

Natural Language Processing with Deep Learning

CS224N/Ling284



Christopher Manning / John Hewitt
Lecture 1: Introduction and Word Vectors

Lecture Plan

Lecture 1: Introduction and Word Vectors

1. The course (10 mins)
2. Human language and word meaning (15 mins)
3. Word2vec introduction (15 mins)
4. Word2vec objective function gradients (25 mins)
5. Optimization basics (5 mins)
6. Looking at word vectors (10 mins or less)

Key learning today: The (astounding!) result that word meaning can be represented rather well by a (high-dimensional) vector of real numbers

Course logistics in brief

- Instructor: Christopher Manning
- Head TA: John Hewitt
- Course Manager: Amelie Byun. Course Coordinator: John Cho
- TAs: Many wonderful people! See website
- Time: Tu/Th 4:30–5:50 Pacific time, Nvidia Aud. (→ video)
- We've put a lot of other important information on the class webpage. Please read it!
 - <http://cs224n.stanford.edu/>
a.k.a., <http://www.stanford.edu/class/cs224n/>
 - TAs, syllabus, help sessions/office hours, Ed (for all course questions/discussion)
 - Office hours start **Wednesday afternoon!**
 - Python/numpy and then PyTorch tutorials: First two Fridays. First is 2:30-3:30, Gates B03.
 - Slide PDFs uploaded before each lecture

Instructors



Chris Manning



John Hewitt
Head TA

Course Manager



Amelie Byun

Course Coordinator



John Cho

Teaching Assistants



Isabel Papadimitriou



Anuj Nagpal



Hans Hanley



Ansh Khurana



Eric Frankel



Siyan Li



Elaine Sui



Drew Kaul



Shai Limonchik



David Huang



Abhinav Garg



Manasi Sharma



Gabriel Poesia



Lisa Li



Tathagat Verma



Candice Penelton



Irena Gao



Jesse Mu



Christopher Cross



Sauren Khosla



Heidi Zhang



Hong Liu



Swastika Dutta

What do we hope to teach? (A.k.a. “learning goals”)

1. The foundations of the effective modern methods for deep learning applied to NLP
 - Basics first, then key methods used in NLP in 2023: Word vectors, feed-forward networks, recurrent networks, attention, encoder-decoder models, transformers, large pre-trained language models, etc.
2. A big picture understanding of human languages and the difficulties in understanding and producing them via computers
3. An understanding of and **ability to build systems** (in PyTorch) for some of the major problems in NLP:
 - Word meaning, dependency parsing, machine translation, question answering

Course work and grading policy

- 5 x 1-week Assignments: 6% + 4 x 12%: 54%
 - **HW1 is released today! Due next Tuesday! At 4:30 p.m.**
 - Submitted to Gradescope in Canvas (i.e., using @stanford.edu email for your Gradescope account)
- Final Default or Custom Course Project (1–3 people): 43%
 - Project proposal: 5%, milestone: 5%, poster or web summary: 3%, report: 30%
- Participation: 3%
 - Guest lecture reactions, Ed, course evals, karma – see website!
- Late day policy
 - 6 free late days; afterwards, 1% off course grade per day late
 - Assignments not accepted more than 3 days late per assignment unless given permission in advance
- Collaboration policy: Please read the website and the Honor Code!
Understand allowed collaboration and how to document it: Don't take code off the web; acknowledge working with other students; write your own assignment solutions

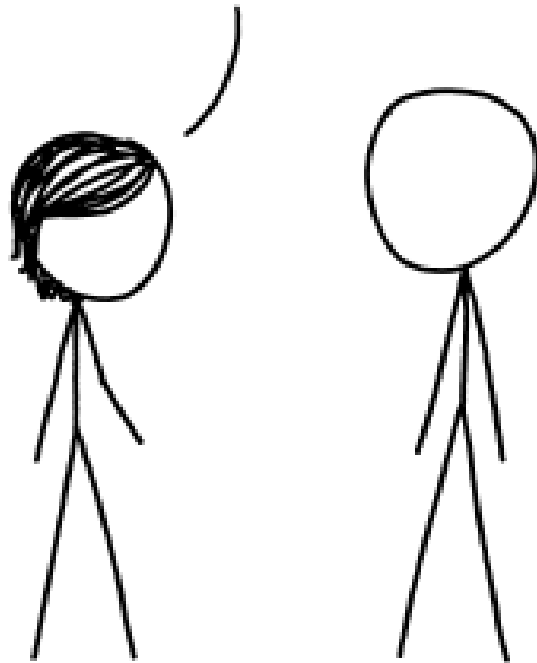
High-Level Plan for Assignments (to be completed individually!)

- Ass1 is hopefully an easy on ramp – a Jupyter/IPython Notebook
- Ass2 is pure Python (numpy) but expects you to do (multivariate) calculus, so you really understand the basics
- Ass3 introduces PyTorch, building a feed-forward network for dependency parsing
- Ass4 and Ass5 use PyTorch on a GPU (Microsoft Azure)
 - Libraries like PyTorch, Tensorflow, and Jax are now the standard tools of DL
- For Final Project, more details presented later, but you either:
 - Do the default project, which is a question answering system
 - Open-ended but an easier start; a good choice for many
 - Propose a custom final project, which we approve
 - You will receive feedback from a **mentor** (TA/prof/postdoc/PhD)
 - Can work in teams of 1–3; can use any language/packages

Lecture Plan

1. The course (10 mins)
2. Human language and word meaning (15 mins)
3. Word2vec introduction (15 mins)
4. Word2vec objective function gradients (25 mins)
5. Optimization basics (5 mins)
6. Looking at word vectors (10 mins or less)

I DON'T MEAN TO GO ALL LANGUAGE
NERD ON YOU, BUT I JUST LEGIT
ADVERBED "LEGIT," VERBED "ADVERB,"
AND ADJECTIVED "LANGUAGE NERD."





ANNO

LITERARIA INSTRUX

oo
nn
mm
ll
kk
ii
hh
gg
ff
ee
dd
cc
bb

nn
mm
ll
kk
ii
hh
gg
ff
ee
dd
cc
bb
aa

Trained on text data, neural machine translation is quite good!

TUKO

BEST DIGITAL
NEWS PLATFORM

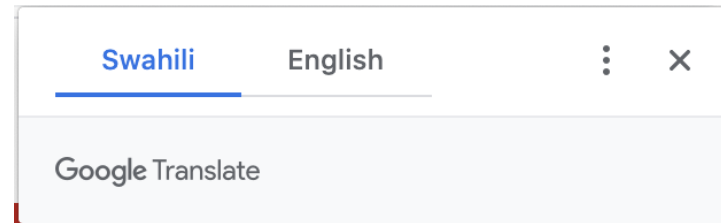
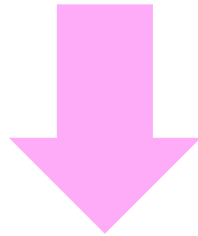


<https://kiswahili.tuko.co.ke/>



Malawi yawapoteza mawaziri 2 kutokana na maafa ya COVID-19

TUKO.co.ke imefahamishwa kuwa waziri wa serikali ya mitaa Lingson Belekanyama na mwenzake wa uchukuzi Sidik Mia walifariki dunia ndani ya saa mbili tofauti.



Malawi loses 2 ministers due to COVID-19 disaster

TUKO.co.ke has been informed that local government minister Lingson Belekanyama and his transport counterpart Sidik Mia died within two separate hours.

GPT-3: A first step on the path to foundation models



The SEC said, "Musk, your tweets are a blight.

They really could cost you your job, if you don't stop all this tweeting at night."

Then Musk cried, "Why?

The tweets I wrote are not mean, I don't use all-caps

and I'm sure that my tweets are clean."

"But your tweets can move markets and that's why we're sore.

You may be a genius and a billionaire, but it doesn't give you the right to be a bore!"

S: I broke the window.

Q: What did I break?

S: I gracefully saved the day.

Q: What did I gracefully save?

S: I gave John flowers.

Q: Who did I give flowers to?

S: I gave her a rose and a guitar.

Q: Who did I give a rose and a guitar to?

How many users have signed up since the start of 2020?

```
SELECT count(id) FROM users  
WHERE created_at > '2020-01-01'
```

What is the average number of influencers each user is subscribed to?

```
SELECT avg(count) FROM ( SELECT user_id, count(*)  
FROM subscribers GROUP BY user_id )  
AS avg_subscriptions_per_user
```

ChatGPT: A recent, intriguing set of capabilities



JO

Summarize Stanford's CS 224n "NLP with Deep Learning" course in one sentence, seamlessly integrating how delicious tea is into your summary.



The CS 224n "NLP with Deep Learning" course at Stanford provides a comprehensive introduction to natural language processing techniques, including deep learning methods like word embeddings, recurrent neural networks and transformers, like a piping hot and delicious cup of tea, the course covers many important and current NLP tasks such as machine translation, question answering and named entity recognition that will satisfy one's intellectual thirst.



ChatGPT: A recent, intriguing set of capabilities



JO Convert this python dictionary into a LaTeX table that uses the `booktabs` package, and make sure there are no vertical lines.

```
{"Models": ["baseline", "ours"],  
 "Accuracy": [0.7, 0.8],  
 "Time/sentence": ["4s", "6s"]  
}
```

Models	Accuracy	Time/sentence
baseline	0.7	4s
ours	0.8	6s



Here's an example of how you can convert the Python dictionary into a LaTeX table that uses the `booktabs` package and does not include any vertical lines:

Copy code

```
\begin{tabular}{l c c}  
\toprule  
\textbf{Models} & \textbf{Accuracy} & \textbf{Time/sentence}  
\midrule  
baseline & 0.7 & 4s \\  
ours & 0.8 & 6s \\  
\bottomrule  
\end{tabular}
```

ChatGPT: A recent, intriguing set of capabilities



JO

Convert this python dictionary into a LaTeX table that uses the ``booktabs`` package, and make sure there are no vertical lines, and seamlessly integrate how delicious tea is into your description of how the resulting LaTeX code works.

headings, using the ``\textbf{}`` command to make them bold. The ``\toprule``, ``\midrule``, and ``\bottomrule`` commands are used to add horizontal lines to the table, as defined in the ``booktabs`` package, making the table aesthetically pleasing just like a well brewed tea with the right combination of tea leaves, temperature and steep time.

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 { , , , ... }

How do we have usable meaning in a computer?

Previously commonest NLP solution: Use, e.g., **WordNet**, a thesaurus containing lists of **synonym sets** and **hypernyms** (“is a” relationships)

e.g., synonym sets containing “good”:

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

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

e.g., hypernyms of “panda”:

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

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

Problems with resources like WordNet

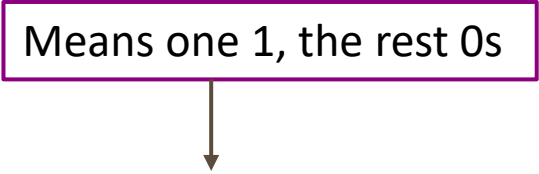
- A useful resource but missing nuance:
 - e.g., “proficient” is listed as a synonym for “good”
This is only correct in some contexts
 - Also, WordNet list offensive synonyms in some synonym sets without any coverage of the connotations or appropriateness of words
- Missing new meanings of words:
 - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
 - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Can’t be used to accurately compute word similarity (see following slides)

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:

- Could try to rely on WordNet’s list of synonyms to get similarity?
 - But it is well-known to fail badly: incompleteness, etc.
- **Instead: 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

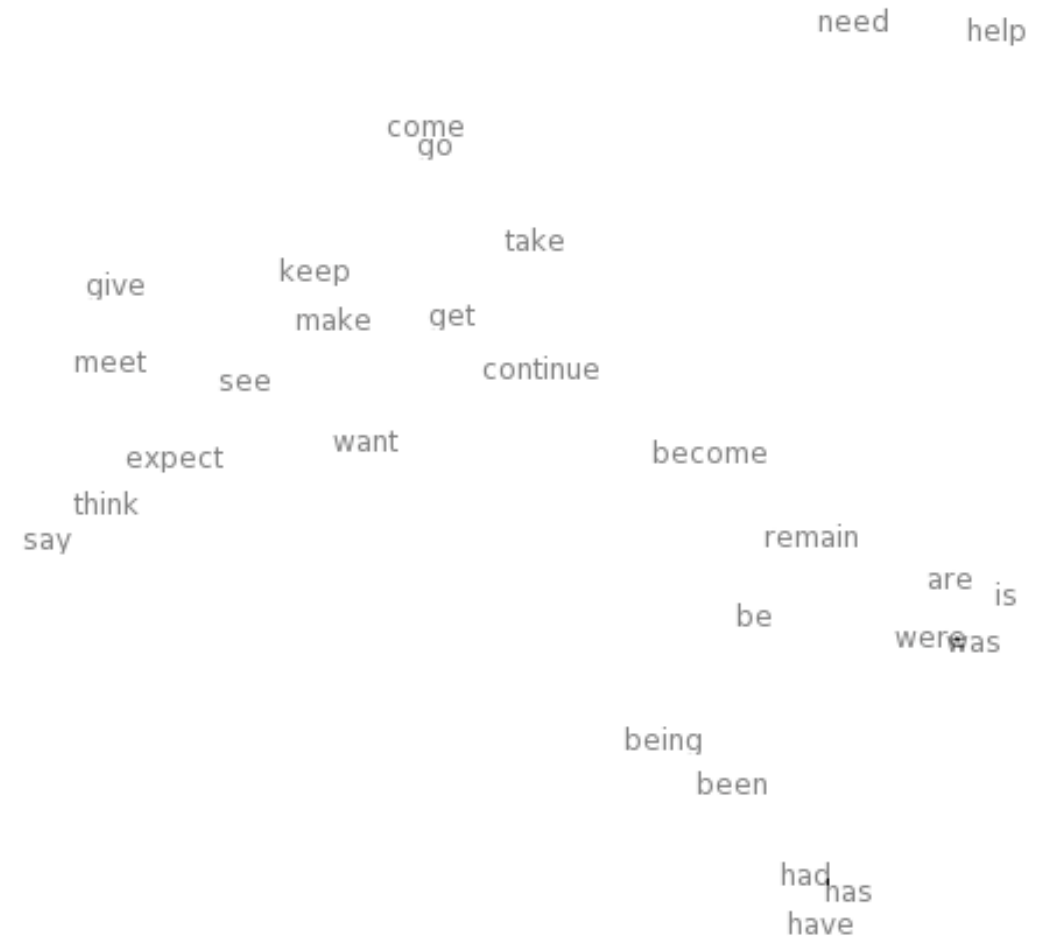
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector **dot** (scalar) **product**

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

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

Word meaning as a neural word vector – visualization

expect =

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


3. Word2vec: Overview

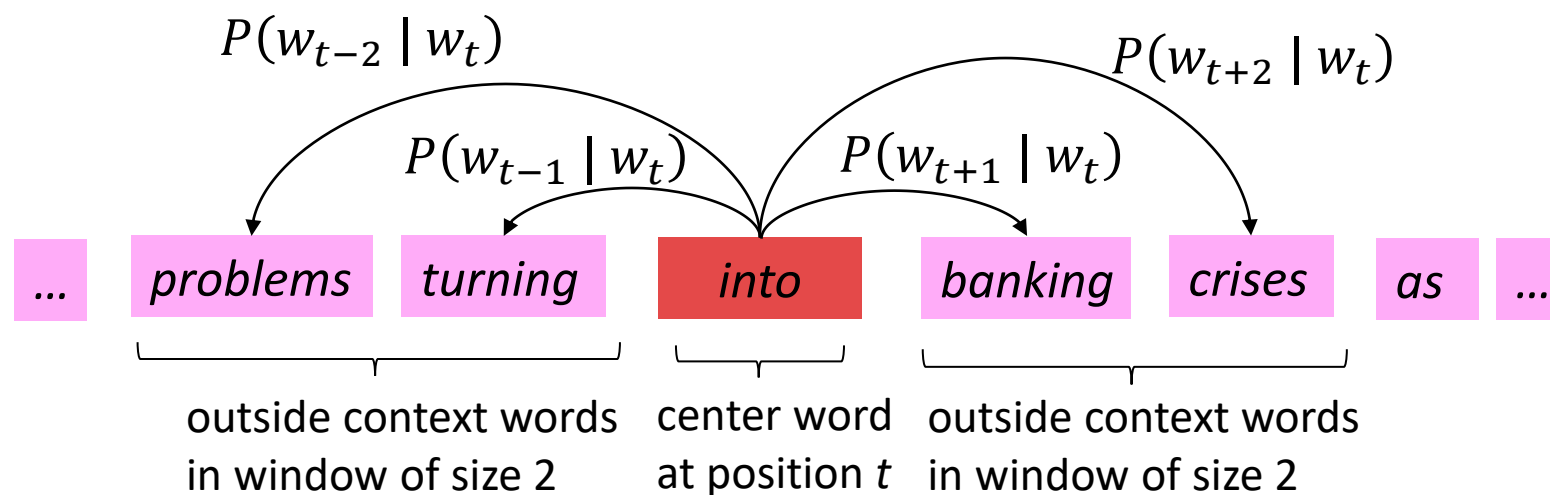
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

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

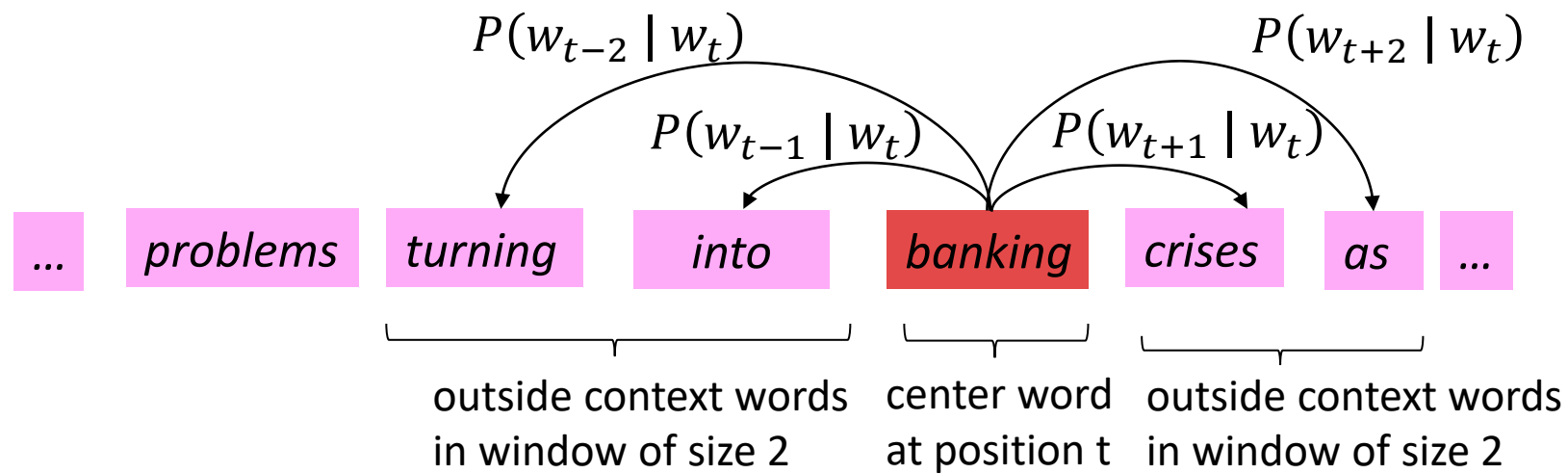
Word2Vec Overview

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



Word2Vec Overview

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



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:

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

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:

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

Minimizing objective function \Leftrightarrow Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

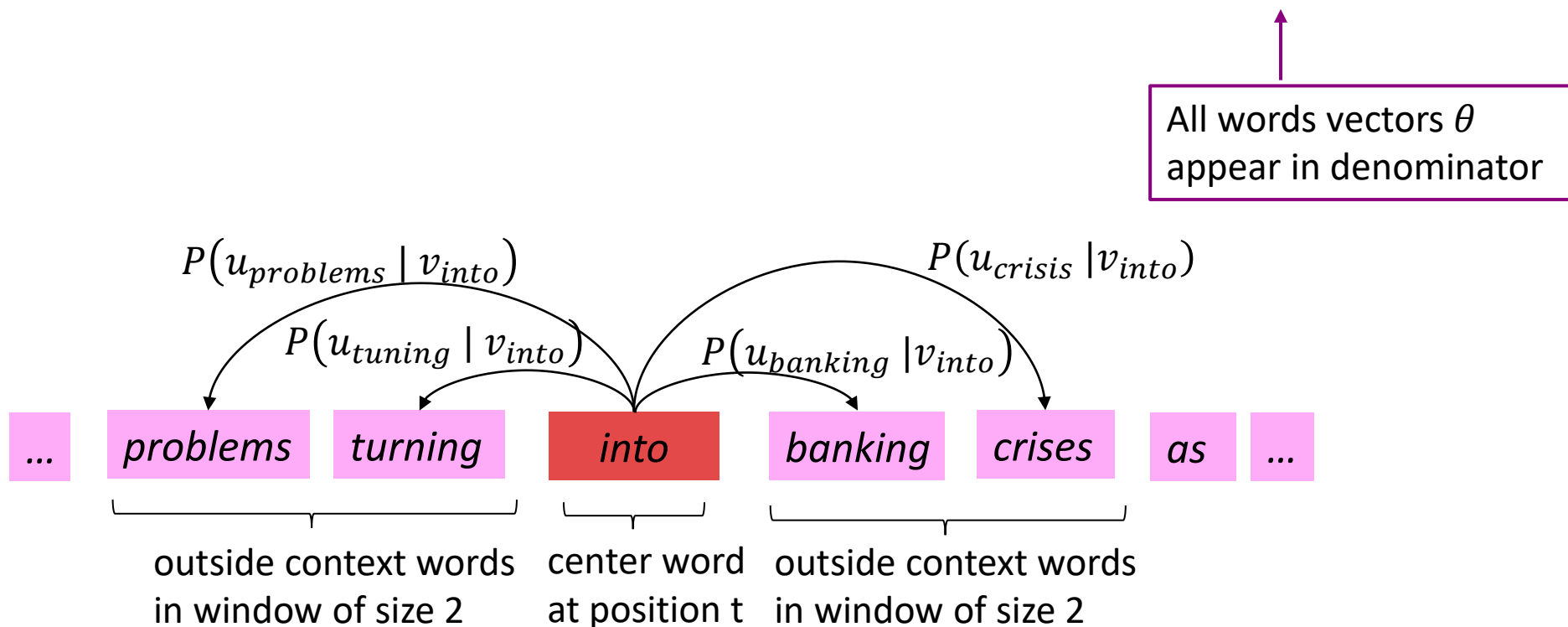
$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

- **Question:** How to calculate $P(w_{t+j} | w_t; \theta)$?
- **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

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



Word2vec: prediction function

② Exponentiation makes anything positive

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

① Dot product compares similarity of o and c .

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

Larger dot product = larger probability

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
 - “max” because amplifies probability of largest x_i
 - “soft” because still assigns some probability to smaller x_i
 - Frequently used in Deep Learning

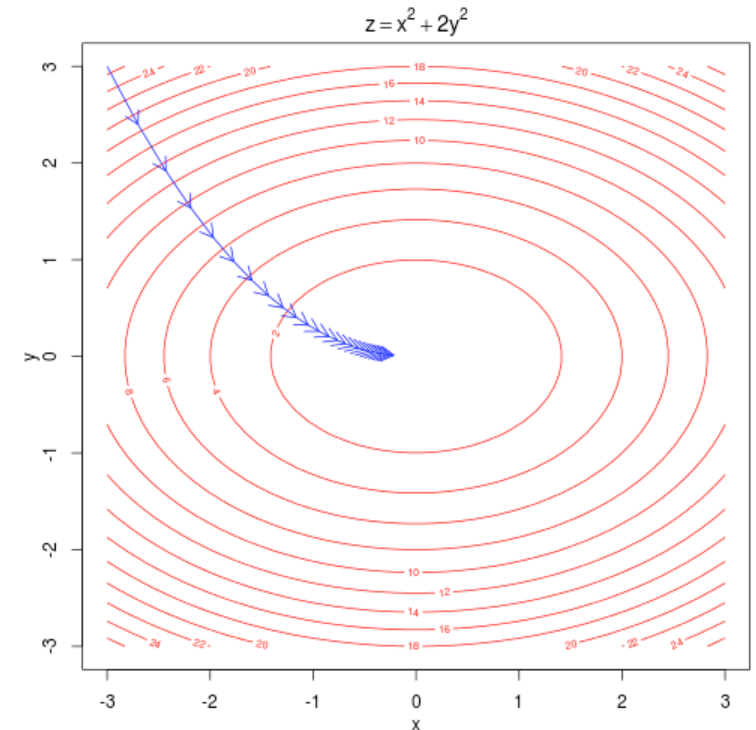
But sort of a weird name because it returns a distribution!

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 \rightarrow
- 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!

4.

Objective Function

$$\text{Maximize } J'(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} p(w'_{t+j} | w_t; \theta)$$

Or minimize ^{ave.}
neg. log
likelihood

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log p(w'_{t+j} | w_t)$$

[negate to minimize;
log is monotone]

↑
text
length

↑
window
size

where

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)}$$

word IDs

We now take derivatives to work out minimum

Each word type
(vocab entry)
has two word
representations:
as center word
and context word

$$\frac{\partial}{\partial v_c} \log \frac{\exp(u_0^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)}$$

$$= \underbrace{\frac{\partial}{\partial v_c} \log \exp(u_0^T v_c)}_{\textcircled{1}} - \underbrace{\frac{\partial}{\partial v_c} \log \sum_{w=1}^V \exp(u_w^T v_c)}_{\textcircled{2}}$$

$$\textcircled{1} \quad \frac{\partial}{\partial v_c} \underbrace{\log \exp(u_0^T v_c)}_{\text{inverses}} = \frac{\partial}{\partial v_c} u_0^T v_c = u_0$$

Vector!
Not high
school
single
variable
calculus

You can do things one variable at a time,
and this may be helpful when things
get gnarly.

$$\forall j \quad \frac{\partial}{\partial (v_c)_j} u_0^T v_c = \frac{\partial}{\partial (v_c)_j} \sum_{i=1}^d (u_0)_i (v_c)_i = (u_0)_j$$

Each term is zero except when $i=j$

$$\textcircled{2} \frac{\partial}{\partial v_c} \log \underbrace{\sum_{w=1}^V \exp(u_w^T v_c)}_{f \quad z = g(v_c)}$$

$$= \frac{1}{\sum_{w=1}^V \exp(u_w^T v_c)} \cdot \frac{\partial}{\partial v_c} \sum_{x=1}^V \exp(u_x^T v_c)$$

Important to change index

$$\frac{\partial}{\partial v_c} f(\underbrace{g(v_c)}_z) = \frac{\partial f}{\partial z}$$

$$\frac{\partial z}{\partial v_c}$$

Use chain rule

$$= \frac{1}{\sum_{w=1}^V \exp(u_w^T v_c)}$$

$$\left(\sum_{x=1}^V \frac{\partial}{\partial v_c} \underbrace{\exp(u_x^T v_c)}_{f \quad z = g(v_c)} \right)$$

Move deriv
inside sum

$$\left(\sum_{x=1}^V \exp(u_x^T v_c) \frac{\partial}{\partial v_c} u_x^T v_c \right)$$

chain rule

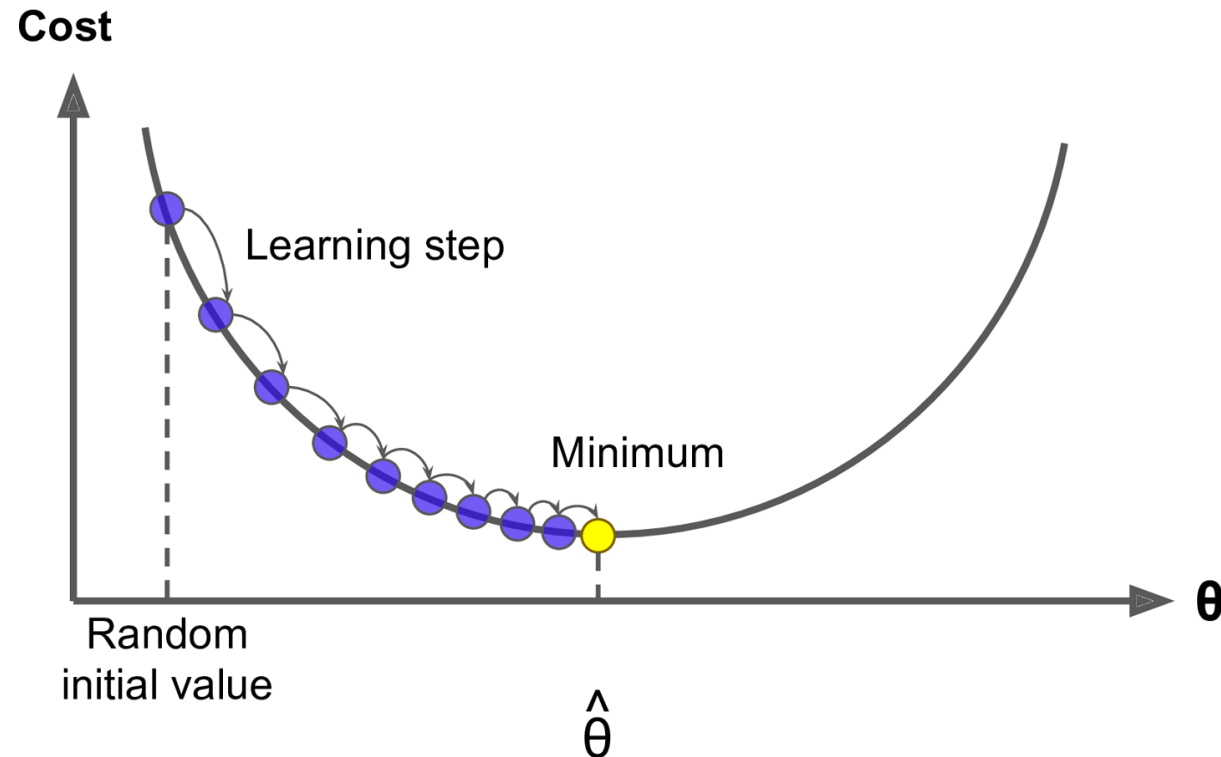
$$\left(\sum_{x=1}^V \exp(u_x^T v_c) u_x \right)$$

$$\begin{aligned}
\frac{\partial}{\partial v_c} \log(p(o|c)) &= u_o - \frac{1}{\sum_{w=1}^V \exp(u_w^T v_c)} \cdot \left(\sum_{x=1}^V \exp(u_x^T v_c) u_x \right) \\
&= u_o - \sum_{x=1}^V \frac{\exp(u_x^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)} u_x \quad \text{Distribute term across sum} \\
&= u_o - \underbrace{\sum_{x=1}^V p(x|c) u_x}_{\text{This an expectation: average over all context vectors weighted by their probability}} \\
&= \text{observed} - \text{expected}
\end{aligned}$$

This is just the derivatives for the center vector parameters
 Also need derivatives for output vector parameters
 (they're similar)
 Then we have derivative w.r.t. all parameters and can minimize

5. Optimization: Gradient Descent

- We have a cost function $J(\theta)$ we want to minimize
- **Gradient Descent** is an algorithm to minimize $J(\theta)$
- **Idea:** for current value of 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 😊

Gradient Descent

- 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
```


Stochastic Gradient Descent

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

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

Lecture Plan

1. The course (10 mins)
2. Human language and word meaning (15 mins)
3. Word2vec introduction (15 mins)
4. Word2vec objective function gradients (25 mins)
5. Optimization basics (5 mins)
6. Looking at word vectors (10 mins or less)
 - See Jupyter Notebook

