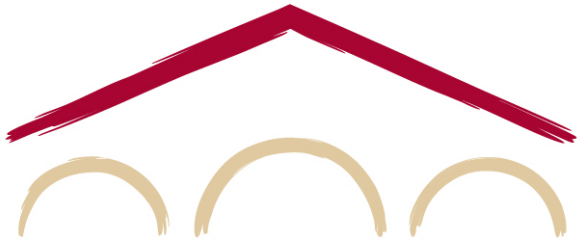
Natural Language Processing with Deep Learning
CS224N/Ling284



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)

Key learning today: The (really surprising!) result that word meaning can be represented rather well by a large vector of real numbers

Course logistics in brief

- Instructor: Christopher Manning
- Head TA: John Hewitt
- Coordinator: Amelie Byun
- TAs: Many wonderful people! See website
- Time: TuTh 4:30–5:50 Pacific time, Zoom U. (→ video)

- Other information: see the class webpage:
  - http://cs224n.stanford.edu/
    a.k.a., http://www.stanford.edu/class/cs224n/
  - TAs, syllabus, office hours (using Nooks), Ed (for all course questions/discussion)
    - Office hours start **tomorrow evening!**
    - Python/numpy and then PyTorch tutorials: First two Fridays 10:00–11:20 Pacific time on Zoom U.
  - Slide PDFs uploaded before each lecture

What do we hope to teach?

1. The foundations of the effective modern methods for deep learning applied to NLP
   - Basics first, then key methods used in NLP: Recurrent networks, attention, transformers, 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

Course work and grading policy

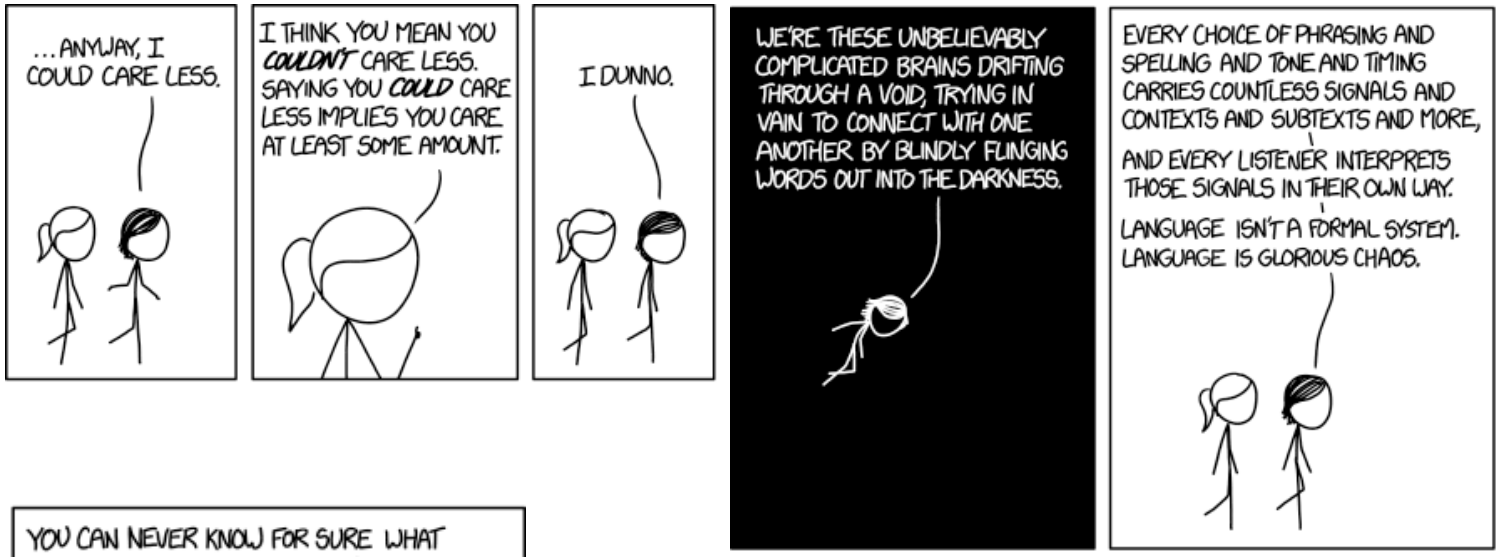
- 5 x 1-week Assignments: 6% + 4 x 12%: 54%
  - **HW1 is released today! Due next Tuesday! At 4:30 p.m.**
  - Please use @stanford.edu email for your Gradescope account
- Final Default or Custom Course Project (1–3 people): 43%
  - Project proposal: 5%, milestone: 5%, web summary: 3%, report: 30%
- Participation: 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

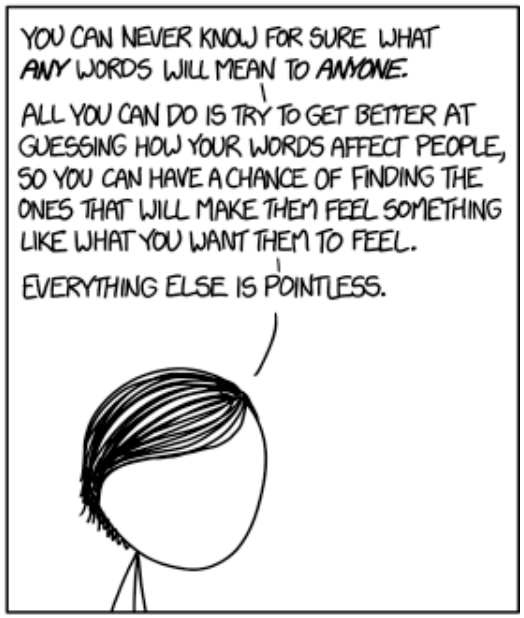
High-Level Plan for Assignments (to be completed individually!)

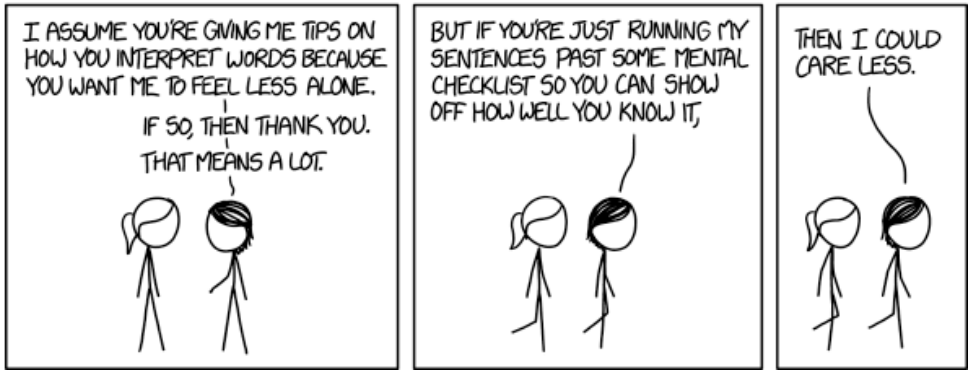
- Ass1 is hopefully an easy on ramp – a Jupyter/IPython Notebook
- Ass2 is pure Python (numpy) but expects you to do (multivariate) calculus so you really understand the basics
- Ass3 introduces PyTorch, building a feed-forward network for dependency parsing
- Ass4 and Ass5 use PyTorch on a GPU (Microsoft Azure)
  - Libraries like PyTorch and Tensorflow are now the standard tools of DL
- For Final Project, 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

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)








ANNO
LITERARIA INSTRUXI
oo
nn
mm
ll
kk
ii
hh
gg
ff
ee
dd
cc
bb
aa

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

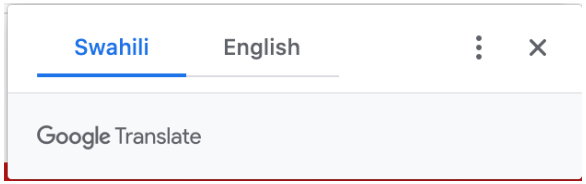
TUKO

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 universal models

OpenAI

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
```

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?

Common 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

- 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

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 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).
- Use the many contexts of *w* to build up a representation of *w*

w = banking

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

A very common size for word vectors is actually 300 dimensions. The following example is 8 dimension.

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

This vector helps us finding the position for the word in a high dimensional vector space.

expect =

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

Similiar words are closer to one another

2D representation of 8 dimensional vector space.

3. Word2vec: Overview

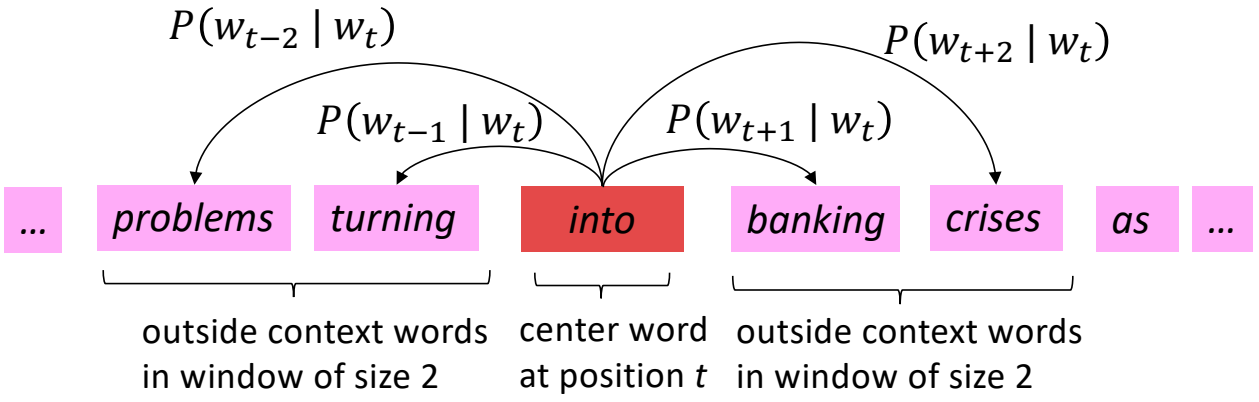
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus ("body") 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

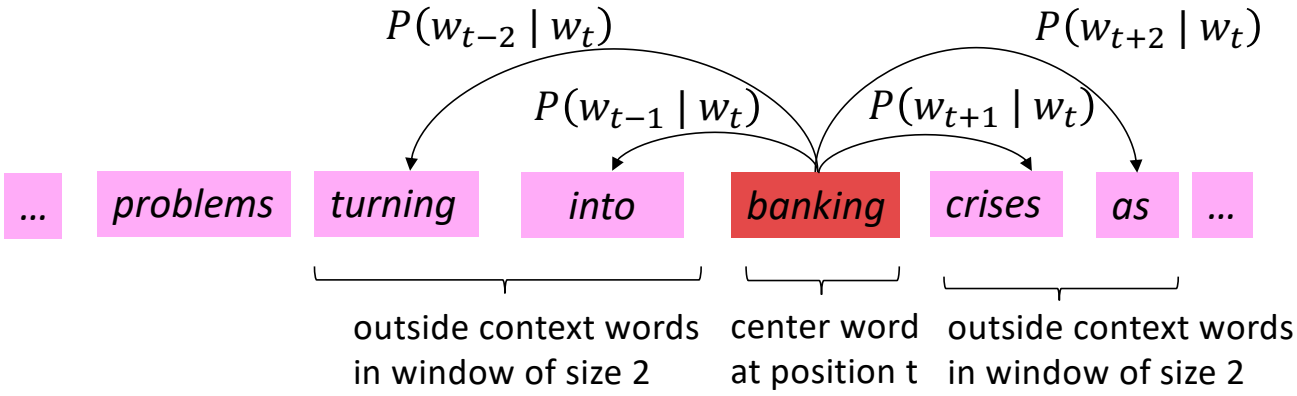
Word2Vec Overview

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



Word2Vec Overview

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



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. Data likelihood:

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

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:

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

Done because its easier to deal with sum vs product.

Minimizing objective function ⟺ Maximizing predictive accuracy

* There is no particular reason why we like to minimize the objective function.

Word2vec: objective function

- We want to minimize the objective function:

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

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

=) Done to simplify the math & optimization

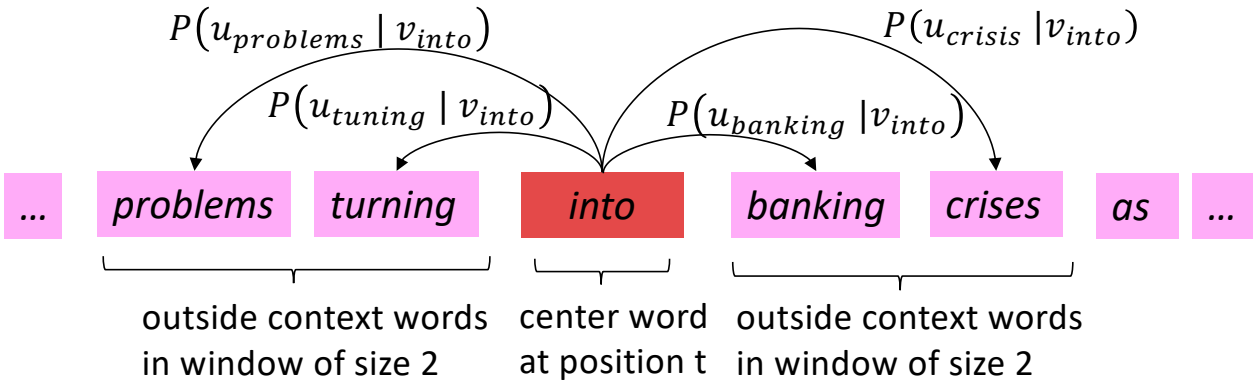
vectors

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



Word2vec: prediction function

② Exponentiation makes anything positive

① 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

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

vocabulary

③ Normalize over entire vocabulary to give probability distribution

Converts any vector ($\mathbb{R}^n$) into a value between 0 & 1.

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

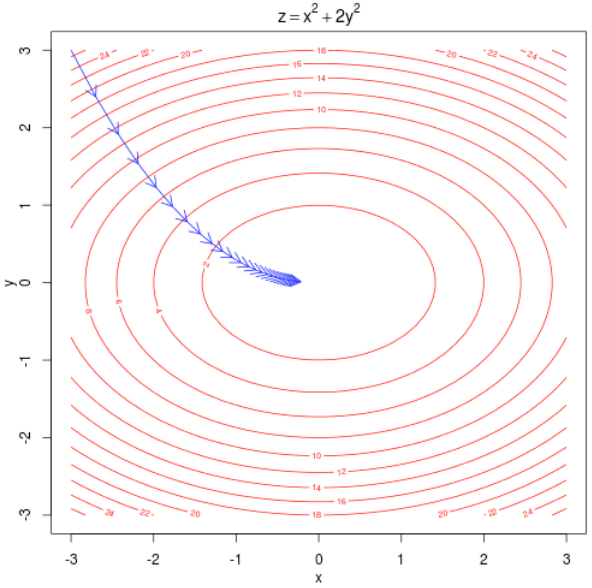
But sort of a weird name because it returns a distribution!

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:
- 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!

Partial differentiation of $J(\theta)$ w.r.t to each term.

4. Word2vec derivations of gradient

- Zoom U. Whiteboard – see video or revised slides
- The basic Lego piece: The chain rule
- Useful basic fact: $\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$
- If in doubt: write it out with indices

$$\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)} = \frac{\partial}{\partial v_c} \log \exp(u_o^T v_c) - \frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c) \quad \left(\log a/b = \log a - \log b\right)$$

① ② ③

For any particular word

① $\frac{\partial}{\partial v_c} \log \exp(u_o^T v_c) = \frac{\partial}{\partial v_c} (u_o^T v_c) = u_o$ [Transpose is removed so that vector shape in final stays the same as original]

② $\frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c) = \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \frac{\partial}{\partial v_c} \sum_{x=1}^{V} \exp(u_x^T v_c)$

③ $\frac{\partial}{\partial v_c} \sum_{x=1}^{V} \exp(u_x^T v_c) = \sum_{x=1}^{V} \frac{\partial}{\partial v_c} \exp(u_x^T v_c) = \sum_{x=1}^{V} \exp(u_x^T v_c) \cdot \frac{\partial}{\partial v_c} (u_x^T v_c) = \sum_{x=1}^{V} \exp(u_x^T v_c) \cdot u_x$

Final: $\frac{\partial}{\partial v_c} \log p(o|c) = u_o - \frac{\sum_{x=1}^{V} \exp(u_x^T v_c)\, u_x}{\sum_{w=1}^{V} \exp(u_w^T v_c)} = u_o - \sum_{x=1}^{V} p(x|c)\, u_x$

$= u_o - \sum_{x=1}^{V} p(x|c)\, u_x$ (expected value)

$=$ Observed $-$ expected

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: $\frac{dy}{dx} = \frac{d}{dx}5(x^3+7)^4$

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

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

$$\frac{dy}{dx} = 20(x^3+7)^3.3x^2$$

Interactive Whiteboard Session! (Math done on Slide 28)

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

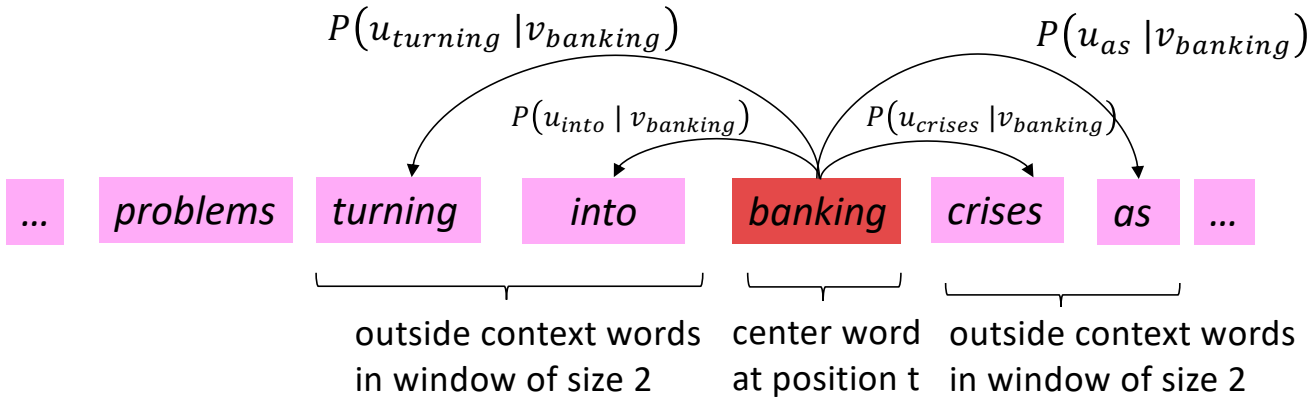
Let's derive gradient for center word together
For one example window and one example outside word:

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

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 the 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:



Summary of PROCESS:

* Start with random word vectors
* Iterate through each word in word corpus.
* Try predicting surrounding words using word vectors : $P(O/C)$

* Update vectors so that they can actually predict surrounding words better.

** Lecture ENDS HERE

Word2vec: More details

Why two vectors? → Easier optimization. Average both at the end. => gives you their position in vector space. i.e

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**

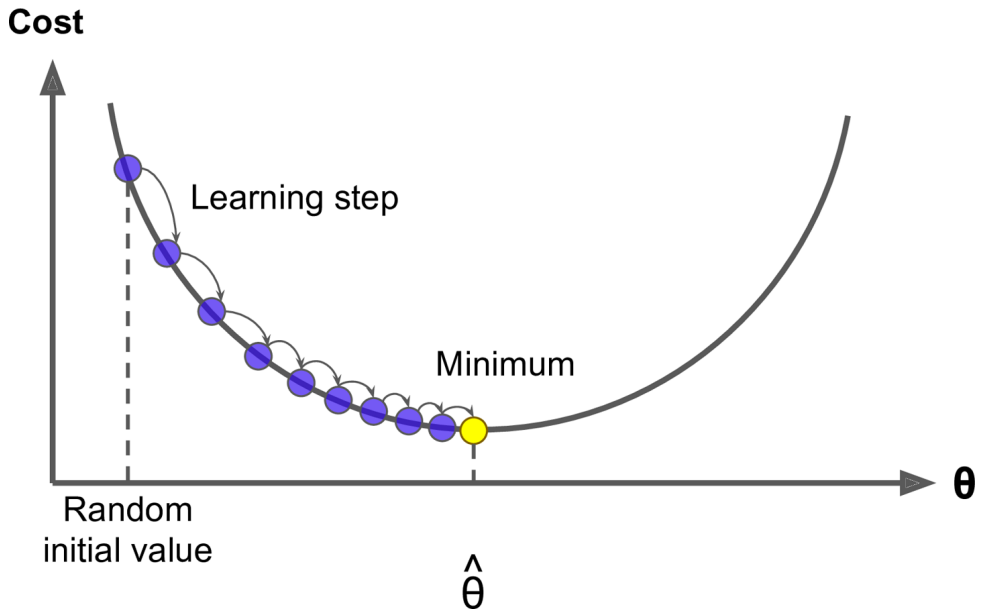
Additional efficiency in training:

1. Negative sampling

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

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)