



香港中文大學(深圳)  
The Chinese University of Hong Kong, Shenzhen

**CSC6052/5051/4100/DDA6307/  
MDS5110**

**Natural Language Processing**

**Lecture 3-1: Word Vectors**

Spring 2025  
Benyou Wang  
School of Data Science

before the lecture

# DeepSeek and Spring Festival



Founder of DeepSeek, Wenfeng Liang becomes famous (年轻人的偶像)

# For DeepSeek

## DeepSeek V3

- MLA and fine-grained experts for MoE (old DeepSeek also has this)
- auxiliary-loss-free strategy for load balancing and a multi-token prediction training objective
- FP8 training mixed precision (previously FP16/BF16) and many other optimization of training

## DeepSeek R1

- New way to achieve o1-level reasoning (without supervised finetuning): 1) DeepSeek-R1-Zero, a model trained via large-scale reinforcement learning (RL) without supervised fine-tuning (SFT) as a preliminary step; plus DeepSeek-R1 with multi-stage training and cold-start data before RL

<https://arxiv.org/html/2412.19437v1> DeepSeek-V3 Technical Report

<https://arxiv.org/abs/2501.12948> DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

# Remarks

- In total **6M \$** for training budget (1/30 of GPT-4o)
  - Reaching **GPT-4o and O1 –level** performance
  - DeepSeek App ranks first in both China and US APP store. Breaking APP downloading record (previously set by ChatGPT)
  - Every big guys in US came out to talk about DeepSeek, negatively affecting the US stock
  - DeepSeek's AI breakthrough 'is biggest shock to come out of China in 185 years' 首都医科大学校长饶毅认为，在科学和技术相关的方面，185年来中国出现的对人类最大的震撼是DeepSeek
- 
- It is open-source and it could be deployed by Huawei TPU Devices (inference without NVIDIA)

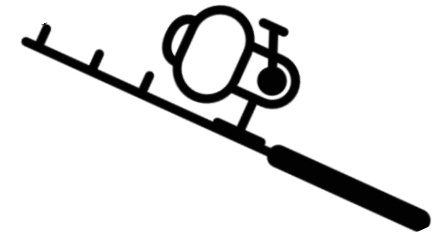
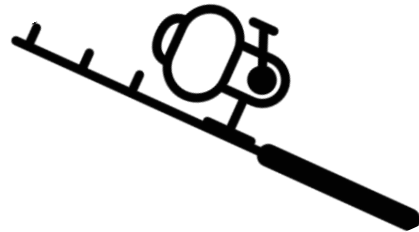
It is time for LLM applications

Prompting, agent and build you own APP **right now**

# Rethinking DeepSeek

- Budget is only about the final-round training, but this is a part of cost (exploring, testing and others are also costly).
- The training efficiency could be or was already achieved by other companies; but these companies did not claim from this aspect (Demis mentioned Google Gemini).
- Distilled from western models (Demis claimed)
- Applied existing technologies well (did not invent something new)

# An analogy of distillation



ChatGPT: Fishing from the sea to a pool

Others : directly fishing from such a small pool



We should not be always following,  
we should try to **lead**.



# Recap and overview

Relation between Word vectors and language modeling

# How do we represent words in NLP models?

- n-gram models

$$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^n P(w_i | w_{i-1})$$

$$P(w_i | w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha |V|}$$

Each word is just a string or indices  $w_i$  in the vocabulary list

- Naive Bayes

$$\hat{P}(w_i | c_j) = \frac{\text{Count}(w_i, c_j) + \alpha}{\sum_{w \in V} \text{Count}(w, c_j) + \alpha |V|}$$

# How do we represent words in NLP models?

- Logistic regression

	Var	Definition	Value in Fig. 5.2
	$x_1$	count(positive lexicon) $\in$ doc)	3
	$x_2$	count(negative lexicon) $\in$ doc)	2
string match	$x_3$	$\begin{cases} 1 & \text{if "no" } \in \text{ doc} \\ 0 & \text{otherwise} \end{cases}$	1
	$x_4$	count(1st and 2nd pronouns $\in$ doc)	3
	$x_5$	$\begin{cases} 1 & \text{if "!" } \in \text{ doc} \\ 0 & \text{otherwise} \end{cases}$	0
	$x_6$	log(word count of doc)	$\ln(64) = 4.15$

# Why word meaning in NLP models?

- With words, a feature is a word identity (= string)
  - Feature 5: `The previous word was “terrible”
  - Requires **exact same word** to be in the training and testing set

“terrible” ≠ “horrible”

- If we can represent word meaning in vectors:
  - The previous word was vector [35, 22, 17, ...]
  - Now in the test set we might see a similar vector [34, 21, 14, ...]
  - We can generalize to **similar but unseen** words!!!

# What do words mean?

- **Synonyms:** couch/sofa, car/automobile, filbert/hazelnut
- **Antonyms:** dark/light, rise/fall, up/down
- Some words are not synonyms but they share some element of meaning
  - cat/dog, car/bicycle, cow/horse
- Some words are not similar but they are **related**
  - coffee/cup, house/door, chef/menu
- **Affective meanings** or **connotations:**

**valence:** the pleasantness of the stimulus

**arousal:** the intensity of emotion provoked by the stimulus

**dominance:** the degree of control exerted by the stimulus

vanish	disappear	9.8
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

SimLex-999

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24

(Osgood et al., 1957)

# Lexical resources

## WordNet Search - 3.1

- [WordNet home page](#) - [Glossary](#) - [Help](#)

Word to search for:

Display Options:

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations  
Display options for sense: (gloss) "an example sentence"

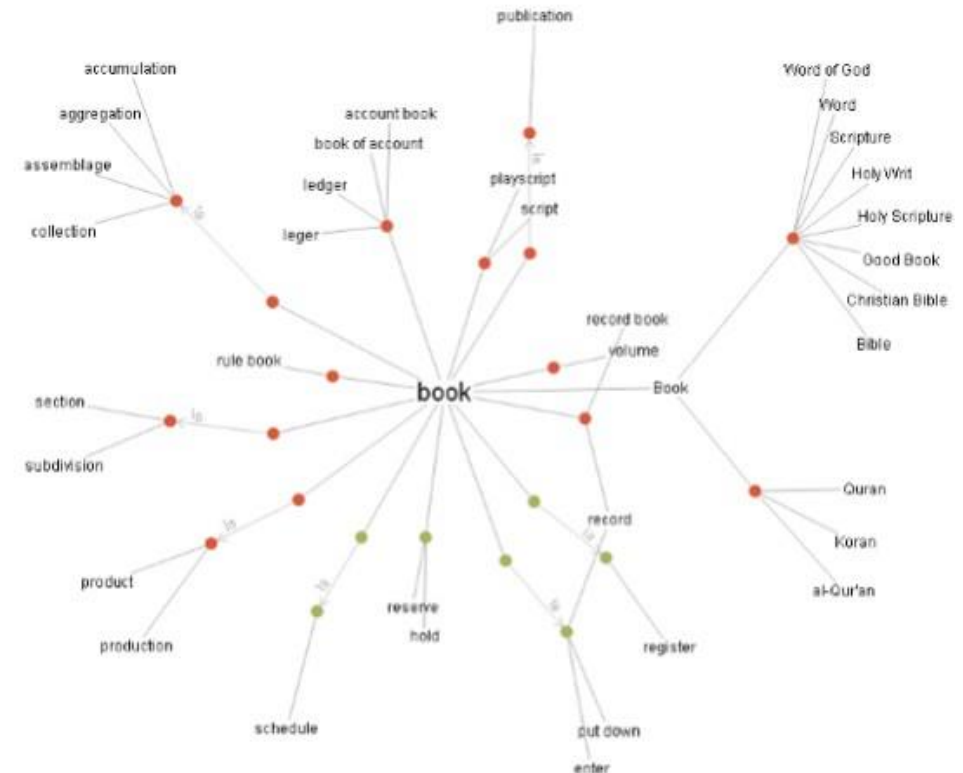
### Noun

- **S: (n) mouse** (any of numerous small rodents typically resembling diminutive rats having pointed snouts and small ears on elongated bodies with slender usually hairless tails)
- **S: (n) shiner, black eye, mouse** (a swollen bruise caused by a blow to the eye)
- **S: (n) mouse** (person who is quiet or timid)
- **S: (n) mouse, computer mouse** (a hand-operated electronic device that controls the coordinates of a cursor on your computer screen as you move it around on a pad; on the bottom of the device is a ball that rolls on the surface of the pad) *"a mouse takes much more room than a trackball"*

### Verb

- **S: (v) sneak, mouse, creep, pussyfoot** (to go stealthily or furtively) *"..stead of sneaking around spying on the neighbor's house"*
- **S: (v) mouse** (manipulate the mouse of a computer)

<http://wordnetweb.princeton.edu/>



(-) Huge amounts of human labor to create and maintain

# The big idea: model of meaning focusing on similarity

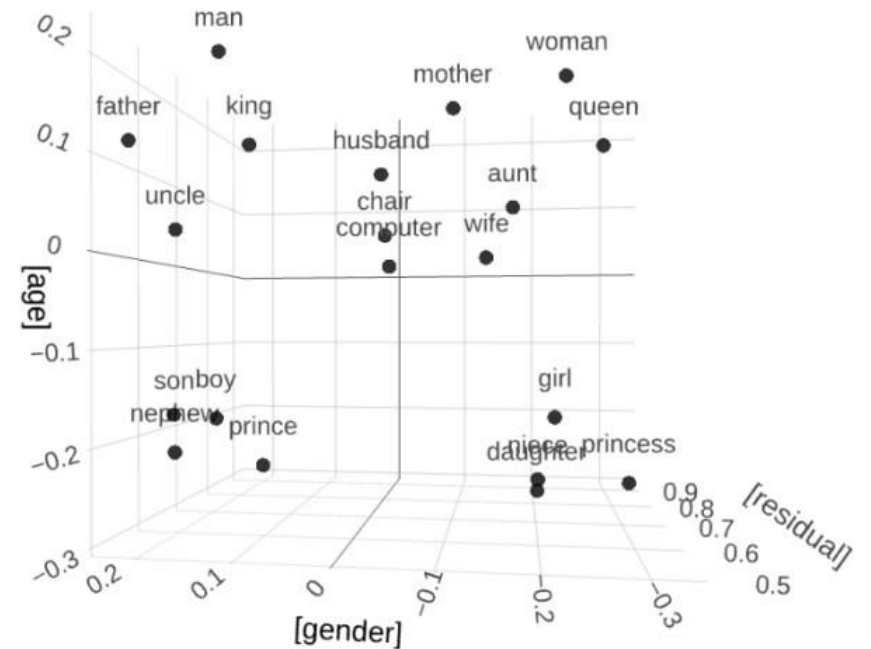
$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix}$$

$$v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix}$$

$$v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

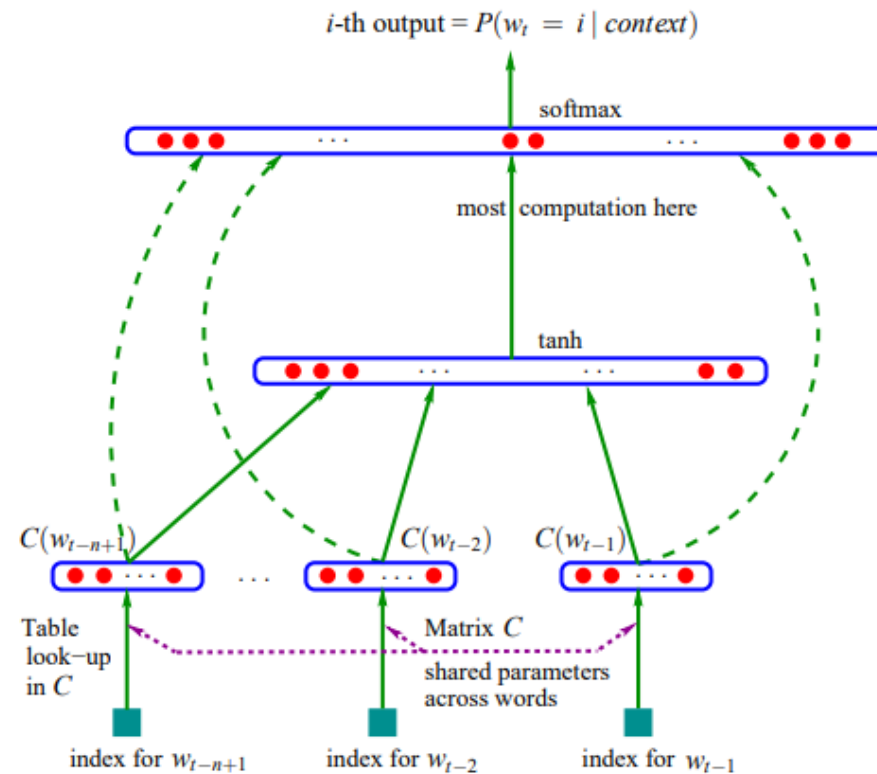
Similar words are “**nearby in the vector space**”



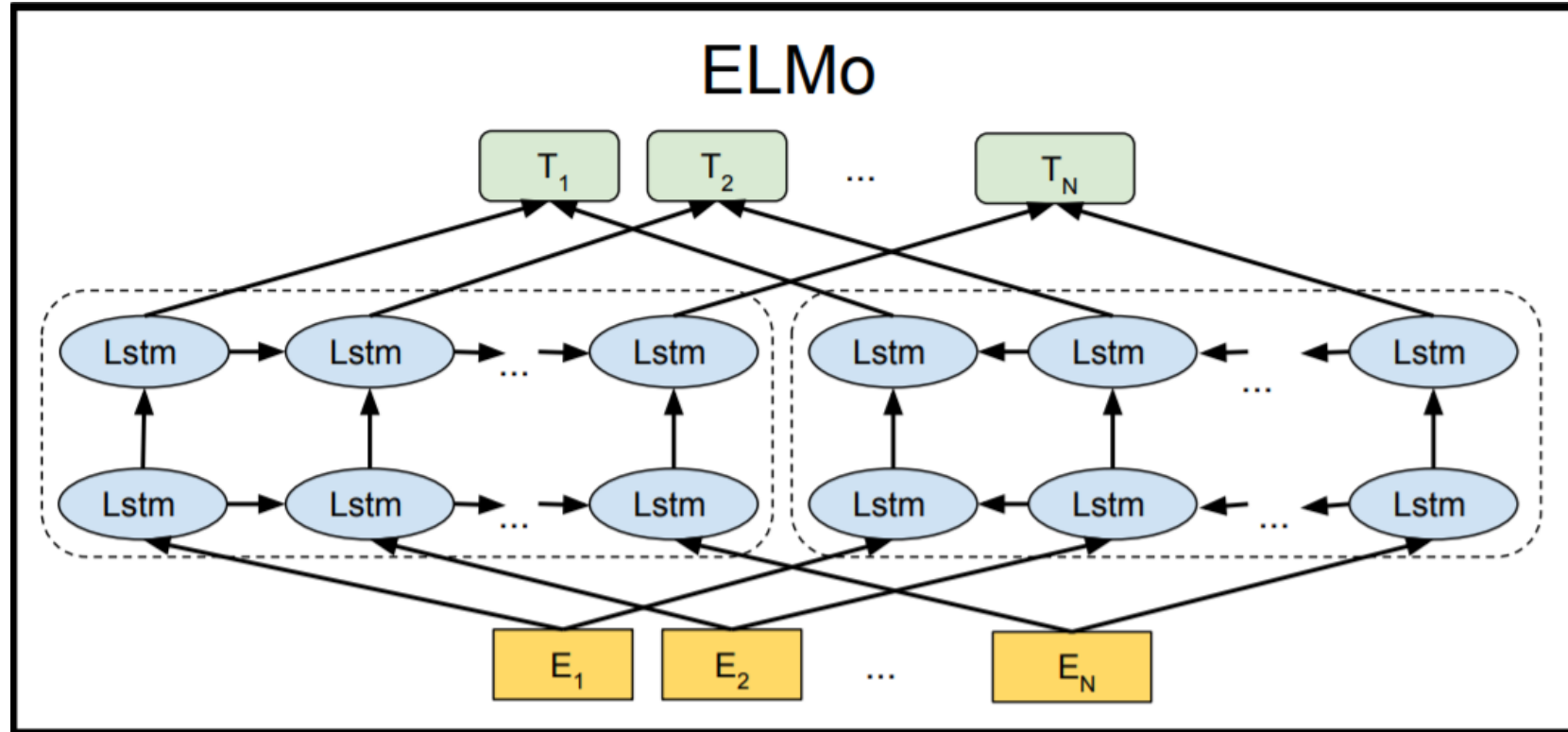
(Bandyopadhyay et al. 2022)



# Learning LMs via neural network brings word embedding



# To contextualized word vectors using LMs



Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer. Deep contextualized word representations. <https://arxiv.org/abs/1802.05365>.

# Language models: Narrow Sense

A probabilistic model that assigns a probability to every finite sequence (grammatical or not)

Sentence: "the cat sat on the mat"

$$P(\text{the cat sat on the mat}) = P(\text{the}) * P(\text{cat}|\text{the}) * P(\text{sat}|\text{the cat}) \\ * P(\text{on}|\text{the cat sat}) * P(\text{the}|\text{the cat sat on}) \\ * P(\text{mat}|\text{the cat sat on the})$$

Implicit order

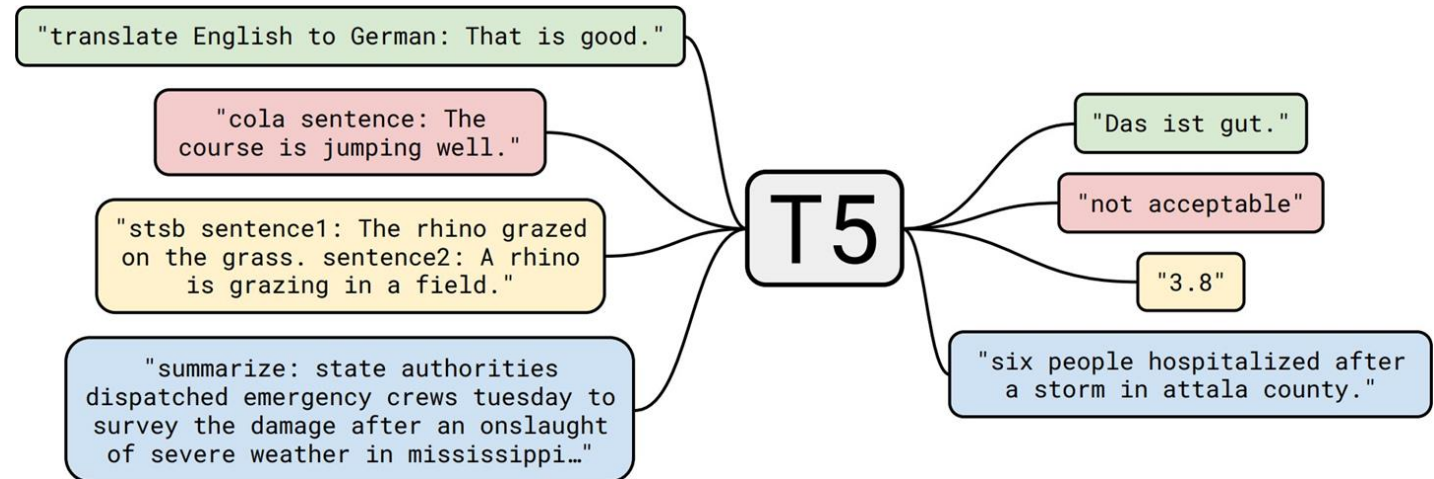
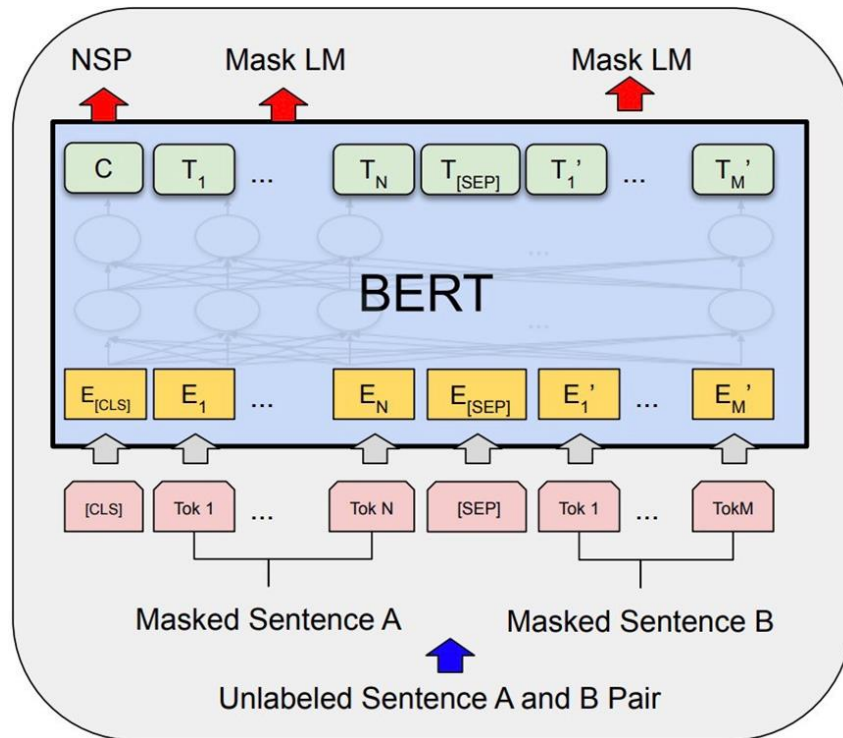


GPT-3 still acts in this way but the model is implemented as a very large neural network of 175-billion parameters!

# Language models: Broad Sense

- ❖ Decoder-only models (GPT-x models)
- ❖ Encoder-only models (BERT, RoBERTa, ELECTRA)
- ❖ Encoder-decoder models (T5, BART)

The latter two usually involve a different **pre-training** objective.



# Contents

- **Motivations to word embedding/word vectors**
- **Word embedding and word vectors**
- **Some variants**
- **Evaluations**

# N-gram Language Models

*the students opened their*

- **Question:** How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn an *n*-gram Language Model!
- **Definition:** An *n*-gram is a chunk of *n* consecutive words.
  - **uni**grams: “the”, “students”, “opened”, “their”
  - **bi**grams: “the students”, “students opened”, “opened their”
  - **tri**grams: “the students opened”, “students opened their”
  - **four**-grams: “the students opened their”
- **Idea:** Collect statistics about how frequent different n-grams are and use these to predict next word.

# N-gram Language Models

- First we make a **Markov assumption**:  $x^{(n)}$  depends only on the preceding  $n-1$  words

$$P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)}) = P(\mathbf{x}^{(t+1)} | \overbrace{\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}}^{n-1 \text{ words}}) \quad \text{(assumption)}$$

prob of an-gram  $\rightarrow$

$$= P(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})$$

prob of a (n-1)-gram  $\rightarrow$

$$P(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})$$

(definition of conditional prob)

- **Question:** How do we get these  $n$ -gram and  $(n-1)$ -gram probabilities?
- **Answer:** By **counting** them in some large corpus of text!

$$\approx \frac{\text{count}(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})}{\text{count}(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})} \quad \text{(statistical approximation)}$$

# N-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ students opened their —  
discard condition on this

$$P(\mathbf{w} | \text{students opened their}) = \frac{\text{count}(\text{students opened their } \mathbf{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- “students opened their” occurred 1000 times
- “students opened their **books**” occurred 400 times
  - $P(\text{books} | \text{students opened their}) = 0.4$
- “students opened their **exams**” occurred 100 times
  - $P(\text{exams} | \text{students opened their}) = 0.1$



# Sparsity Problems with n-gram Language Models

## Sparsity Problem 1

**Problem:** What if “*students opened their w*” never occurred in data? Then  $w$  has probability 0!

**(Partial) Solution:** Add small  $\delta$  to the count for every  $w \in V$ . This is called *smoothing*.

$$P(w|\text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

## Sparsity Problem 2

**Problem:** What if “*students opened their*” never occurred in data? Then we can’t calculate probability for *any*  $w$ !

**(Partial) Solution:** Just condition on “*opened their*” instead. This is called *backoff*.

**Note:** Increasing  $n$  makes sparsity problems worse. Typically, we can’t have  $n$  bigger than 5.

# Storage Problems with $n$ -gram Language Models

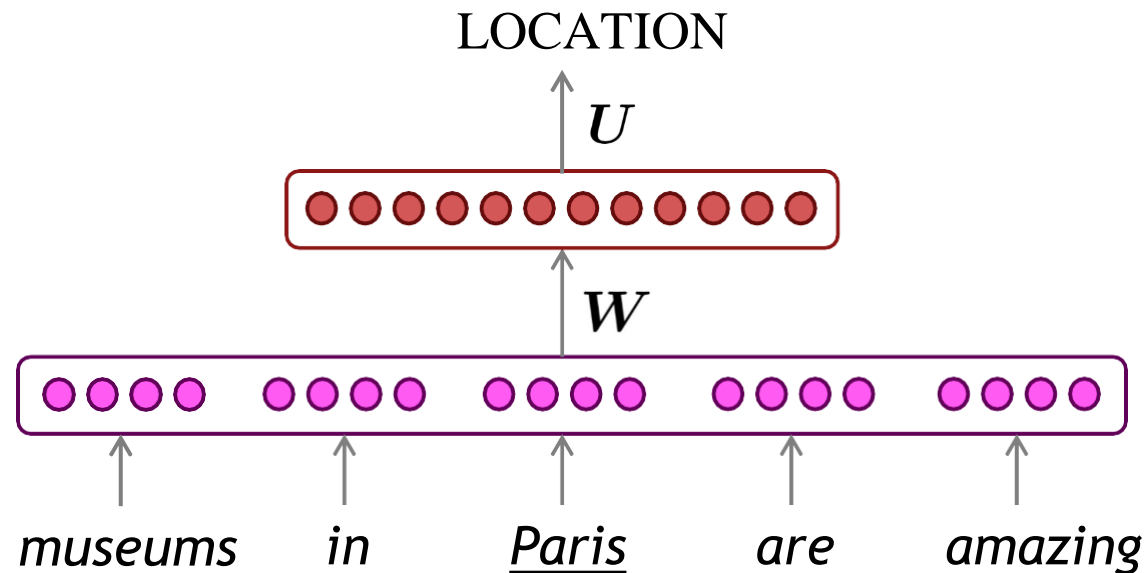
**Storage**: Need to store count for all  $n$ -grams you saw in the corpus.

$$P(\mathbf{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \mathbf{w})}{\text{count}(\text{students opened their})}$$

Increasing  $n$  or increasing corpus increases model size!

# How to build a *neural* language model?

- Recall the Language Modeling task:
  - Input: sequence of words  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}$
  - Output: prob. dist. of the next word  $P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})$
- How about a **window-based neural model**?
  - We saw this applied to Named Entity Recognition :



# A fixed-window neural Language Model

output distribution

$$\hat{y} = \text{softmax}(U\mathbf{h} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

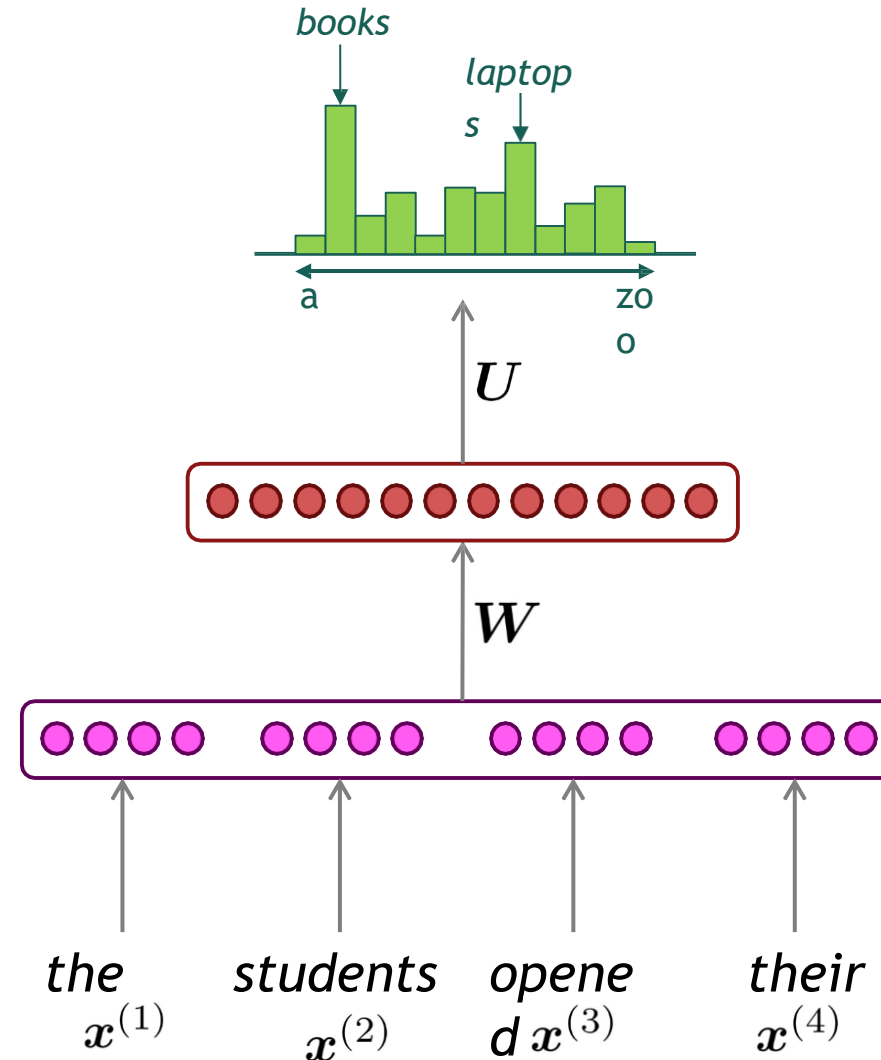
$$\mathbf{h} = f(\mathbf{W}\mathbf{e} + \mathbf{b}_1)$$

concatenated word embeddings

$$\mathbf{e} = [\mathbf{e}^{(1)}; \mathbf{e}^{(2)}; \mathbf{e}^{(3)}; \mathbf{e}^{(4)}]$$

words / one-hot vectors

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}$$



# A fixed-window neural Language Model

Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

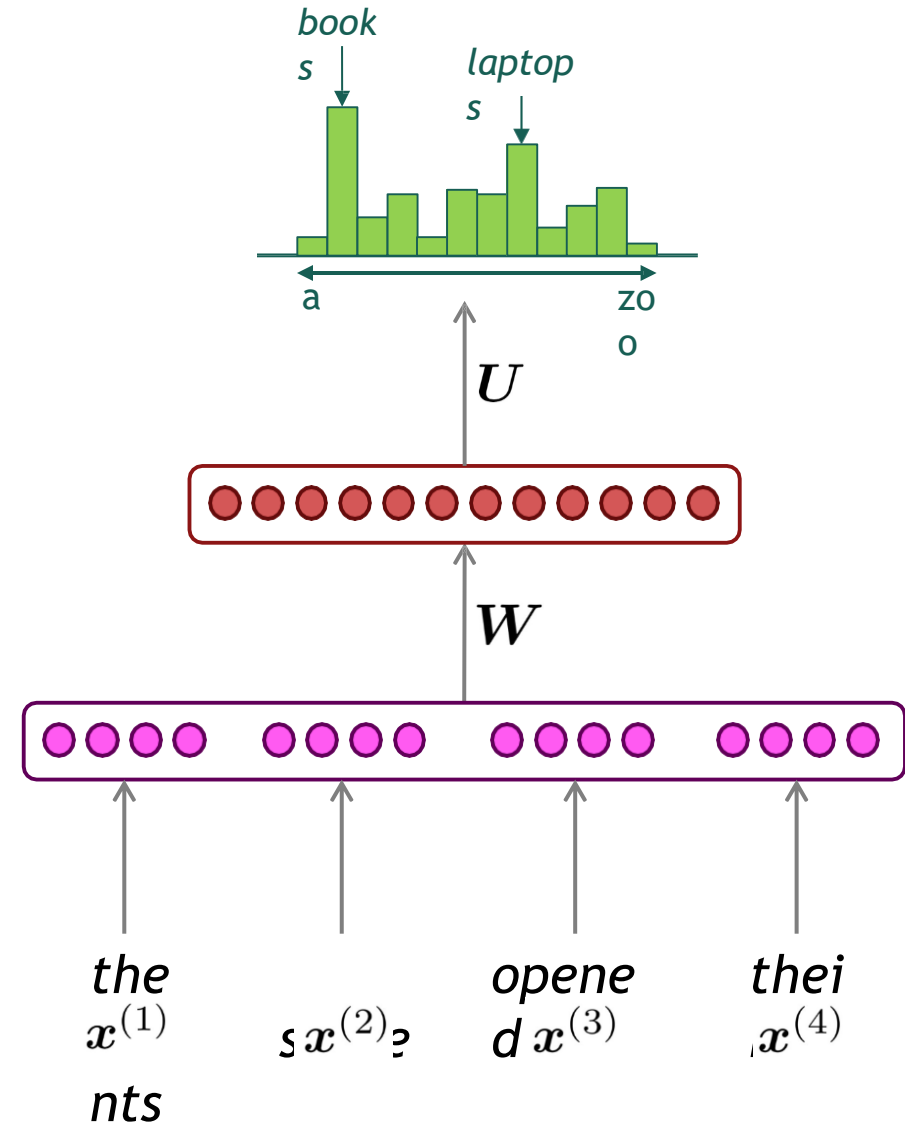
**Improvements** over  $n$ -gram LM:

- No sparsity problem
- Don't need to store all observed  $n$ -grams

Remaining **problems**:

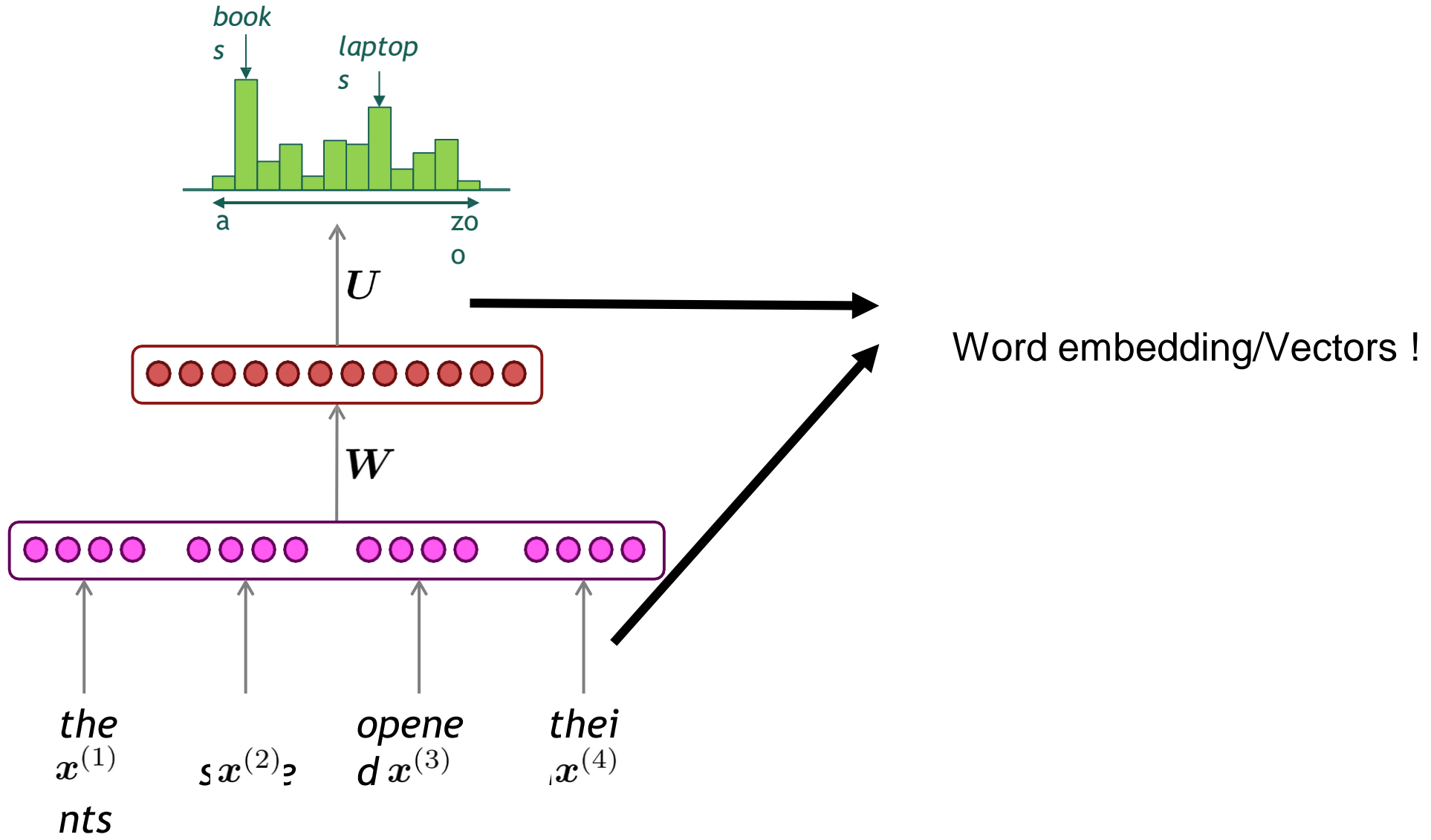
- Fixed window is **too small**
- Enlarging window enlarges  $W$
- Window can never be large enough!
- $x^{(1)}$  and  $x^{(n)}$  are multiplied by completely different weights in  $W$ . **No symmetry** in how the inputs are processed.

We need a neural architecture that can process *any length input*  
**Recurrent NN is the solution!**



# Word Embedding and Word vectors

# Byproducts of NNLM : word embedding






# How do we represent the meaning of a word?

Definition: meaning (Webster dictionary)

- ❑ the idea that is represented by a word, phrase, etc.
- ❑ the idea that a person wants to express by using words, signs, etc.
- ❑ the idea that is expressed in a work of writing, art, etc.

Commonest linguistic way of thinking of meaning:

- ❑ signifier (symbol)  $\Leftrightarrow$  signified (idea or thing)  
= denotational semantics
- ❑ Tree  $\Leftrightarrow$  { , , , ... }



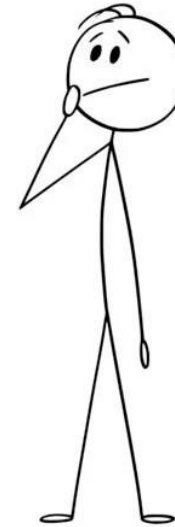
# Representing words as discrete symbols

- ❑ In traditional NLP, we regard words as discrete symbols:  
hotel, conference, motel – a localist representation
- ❑ Such symbols for words can be represented by one-hot vectors:  
motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]  
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
- ❑ Vector dimension = number of words in vocabulary (e.g., 500,000+)

These two vectors are orthogonal

**There is no natural notion of similarity for one-hot vectors!**

# Distributional hypothesis



- “The meaning of a word is its use in the language”
- “If A and B have almost identical environments we say that they are synonyms.”
- “You shall know a word by the company it keeps”

[Wittgenstein PI 43]

[Harris 1954]

[Firth 1957]

# Representing words by their context

Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**

- “*You shall know a word by the company it keeps*” (J. R. Firth 1957: 11)
- **One of the most successful ideas of modern statistical NLP!**
- When a word  $w$  appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of  $w$  to build up a representation of  $w$

...government debt problems turning into **banking** crises as happened in 2009...

...saying that Europe needs unified **banking** regulation to replace the hodgepodge...

...India has just given its **banking** system a shot in the arm...

These **context words** will represent **banking**

# Distributional hypothesis

“Ongchoi”

Ongchoi is delicious sautéed with garlic 蒜蓉Ongchoi味道鲜美

Ongchoi is superb over rice Ongchoi配米饭吃起来味道极佳

Ongchoi leaves with salty sauces Ongchoi叶配咸酱

# Distributional hypothesis

“Ongchoi”

Ongchoi is delicious sautéed with garlic

Ongchoi is superb over rice

Ongchoi leaves with salty sauces

Q: What do you think ‘Ongchoi’ means?

- A) a savory snack
- B) a green vegetable
- C) an alcoholic beverage
- D) a cooking sauce

# Distributional hypothesis

“Ongchoi”

Ongchoi is delicious sautéed with garlic

Ongchoi is superb over rice

Ongchoi leaves with salty sauces

You may have seen these sentences before:

spinach **sautéed with garlic over rice** chard stems

and **leaves** are **delicious** collard greens and other

**salty** leafy greens

Something similar to **Spinach** (菠菜)?

# Distributional hypothesis

“Ongchoi”

Ongchoi is a leafy green like spinach, chard or collard greens

空心菜  
*kangkong*  
rau muống  
...



## How can do the same thing computationally?

- Count the words in the context of ongchoi
- See what other words occur in those contexts

We can represent a word's context using vectors!



# Word embeddings


Goal: represent words as **short** (50-300 dimensional) & **dense** (real-valued) vectors

## Count-based approaches

- Used since the 90s
- Sparse word-word co-occurrence PPMI matrix
- Decomposed with SVD

## Prediction-based approaches

- Formulated as a machine learning problem
- Word2vec (Mikolov et al., 2013)
- GloVe (Pennington et al., 2014)



Underlying theory: Distributional Hypothesis (*Firth, '57*)  
“**Similar words occur in similar contexts**”

# Word embeddings: the learning problem

**Learning** vectors from text for representing words

- **Input:** a large text corpus, vocabulary  $V$ , vector dimension  $d$  (e.g., 300)
- **Output:**  $f : V \rightarrow \mathbb{R}^d$

Each coordinate/dimension of the vector doesn't have a particular interpretation

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

# Word embeddings

- Basic property: similar words have similar vectors

word  $w^*$  = "sweden"

$$\arg \max_{w \in V} \cos(e(w), e(w^*))$$

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

$\cos(u, v)$  ranges between -1 and 1

# Word embeddings

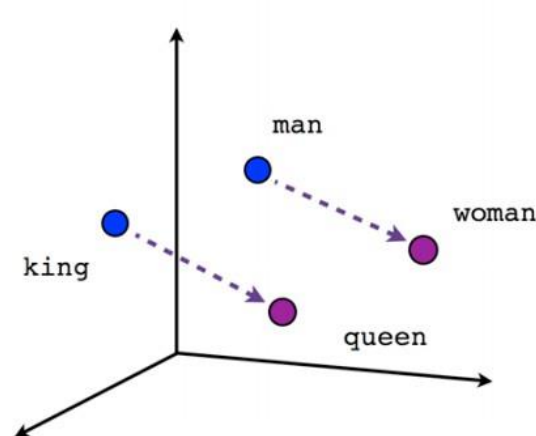
- They have some other nice properties too!

Kawin Ethayarajh, David Duvenaud<sup>†</sup>, Graeme Hirst

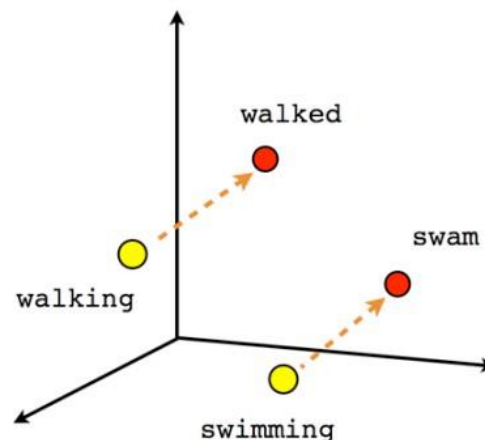
University of Toronto

<sup>†</sup>Vector Institute

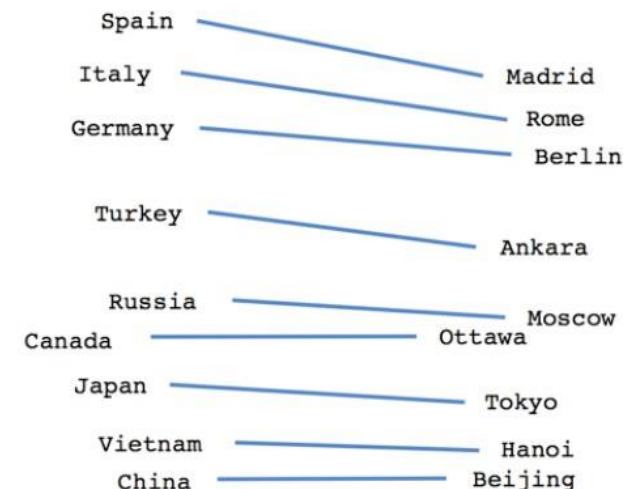
{kawin, duvenaud, gh}@cs.toronto.edu



Male-Female



Verb tense



Country-Capital

$$v_{\text{man}} - v_{\text{woman}} \approx v_{\text{king}} - v_{\text{queen}}$$

$$v_{\text{Paris}} - v_{\text{France}} \approx v_{\text{Rome}} - v_{\text{Italy}}$$

Word analogy test:  $a : a^* :: b : b^*$

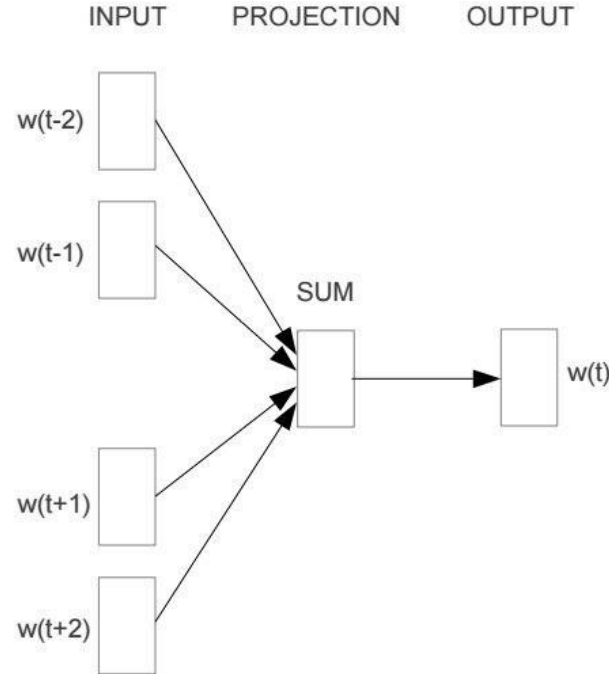
$$b^* = \arg \max_{w \in V} \cos(e(w), e(a^*) - e(a) + e(b))$$

# word2vec

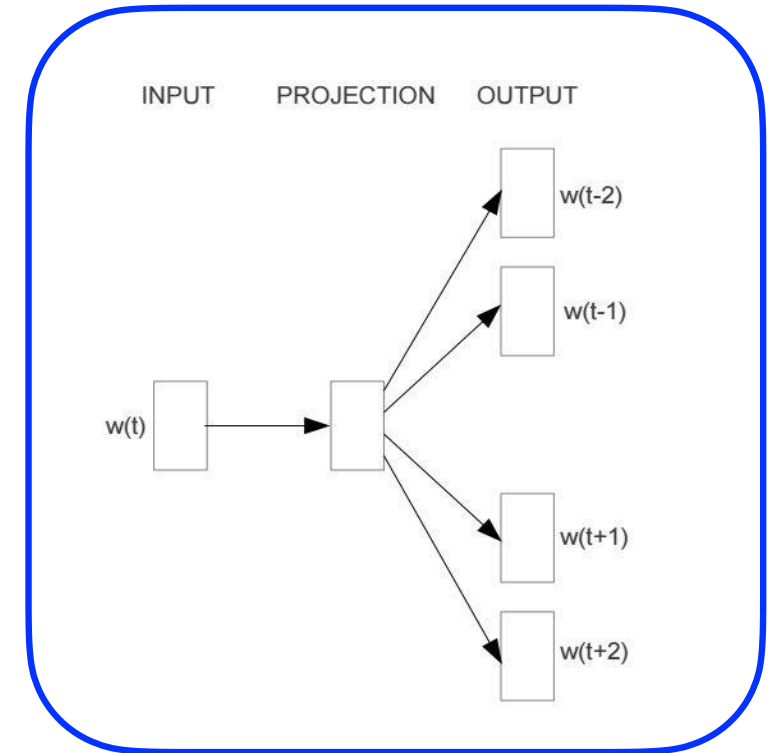
- (Mikolov et al 2013a): Efficient Estimation of Word Representations in Vector Space
- (Mikolov et al 2013b): Distributed Representations of Words and Phrases and their Compositionality



Thomas Mikolov



Continuous Bag of Words  
(CBOW)

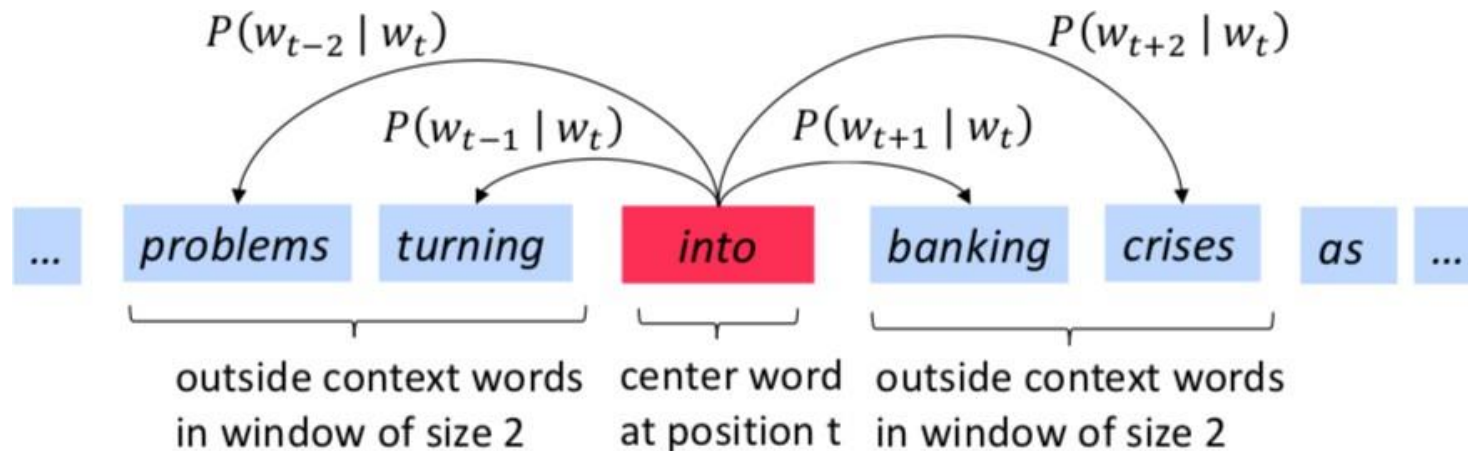


Skip-gram

# Skip-gram

- Assume that we have a large corpus  $w_1, w_2, \dots, w_T \in V$
- **Key idea:** Use each word to **predict** other words in its context
- Context: a fixed window of size  $2m$  ( $m = 2$  in the example)

A classification problem!

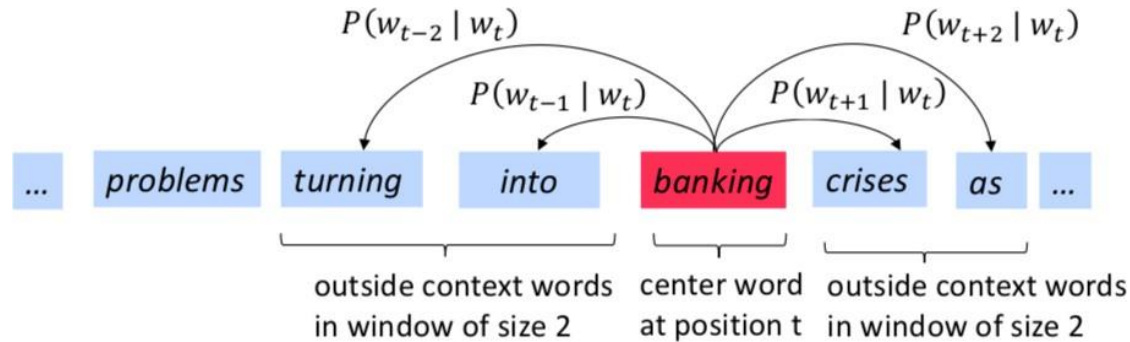
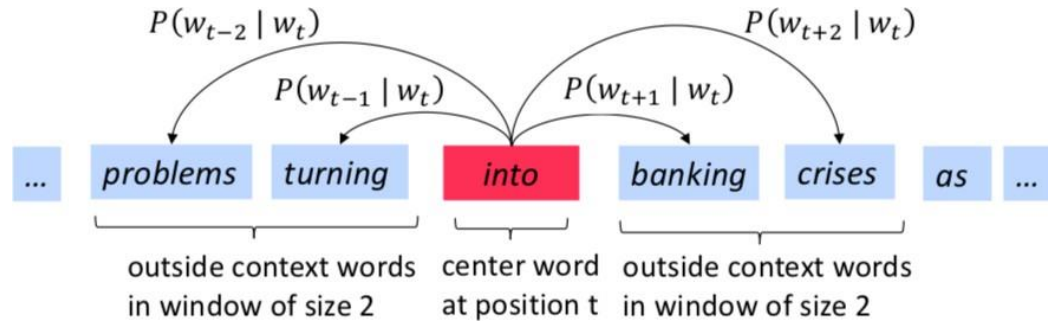


$P(b | a)$  = given the center word is  $a$ , what is the probability that  $b$  is a context word?

$P(\cdot | a)$  is a probability distribution defined  $\sum_{w \in V} P(w | a) = 1$

We are going to define this distribution soon!

# Skip-gram



Convert the training data into:

(into, problems)

(into, turning)

(into, banking)

(into, crises)

(banking, turning)

(banking, into)

(banking, crises)

(banking, as)

...

Our goal is to find parameters that can maximize

$P(\text{problems} | \text{into}) \times P(\text{turning} | \text{into}) \times P(\text{banking} | \text{into}) \times P(\text{crises} | \text{into}) \times P(\text{turning} | \text{banking}) \times P(\text{into} | \text{banking}) \times P(\text{crises} | \text{banking}) \times P(\text{as} | \text{banking}) \dots$

# Skip-gram: objective function

- For each position  $t = 1, 2, \dots, T$ , predict context words within context size  $m$ , given center word  $w_t$ :

$$\mathcal{L}(\theta) = \prod_{t=1}^T \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} \mid w_t; \theta)$$

all the parameters to be optimized



- It is equivalent as minimizing the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log \mathcal{L}(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} \mid w_t; \theta)$$



# How to define $P(w_{t+j} \mid w_t; \theta)$ ?

- Use two sets of vectors for each word in the vocabulary

$\mathbf{u}_a \in \mathbb{R}^d$ : vector for center word  $a$ ,  $\forall a \in V$

$\mathbf{v}_b \in \mathbb{R}^d$ : vector for context word  $b$ ,  $\forall b \in V$

- Use inner product  $\mathbf{u}_a \cdot \mathbf{v}_b$  to measure how likely word  $a$  appears with context word  $b$

Softmax we have seen in multinomial logistic regression!

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Recall that  $P(\cdot \mid a)$  is a probability distribution defined over  $V$ ...

# ... vs multinomial logistic regression

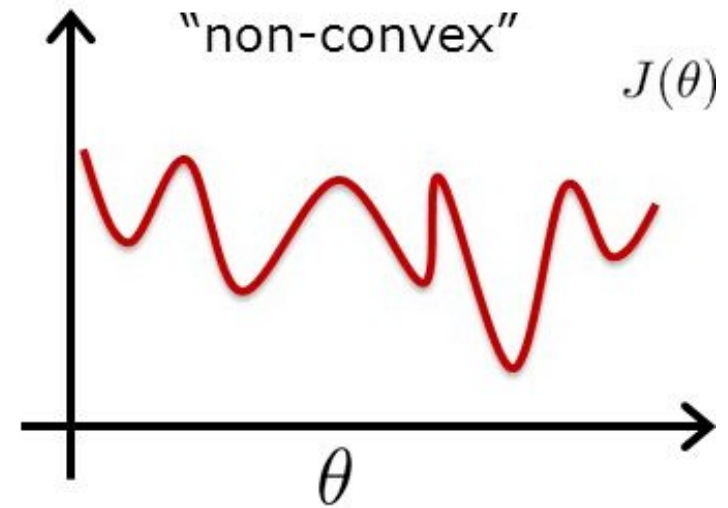
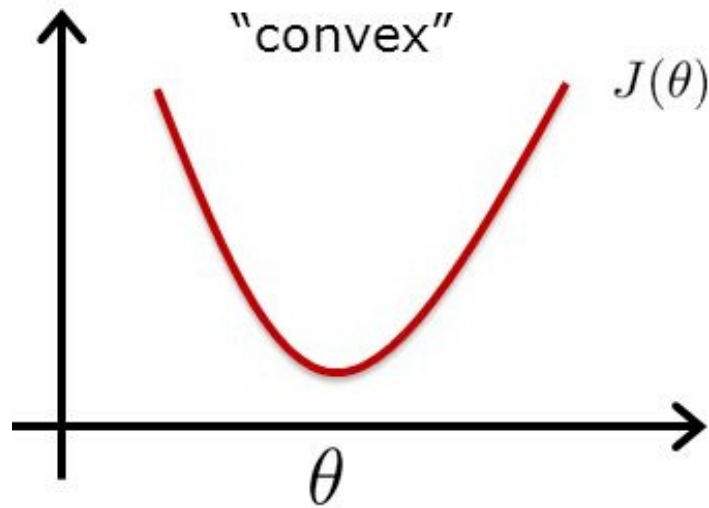
Multinomial logistic regression:

$$P(y = c | \mathbf{x}) = \frac{\exp(\mathbf{w}_c \cdot \mathbf{x} + b_c)}{\sum_{j=1}^m \exp(\mathbf{w}_j \cdot \mathbf{x} + b_j)}$$

$$P(w_{t+j} | w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

- Essentially a  $|V|$ -way classification problem
- If we fix  $\mathbf{u}_{w_t}$ , it is reduced to a multinomial logistic regression problem.
- However, since we have to learn both and together, the training objective is **non-convex**.

# ... vs multinomial logistic regression



- It is hard to find a global minimum
- But can still use stochastic gradient descent to optimize  
:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} J(\theta)$$

# Important note

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

- In this formulation, we don't care about the classification task itself like we do for the logistic regression model we saw previously.
- The key point is that the *parameters* used to optimize this training objective—when the training corpus is large enough—can give us very good representations of words (following the principle of distributional hypothesis)!

# How many parameters in this model?

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

How many parameters does this model have (i.e. what is size of )?

- (a)  $d |V|$
- (b)  $2d |V|$
- (c)  $2m |V|$
- (d)  $2md |V|$

$d =$  dimension of each vector

# How many parameters in this model?

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

How many parameters does this model have (i.e. what is size of )?

- (a)  $d |V|$
- (b)  $2d |V|$
- (c)  $2m |V|$
- (d)  $2md |V|$

$d =$  dimension of each vector

The answer is (b).

Each word has two  $d$ -dimensional vectors, so it is  $2 \times |V| \times d$ .

# word2vec formulation

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Q: Why do we need two vectors for each word instead of one?

A: because one word is not likely to appear in its own context window, e.g.,  $P(\text{dog} \mid \text{dog})$  should be low. If we use one set of vectors only, it essentially needs to minimize  $\mathbf{u}_{\text{dog}} \cdot \mathbf{u}_{\text{dog}}$ .

Q: Which set of vectors are used as word embeddings?

A: This is an empirical question. Typically just  $\mathbf{u}_w$  but you can also concatenate the two vectors..

Skip-gram with negative sampling  
(SGNS) and other variants



# Skip-gram with negative sampling (SGNS)

**Problem:** every time you get one pair of  $(t, c)$ , you need to update  $\mathbf{v}_k$  with all the words in the vocabulary! This is very expensive computationally.

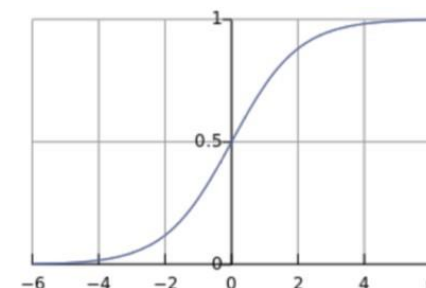
$$\frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k | t) \mathbf{v}_k \quad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k | t) - 1) \mathbf{u}_t & k = c \\ P(k | t) \mathbf{u}_t & k \neq c \end{cases}$$

**Negative sampling:** instead of considering all the words in  $V$ , let's randomly sample  $K$  (5-20) negative examples.

softmax: 
$$y = -\log \left( \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \right)$$

Negative sampling: 
$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^K \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

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



# Skip-gram with negative sampling (SGNS)

**Key idea: Convert the  $|V|$  -way classification into a set of binary classification tasks.**

Every time we get a pair of words  $(t, c)$ , we don't predict  $c$  among all the words in the vocabulary. Instead, we predict  $(t, c)$  is a positive pair, and  $(t, c')$  is a negative pair for a small number of sampled  $c'$ .

**positive examples +**

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

**negative examples -**

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^K \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

$P(w)$ : sampling according to the frequency of words

Similar to **binary logistic regression**, but we need to optimize and together.

$$P(y = 1 \mid t, c) = \sigma(\mathbf{u}_t \cdot \mathbf{v}_c) \quad p(y = 0 \mid t, c') = 1 - \sigma(\mathbf{u}_t \cdot \mathbf{v}_{c'}) = \sigma(-\mathbf{u}_t \cdot \mathbf{v}_{c'})$$

# Understanding SGNS

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^K \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

In skip-gram with negative sampling (SGNS), how many parameters need to be updated in  $\theta$  for every  $(t, c)$  pair?

- (a)  $Kd$
- (b)  $2Kd$
- (c)  $(K + 1)d$
- (d)  $(K + 2)d$

# Understanding SGNS

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^K \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

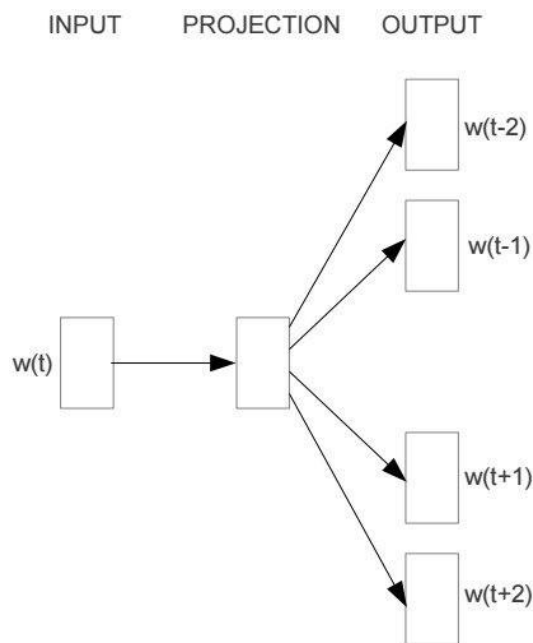
In skip-gram with negative sampling (SGNS), how many parameters need to be updated in  $\theta$  for every  $(t, c)$  pair?

- (a)  $Kd$
- (b)  $2Kd$
- (c)  $(K + 1)d$
- (d)  $(K + 2)d$

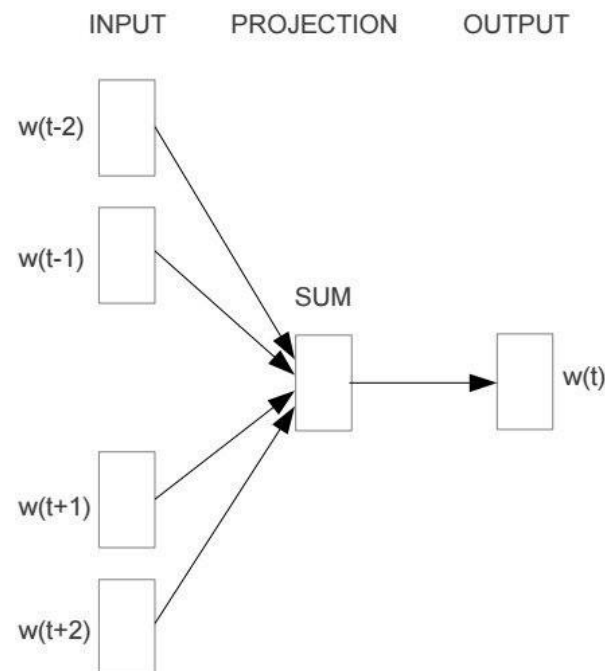
The answer is (d).

We need to calculate gradients with respect to  $\mathbf{u}_t$  and  $(K + 1)$   $\mathbf{v}_i$  (one positive and  $K$  negatives).

# Continuous Bag of Words (CBOW)



Skip-gram



Continuous Bag of Words (CBOW)

$$L(\theta) = \prod_{t=1}^T P(w_t | \{w_{t+j}\}, -m \leq j \leq m, j \neq 0)$$

$$\bar{\mathbf{v}}_t = \frac{1}{2m} \sum_{-m \leq j \leq m, j \neq 0} \mathbf{v}_{t+j}$$

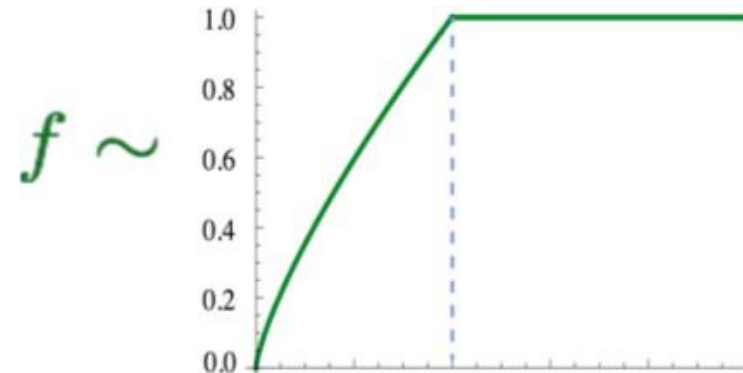
$$P(w_t | \{w_{t+j}\}) = \frac{\exp(\mathbf{u}_{w_t} \cdot \bar{\mathbf{v}}_t)}{\sum_{k \in V} \exp(\mathbf{u}_k \cdot \bar{\mathbf{v}}_t)}$$

# GloVe: Global Vectors

- Key idea: let's approximate  $\mathbf{u}_i \cdot \mathbf{v}_j$  using their co-occurrence counts directly
- Take the global co-occurrence statistics:  $X_{i,j}$

$$J(\theta) = \sum_{i,j \in V} f(X_{i,j}) \left( \mathbf{u}_i \cdot \mathbf{v}_j + b_i + \tilde{b}_j - \log X_{i,j} \right)^2$$

- Training faster
- Scalable to very large corpora



# FastText: Subword Embeddings

- Similar to Skip-gram, but break words into n-grams with  $n = 3$  to  $6$

where:

- 3-grams: <wh, whe, her, ere, re>
- 4 grams: <whe, wher, here, ere>
- 5 grams: <wher, where, here>
- 6 grams: <where, where>

- Replace  $\mathbf{u}_i \cdot \mathbf{v}_j$  by  $\sum_{g \in n\text{-grams}(w_i)} \mathbf{u}_g \cdot \mathbf{v}_j$

# Trained word embeddings available

- word2vec: <https://code.google.com/archive/p/word2vec/>
- GloVe: <https://nlp.stanford.edu/projects/glove/>
- FastText: <https://fasttext.cc/>

## Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the [Public Domain Dedication and License](http://www.opendatacommons.org/licenses/pddl/1.0/) v1.0 whose full text can be found at: <http://www.opendatacommons.org/licenses/pddl/1.0/>.
  - [Wikipedia 2014 + Gigaword 5](#) (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): [glove.6B.zip](#)
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): [glove.42B.300d.zip](#)
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): [glove.840B.300d.zip](#)
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): [glove.twitter.27B.zip](#)
- Ruby [script](#) for preprocessing Twitter data

Differ in algorithms, text corpora, dimensions, cased/uncased...

Applied to many other languages



# Easy to use!

```
from gensim.models import KeyedVectors
# Load vectors directly from the file
model = KeyedVectors.load_word2vec_format('data/GoogleGoogleNews-vectors-negative300.bin', binary=True)
# Access vectors for specific words with a keyed lookup:
vector = model['easy']
```

```
In [17]: model.similarity('straightforward', 'easy')
```

```
Out[17]: 0.5717043285477517
```

```
In [18]: model.similarity('simple', 'impossible')
```

```
Out[18]: 0.29156160264633707
```

```
In [19]: model.most_similar('simple')
```

```
Out[19]: [('straightforward', 0.7460169196128845),
          ('Simple', 0.7108174562454224),
          ('uncomplicated', 0.6297484636306763),
          ('simplest', 0.6171397566795349),
          ('easy', 0.5990299582481384),
          ('fairly_straightforward', 0.5893306732177734),
          ('deceptively_simple', 0.5743066072463989),
          ('simpler', 0.5537199378013611),
          ('simplistic', 0.5516539216041565),
          ('disarmingly_simple', 0.5365327000617981)]
```

# Evaluating Word vectors

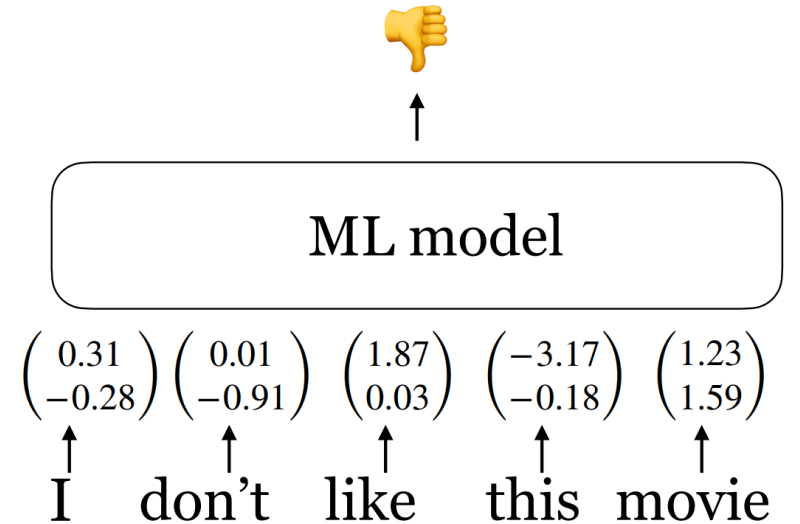
# Extrinsic vs intrinsic evaluation

## Extrinsic evaluation

- Let's plug these word embeddings into a real NLP system and see whether this improves performance
- Could take a long time but still the most important evaluation metric

## Intrinsic evaluation

- Evaluate on a specific/intermediate subtask
- Fast to compute
- Not clear if it really helps downstream tasks



# Extrinsic evaluation



ML model

$$\begin{pmatrix} 0.31 \\ -0.28 \end{pmatrix} \begin{pmatrix} 0.01 \\ -0.91 \end{pmatrix} \begin{pmatrix} 1.87 \\ 0.03 \end{pmatrix} \begin{pmatrix} -3.17 \\ -0.18 \end{pmatrix} \begin{pmatrix} 1.23 \\ 1.59 \end{pmatrix}$$

I don't like this movie

A straightforward solution: given an input sentence  $x_1, x_2, \dots, x_n$

Instead of using a bag-of-words model, we can compute  $vec(x) = e(x_1) + e(x_2) + \dots + e(x_n)$

And then train a logistic regression classifier on  $vec(x)$  as we did before!

There are much better ways to do this e.g., take word embeddings as input of neural networks

# Intrinsic evaluation: word similarity

## Word similarity

Example dataset: wordsim-353

353 pairs of words with human judgement

<http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353>

/

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Cosine similarity:

$$\cos(\mathbf{u}_i, \mathbf{u}_j) = \frac{\mathbf{u}_i \cdot \mathbf{u}_j}{\|\mathbf{u}_i\|_2 \times \|\mathbf{u}_j\|_2}$$

Metric: Spearman rank correlation

# Intrinsic evaluation: word similarity

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	<u>72.7</u>	75.1	56.5	37.0
CBOW <sup>†</sup>	6B	57.2	65.6	68.2	57.0	32.5
SG <sup>†</sup>	6B	62.8	65.2	69.7	<u>58.1</u>	37.2
GloVe	6B	<u>65.8</u>	<u>72.7</u>	<u>77.8</u>	53.9	<u>38.1</u>
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	<b><u>75.9</u></b>	<b><u>83.6</u></b>	<b><u>82.9</u></b>	<b><u>59.6</u></b>	<b><u>47.8</u></b>
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

SG: Skip-gram

# Intrinsic evaluation: word analogy

Word analogy test:  $a : a^* :: b : b^*$

$$b^* = \arg \max_{w \in V} \cos(e(w), e(a^*) - e(a) + e(b))$$

semantic

Chicago:Illinois Philadelphia: ?

syntactic

bad:worst cool: ?

More examples at

<http://download.tensorflow.org/data/questions-words.txt>

Metric: accuracy

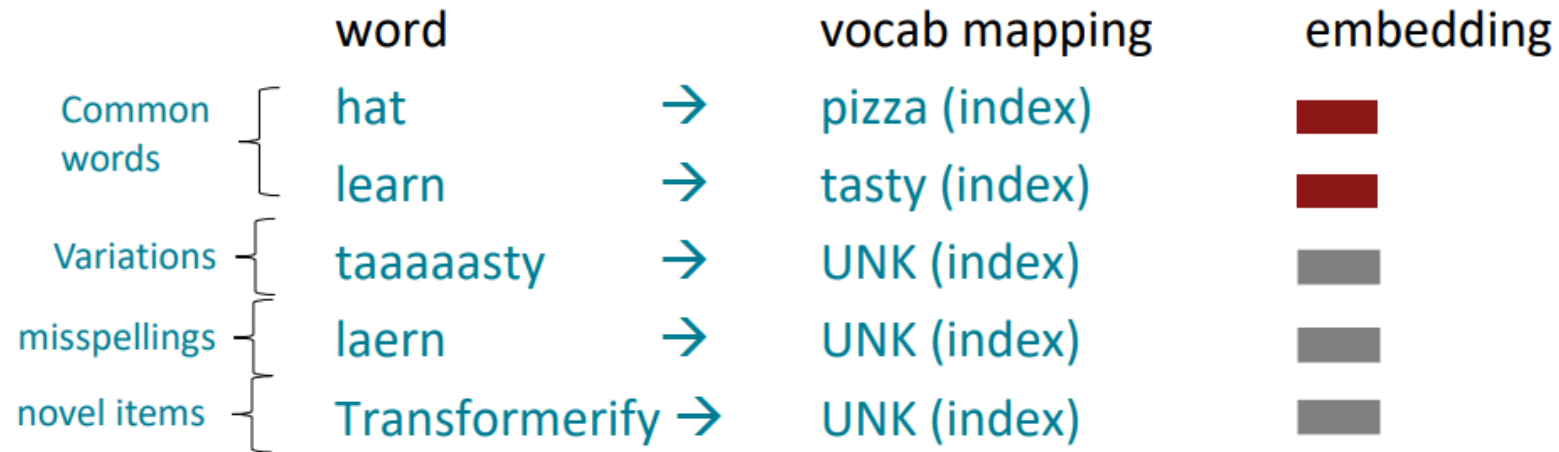
# Intrinsic evaluation: word analogy

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	<u>67.5</u>	<u>54.3</u>	<u>60.3</u>
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	<u>64.8</u>	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	<u>80.8</u>	61.5	<u>70.3</u>
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW <sup>†</sup>	300	6B	63.6	<u>67.4</u>	65.7
SG <sup>†</sup>	300	6B	73.0	66.0	69.1
GloVe	300	6B	<u>77.4</u>	67.0	<u>71.7</u>
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	<b><u>81.9</u></b>	<b><u>69.3</u></b>	<b><u>75.0</u></b>



# Word structure and subword models

We assume a fixed vocab of tens of thousands of words, built from the training set. All novel words seen at test time are mapped to a single UNK.



Finite vocabulary assumptions make even less sense in many languages.

- Many languages exhibit complex morphology, or word structure.
- The effect is more word types, each occurring fewer times.

# More on Word vectors

# Interesting characters/words

- 柸 《广韵》《集韵》并以冉切，音琰 (yan3)。物上大下小也。  
又《集韵》他刀切，音叨 (tao1)。进也。
- LGUer
- Loooooooooong

# A paper from ours: MorphTE

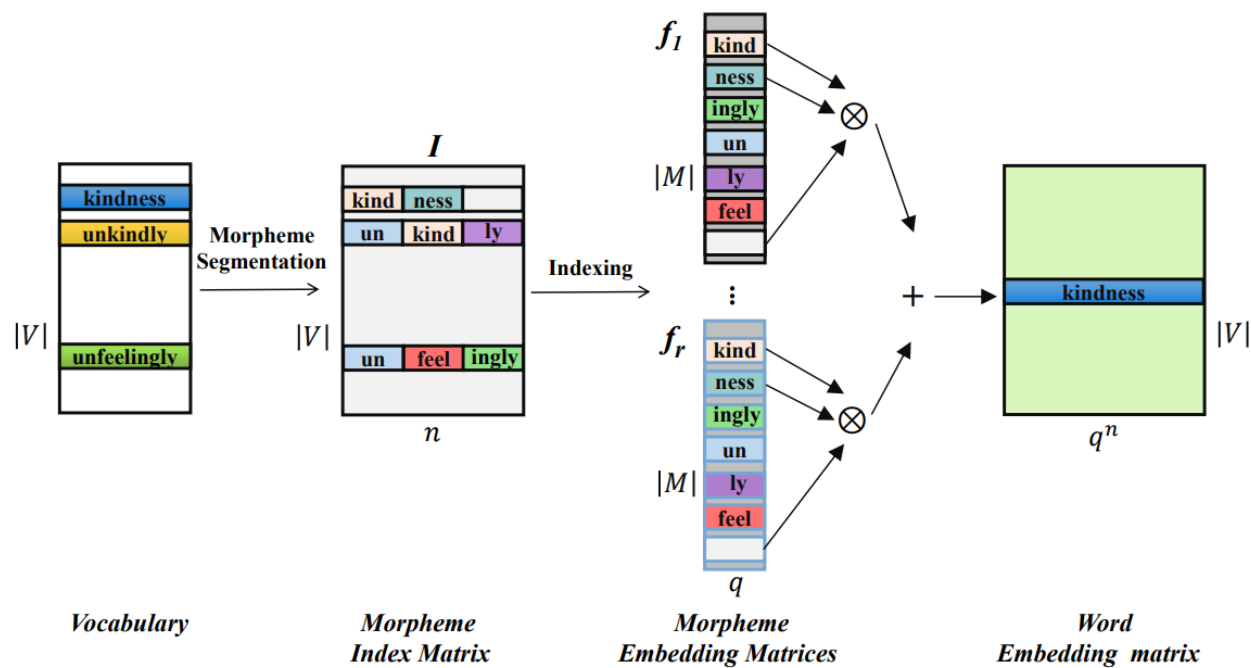
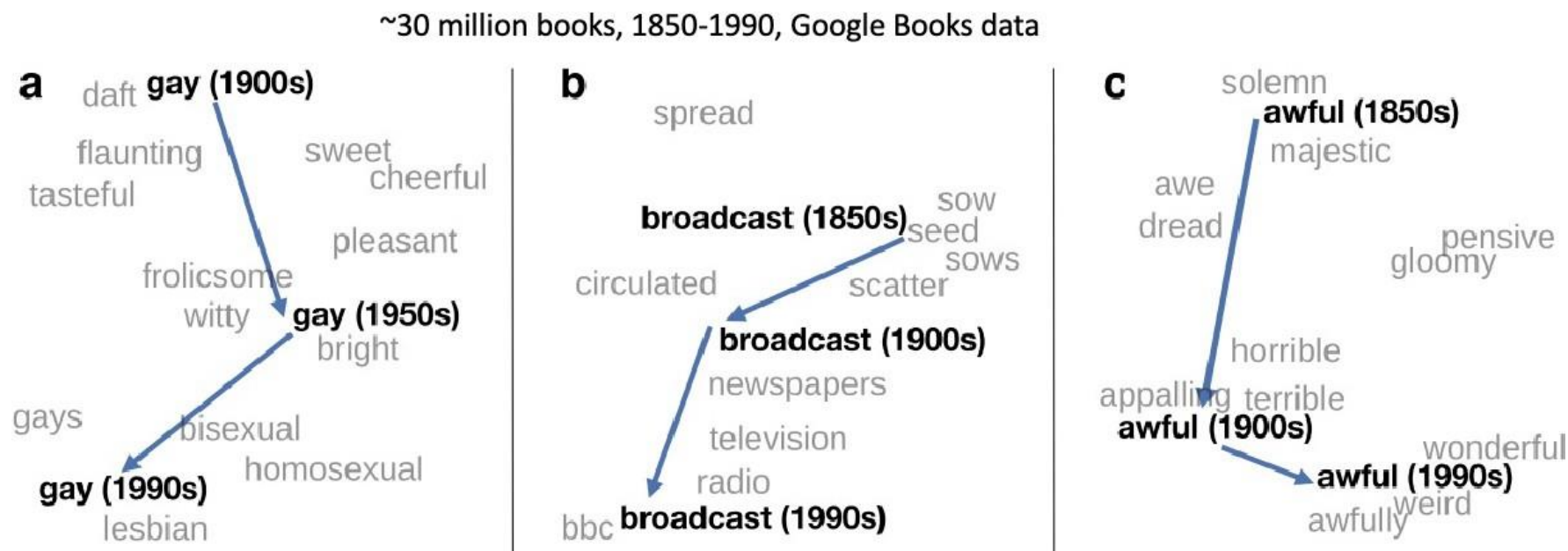


Figure 3: The workflow of MorphTE.  $n$  is the order (number of morphemes for a word).  $q$  is the size of morpheme vectors.  $|V|$  and  $|M|$  denote the size of word vocabulary and morpheme vocabulary.

# Embeddings as a window onto historical semantics

Train embeddings on different decades of historical text to see meanings shift



William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. Proceedings of ACL.

# Embeddings reflect cultural bias!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

Ask "Paris : France :: Tokyo : x"

- x = Japan

Ask "father : doctor :: mother : x"

- x = nurse

Ask "man : computer programmer :: woman : x"

- x = homemaker

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

Thanks

# Training Word vectors



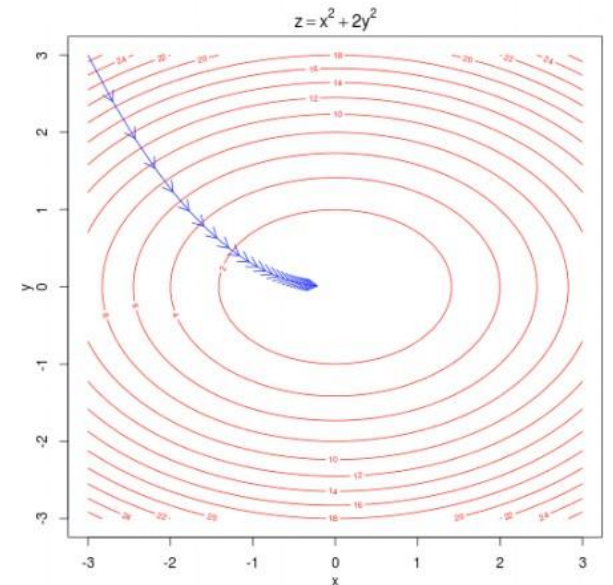
# How to train this model?

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

- To train such a model, we need to compute the vector gradient  $\nabla_{\theta} J(\theta) = ?$

- Again,  $\theta$  represents all  $2d \mid V \mid$  model parameters, in one vector.

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix}$$



# Vectorized gradients

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{a}$$
$$\mathbf{x}, \mathbf{a} \in \mathbb{R}^n$$

$$\frac{\partial f}{\partial \mathbf{x}} = \mathbf{a}$$

$$f = x_1 a_1 + x_2 a_2 + \dots + x_n a_n$$

$$\frac{\partial f}{\partial \mathbf{x}} = \left[ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]$$

# Vectorized gradients: exercises

Let  $f = \exp(\mathbf{w} \cdot \mathbf{x})$ , what is the value of  $\frac{\partial f}{\partial \mathbf{x}}$ ?  $\mathbf{w}, \mathbf{x} \in \mathbb{R}^n$

- (a)  $\mathbf{w}$
- (b)  $\exp(\mathbf{w} \cdot \mathbf{x})$
- (c)  $\exp(\mathbf{w} \cdot \mathbf{x})\mathbf{w}$
- (d)  $\mathbf{x}$

# Vectorized gradients: exercises

Let  $f = \exp(\mathbf{w} \cdot \mathbf{x})$ , what is the value of  $\frac{\partial f}{\partial \mathbf{x}}$ ?  $\mathbf{w}, \mathbf{x} \in \mathbb{R}^n$

- (a)  $\mathbf{w}$
- (b)  $\exp(\mathbf{w} \cdot \mathbf{x})$
- (c)  $\exp(\mathbf{w} \cdot \mathbf{x})\mathbf{w}$
- (d)  $\mathbf{x}$

The answer is (c).

$$\frac{\partial}{\partial x_i} = \frac{\exp(\sum_{k=1}^n w_k x_k)}{\partial x_i} = \exp(\sum_{k=1}^n w_k x_k) w_i$$

# Let's compute gradients for word2vec

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Consider one pair of center/context words  $(t, c)$ :

$$y = -\log \left( \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \right)$$

We need to compute the gradient of with respect to

$$\mathbf{u}_t \text{ and } \mathbf{v}_k, \forall k \in V$$

# Let's compute gradients for word2vec

$$y = -\log \left( \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \right)$$

$$\begin{aligned} y &= -\log(\exp(\mathbf{u}_t \cdot \mathbf{v}_c)) + \log\left(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)\right) \\ &= -\mathbf{u}_t \cdot \mathbf{v}_c + \log\left(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)\right) \end{aligned}$$

# Let's compute gradients for word2vec

$$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

$$\begin{aligned} y &= -\log(\exp(\mathbf{u}_t \cdot \mathbf{v}_c)) + \log\left(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)\right) \\ &= -\mathbf{u}_t \cdot \mathbf{v}_c + \log\left(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)\right) \end{aligned}$$

$$\begin{aligned} \frac{\partial y}{\partial \mathbf{u}_t} &= \frac{\partial(-\mathbf{u}_t \cdot \mathbf{v}_c)}{\partial \mathbf{u}_t} + \frac{\partial(\log \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))}{\partial \mathbf{u}_t} \\ &= -\mathbf{v}_c + \frac{\frac{\partial \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\partial \mathbf{u}_t}}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \\ &= -\mathbf{v}_c + \frac{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k) \cdot \mathbf{v}_k}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \end{aligned}$$

# Let's compute gradients for word2vec

$$y = -\log \left( \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \right)$$

$$\begin{aligned} y &= -\log(\exp(\mathbf{u}_t \cdot \mathbf{v}_c)) + \log\left(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)\right) \\ &= -\mathbf{u}_t \cdot \mathbf{v}_c + \log\left(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)\right) \end{aligned}$$

$$\begin{aligned} \frac{\partial y}{\partial \mathbf{u}_t} &= \frac{\partial(-\mathbf{u}_t \cdot \mathbf{v}_c)}{\partial \mathbf{u}_t} + \frac{\partial(\log \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))}{\partial \mathbf{u}_t} \\ &= -\mathbf{v}_c + \frac{\frac{\partial \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\partial \mathbf{u}_t}}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \\ &= -\mathbf{v}_c + \frac{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k) \cdot \mathbf{v}_k}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \end{aligned}$$

Recall that

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$



# Let's compute gradients for word2vec

$$y = -\log \left( \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \right)$$

$$\begin{aligned} y &= -\log(\exp(\mathbf{u}_t \cdot \mathbf{v}_c)) + \log\left(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)\right) \\ &= -\mathbf{u}_t \cdot \mathbf{v}_c + \log\left(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)\right) \end{aligned}$$

Recall that

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

$$\begin{aligned} \frac{\partial y}{\partial \mathbf{u}_t} &= \frac{\partial(-\mathbf{u}_t \cdot \mathbf{v}_c)}{\partial \mathbf{u}_t} + \frac{\partial(\log \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))}{\partial \mathbf{u}_t} \\ &= -\mathbf{v}_c + \frac{\frac{\partial \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\partial \mathbf{u}_t}}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \\ &= -\mathbf{v}_c + \frac{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k) \cdot \mathbf{v}_k}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \\ &= -\mathbf{v}_c + \sum_{k \in V} \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\sum_{k' \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_{k'})} \mathbf{v}_k \\ &= -\mathbf{v}_c + \sum_{k \in V} P(k \mid t) \mathbf{v}_k \end{aligned}$$

# Overall algorithm

- Input: text corpus, embedding size  $d$ , vocabulary  $V$ , **context size  $m$**
- Initialize  $\mathbf{u}_i, \mathbf{v}_i$  randomly  $\forall i \in V$
- Run through the training corpus and for each training instance  $(t, c)$ :

- Update  $\mathbf{u}_t \leftarrow \mathbf{u}_t - \eta \frac{\partial y}{\partial \mathbf{u}_t} \quad \frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k | t) \mathbf{v}_k$

- Update  $\mathbf{v}_k \leftarrow \mathbf{v}_k - \eta \frac{\partial y}{\partial \mathbf{v}_k}, \forall k \in V \quad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k | t) - 1) \mathbf{u}_t & k = c \\ P(k | t) \mathbf{u}_t & k \neq c \end{cases}$

Convert the training data into:  
(into, problems)  
(into, turning)  
(into, banking)  
(into, crises)  
(banking, turning)  
(banking, into)  
(banking, crises)  
(banking, as)

...

# Overall algorithm

- Input: text corpus, embedding size  $d$ , vocabulary  $V$ , **context size  $m$**
- Initialize  $\mathbf{u}_i, \mathbf{v}_i$  randomly  $\forall i \in V$
- Run through the training corpus and for each training instance  $(t, c)$ :

- Update  $\mathbf{u}_t \leftarrow \mathbf{u}_t - \eta \frac{\partial y}{\partial \mathbf{u}_t} \quad \frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k | t) \mathbf{v}_k$

- Update  $\mathbf{v}_k \leftarrow \mathbf{v}_k - \eta \frac{\partial y}{\partial \mathbf{v}_k}, \forall k \in V \quad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k | t) - 1) \mathbf{u}_t & k = c \\ P(k | t) \mathbf{u}_t & k \neq c \end{cases}$

Convert the training data into:  
(into, problems)  
(into, turning)  
(into, banking)  
(into, crises)  
(banking, turning)  
(banking, into)  
(banking, crises)  
(banking, as)  
...

Q: Can you think of any issues with this algorithm?