Natural Language Processing with Deep Learning
CS224N/Ling284

Christopher Manning

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)

Course logistics in brief

- Instructor: Christopher Manning
- Head TA and co-instructor: Abigail See
- TAs: Many wonderful people! See website
- Time: TuTh 4:30–5:50, Nvidia Aud ($\rightarrow$ video)

- Other information: see the class webpage:
  - http://cs224n.stanford.edu/
    a.k.a., http://www.stanford.edu/class/cs224n/
  - Syllabus, **office hours**, "handouts", TAs, Piazza
    - Office hours start **this Thursday**
  - Slides uploaded before each lecture

3

What do we hope to teach?

1. An understanding of the effective modern methods for deep learning
   - Basics first, then key methods used in NLP: Recurrent networks, attention, etc.

2. A big picture understanding of human languages and the difficulties in understanding and producing them

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

What's different this year?

- Lectures (including guest lectures) covering new material: character models, transformers, safety/fairness, multitask learn.
- 5x one-week assignments instead of 3x two-week assignments
- Assignments covering new material (NMT with attention, ConvNets, subword modeling)
- Using PyTorch rather than TensorFlow
- Assignments due *before class* (4:30pm) not at midnight!
- Gentler but earlier ramp-up
  - First assignment is easy, but due *one week from today*!
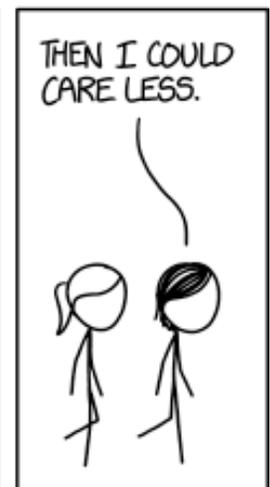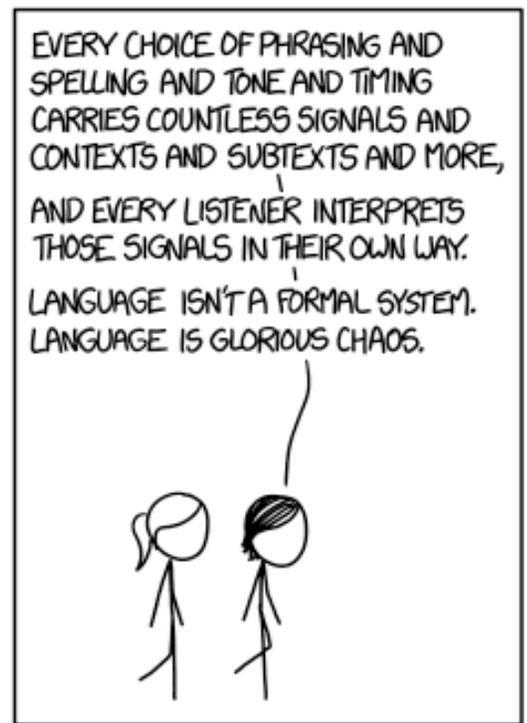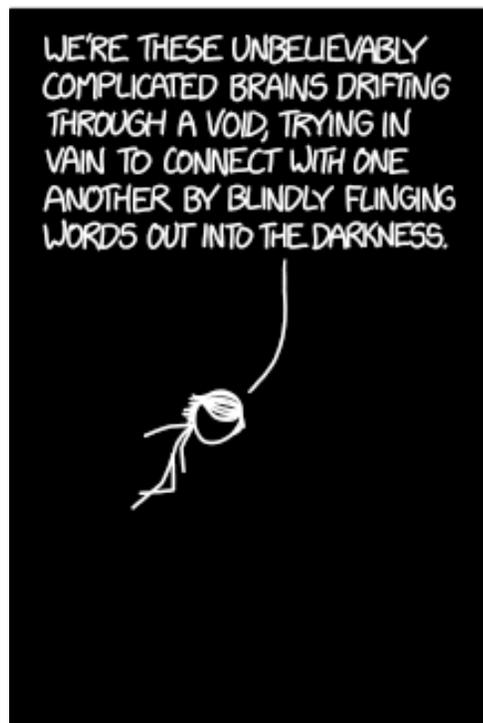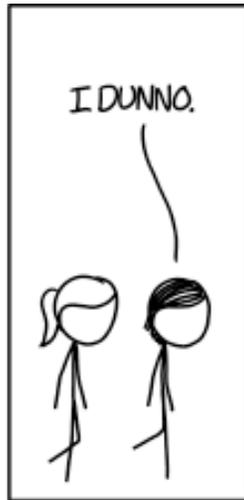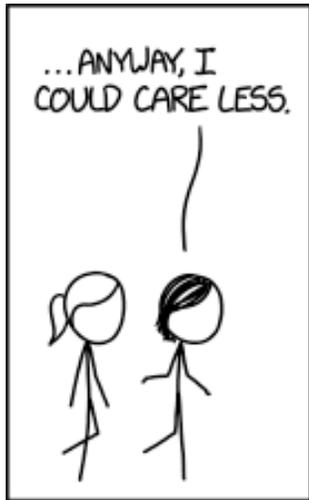- No midterm

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.
  - Please use @stanford.edu email for your Gradescope account
- Final Default or Custom Course Project (1–3 people): 43%
  - Project proposal: 5%, milestone: 5%, poster: 3%, report: 30%
  - Final poster session attendance expected! (See website.)
  - **Wed Mar 20, 5pm-10pm** (put it in your calendar!)
- Participation: 3%
  - (Guest) lecture attendance, Piazza, eval, karma – see website!
- Late day policy
  - 6 free late days; afterwards, 10% off per day late
  - Assignments not accepted after 3 late days per assignment
- Collaboration policy: Read the website and the Honor Code! Understand allowed 'collaboration' and how to document it

High-Level Plan for Problem Sets

- HW1 is hopefully an easy on ramp – an IPython Notebook

- HW2 is pure Python (numpy) but expects you to do (multivariate) calculus so you really understand the basics

- HW3 introduces PyTorch

- HW4 and HW5 use PyTorch on a GPU (Microsoft Azure)
  - Libraries like PyTorch, Tensorflow (and Chainer, MXNet, CNTK, Keras, etc.) are becoming the standard tools of DL

- For FP, you either
  - Do the default project, which is SQuAD question answering
    - Open-ended but an easier start; a good choice for most
  - 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

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)

...ANYWAY, I COULD CARE LESS.

I THINK YOU MEAN YOU *COULDN'T* CARE LESS. SAYING YOU *COULD* CARE LESS IMPLIES YOU CARE AT LEAST SOME AMOUNT.

I DUNNO.

WE'RE THESE UNBELIEVABLY COMPLICATED BRAINS DRIFTING THROUGH A VOID, TRYING IN VAIN TO CONNECT WITH ONE ANOTHER BY BLINDLY FLINGING WORDS OUT INTO THE DARKNESS.

EVERY CHOICE OF PHRASING AND SPELLING AND TONE AND TIMING CARRIES COUNTLESS SIGNALS AND CONTEXTS AND SUBTEXTS AND MORE,

AND EVERY LISTENER INTERPRETS THOSE SIGNALS IN THEIR OWN WAY.

LANGUAGE ISN'T A FORMAL SYSTEM. LANGUAGE IS GLORIOUS CHAOS.

YOU CAN NEVER KNOW FOR SURE WHAT *ANY* WORDS WILL MEAN TO *ANYONE*.

ALL YOU CAN DO IS TRY TO GET BETTER AT GUESSING HOW YOUR WORDS AFFECT PEOPLE, SO YOU CAN HAVE A CHANCE OF FINDING THE ONES THAT WILL MAKE THEM FEEL SOMETHING LIKE WHAT YOU WANT THEM TO FEEL.

EVERYTHING ELSE IS POINTLESS.

https://xkcd.com/1576/ Randall Munroe CC BY NC 2.5

I ASSUME YOU'RE GIVING ME TIPS ON HOW YOU INTERPRET WORDS BECAUSE YOU WANT ME TO FEEL LESS ALONE.

IF SO, THEN THANK YOU. THAT MEANS A LOT.

BUT IF YOU'RE JUST RUNNING MY SENTENCES PAST SOME MENTAL CHECKLIST SO YOU CAN SHOW OFF HOW WELL YOU KNOW IT,

THEN I COULD CARE LESS.

1. 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) ⟺ signified (idea or thing)

= denotational semantics

How do we have usable meaning in a computer?

<u>Common solution</u>: 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

- Great as a resource but missing nuance

  - e.g. "proficient" is listed as a synonym for "good".
    This is only correct in some contexts.

- 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 compute accurate word similarity →

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

Words can be represented by one-hot vectors:

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

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

14

Problem with words as discrete symbols

Example: in web search, if 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**

15

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).

- 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

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

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

Word meaning as a neural word vector – visualization

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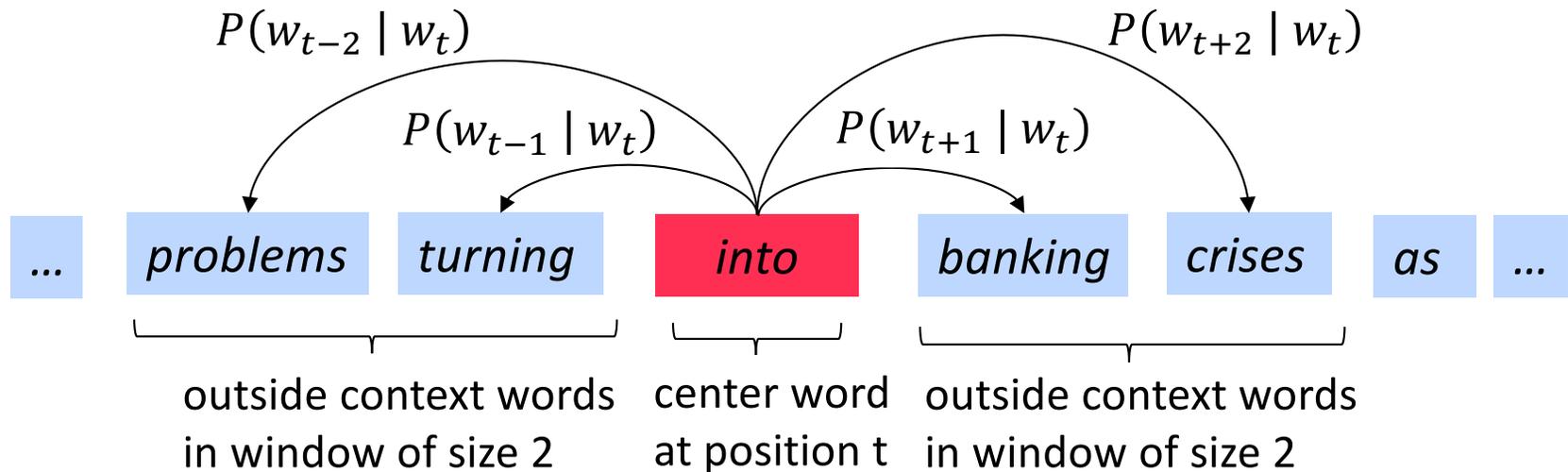$$

3. Word2vec: Overview

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

Idea:

- We have a large corpus of text
- 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)
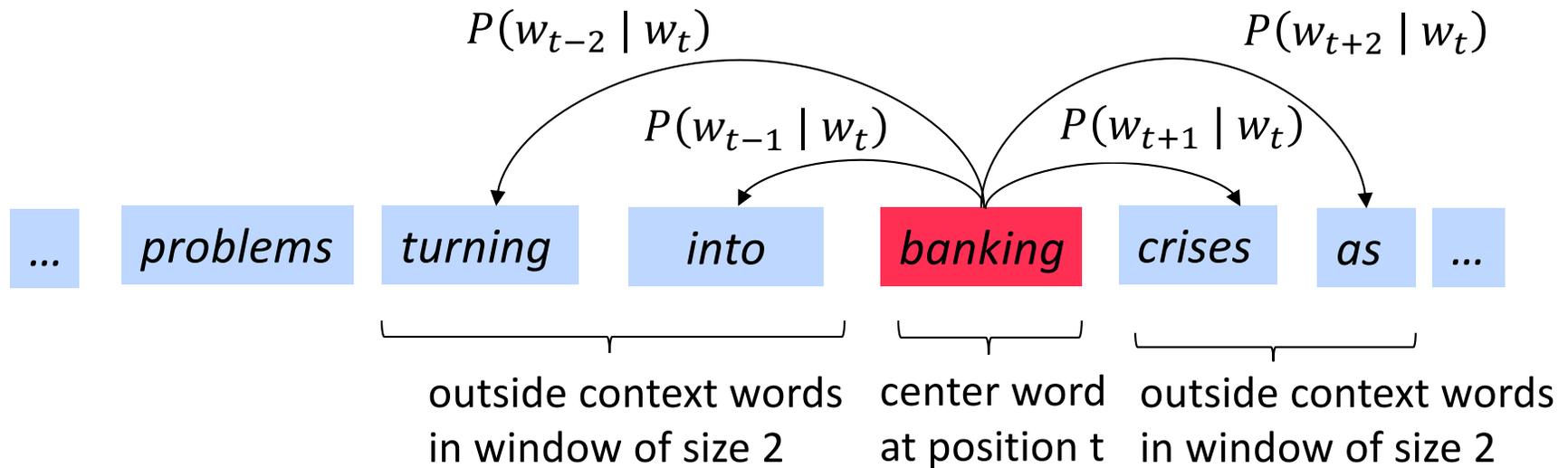- Keep adjusting the word vectors to maximize this probability

Word2Vec Overview

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

$$P(w_{t-2} \mid w_t)$$

$$P(w_{t-1} \mid w_t)$$

$$P(w_{t+1} \mid w_t)$$

$$P(w_{t+2} \mid w_t)$$

... | *problems* | *turning* | *into* | *banking* | *crises* | *as* | ...

outside context words in window of size 2

center word at position t

outside context words in window of size 2

Word2Vec Overview

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

$P(w_{t-2} \mid w_t)$ $P(w_{t+2} \mid w_t)$

$P(w_{t-1} \mid w_t)$ $P(w_{t+1} \mid w_t)$

… | problems | turning | into | banking | crises | as | …

outside context words in window of size 2   center word at position t   outside context words in window of size 2

Word2vec: objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j.

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

θ is all variables to be optimized

sometimes called *cost* or *loss* function

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

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

Minimizing objective function $\Longleftrightarrow$ Maximizing predictive accuracy

Word2vec: objective function
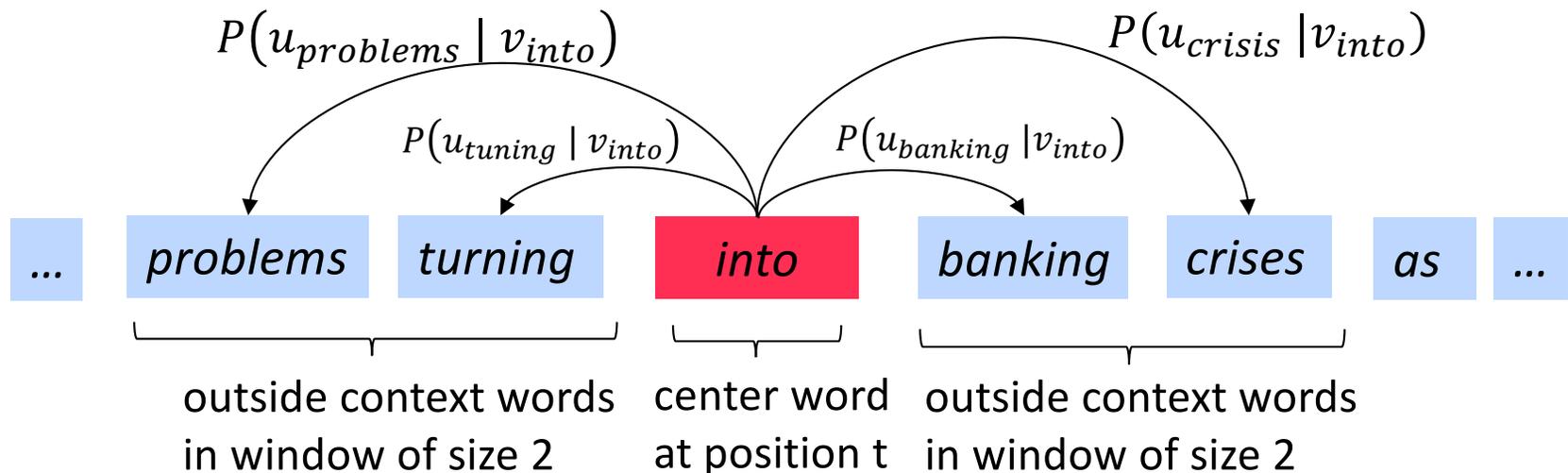
- We want to minimize the objective function:

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

- <u>Question:</u> How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?

- <u>Answer:</u> 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 Overview with Vectors

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

- $P(u_{problems} \mid v_{into})$ short for $P(problems \mid into \, ; u_{problems}, v_{into}, \theta)$

$P(u_{problems} \mid v_{into})$ $\qquad$ $P(u_{crisis} \mid v_{into})$

$P(u_{tuning} \mid v_{into})$ $\qquad$ $P(u_{banking} \mid v_{into})$

| … | *problems* | *turning* | *into* | *banking* | *crises* | *as* | … |

outside context words
in window of size 2

center word
at position t

outside context words
in window of size 2

Word2vec: prediction function

Exponentiation makes anything positive

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

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

Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \to \mathbb{R}^n$

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

- 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

Training a model by optimizing parameters

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

E.g., below, for a simple convex function over two parameters

Contour lines show levels of objective function

$z = x^2 + 2y^2$
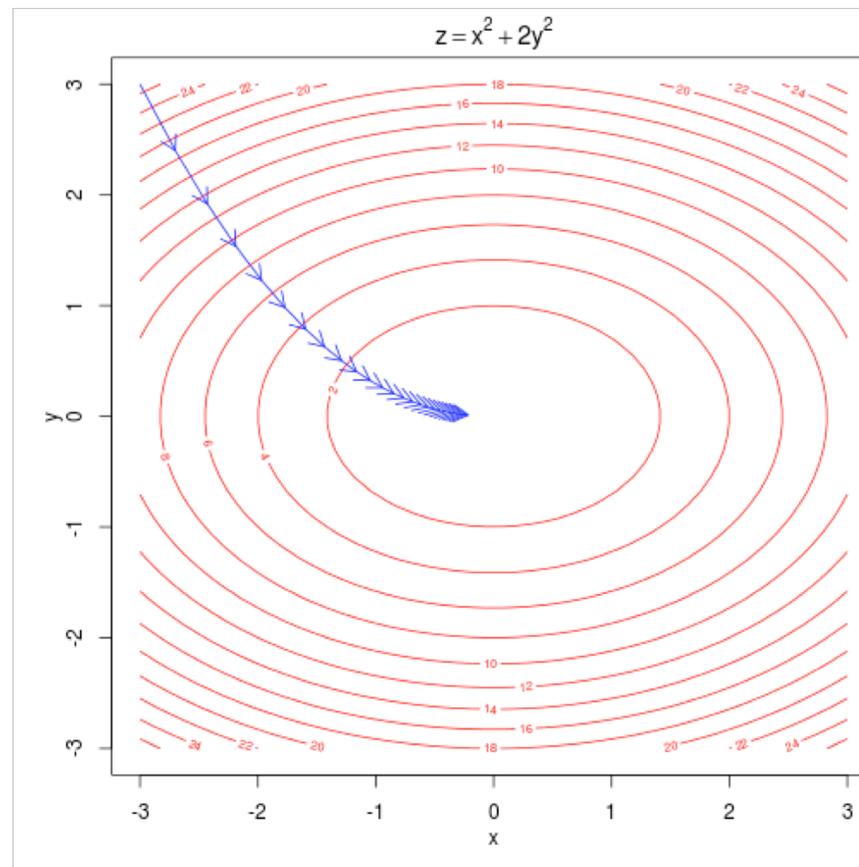
To train the model: Compute **all** vector gradients!

- Recall: θ represents **all** model parameters, in one long vector
- In our case with d-dimensional vectors and V-many words:

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

- Remember: every word has two vectors
- We optimize these parameters by walking down the gradient

4. Word2vec derivations of gradient

- Whiteboard – see video if you're not in class ;)

- The basic Lego piece

- Useful basics: $\dfrac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \dfrac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$

- If in doubt: write out with indices

- Chain rule! If $y = f(u)$ and $u = g(x)$, i.e. $y = f(g(x))$, then:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

Chain Rule

- Chain rule! If $y = f(u)$ and $u = g(x)$, i.e. $y = f(g(x))$, then:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} = \frac{df(u)}{du}\frac{dg(x)}{dx}$$

- Simple example: $\quad \dfrac{dy}{dx} = \dfrac{d}{dx}5(x^3 + 7)^4$

$$y = f(u) = 5u^4 \qquad\qquad u = g(x) = x^3 + 7$$

$$\frac{dy}{du} = 20u^3 \qquad\qquad \frac{du}{dx} = 3x^2$$

$$\frac{dy}{dx} = 20(x^3 + 7)^3 . 3x^2$$

Interactive Whiteboard Session!

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m, j \ne 0} \log p(w_{t+j}|w_t)$$

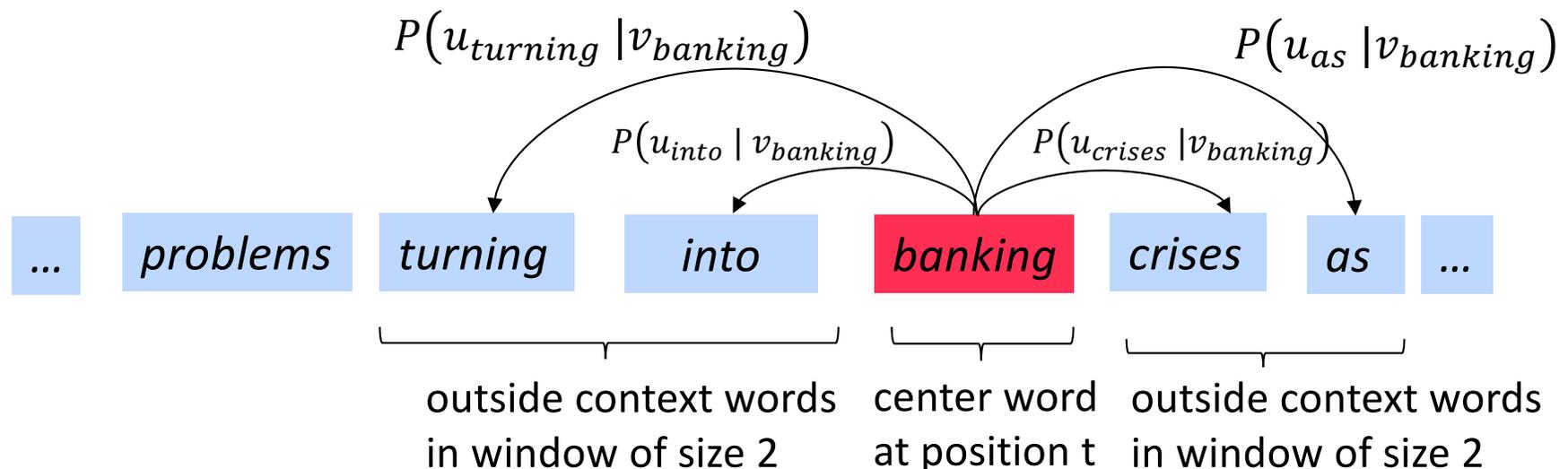Let's derive gradient for center word together
For <u>one example window</u> and <u>one example outside word</u>:

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

You then also need the gradient for context words (it's similar; left for homework). That's all of the parameters θ here.

Calculating all gradients!

- We went through gradient for each center vector v in a window
- We also need gradients for outside vectors u
  - Derive at home!
- Generally in each window we will compute updates for all parameters that are being used in that window. For example:

$$P(u_{turning}\,|v_{banking})$$

$$P(u_{as}\,|v_{banking})$$

$$P(u_{into}\,|\,v_{banking})$$

$$P(u_{crises}\,|v_{banking})$$

| … | *problems* | *turning* | *into* | *banking* | *crises* | *as* | … |

outside context words in window of size 2

center word at position t

outside context words in window of size 2

Word2vec: More details

Why two vectors? → Easier optimization. Average both at the end.

Two model variants:

1. Skip-grams (SG)

   Predict context ("outside") words (position independent) given center word

2. Continuous Bag of Words (CBOW)

   Predict center word from (bag of) context words

This lecture so far: **Skip-gram model**

Additional efficiency in training:

1. Negative sampling

So far: Focus on **naïve softmax** (simpler training method)

32

5. Optimization: Gradient Descent

- We have a cost function $J(\theta)$ we want to minimize
- **Gradient Descent** is an algorithm to minimize $J(\theta)$
- <u>Idea</u>: for current value of θ, calculate gradient of $J(\theta)$, then take small step in direction of negative gradient. Repeat.
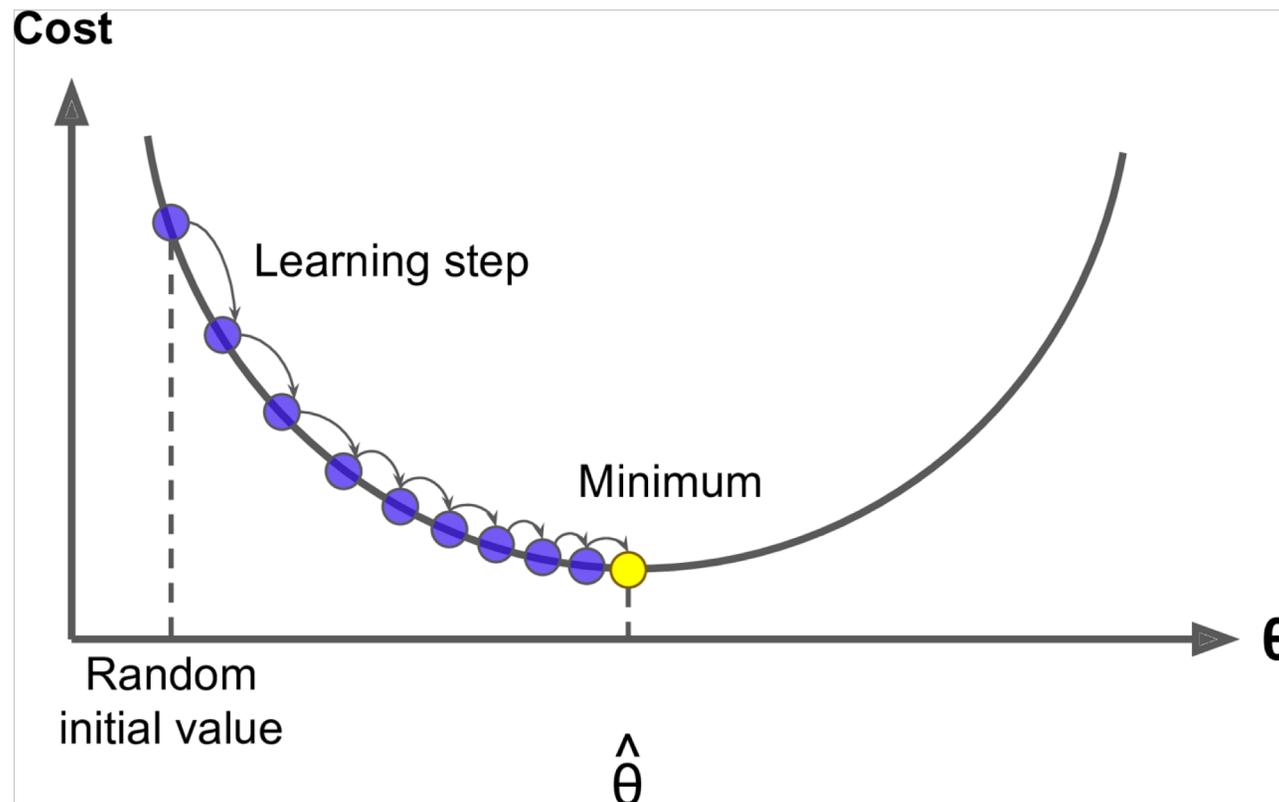
Note: Our objectives may not be convex like this :(

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:

```python
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)**