



# Machine Learning

## Lecture 10: Representation Learning of Words

Fall 2023

Instructor: Xiaodong Gu

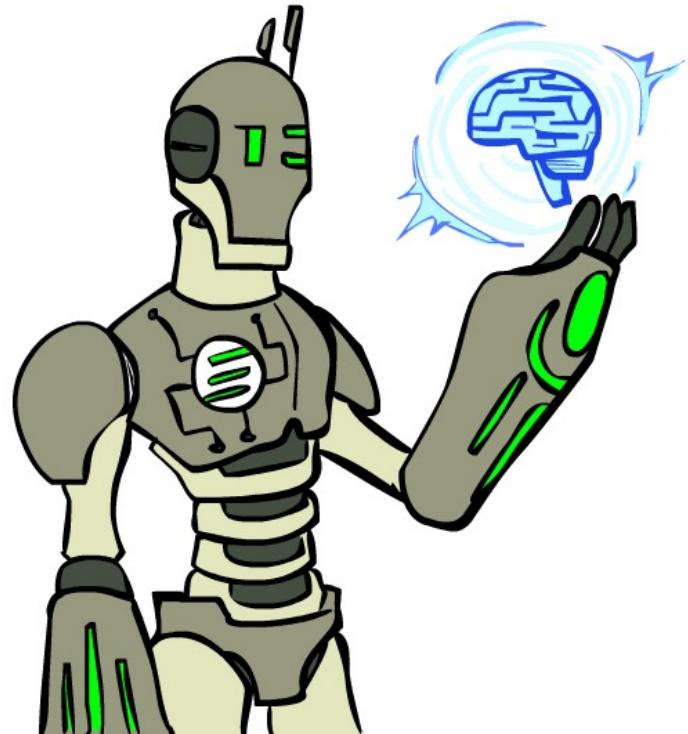


# Today

---



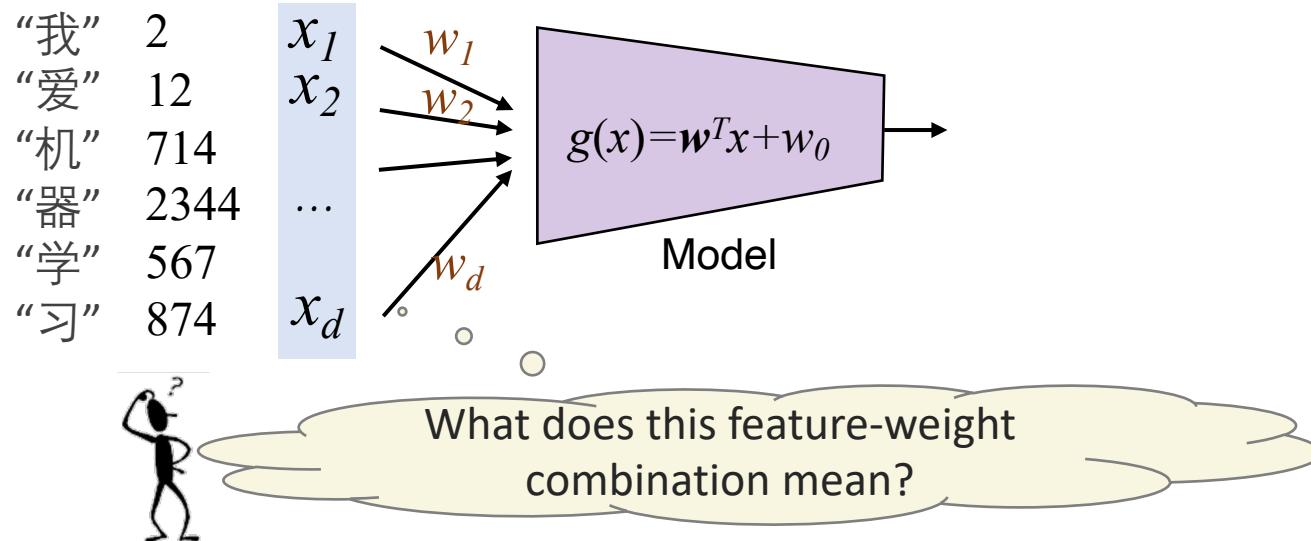
- Word Embedding (word2vec)  
Application of **neural networks** for distributed representations of words
- Language Model  
A key framework in NLP that is important in the follow-up lectures.



# Before learning word embedding...



- How to represent words in computers?
  - discrete-symbols (ASCII, id, etc.)
  - e.g., “hotel”:11, “conference”:81, “motel”: 1203, ...
  - a **localist** representation
- How to represent words in machine learning models?



# Words as Vectors



- Words can be represented by **one-hot** vectors:

## One-Hot Encoding

Means one 1, the rest 0s

hotel = [ 1 0 0 0 0 0 0 0 0 ]

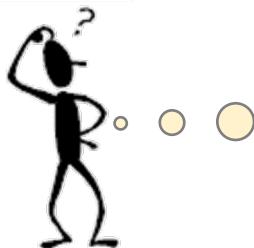
flower = [ 0 1 0 0 0 0 0 0 0 ]

tree = [ 0 0 0 1 0 0 0 0 0 ]

motel = [ 0 0 0 0 0 0 0 1 0 ]

elephant = [ 0 0 0 0 0 0 0 0 1 ]

Vector dimensionality =  
number of words in  
vocabulary (e.g., 500,000)



How can we capture relationships  
(similarity) between words?

# Word Embeddings



- Represent words as **low-dimensional dense vectors** that can reflect their semantic similarities.

## One-Hot Encoding

hotel = [ 1 0 0 0 0 ]

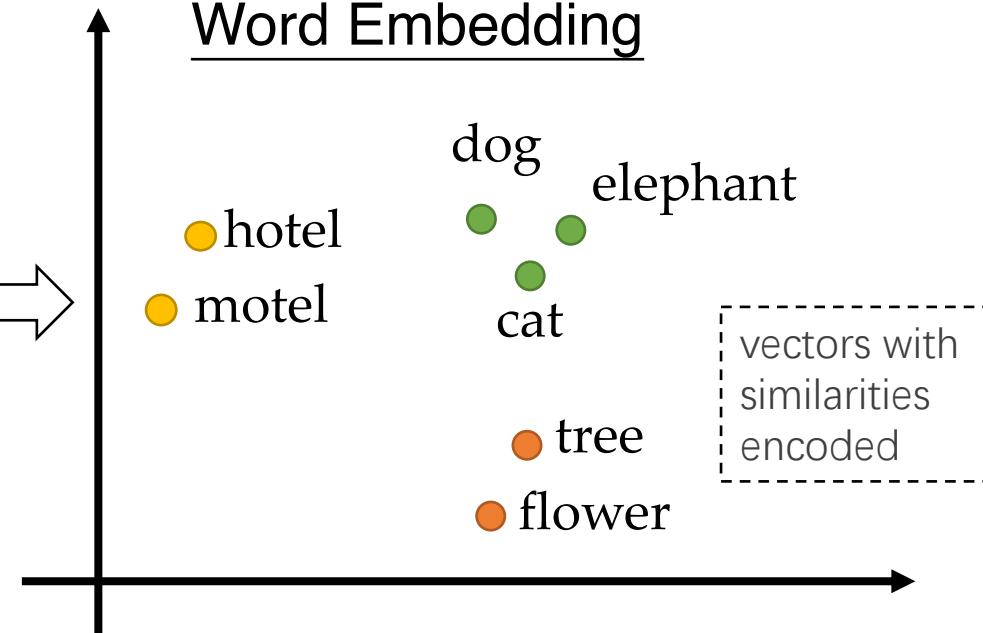
flower = [ 0 1 0 0 0 ]

business = [ 0 0 1 0 0 ]

motel = [ 0 0 0 1 0 ]

elephant = [ 0 0 0 0 1 ]

## Word Embedding

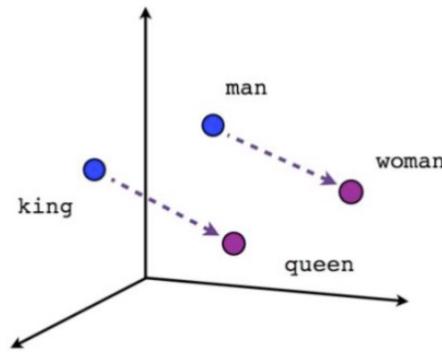


**Note:** word vectors are sometimes called **word embeddings** or **word representations**. They are **distributed** representations.

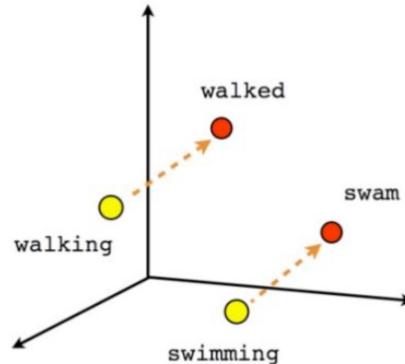
# Why Word Embeddings?



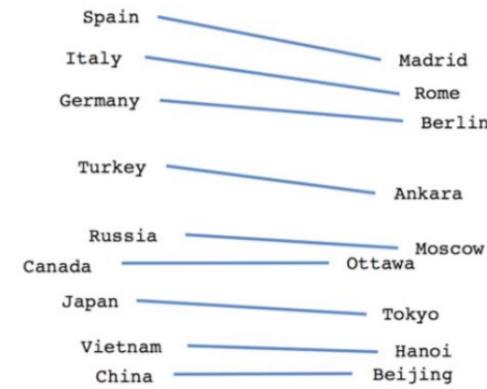
- Can capture the **rich** relational structure of the lexicon.



Male-Female



Verb tense



Country-Capital



# How to obtain word embeddings?

# How to obtain word embeddings?



- A word can be understood by its context

“A bottle of ***tesgüino*** is on the table”

“Everybody likes ***tesgüino***”

“***Tesgüino*** makes you drunk”

“We make ***tesgüino*** out of corn”



*“You shall know  
a word by the  
company it keeps”*  
(J. R. Firth 1957)

- From context words we can guess ***tesgüino*** means an alcoholic beverage such as beer.
- Intuition for an algorithm:



Two words are similar if they have similar word contexts

# How to Exploit the Context?



## Counting-based Approach

two words have similar vectors if they frequently **co-occur** in the same context (sentence, document, etc.).

e.g. Glove:

## Prediction-based Approach

train **neural networks** to predict a word (vector) given its context words (vector).

e.g., word2vec



# Counting based: the vector space model

# Vector Space Model



- The cornerstone technology in **information retrieval**.
- Term-Document Matrix (词项-文档矩阵)

	$d_1$	$d_2$	$d_3$	$d_4$	$d_5$
ekonomi	0	1	40	38	1
pusing	4	5	1	3	30
keuangan	1	2	30	25	2
sakit	4	6	0	4	25
Inflasi	8	1	15	14	1

the count of word t  
in document d

vector of  $d_3 = [40, 1, 30, 0, 15]$

- Two documents are similar if they have similar vectors!

$$d_3 = [40, 1, 30, 0, 15]$$

$$d_4 = [38, 3, 25, 4, 14]$$

# Vector Space Model



- Term-Document Matrix

	$d_1$	$d_2$	$d_3$	$d_4$	$d_5$
ekonomi	0	1	40	38	1
pusing	4	5	1	3	30
keuangan	1	2	30	25	2
sakit	4	6	0	4	25
Inflasi	8	1	15	14	1

vector of “sakit” =  
[4, 6, 0, 4, 25]

- Two words are similar if they have similar vectors!

$$\text{pusing} = [4, 5, 1, 3, 30]$$

$$\text{sakit} = [4, 6, 0, 4, 25]$$

# Vector Space Model



- **Weighting:** in practice, we usually use weights such as **TF-IDF** (逆文档词频), instead of simply **raw counts** (only TF).

$$\text{tf-idf}_{w,d} = \text{tf}_{w,d} \times \log(N / \text{df}_w)$$

$\text{tf}_{w,d}$  = frequency of  $w$  in  $d$

$\text{df}_w$  = number of documents containing  $w$

$N$  = total number of documents

	$d_1$	$d_2$	$d_3$	$d_4$	$d_5$
ekonomi	0	1	40	38	1
pusing	4	5	1	3	30
keuangan	1	2	30	25	2
sakit	4	6	0	4	25
Inflasi	8	1	15	14	1

$$\text{TF}(\text{sakit}) = [4, 6, 0, 4, 25]$$

$$\begin{aligned} \text{DF}(\text{sakit}) &= 4 \\ N &= 5 \end{aligned} \quad \left. \right\} \quad \text{IDF}(\text{sakit}) = \log(5/4)$$

$$\text{TF-IDF}(\text{sakit}) = [\dots\dots]$$

# Limitations

---



- TF-IDF vectors are
  - **long** (length  $|V| = 20,000$  to  $50,000$ )
  - **sparse** (most elements are zero)
  
- difficult to use as features in machine learning (more weights to tune)
- storing explicit counts can be difficult for generalization



# Prediction based: word2vec

# Word2Vec: Overview



- **Word2vec** (Mikolov et al. 2013) is a framework for learning word vectors.

## Idea:

- we have a large corpus of text.
- every word in a fixed vocabulary is assigned with a **learnable vector**;
- go through each position  $t$  in the text, which has a center word  $c$  and context (“outside”) words  $o$ ;
- calculate the **probability** of  $o$  given  $c$  (or vice versa) using their vector **similarities**;
- keep **updating** the word vectors to maximize this probability.

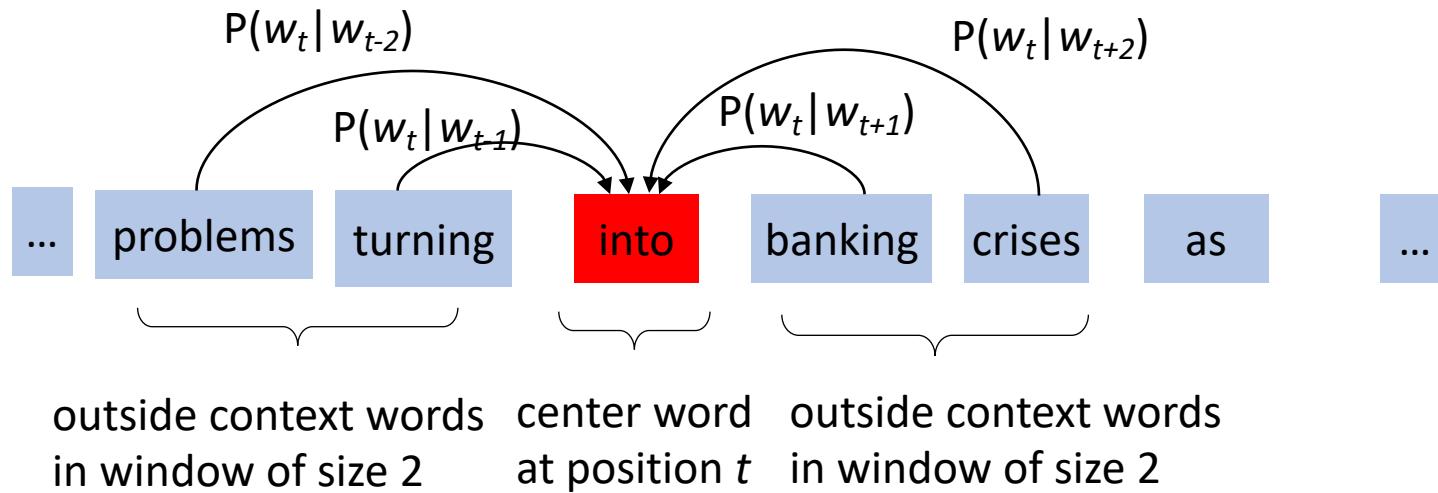


Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality. NIPS 2013.

# Word2Vec: Overview



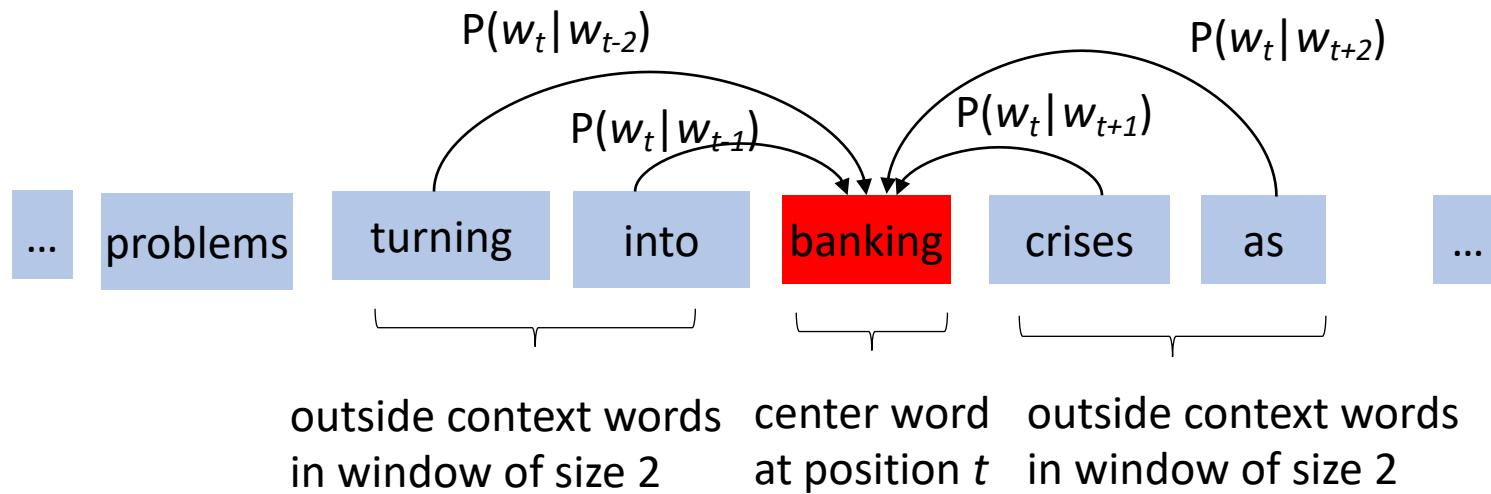
Example: window and process for computing  $P(w_t | w_{t+j})$



# Word2Vec: Overview



Example: window and process for computing  $P(w_t | w_{t+j})$



# Mikolov's CBOW

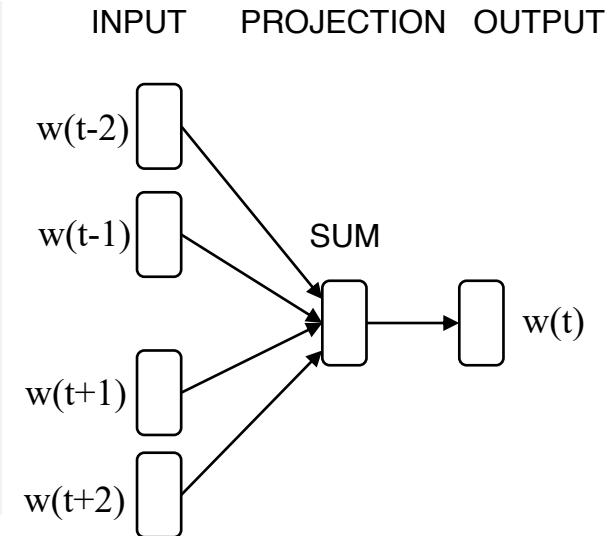


- **CBOW**: the distributed representations of context (or surrounding words) are combined to **predict the word in the middle**.

$$P(w_t | w_{t-k}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+k}) = \text{softmax} (\mathbf{NN} (V(w_{t-k}) + \dots + V(w_{t+k})))$$

This can be represented as a **neural network**:

- An **input layer** which converts each word (one-hot) into a dense vector.
- A **projection layer** which combines the vectors of input words.
- An **output layer** which predicts the target word  $w_t$  given the combined context vector.



**CBOW = Continuous-Bag-of-Words**

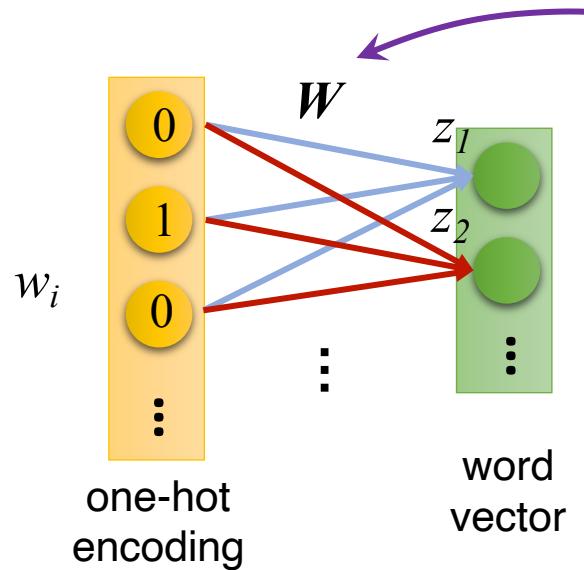
the order of words in the context does not influence the projection.

# Model Architecture

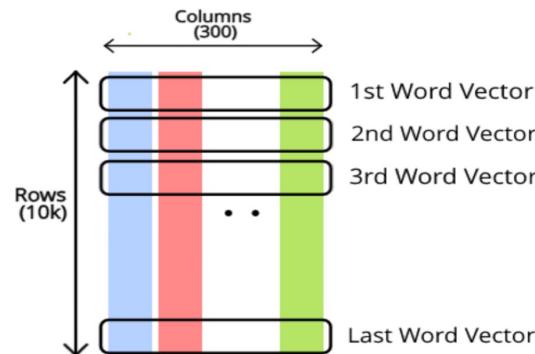


- Input Layer: represent any word into a vector.

$$z_i = \mathbf{W}x_i = \mathbf{W}_i$$



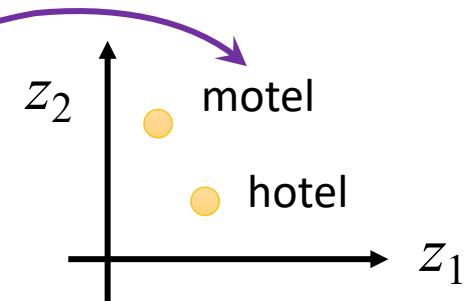
The weight matrix  $\mathbf{W} \in R^{|V| \times d}$  is a **lookup table** with each row  $\mathbf{W}_i$  being the vector for word  $w_i$ .



## Example

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

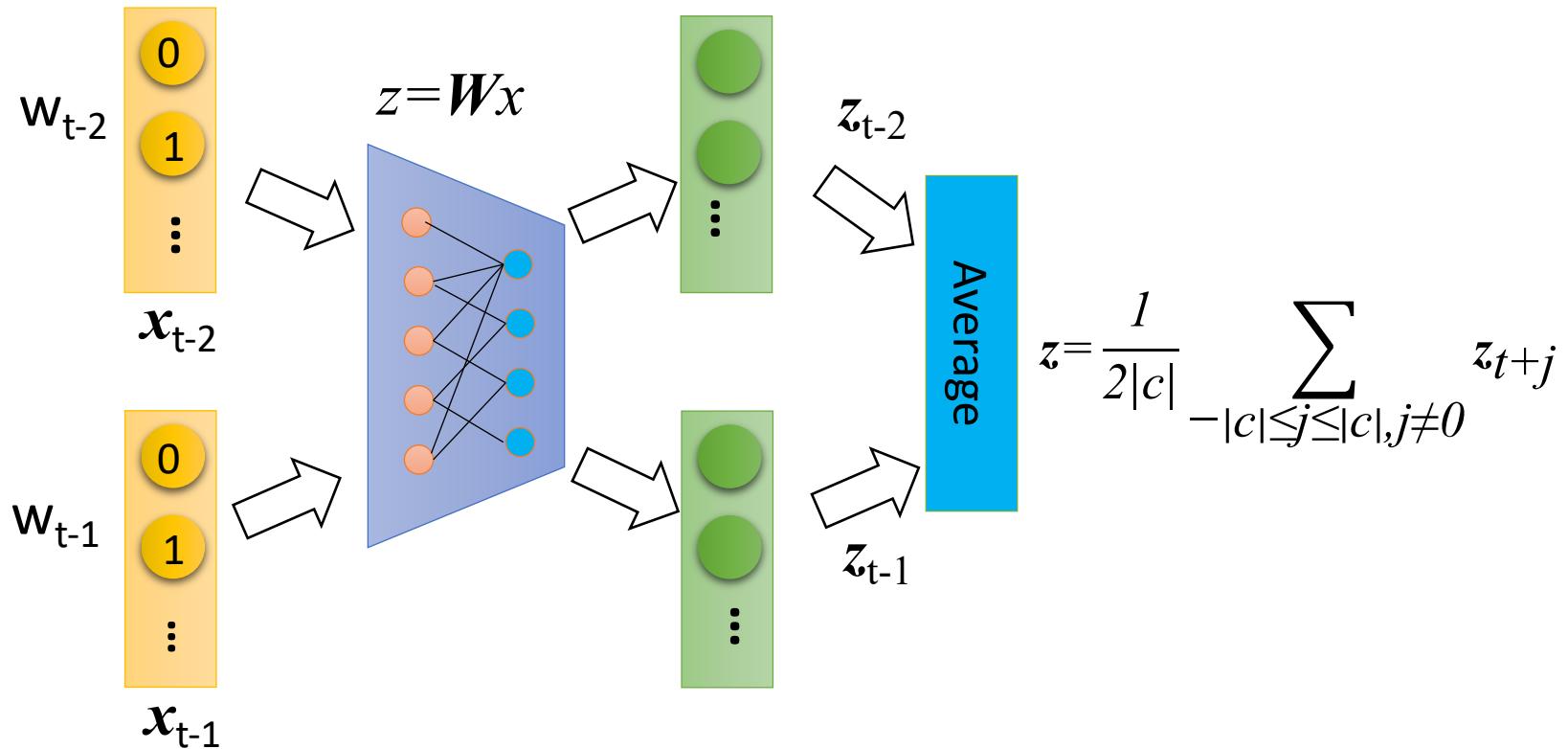
"motel"



# Architecture



- Projection Layer: combines context vectors into one vector.

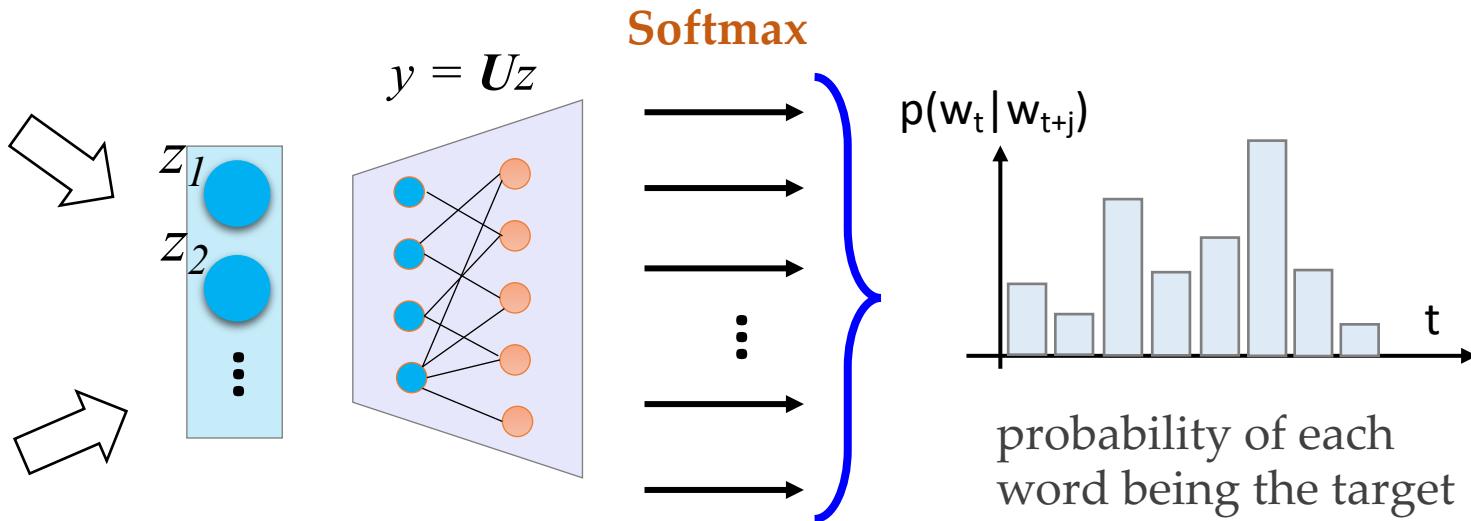


# Architecture

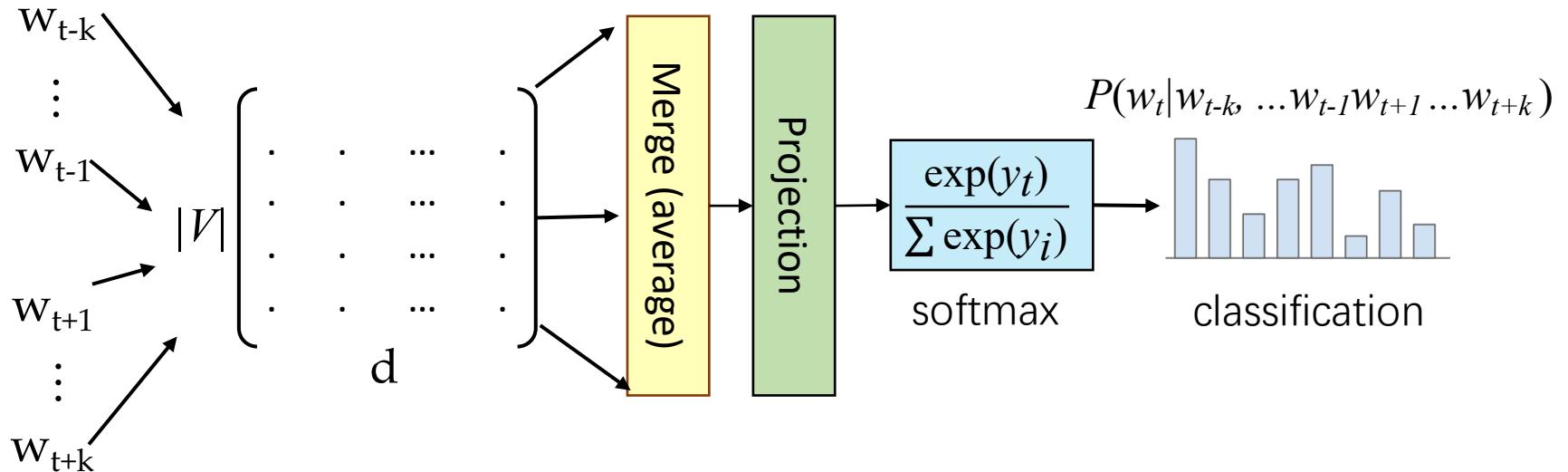


- Output Layer: predicts the probability of the target word.

$$P(w_t | w_{t-|c|}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+|c|}) = \frac{\exp(y_t)}{\sum_{i=1}^{|V|} \exp(y_i)}$$



# The Big Picture



# Training



- Given  $D = \{w_1, w_2, \dots, w_N\}$ , minimize the **negative log likelihood** (NLL) loss function:

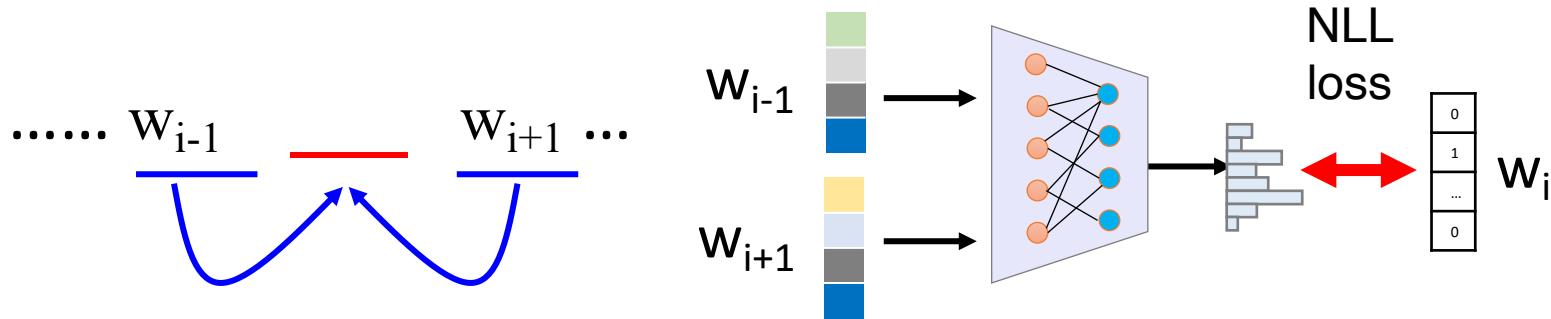
$$L(W, U | D) = -\frac{1}{N} \sum_{t=1}^N \log p(w_t | w_{t-k}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+k})$$

using gradient descend.

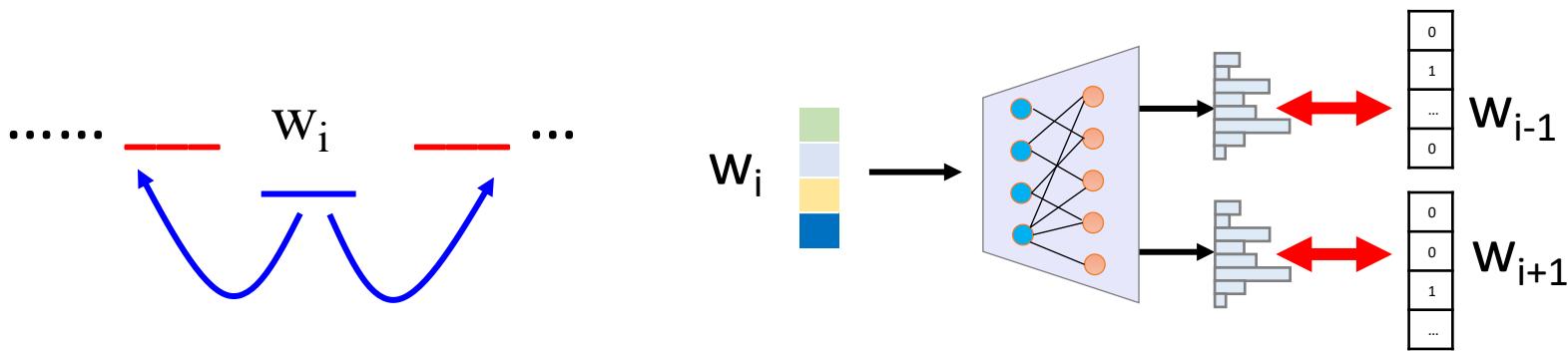
# Other Models



- CBOW predict the word given its context



- Skip-gram predict the context given a word



# The Skip-Gram Model

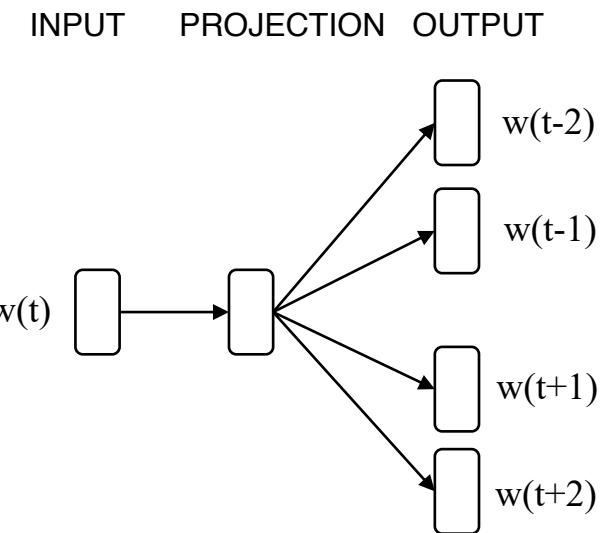


- We seek a model for  $P(w_{t+j} | w_t)$ .

$$P(w_{t+j} | w_t) = \frac{\exp(y_{t+j})}{\sum_{i=1}^{|V|} \exp(y_i)}$$

$$y = Uz$$

$$z = Wx$$

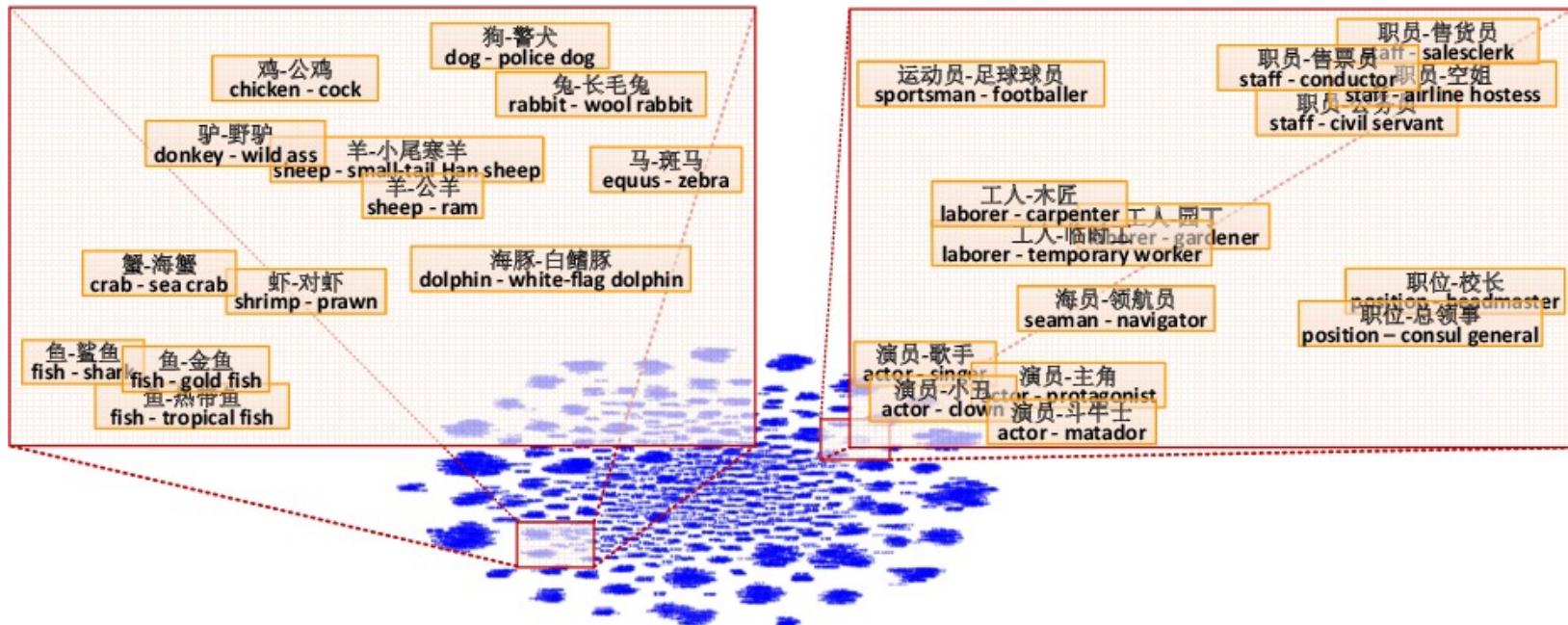


$$L(W, U | \chi) = -\frac{1}{N} \sum_{t=1}^N \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$$

# Example



- Word2vec maximizes an objective function by putting similar words nearby in space.



Fu, Ruiji, et al. "Learning semantic hierarchies via word embeddings." *ACL 2014*.

# The Word Analogy Task



- Word Analogy

a:b :: c:?

man:woman :: king:?

## Examples

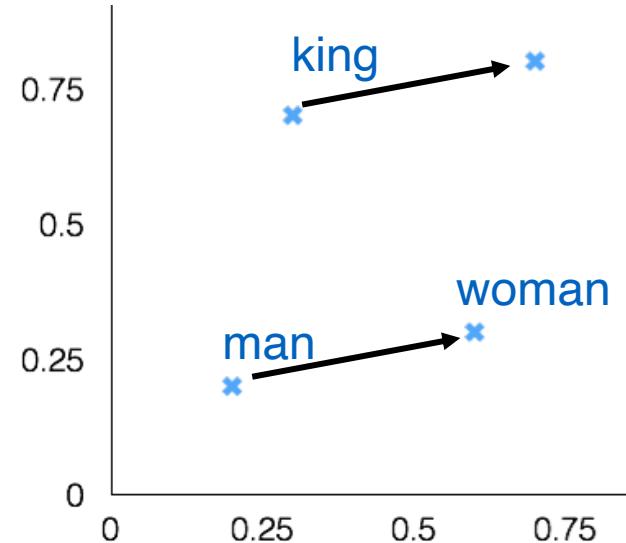
- Man is to Woman as King is to \_\_ ?
- Good is to Best as Smart is to \_\_ ?
- China is to Beijing as America is to \_\_ ?

How to find d ?

$$d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

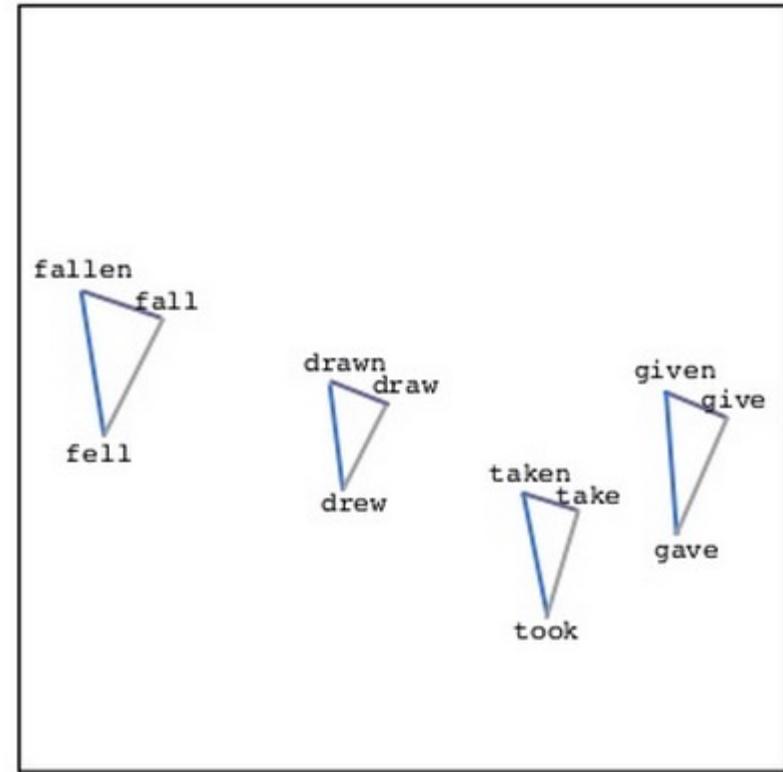
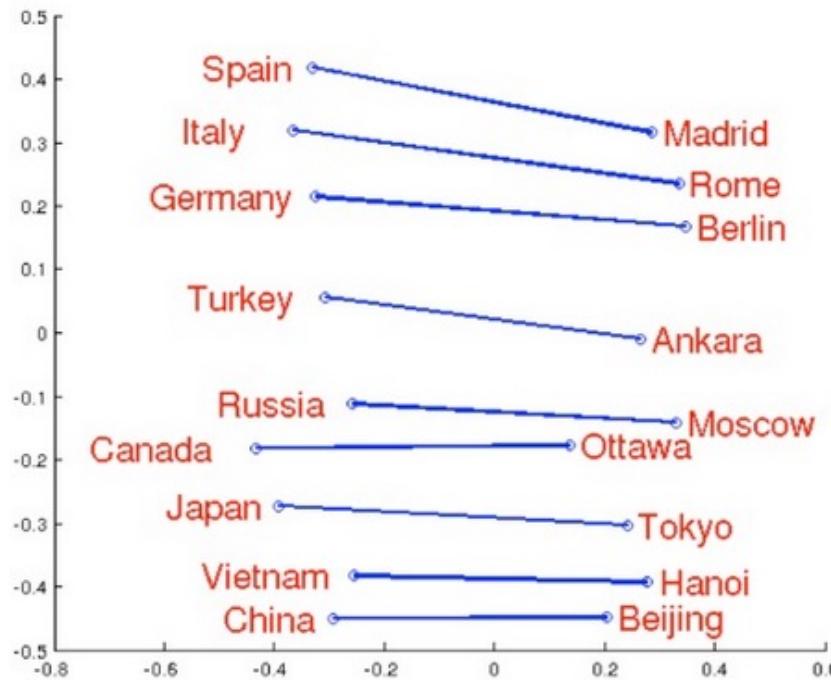
- It turns out that word2vec is good for such an analogy task.

$$V_{\text{king}} - V_{\text{man}} + V_{\text{woman}} = V_{\text{queen}}$$



Mikolov, T., et al. Distributed Representations of Words and Phrases and Their Compositionality. NIPS 2013.

# Example

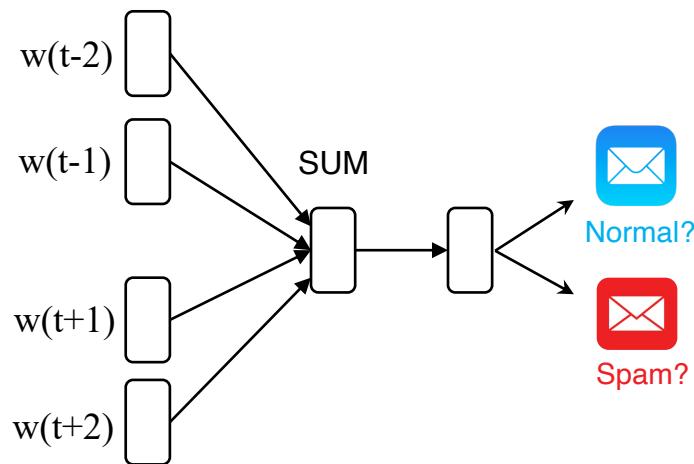


Source: <http://www.slideshare.net/hustwj/cikm-keynotenov2014>

# Thinking



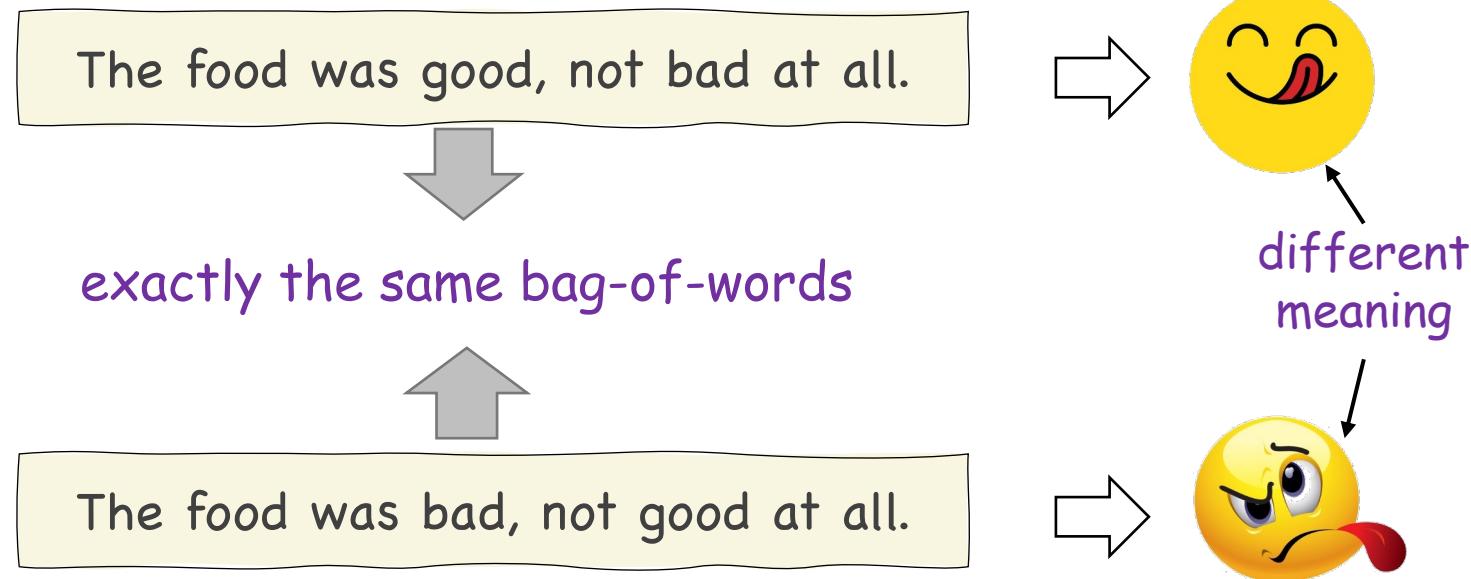
- Is word2vec supervised or unsupervised learning ?
- What limitations does word2vec have if we use it for text classification ?



# Beyond Bag-of-Words



- To understand the meaning of a sentence, the **order** of the words can not be ignored.



Paragraph Vector: Le, Quoc, and Tomas Mikolov. "Distributed Representations of Sentences and Documents." ICML, 2014  
Seq2seq Auto-encoder: Li, Jiwei, Minh-Thang Luong, and Dan Jurafsky. "A hierarchical neural autoencoder for paragraphs and documents." arXiv preprint, 2015

Skip Thought: Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, Sanja Fidler, "Skip Thought Vectors" arXiv preprint, 2015.

# Sequence in the Wild



# Sequence in the Wild



- Audio:



- Word:

自然, 语言

- Sentence:

我爱自然语言处理

- Program:

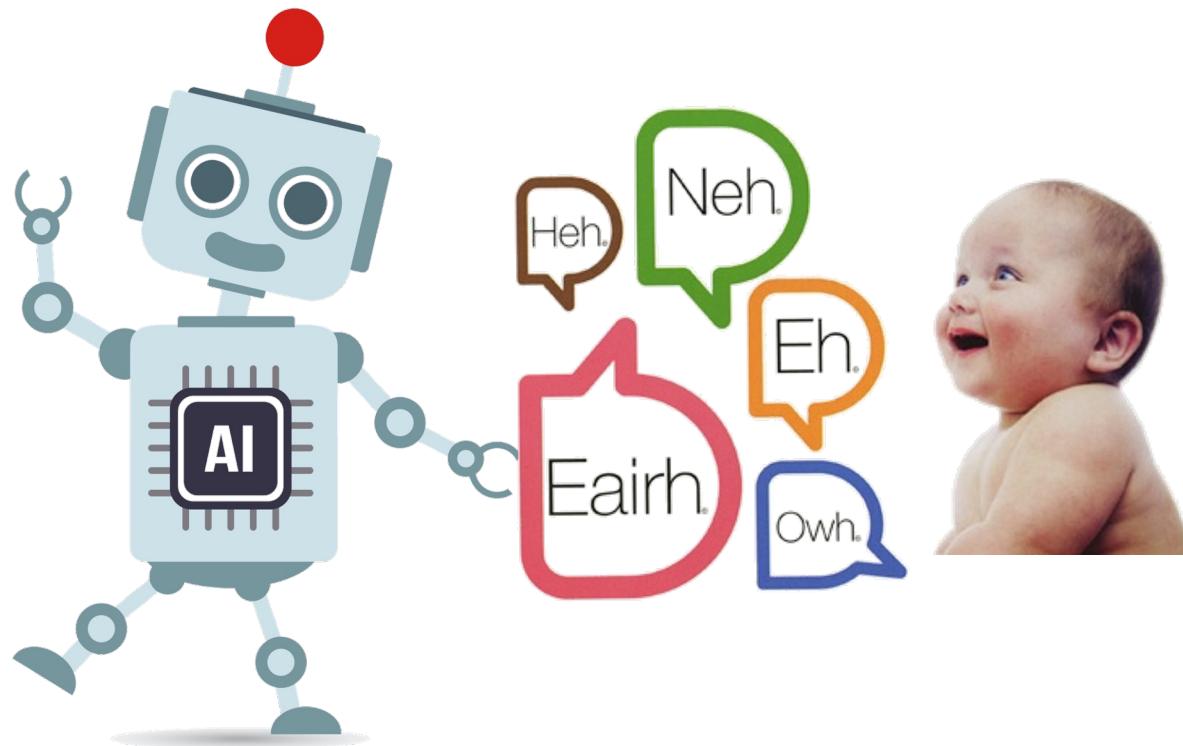
```
for (int i = 0; i < 10 ; i ++ ) {
```

...

# How can computers talk?



- Teach computers to talk like a baby?

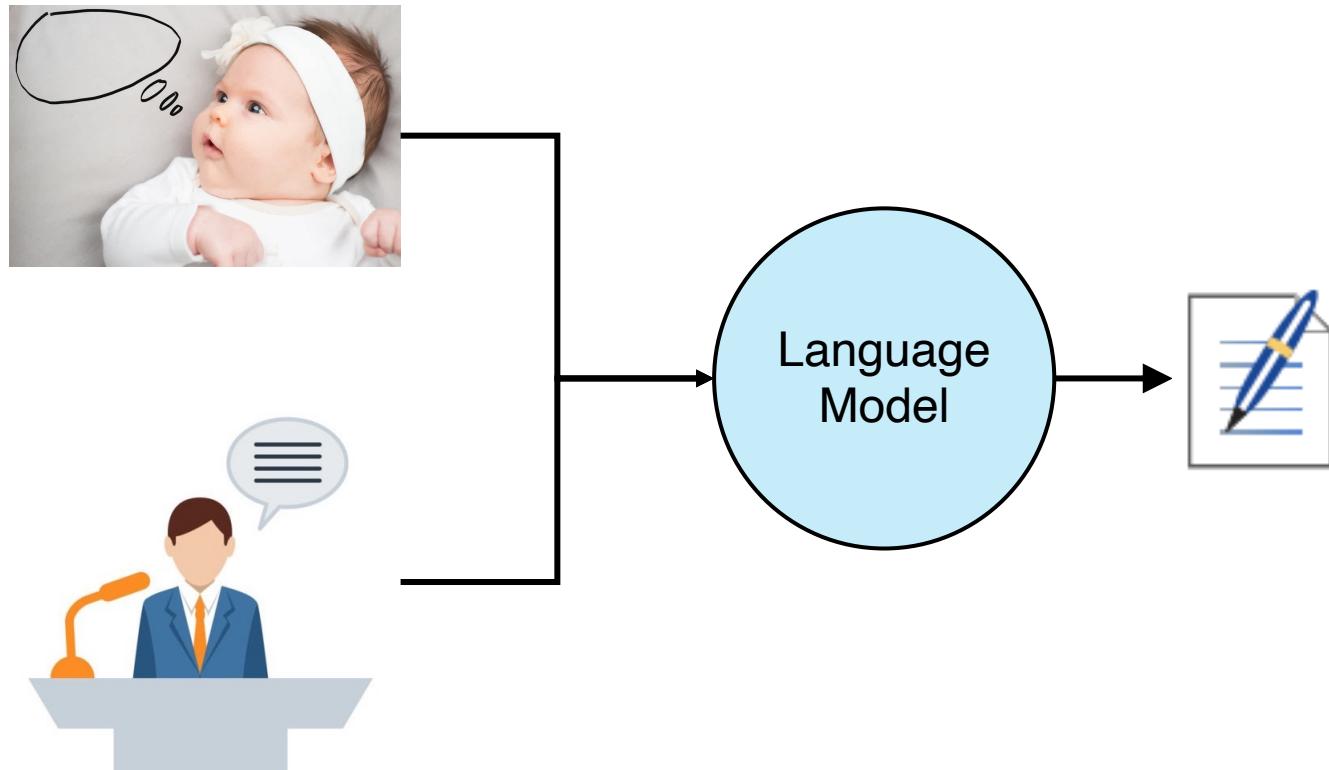


How to measure the linguistic ability of a computer?

# Language Model



- How proficiency do you acquire a language?



# Language Model



- A probabilistic model of **how likely** a given string appear in a given “**language**”.
- For any sequence  $x = (w_1, w_2, \dots, w_N)$ , a **language model** can be defined as:

$$p(x) = p(w_1, w_2, \dots, w_N)$$

## Example:

$P_1=P(\text{“他明天去上课”})$

$P_2=P(\text{“明天他将要去上课”})$

$P_3=P(\text{“明儿个他上课”})$

$P_4=P(\text{“依明朝得返课啦, 唔好迟到啊嘅”})$

Chinese:  $P_1 > P_2 > P_3 > P_4$

他明天将去上\_\_?

## • Applications:

message suggestion; document generation; spelling correction; machine translation; speech recognition;...

# Language Model



- What is the probability of  $P(w_1, \dots, w_N)$ ?

$$p(\text{我爱机器学习}) = ?$$

- Chain Rule:

$$p(w_1, \dots, w_N) = p(w_1)p(w_2 | w_1) \dots p(w_N | w_1, \dots, w_{N-1})$$

$$p(\text{我爱机器学习}) = p(\text{我})p(\text{爱} | \text{我})p(\text{机} | \text{我爱})p(\text{器} | \text{我爱}) \\ p(\text{机})p(\text{学} | \text{我爱机器})p(\text{习} | \text{我爱机器学})$$

- Markov Assumption: (only consider the last  $n-1$  words)

$$p(w_i | w_1, \dots, w_{i-1}) = p(w_i | w_{i-n+1}, \dots, w_{i-1})$$

$$p(\text{习} | \text{我爱机器学}) \approx p(\text{习} | \text{机器学}) \approx p(\text{习} | \text{学})$$

# Bigram Language Model



So that's what we get for n=2:

$$p(w) \approx p(w_1)p(w_2|w_1)\dots p(w_N|w_{N-1})$$

$$1/18 \times 1/8 \times 1/120 \times 1/4 \times 1/420 \times 1/2$$

$p(\text{我爱机器学习}) = p(\text{我})p(\text{爱}|\text{我})p(\text{机}|\text{爱})p(\text{器}|\text{机})p(\text{学}|\text{器})p(\text{习}|\text{学})$

Q: How to estimate these probabilities?



A: Straightforward counting.

Q: But remember in word embedding, counting discrete words have many drawbacks?

A: Don't worry, we have neural networks as language models.

# What's Next?

WHAT'S  
NEXT?

## Recurrent Neural Networks

- A deep neural network for sequence (language) modeling.

