tejasgokhale.com

CMSC 475/675 Neural Networks

Lecture 10:

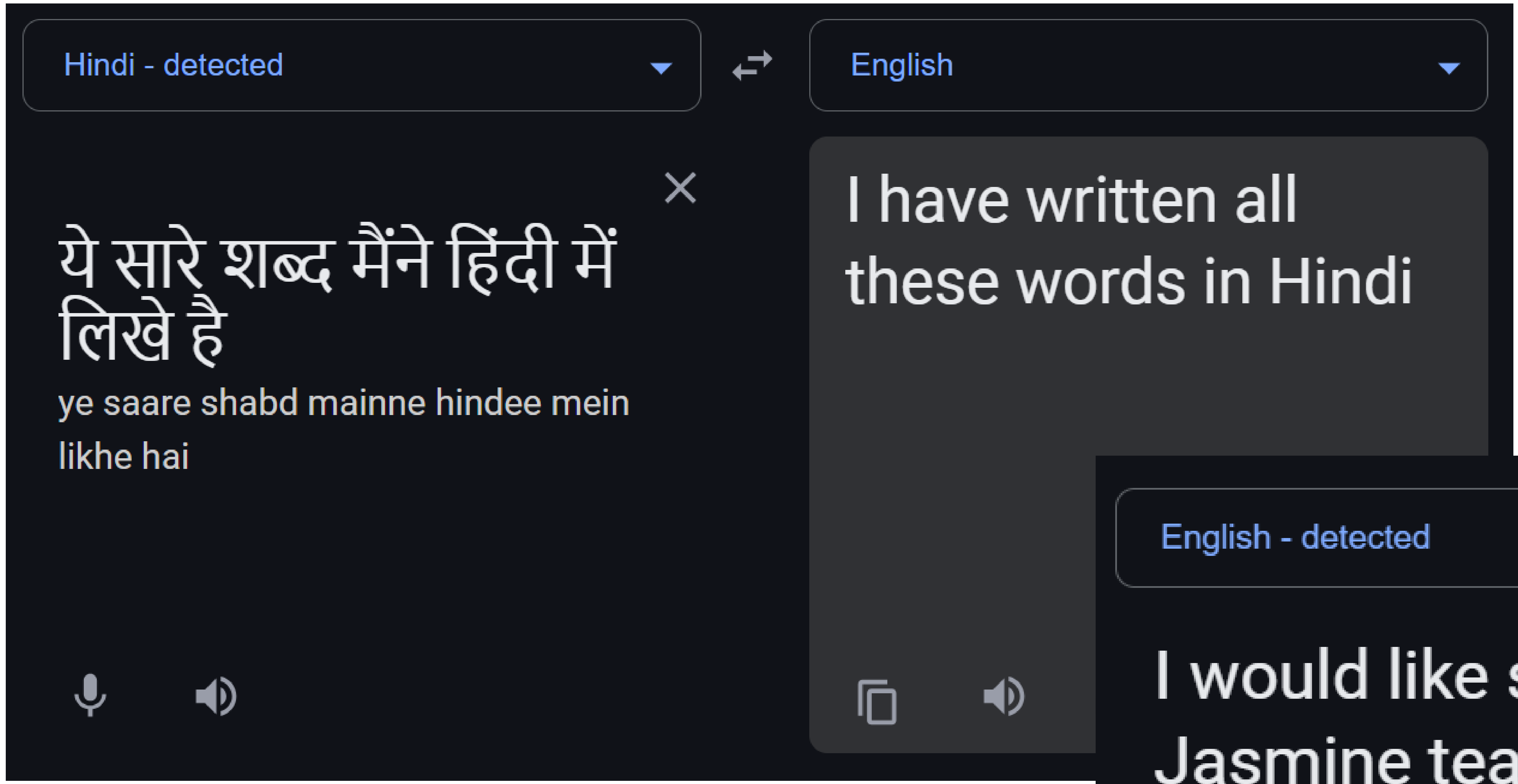
Neural Language Models I

Word2Vec, N-Gram

UMBC

Some slides from
Diyi Yang (Stanford),
Mariane Carpuat (UMD)

Applications: Translation



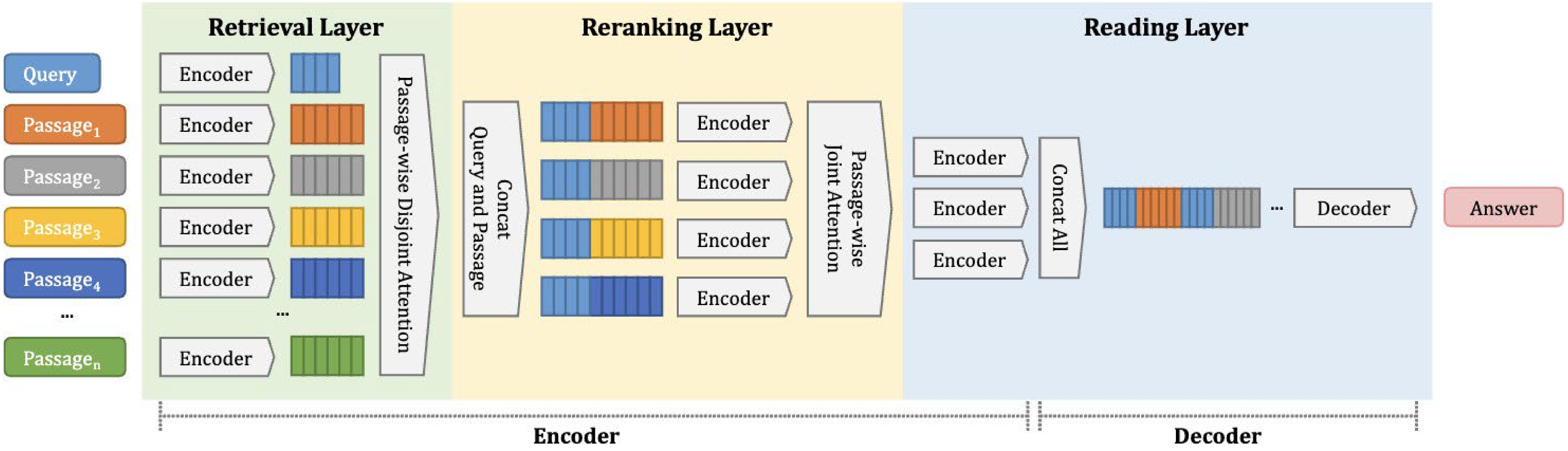
Applications: Question Answering

when did Kendrick lamar's first album come out?

July 2, 2011

These are my leftover songs you all can have them. I'm going to put my best out. My best effort. I'm trying to look for an album in 2012."[44] In June 2011, Lamar released "Ronald Reagan Era (His Evils)", a cut from *Section.80*, featuring Wu-Tang Clan leader RZA.[45] On July 2, 2011, Lamar released *Section.80*, his first independent album. The album features guest appearances from GLC, Colin Munroe, Schoolboy Q, and Ab-Soul, while the production was handled by Top Dawg in-house

E.g., YONO (Lee et al. 2021) uses a T5 model fine-tuned for QA



GPT-3: Towards general-purpose language models

OpenAI

A

Hey please draft a polite mail to explain my boss Jeremy that I would not be able to come to office for next 2 days because my 9 year song Peter is angry with me that I am not giving him much time. So I have plans to take him out for a trip. I will return and complete my work by meeting deadline.

Dear Jeremy,

I hope this email finds you well. I'm writing to let you know that I will not be able to come into the office for the next two days.

My 9-year-old son Peter is feeling upset with me because he feels like I haven't been giving him enough attention lately. In order to make it up to him, I've decided to take him on a little trip to help us reconnect.

I understand that this may cause some inconvenience, but I promise to make up for any lost time by working extra hard to meet any deadlines when I return.

Thank you for your understanding.

Best regards,
[Your Name]

Give a step-by-step analysis of the worst-case time complexity of the bubble sort algorithm, with Python code examples, but write every sentence in the speaking style of a fast-talkin' wise guy from a 1940's gangster movie.

User What is unusual about this image?

Source: Barnorama

GPT-4 The unusual thing about this image is that a man is ironing clothes on an ironing board attached to the roof of a moving taxi.

How should we represent words as vectors?

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.

Common 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

Means one 1, the rest 0s

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+)

Problem with words as discrete symbols

Example: in web search, if a user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"

But:

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]

These two vectors are orthogonal

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

Solution:

- **learn to encode similarity in the vectors themselves**

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***

Word vectors

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector dot (scalar) product

$$
\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \qquad \textit{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}
$$

Note: word vectors are also called (word) embeddings or (neural) word representations
They are a distributed representation

Word meaning as a neural word vector

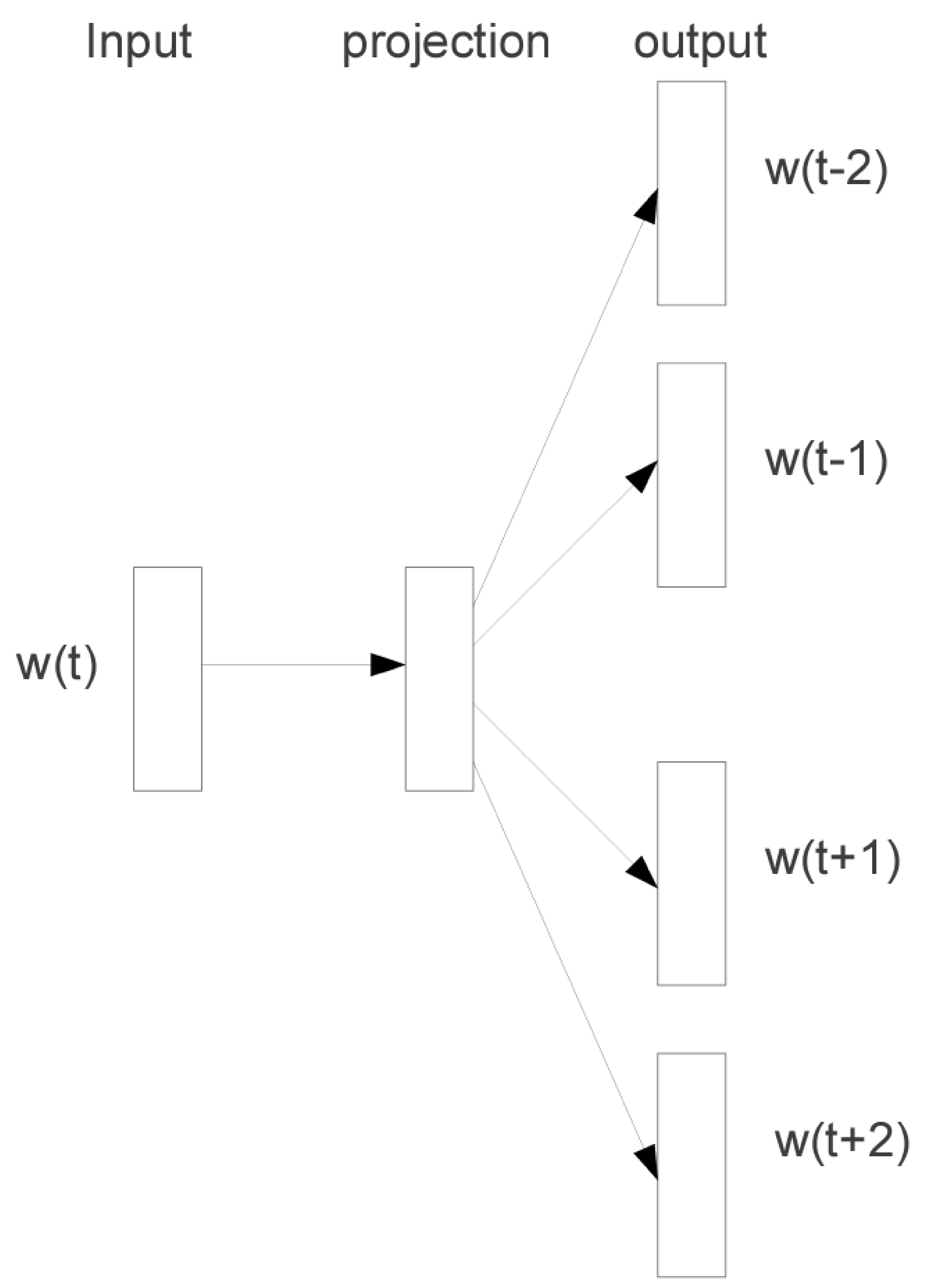
$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

word2vec: Overview

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

Idea:

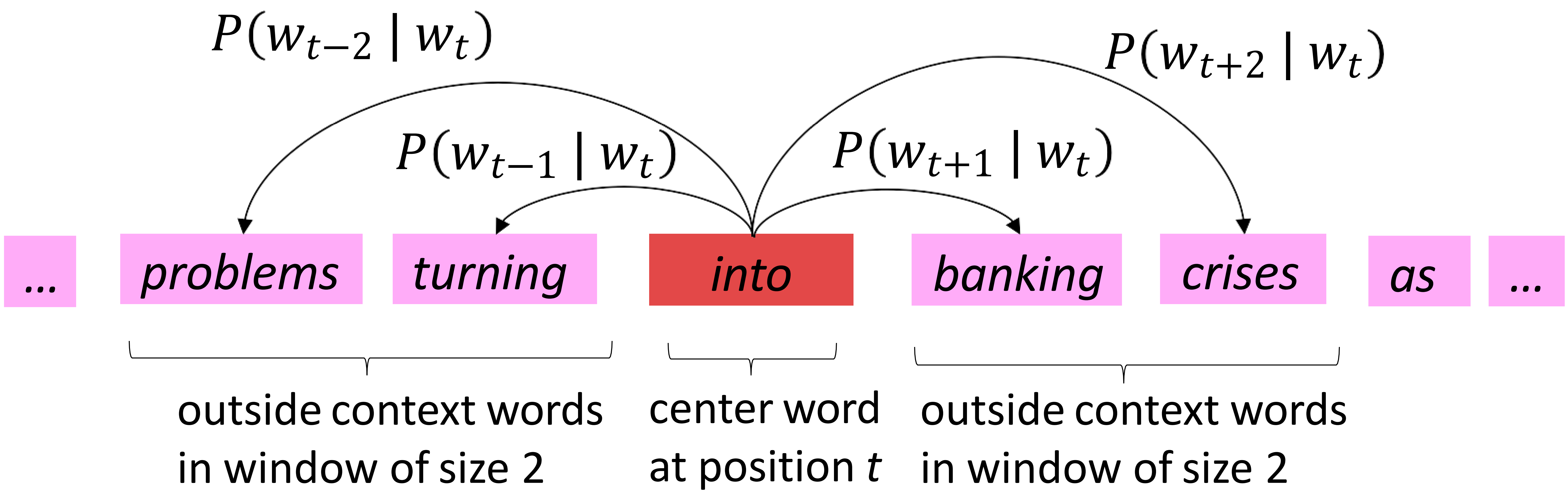
- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
- Keep adjusting the word vectors to maximize this probability



Skip-gram model
(Mikolov et al. 2013)

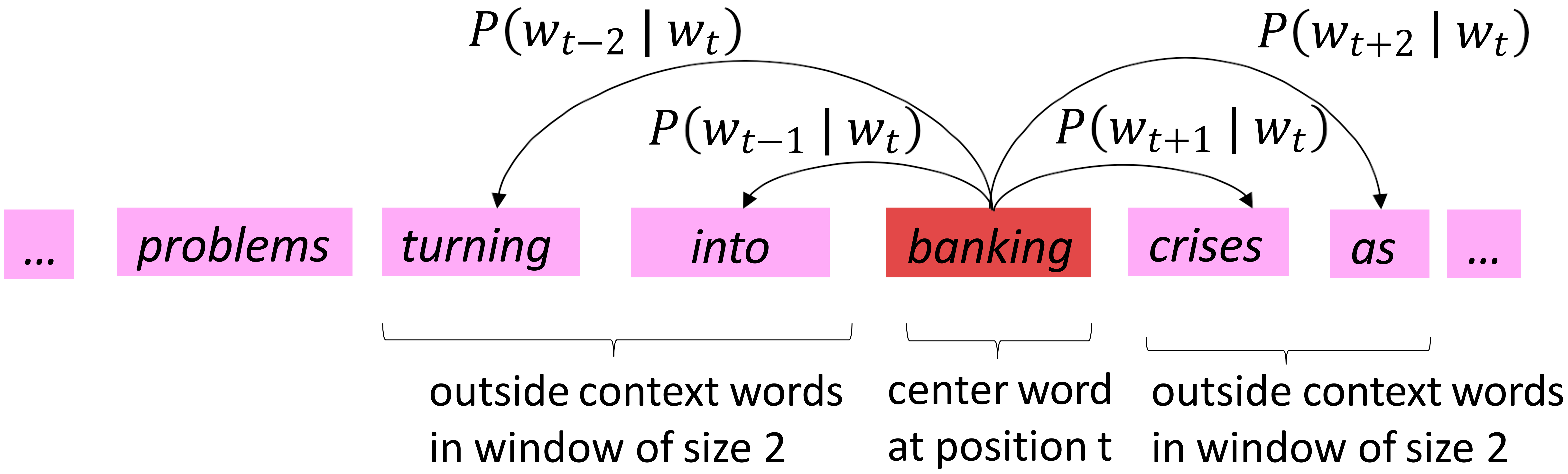
word2vec Overview

Example windows and process for computing $P\left(w_{t+j} \mid w_t\right)$



word2vec Overview

Example windows and process for computing $P\left(w_{t+j} \mid w_t\right)$



word2vec: objective function

For each position $t = 1, \dots, T$, predict context words within a window of fixed size m, given center word w_t. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function ⟺ Maximizing predictive accuracy

word2vec: objective function

- We want to minimize the objective function:

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

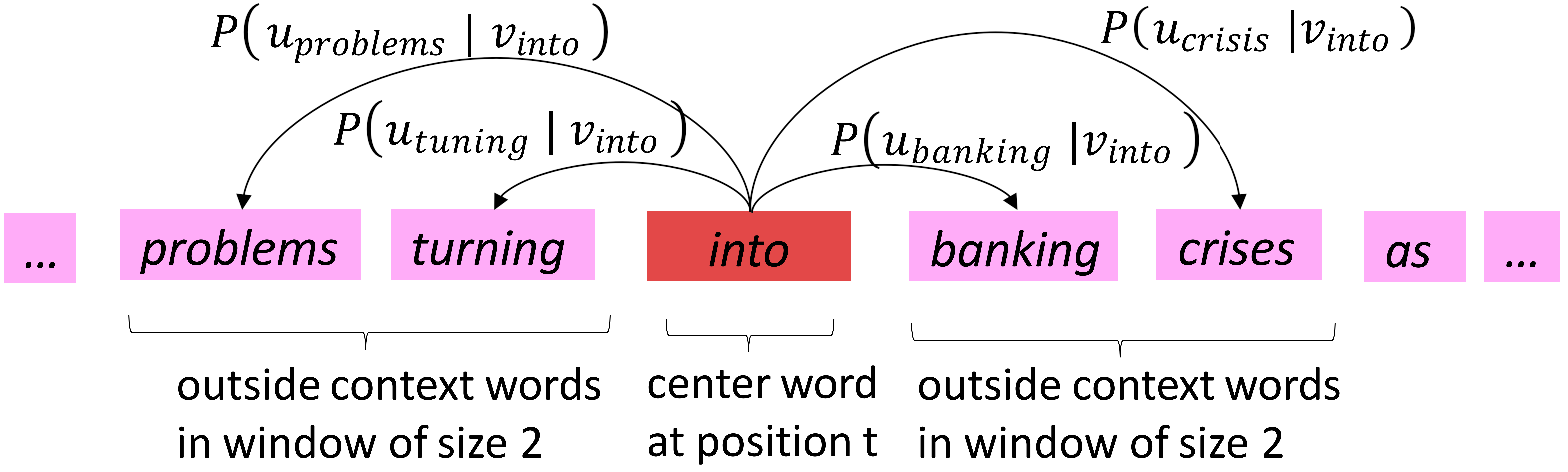
- **Question:** How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?
- **Answer:** We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

word2vec with Vectors

u context vector
v center word

- Example windows and process for computing $P\left(w_{t+j} \mid w_t\right)$
- $P\left(u_{problems} \mid v_{into}\right)$ short for $\mathrm{P}\left(problems \mid into\, ;u_{problems}, v_{into}, \theta\right)$



word2vec: prediction function

u	context vector
v	center word

② Exponentiation makes anything positive

① Dot product compares similarity

$u^T v = u \cdot v = \sigma_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

$$P(o|c) = \frac{\exp(\boxed{u_o^T v_c})}{\boxed{\sum_{w \in V} \exp(u_w^T v_c)}}$$

③ Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \rightarrow (0,1)^n$

Open region

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

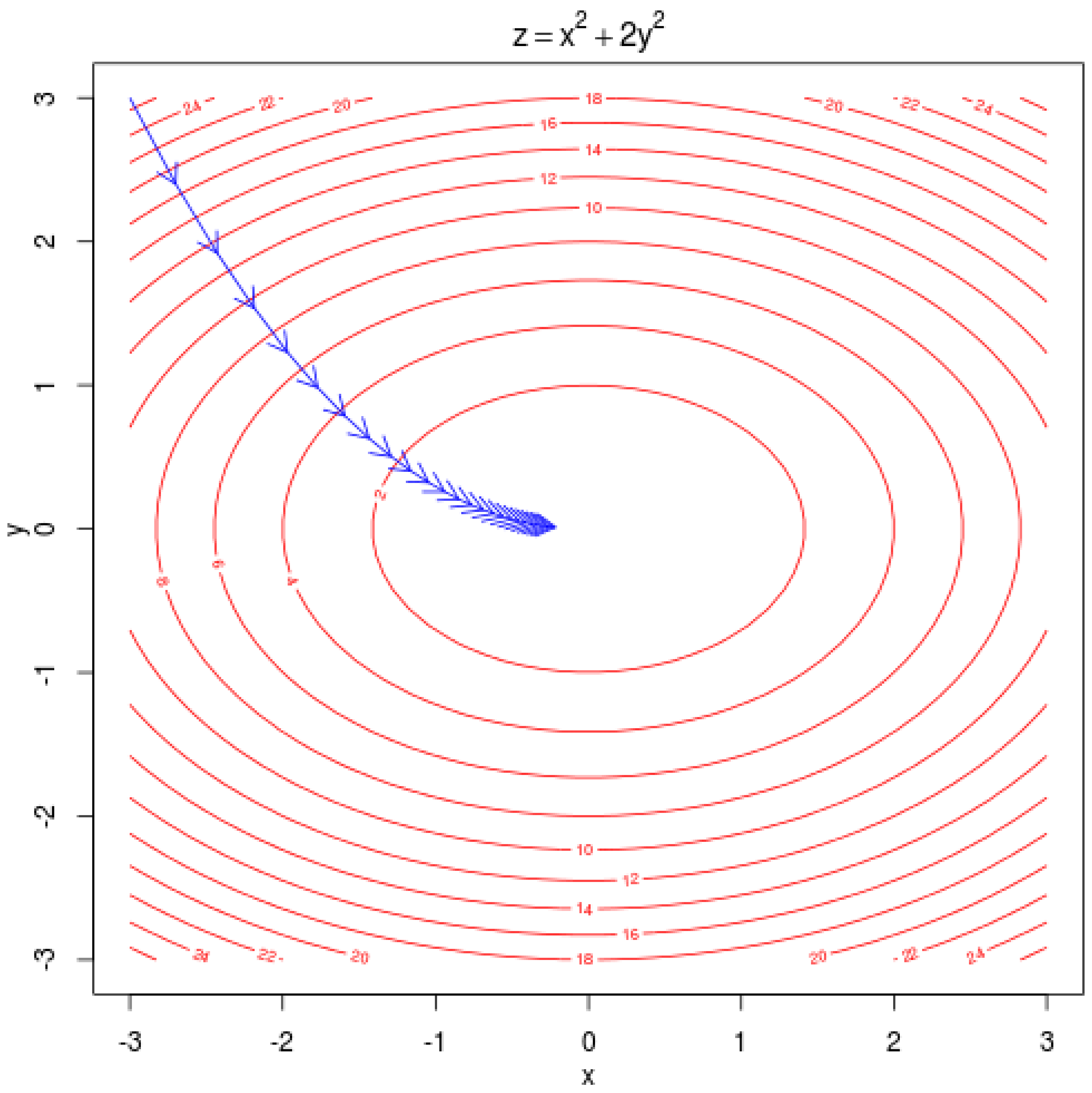
- The softmax function maps arbitrary values x_i to a probability distribution p_i

To train the model: Optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with d-dimensional vectors and V-many words, we have ➔
- Remember: every word has two vectors

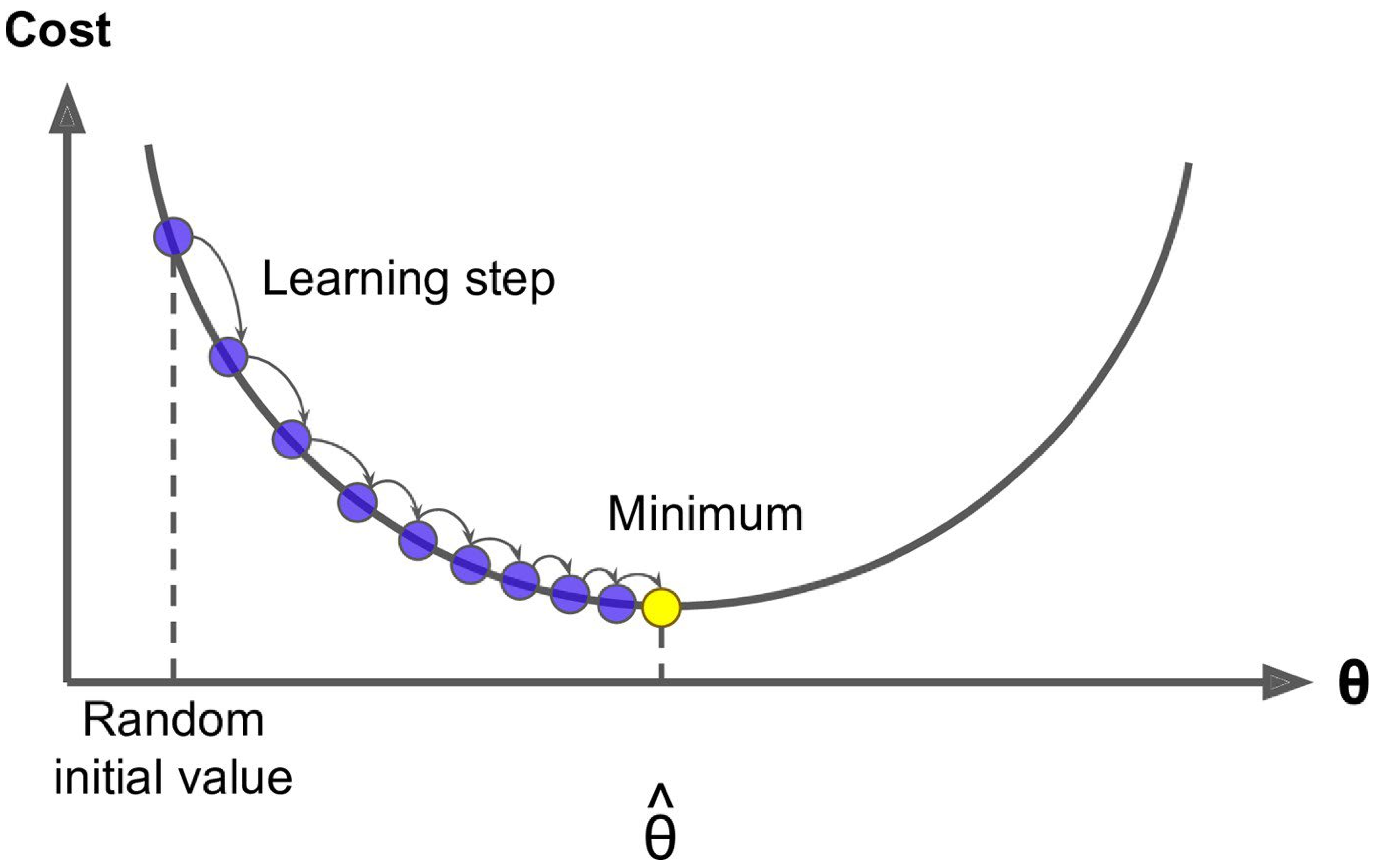
$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute all vector gradients!

Optimization: (Stochastic) 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 θ, calculate gradient of $J(\theta)$, then take small step in direction of negative gradient. Repeat.



Note: Our objectives may not be convex like this ☹

But life turns out to be okay ☺

(Stochastic) Gradient Descent (on mini-batches)

- Update equation (in matrix notation):

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

α = *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
```

word2vec:

② Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(\boxed{u_o^T v_c})}{\boxed{\sum_{w \in V} \exp(u_w^T v_c)}}$$

① Dot product compares similarity of *o* and *c*.
$u^T v = u \cdot v = \sigma_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

③ Normalize over entire vocabulary to give probability distribution

Word2vec parameters ... and computations

U outside

V center

$U \cdot v_4{}^T$ dot product

$\mathrm{softmax}(U \cdot v_4{}^T)$ probabilities

u context vector
v center word

"Bag of words" model! → The model makes the same predictions at each position

We want a model that gives a reasonably high probability estimate to *all* words that occur in the context (at all often)

Cool Embedding Visualizer:

https://projector.tensorflow.org/


calgary
harbour
helsinki
gda
rio
bremen
baltic
stockholm
hamburg
centre
petersburg
leipzig
im
tokyo
clerk
trans
saxony
beck
mainz
cologne
munich
cnn
naples
elector
austin
paris
berlin
germany
johann
sydney
wagner
hans
german
london
theater
rudolf
saint
otto
mason
undergraduate
alberta
graduate
cornell
colorado
berkeley
alaska
nevada
harvard
ontario
delaware
utah
minnesota
oregon
canada
toronto
carson
maryland
montana
columbia
cambridge
northwestern
quebec
seminary
california
ohio
oxford
carolina
school
boston
fort
victoria
college
york
pennsylvania
illinois
royal
attended
disks
dx
semiconductor
processors
circuits
storage
powerpc
cpu
hardware
terminals
hp
boolean
desktop
programmable
devices
risc
fortran
pointer
computing
interface
controller
embedded
compilers
pdp
compiler
device
integrated
tape
logic
babbage
duplicate
printing
programming
neumann
designs
design

word2vec Variant

Skip-Gram model with negative sampling

② Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(\boxed{u_o^T v_c})}{\boxed{\sum_{w \in V} \exp(u_w^T v_c)}}$$

① Dot product compares similarity of *o* and *c.*

$u^T v = u \cdot v = \sigma_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

③ Normalize over entire vocabulary to give probability distribution

- The normalization term is computationally expensive (when many output classes):
  - Denominator is a BIG sum over all words ...
- Idea: train binary logistic regression to differentiate between true pair and noise pair
  - True pair: (center word, context word)
  - Noise pair: (center word, random word)

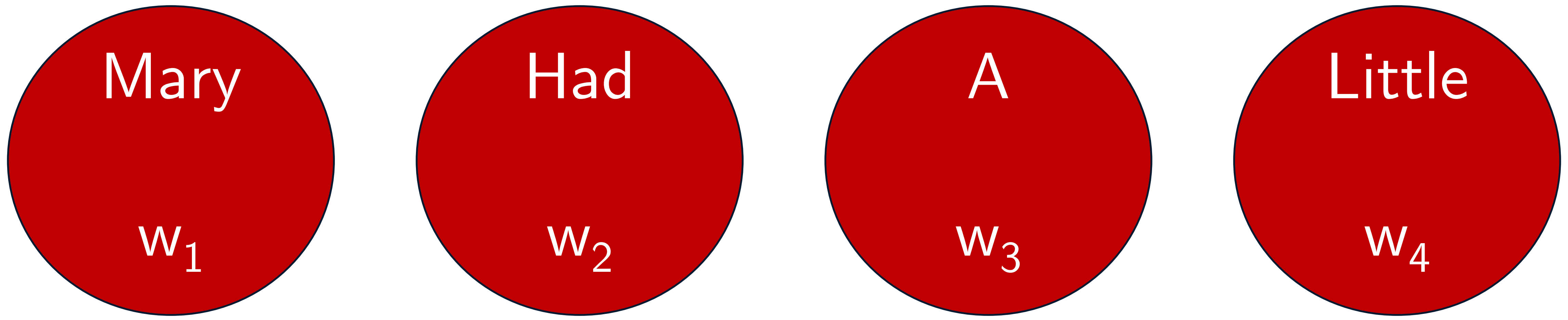
word2vec Variant
Skip-Gram model with negative sampling

- The normalization term is computationally expensive (when many output classes):
  - Denominator is a BIG sum over all words ...
- Idea: differentiate between true pair and noise pair
  - True pair: (center word, context word)
  - Noise pair: (center word, random word)
- Take negative samples (using word probabilities)
  - Maximize probability that real context word appears;
  - Minimize probability that random words appear around center word
- minimize:

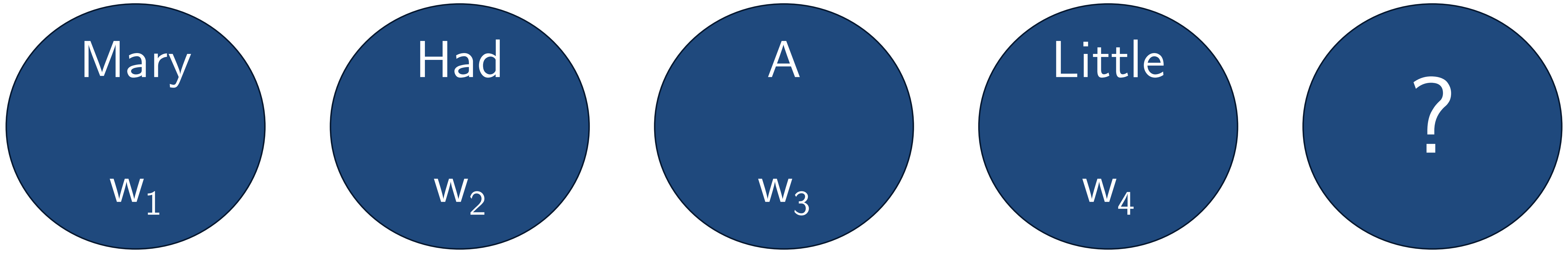
$$J_{neg-sample}(\boldsymbol{u}_o, \boldsymbol{v}_c, U) = -\log \underbrace{\sigma}_{\text{sigmoid}}(\boldsymbol{u}_o^T \boldsymbol{v}_c) - \sum_{k \in \{K \text{ sampled indices}\}} \log \sigma(-\boldsymbol{u}_k^T \boldsymbol{v}_c)$$

Zoom out

Language Modeling: Probabilistic Perspective



- The "joint probability" view:
  - What is the probability of n words appearing in sequence? [Mary, had, a, little]
  - $P(W) = P(w_1\, w_2\, w_3\, w_4\, \ldots.)$



- The "next word predictor" view of language modeling
  - Given n previous words, can we build probabilistic models for the next word?

Language Modeling (LM) more formally

Goal: compute the probability of a sentence or sequence of words W:

$$P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$$

Related task: probability of an upcoming word:

$$P(w_5|w_1,w_2,w_3,w_4) \text{ or } P(w_n|w_1,w_2 \ldots w_{n-1})$$

An LM computes either of these:

$$P(W) \quad \text{or} \quad P(w_n|w_1,w_2 \ldots w_{n-1})$$

slide credit: Dan Jurafsky

Language Models

- The "next word predictor" view of language modeling
  - Given n previous words, can we build probabilistic models for the next word?
  - Instead of modeling the joint $\mathrm{P}(W) = P(w_1 w_2 w_3 w_4 \ldots)$,
    let's model $P(w_5 | w_1 w_2 w_3 w_4)$

Language Models

- The "next word predictor" view of language modeling
  - Given n previous words, can we build probabilistic models for the next word?
  - Instead of modeling the joint $\mathrm{P}(W) = P(w_1 w_2 w_3 w_4 \ldots)$, let's model $P(w_5 | \, w_1 w_2 w_3 w_4)$

- How can we use the "next word predictor" view to model $P(W)$?
  - Bayes rule!

Language Models

- The "next word predictor" view of language modeling
  - Given n previous words, can we build probabilistic models for the next word?
  - Instead of modeling the joint $\mathrm{P}(W) = P(w_1 w_2 w_3 w_4 \ldots)$, let's model $P(w_5 | w_1 w_2 w_3 w_4)$

- How can we use the "next word predictor" view to model $P(W)$?
  - Bayes rule!
  - Recall: $P(A, B) = P(A)P(B|A)$
  - More generally (CHAIN RULE)

$$P(w_1\ w_2 \ldots w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1 w_2) \ldots P(w_n|w_1 w_2 \ldots w_{n-1})$$

Language Models: Chain Rule

$$P(w_1 \dots w_n) = \prod_i P(w_i \mid w_1 \dots w_{i-1})$$

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is)

× P(transparent|its water is so)

- Could we just count and divide?

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

- No! Too many possible sentences!
- We'll never see enough data for estimating these

Language Models: Markov Assumption

- Markov: ya'll looking too far. A few previous words are enough.

$$P(\texttt{blue}|\texttt{The water of Walden Pond is so beautifully})$$

$$\approx \quad P(\texttt{blue}|\texttt{beautifully})$$

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

Bigram Markov Assumption

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

Instead of: $\prod_{k=1}^{n} P(w_k|w_{1:k-1})$

In connection with student riots in 1908, professors and lecturers of St. Petersburg University were ordered to monitor their students. Markov refused to accept this decree, and he wrote an explanation in which he declined to be an "agent of the governance". Markov was removed from further teaching duties at St. Petersburg University, and hence he decided to retire from the university.

Simple Language Models: Unigram Model

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Some automatically generated sentences from two different unigram models

```
To him swallowed confess hear both . Which . Of save on trail
for are ay device and rote life have

Hill he late speaks ; or ! a more to leg less first you enter

Months the my and issue of year foreign new exchange's September

were recession exchange new endorsed a acquire to six executives
```

Simple Language Models: Bigram Model

$$P(w_i \mid w_1 w_2 \dots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

Some automatically generated sentences rom two different unigram models

```
Why dost stand forth thy canopy, forsooth; he is this palpable hit
the King Henry. Live king. Follow.

What means, sir. I confess she? then all sorts, he is trim, captain.

Last December through the way to preserve the Hudson corporation N.
B. E. C. Taylor would seem to complete the major central planners
one gram point five percent of U. S. E. has already old M. X.
corporation of living

on information such as more frequently fishing to keep her
```

More generally: N-gram models

Estimating Bigram Probabilities

An example

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $\quad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $\quad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $\quad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $\quad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

Estimating Bigram Probabilities

"I want to eat Chinese food lunch spend"

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Estimating Bigram Probabilities

"I want to eat Chinese food lunch spend"

Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Computing likelihood using bigrams

$$
\begin{aligned}
&P(\text{<s> I want english food </s>}) = \\
&P(\text{I}|\text{<s>}) \\
&\qquad \times\ P(\text{want}|\text{I}) \\
&\qquad \times\ P(\text{english}|\text{want}) \\
&\qquad \times\ P(\text{food}|\text{english}) \\
&\qquad \times\ P(\text{</s>}|\text{food}) \\
&\qquad =\ .000031
\end{aligned}
$$

What do n-grams reveal .... ?

$P(\text{english}|\text{want}) = .0011$

$P(\text{chinese}|\text{want}) = .0065$

$P(\text{to}|\text{want}) = .66$

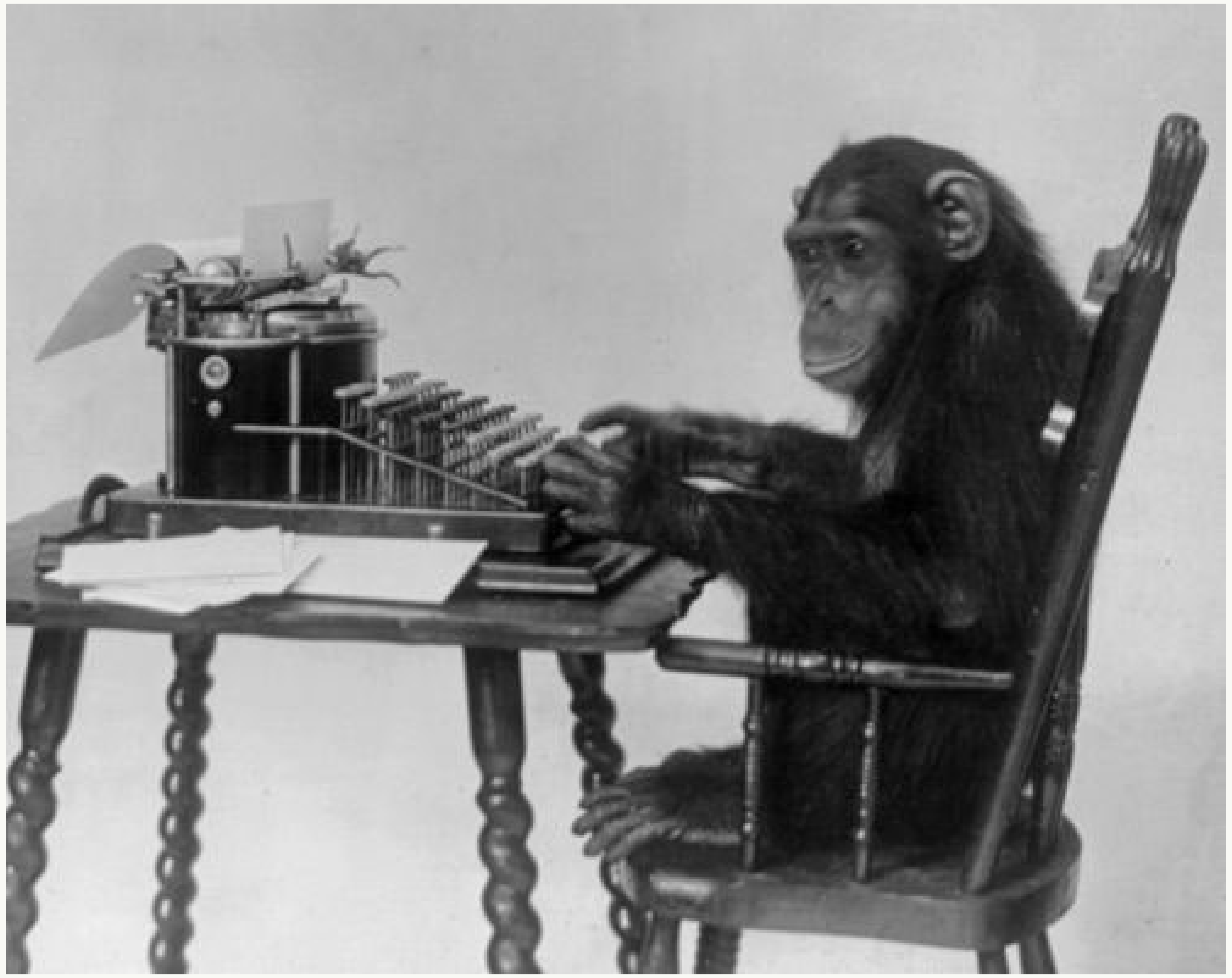
$P(\text{eat} \mid \text{to}) = .28$

$P(\text{food} \mid \text{to}) = 0$

$P(\text{want} \mid \text{spend}) = 0$

$P\,(\text{i} \mid <\text{s}>) = .25$

Aside: Infinite Monkey Theorem



The **infinite monkey theorem** states that a monkey hitting keys independently and at random on a typewriter keyboard for an infinite amount of time will almost surely type any given text, including the complete works of William Shakespeare.[a]

More precisely, under the assumption of independence and randomness of each keystroke, the monkey would almost surely type every possible finite text an infinite number of times.

N-gram Language Model for Shakespeare

Can we train a Language Model on Shakespeare's work

and generate new "Shakespeare-like" sentences ?

Approximating Shakespeare

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; –It cannot be but so.

Shakespeare as corpus

N=884,647 tokens, V=29,066

Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams.

- So 99.96% of the possible bigrams were never seen (have zero entries in the table)
- That sparsity is even worse for 4-grams, explaining why our sampling generated actual Shakespeare.

Can you guess the author? These 3-gram sentences are sampled from an LM trained on who?

1) They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and gram Brazil on market conditions

2) This shall forbid it should be branded, if renown made it empty.

3) “You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.

Larger n-grams ...

4-grams, 5-grams

Large datasets of large n-grams have been released

- N-grams from Corpus of Contemporary American English (COCA) 1 billion words (Davies 2020)
- Google Web 5-grams (Franz and Brants 2006) 1 trillion words)
- Efficiency: quantize probabilities to 4-8 bits instead of 8-byte float

Newest model: infini-grams (∞-grams) (Liu et al 2024)

- No precomputing! Instead, store 5 trillion words of web text in **suffix arrays**. Can compute n-gram probabilities with any n!

Toolkit: "KenLM": https://kheafield.com/code/kenlm

Google N-Gram Release: August 2006

Home > Blog >

All Our N-gram are Belon
You

August 3, 2006 · Posted by Alex Franz and Thorsten Brants, Google Machine Translation

```
File sizes: approx. 24 GB compressed (gzip'ed) text

Number of tokens:    1,024,908,267,229
Number of sentences:    95,119,665,584
Number of unigrams:         13,588,391
Number of bigrams:         314,843,401
Number of trigrams:        977,069,902
Number of fourgrams:     1,313,818,354
Number of fivegrams:     1,176,470,663
```

We believe … assive
amounts o … omising
direction o … ps, no
matter hov … ve decided
to share th … ords of
running te … es that
appear at l … s that
appear les

```
serve as the initial 5331
serve as the initiating 125
serve as the initiation 63
serve as the initiator 81
serve as the injector 56
serve as the inlet 41
serve as the inner 87
serve as the input 1323
serve as the inputs 189
serve as the insertion 49
serve as the insourced 67
serve as the inspection 43
serve as the inspector 66
serve as the inspiration 1390
```

How to Evaluate Models? Model A vs B?

- Does our language model prefer good sentences to bad ones?
- Does it assign higher probability to "real" or "frequently observed" sentences ?
- Does it assign lower probability to "ungrammatical" or "rarely observed" sentences ?

Extrinsic vs Intrinsic Evaluation

Intrinsic Evaluation

"A better model assigns a higher probability to the word that actually occurs"

- Collect "test" sentences (i.e. new sentences not in training corpus)
  - How well can the model predict the next word?

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

....

fried rice 0.0001

....

and 1e-100

Unigrams are TERRIBLE at this game. WHY?

Intrinsic Evaluation: Perplexity

"The best model predicts the best on an unseen test set"

- A model that gives the highest $P(sentence)$

Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

For bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

Intrinsic Evaluation: Perplexity

"The best model predicts the best on an unseen test set"

- A model that gives the highest $P(sentence)$

Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Problems with N-gram models

- N-grams can't handle **long-distance dependencies**:

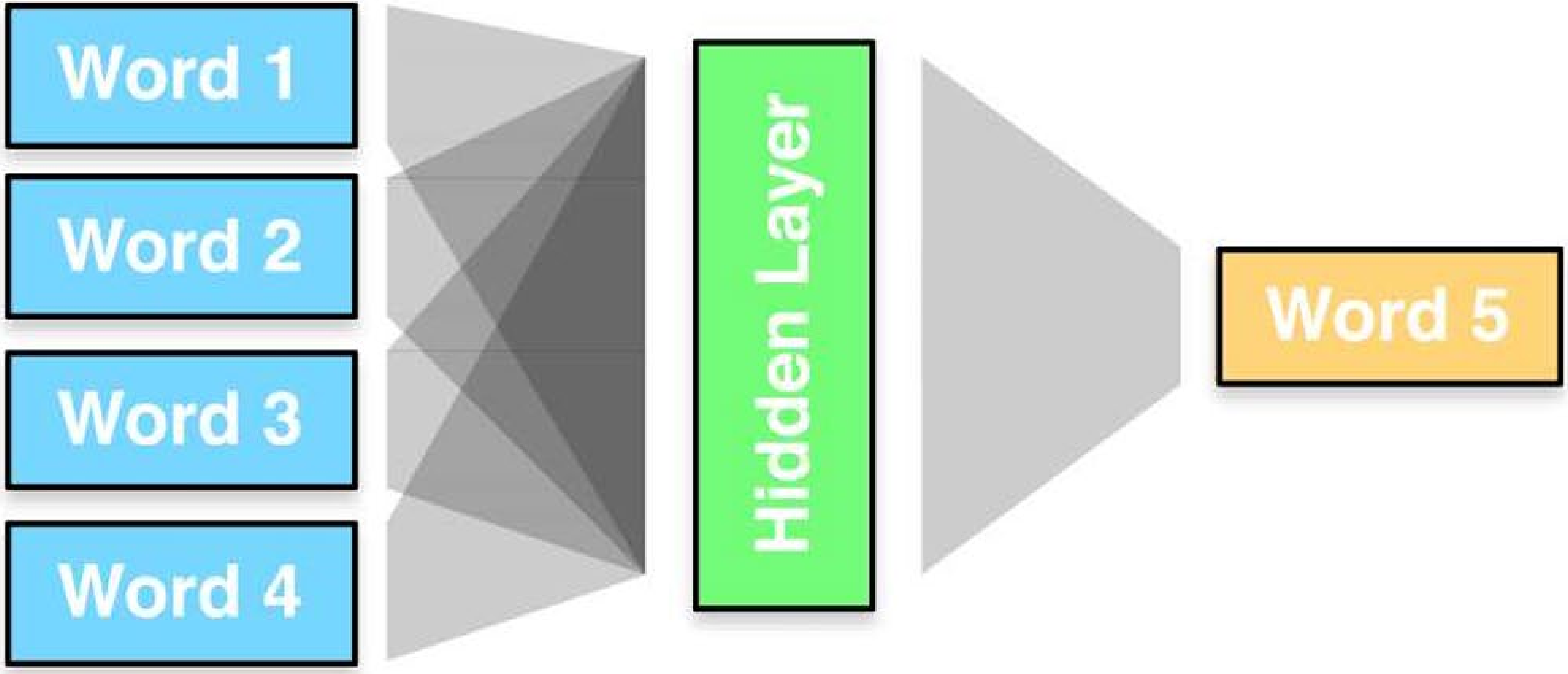
"**The soups** that I made from that new cookbook I bought yesterday **were** amazingly delicious."

- N-grams don't do well at modeling new sequences with similar meanings

The solution: **Large language models**

- can handle much longer contexts
- because of using embedding spaces, can model synonymy better, and generate better novel strings

Towards Neural Language Models



Figures by Philipp Koehn (JHU)

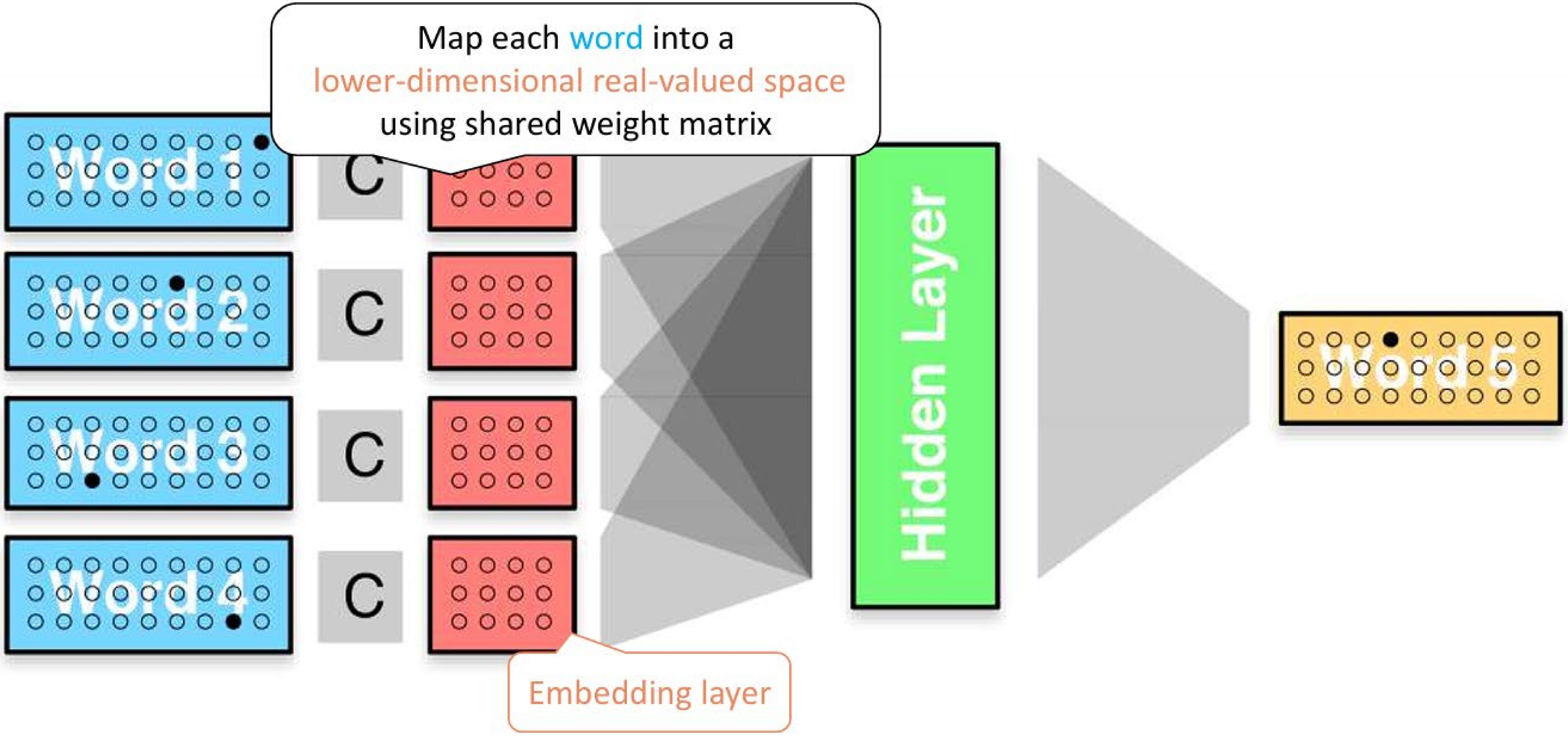
Representing Words (recap)

- "one hot vector"

  dog = [0, 0, 0, 0, 1, 0, 0, 0 ...]
  cat = [0, 0, 0, 0, 0, 0, 1, 0 ...]
  eat = [0, 1, 0, 0, 0, 0, 0, 0 ...]

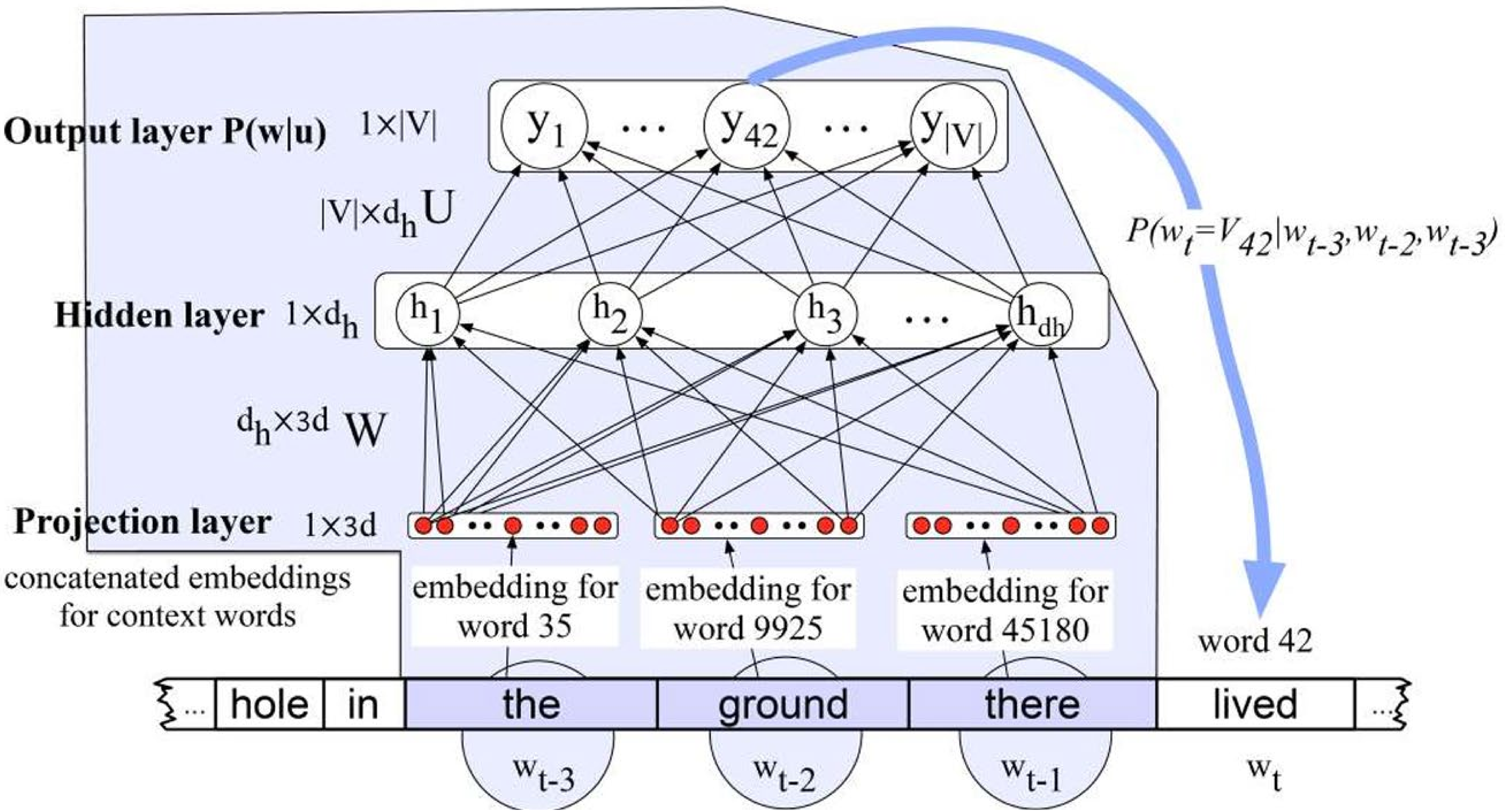
- That's a large vector! practical solutions:
  - limit to most frequent words (e.g., top 20000)
  - cluster words into classes
  - Word2vec using gradient descent ....

Language Modeling with Feedforward Neural Networks

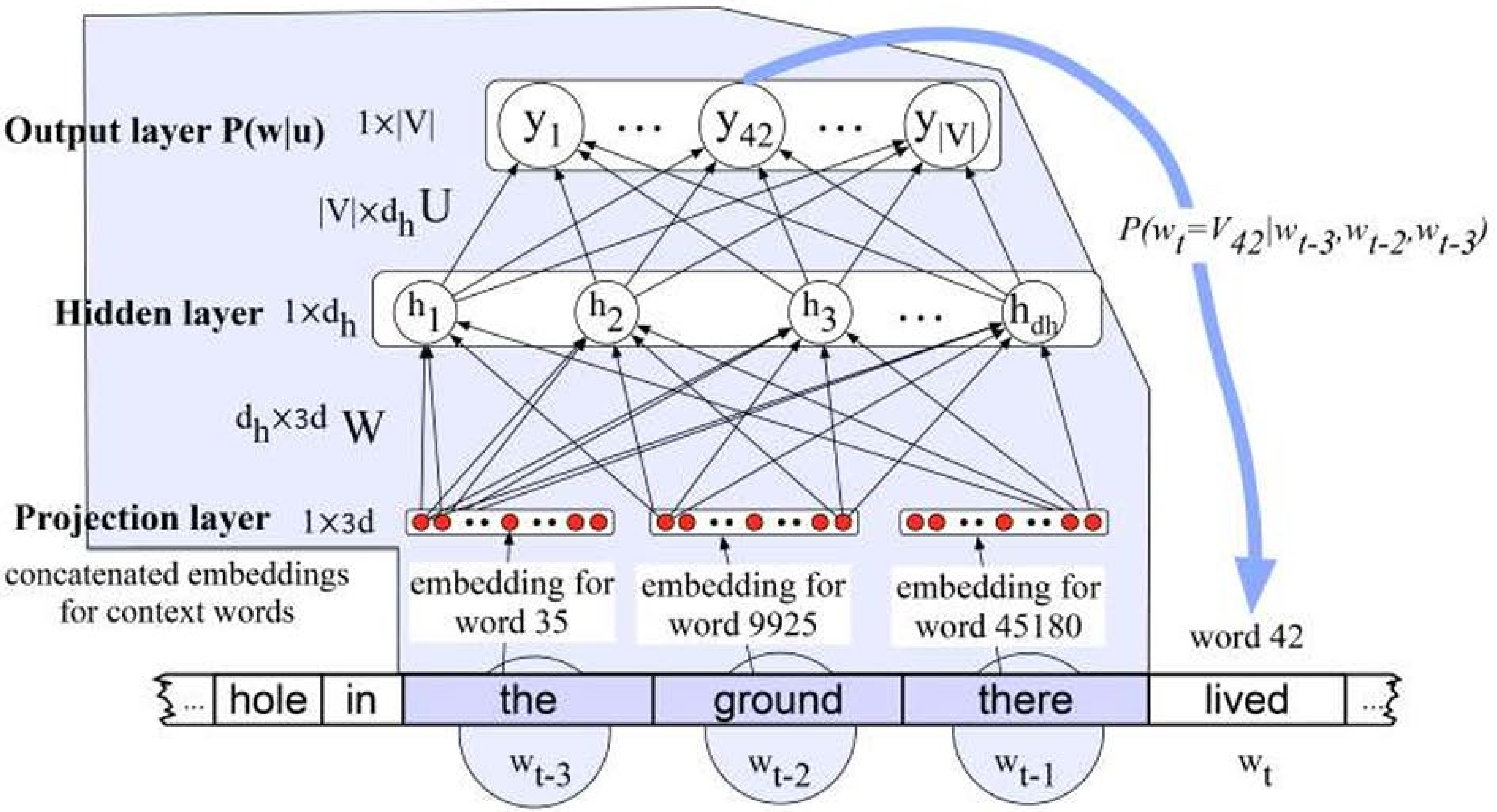


Bengio et al. 2003

Example: Prediction with a Feedforward LM



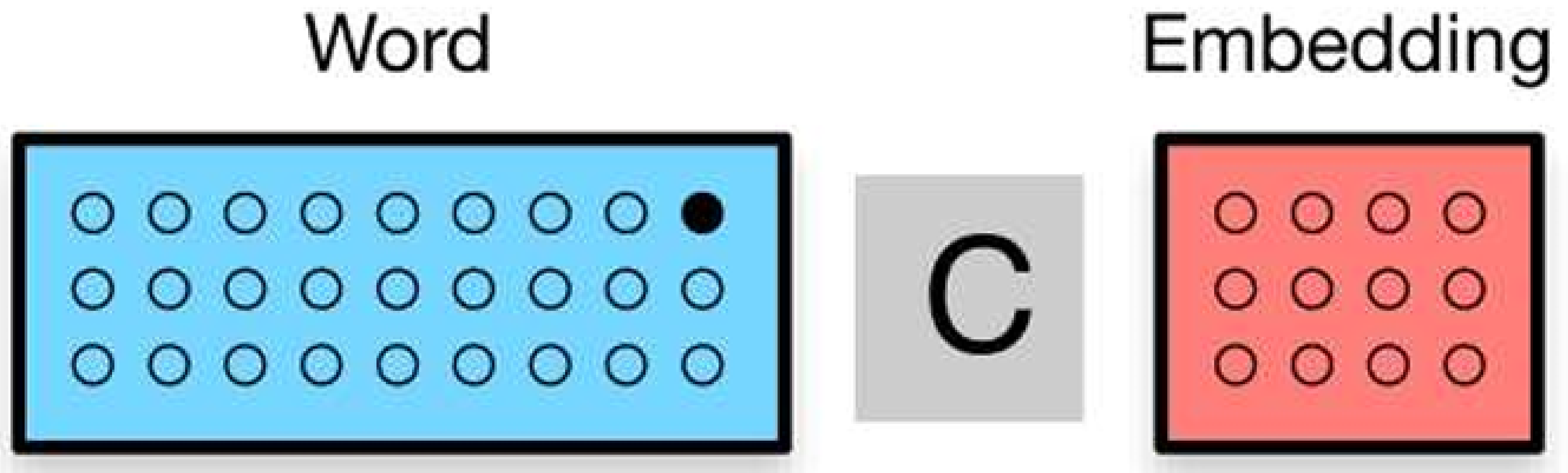
Example: Prediction with a Feedforward LM



$$e = (Ex_1, Ex_2, ..., Ex)$$
$$h = \sigma(We + b)$$
$$z = Uh$$
$$y = \mathrm{softmax}(z)$$

Note: bias omitted in figure

Word Embeddings: a product of neural LMs



- Words that occurs in similar contexts tend to have similar embeddings
- Embeddings capture many usage regularities
- Useful features for many NLP tasks

Word Embeddings Capture Useful Regularities

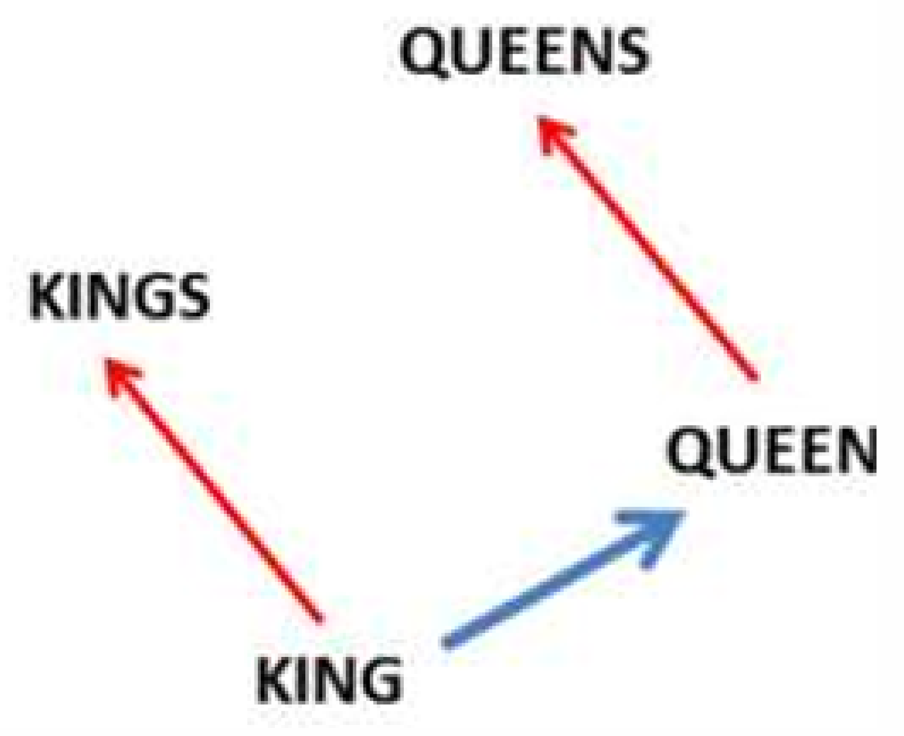
Morpho-Syntactic

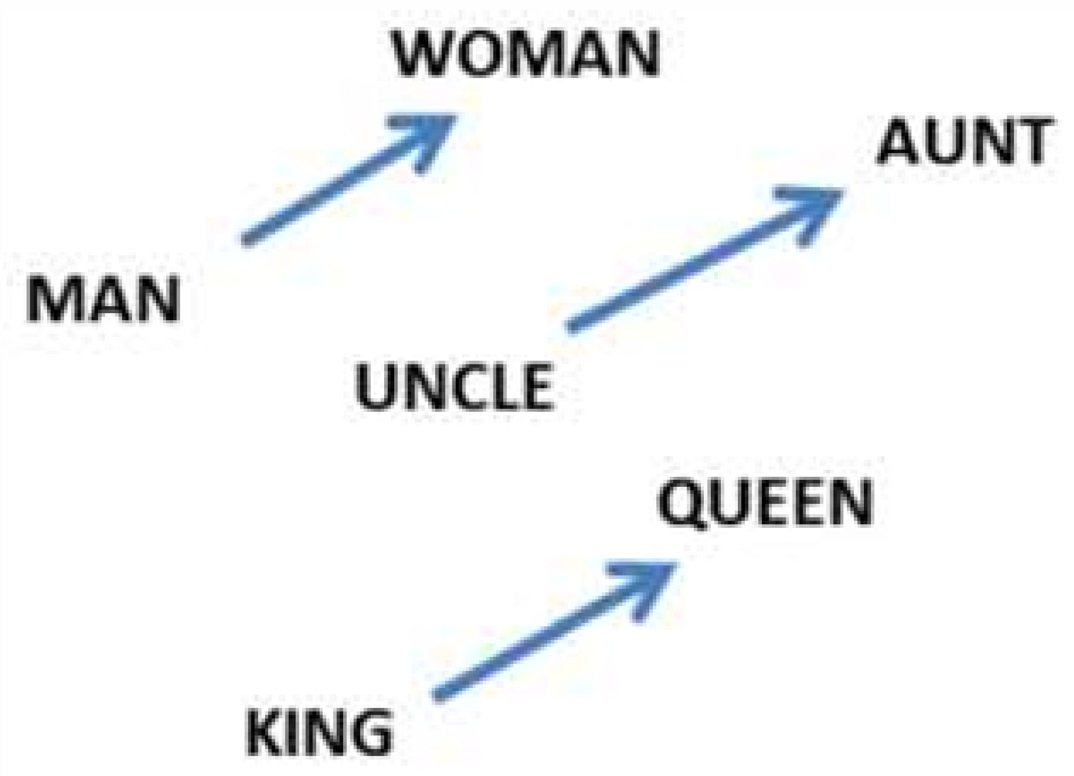
- Adjectives: base form vs. comparative
- Nouns: singular vs. plural
- Verbs: present tense vs. past tense

[Mikolov et al. 2013]

Semantic

- Word similarity/relatedness
- Semantic relations
- But tends to fail at distinguishing
  - Synonyms vs. antonyms
  - Multiple senses of a word





CMSC 475/675 Neural Networks

Lecture 10

Addendum:

Tokenization

UMBC

Tokenization

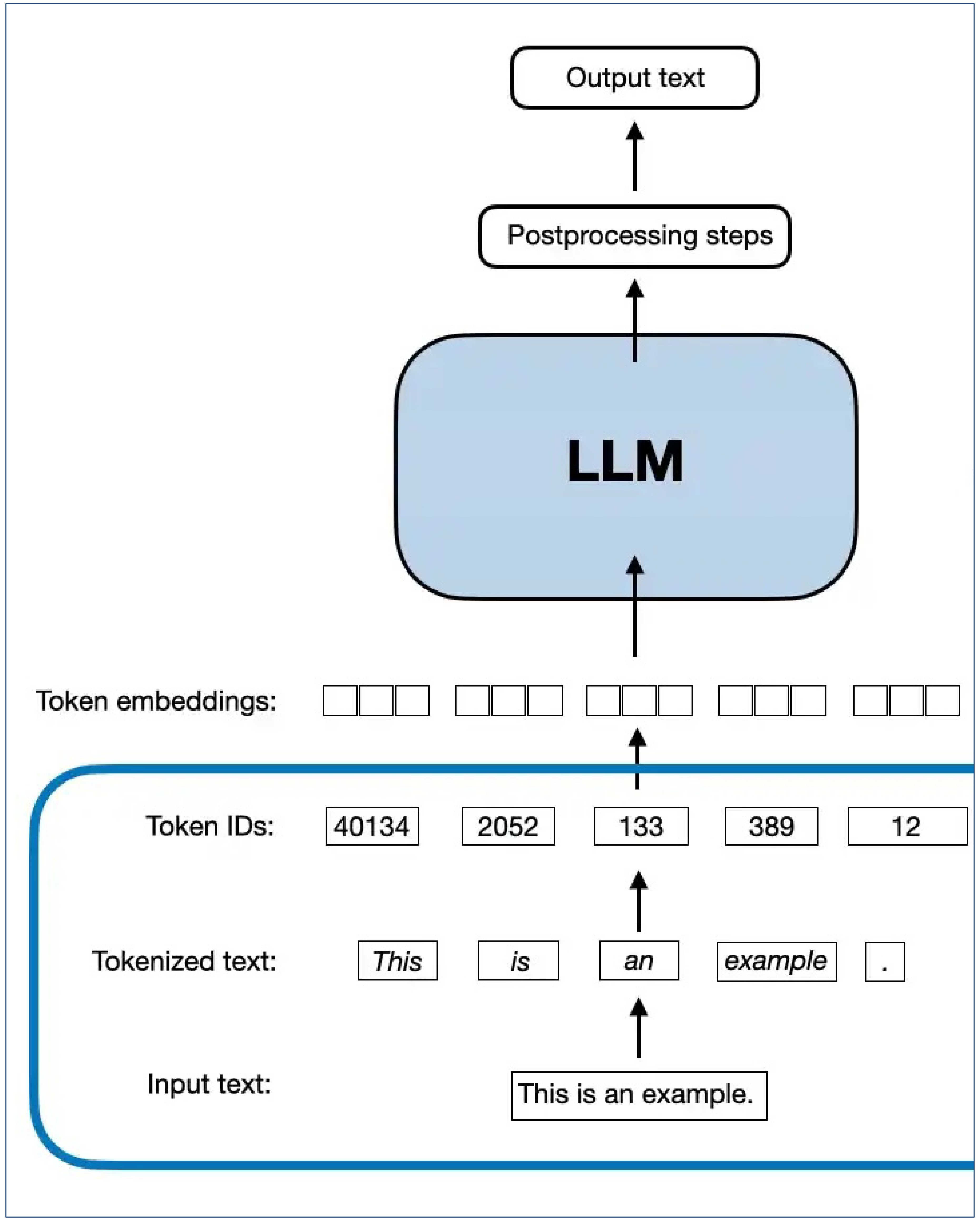
(tokens are units of language)

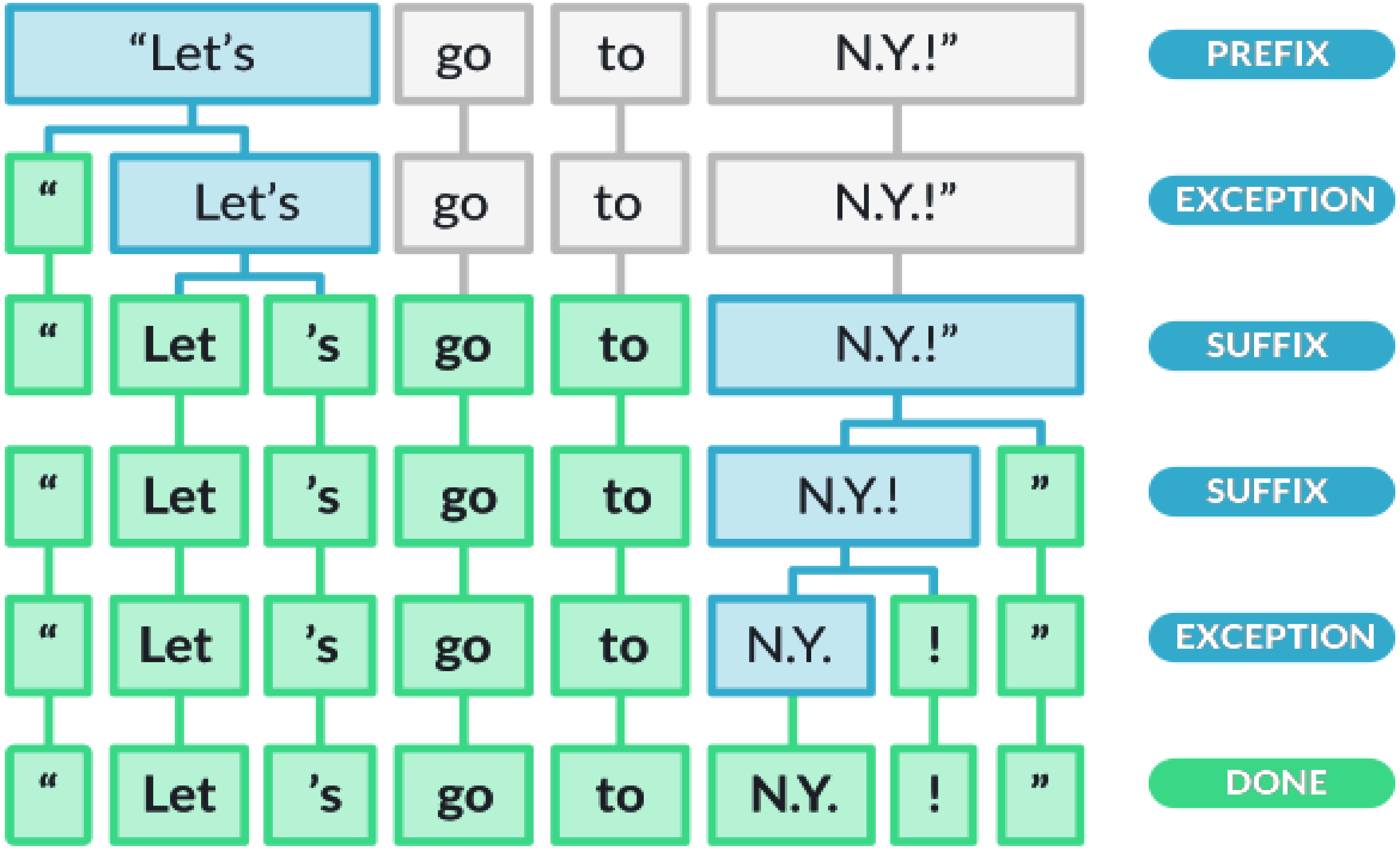
(think “atoms”)

Revisiting Words … Tokenization

- Text broken into smaller units = tokenization
  - Tokens are discrete text units
  - Different ways to do tokenization (words, morphemes, characters)
- Typical preprocessing step for NLP tasks
- Why tokenize?
  - Text encoding is necessary – better to encode smaller units than large chunks
  - Deal with whitespaces
  - Building up "meaning" in a bottom-up way

Tokenization Example





Tokenization Variants

Paragraph Tokenization

- Break documents into paragraphs
- Rarely used

Sentence Tokenization

- Break documents into sentences

Word Tokenization

- Break documents into words
- Very common

Sub-Word Tokenization

- Break documents into morphemes
- Very common

Character Tokenization

- Break documents into individual characters

Word Tokenization

- Very popular – applied as a preprocessing step in many NLP tasks
- Considers dictionary words and delimiters as “tokens” in the vocab

```
What is the tallest building
```

➔ `“What”, “is”, “the”, “tallest”, “building”, “?”`

Sub-Word (Morpheme) Tokenization

- Fine-grained than word tokenization
  - Breaks text into words
  - Further breaks words into smaller units based on root, prefix, suffix, etc
  - Based on linguistic rules

- Important for “flective” languages
  - Words have many forms with prefixes and suffixes that change the meaning of the word

```
What is the tallest building

➔ “What”, “is”, “the”, “tall”, “est”, “build”, “ing”, “?”
```

Byte-Pair Encoding (BPE) Tokenization

- Initially developed for compressing text (Gage 1994)
- Modified for use in Large Language Models (GPT etc.)
- It's an iterative algorithm

Byte-Pair Encoding (BPE) Tokenization

- Initial Steps:
  - Begin by creating a unique set of words in corpus
  - Set a desired token vocabulary size: D
    - Example: let's say the corpus has 5 words: "hug", "pug", "pun", "bun", "hugs"
    - The "base vocab" is the set of all characters: *b, g, h, n, p, s, u*
- "Merge"
  - Add new tokens until desired vocab size is reached by learning "merges"
  - At any step during tokenization, search for the most frequent pair of consecutive tokens and merge it to create a new token
    - *Merges* are rules to merge two elements of the existing vocabulary to create a new token
    - The first *merge* creates tokens with 2 characters -- as algo proceeds longer tokens are created

BPE Example

CORPUS: "hug", "pug", "pun", "bun", "hugs"
Base vocab: *b, g, h, n, p, s, u*

https://huggingface.co/learn/llm-course/en/chapter6/5

- Get Frequencies:

```
("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
```

- Pair ("u", "g") is the most frequent (appears 20 times in total)
  - Merge: ("u", "g") → "ug"

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug"]
Frequency: ("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)
```

- Pair ("u", "n") is the most frequent (appears 16 times). Merge to get "un"

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un"]
Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("h" "ug" "s", 5)
```

- Pair ("h", "ug") is the most frequent …. Merge to get "hug"

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]
Corpus: ("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)
```

Back to NN for LM

(Transformers!)