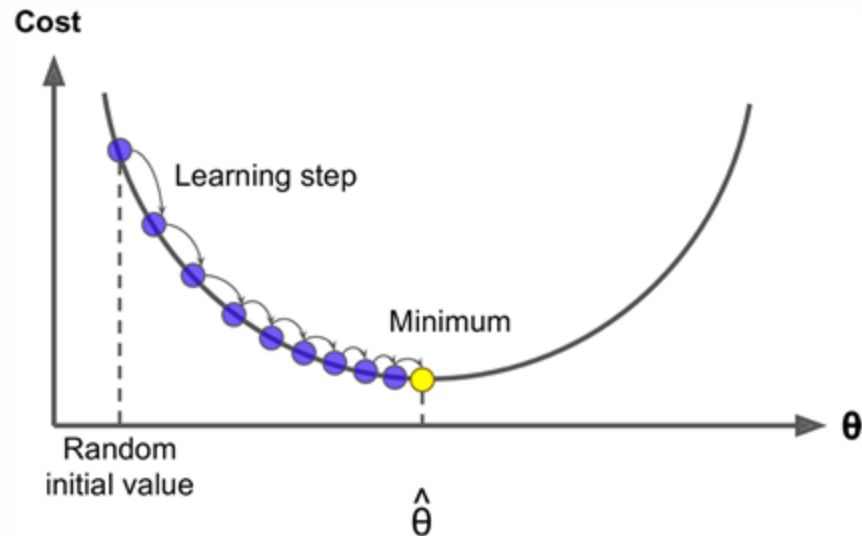
$$J(\theta) = \sum_{i=1}^{len(D)} \left( \sum_{j=-2}^2 -\log P(\text{context}=D[i+j]|\text{center}=D[i]) \right. \\ \left. + \sum_k \log P(\text{context}=n_k|\text{center}=D[i]) \right)$$

negative samples  
for center word  $D[i]$

$$= \text{sigmoid}(u_{D[i+j]}^T \cdot v_{D[i]})$$
$$= \frac{1}{1 + \exp(-u_{D[i+j]}^T \cdot v_{D[i]})}$$

# Optimization: Gradient Descent

- We have a cost function  $J(\theta)$  we want to minimize
- **Gradient Descent** is an algorithm to minimize  $J(\theta)$
- **Idea:** for current value of  $\theta$ , calculate gradient of  $J(\theta)$ , then take **small step in direction of negative gradient**. Repeat.



# Gradient Descent

- Update equation (in matrix notation):

$$\theta^{new} = \theta^{old} - \underline{\alpha} \nabla_{\theta} J(\theta)$$

$\alpha$  = step size or learning rate

- Update equation (for single parameter):

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

- Algorithm:

```
while True:
    theta_grad = evaluate_gradient(J, corpus, theta)
    theta = theta - alpha * theta_grad
```

# Stochastic Gradient Descent

- **Problem:**  $J(\theta)$  is a function of **all** windows in the corpus (potentially billions!)
  - So  $\nabla J(\theta)$  is **very expensive to compute**
- You would wait a very long time before making a single update!
- Very bad idea for pretty much all neural nets!
- **Solution: Stochastic gradient descent** (SGD)
  - Repeatedly sample windows, and update after each one
- **Algorithm:**

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J, window, theta)
    theta = theta - alpha * theta_grad
```

**Mini Batch  
Gradient  
Descent**