Natural Language Processing

CS 3216/UG, AI 5203/PG

Week-4

Semantics, vector embeddiings

Mahindra™ University

Global Thinkers. Engaged Leaders.

Acknowledgments

These slides were adapted from the book

SPEECH and LANGUAGE PROCESSING: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition

Practical Natural Language Processing (A Comprehensive Guide to Building Real-World NLP Systems)Orielly

and

some modifications from presentations and resources found in the WEB by several scholars.

Recap

- NLP
- Applications
- Regular expressions
- Tokenization
- Stemming
    - Porter Stemmer
- Lemmatization
- Normalization
- Stopwords
- Bag-of-Words
- TF-IDF
- NER
- POS tagging

Semantics

- What is semantics?
  - Semantics is the study of meaning

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)

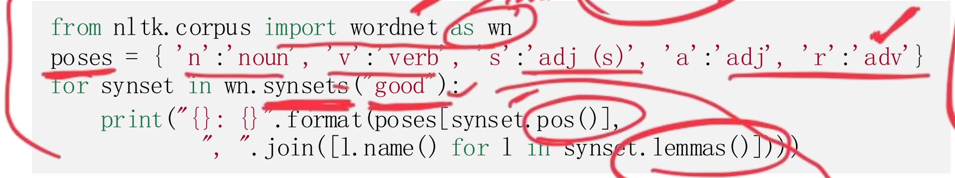
= denotational semantics

⇗ Acknowledgemet :- ALL SLIDES FROM CHRISTOPHER Manning CS224N

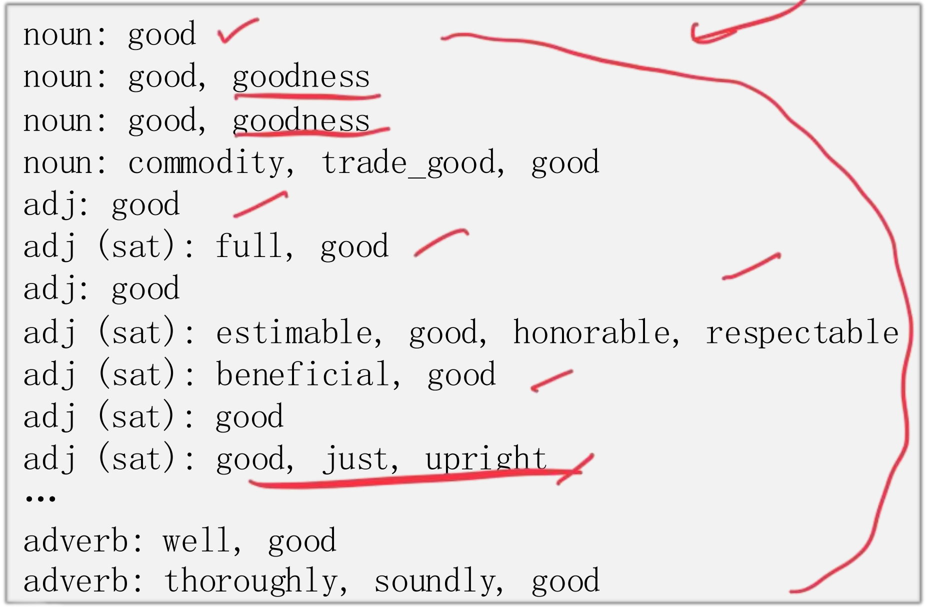
How do we have usable meaning in a computer?

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

e.g. synonym sets containing “good”:

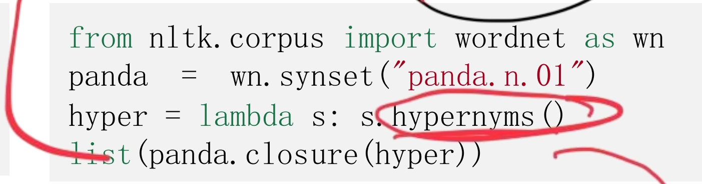


```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
          ", ".join([l.name() for l in synset.lemmas()])))
```

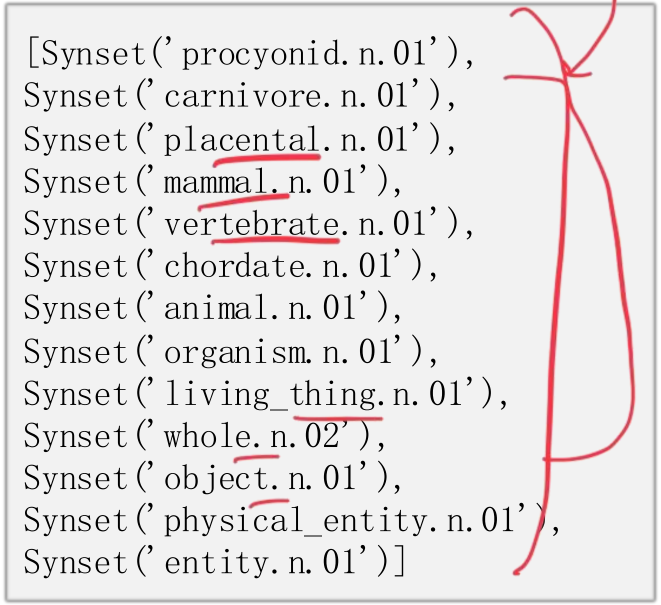


```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly,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

0 0 1 1

In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel – a localist representation

binary representation

Means one 1, the rest 0s

Words can be represented by one-hot vectors:

cosine

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]

dot product of

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

vectors

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!

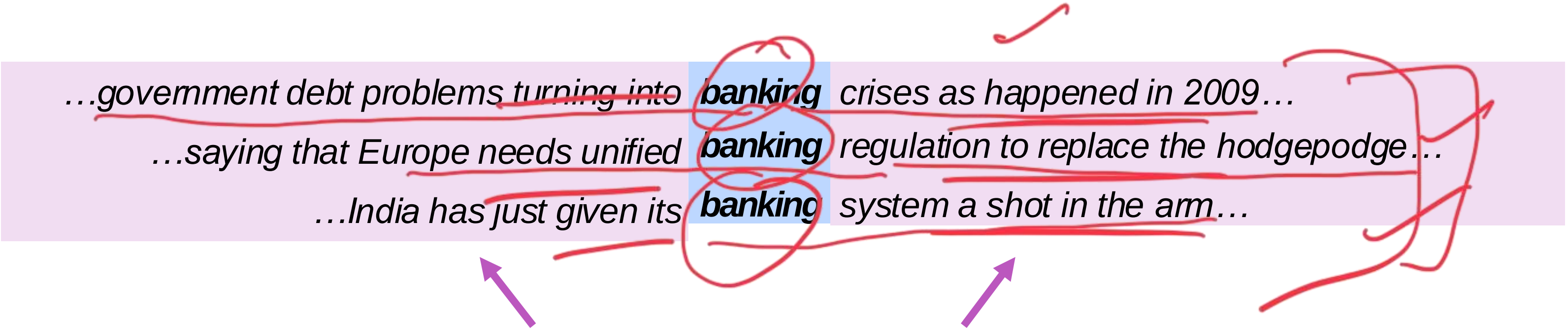
Solution:

- 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**

Distributional Semantics ✓

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

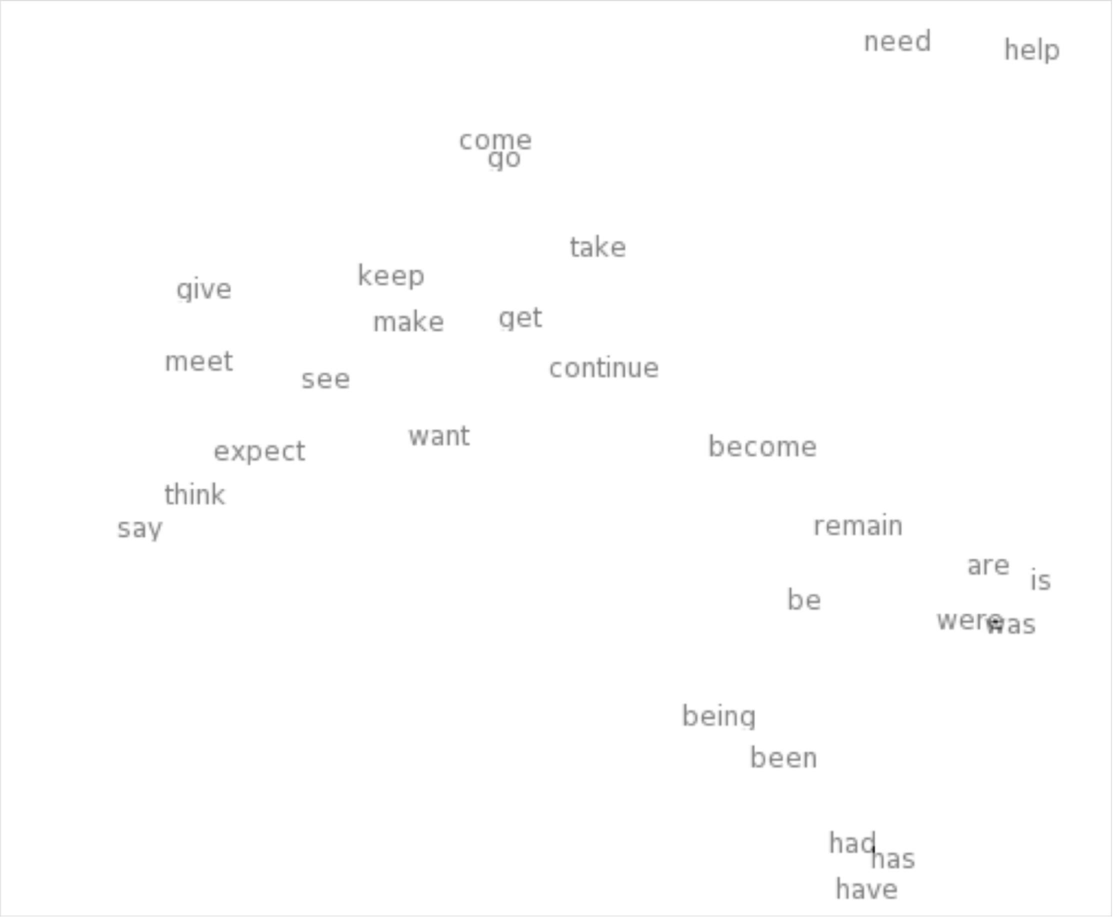
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts

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

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

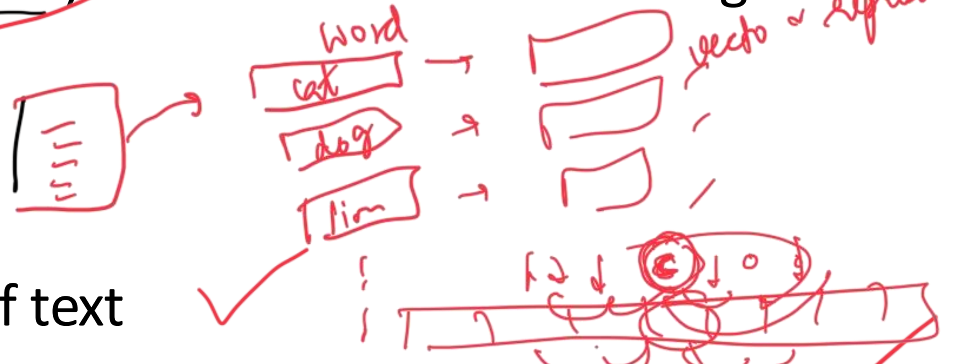
Word meaning as a neural word vector – visualization

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



3. Word2vec: Overview

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



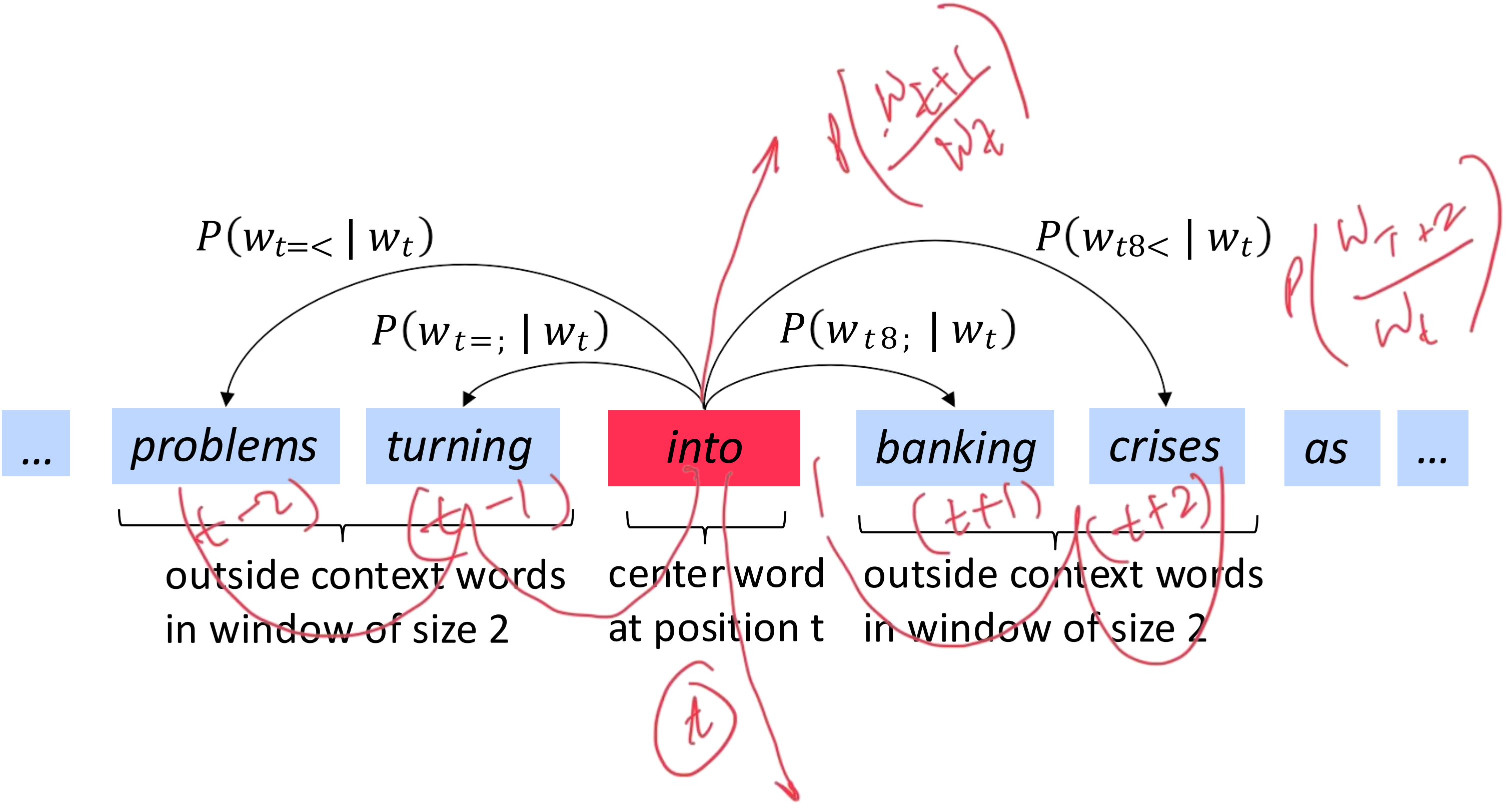
Idea:

- 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

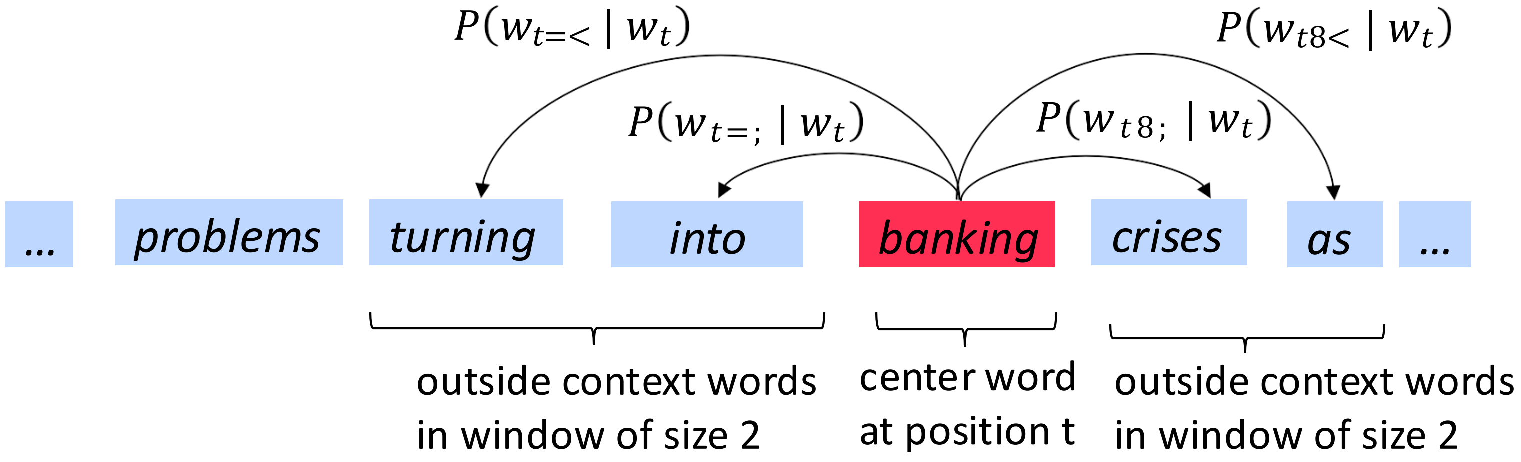
context window=2

- Example windows and process for computing $P(w_{t89} \mid w_t)$


$P(w_{t=<} \mid w_t)$
$P(w_{t=;} \mid w_t)$
$P(w_{t8;} \mid w_t)$
$P(w_{t8<} \mid w_t)$
…
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
$P\left(\frac{w_{t+1}}{w_t}\right)$
$P\left(\frac{w_{T+2}}{w_t}\right)$
(t-2)
(t-1)
(t+1)
(t+2)
t

Word2Vec Overview

- Example windows and process for computing $P\left(w_{t89}\,|\,w_t\right)$



$P(w_{t=<}\,|\,w_t)$

$P(w_{t=;}\,|\,w_t)$

$P(w_{t8;}\,|\,w_t)$

$P(w_{t8<}\,|\,w_t)$

… *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 $t = 1, \dots, T$, predict context words within a window of fixed size m, given center word $w_{(}$.

Positions

$$\text{Likelihood} = L(*) = \underset{t-1}{\overset{T}{\mathrm{G}}} \quad \underset{\substack{-m2(2m\\(34}}{\mathrm{G}} \; P\big(w_{t6(} \mid w_t; *\big)$$

w_j

* is all variables to be optimized

sometimes called *cost* or *loss* function

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

$$J(*) = -\frac{1}{T}\log L(*) = -\frac{1}{T} \underset{t-1}{\overset{T}{@}} \quad \underset{\substack{-m2(2m\\(34}}{@} \; \log P\big(w_{t6(} \mid w_t; *\big)$$

Minimizing objective function ⟺ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

$$J(\text{"}) = -\frac{1}{T}\,'^{\,T}_{t)*}\;'_{\substack{,m./.m\\ /01}}\;\log P\left(w_{t7/} \mid w_t;\text{"}\right)$$

- Question: How to calculate $P\left(w_{t7/} \mid w_t;\text{"}\right)$?

- Answer: We will *use two* vectors per word *w*:
  - $v_;$ when *w* is a center word
  - $u_;$ when *w* is a context word

- Then for a center word *c* and a context word *o*:

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

Vector of context word

Center word W

V_w

Center

U_w

W=C

W=O

V_c

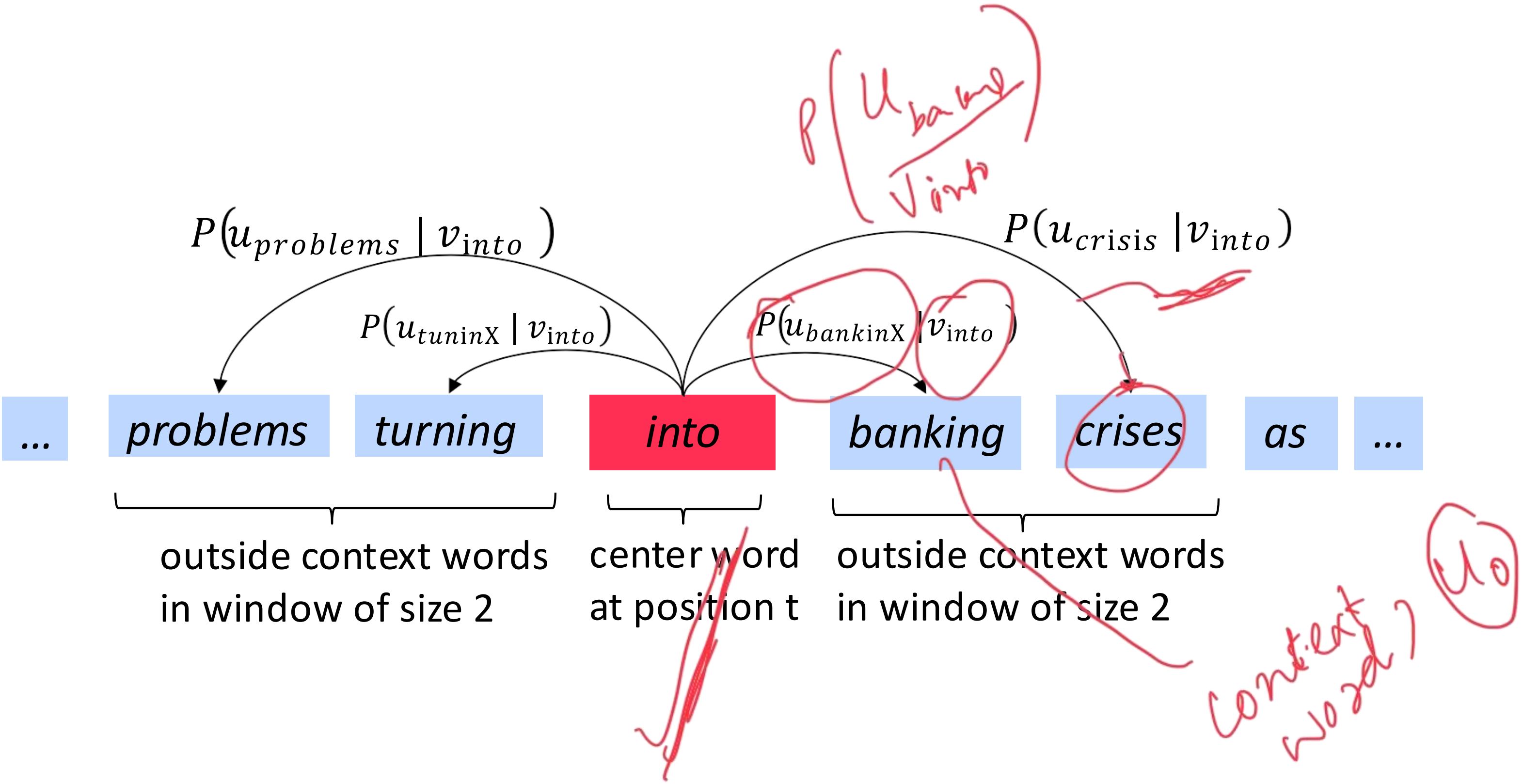
U_O

Vector centre word

context word

Word2Vec Overview with Vectors

- Example windows and process for computing $P(w_{t89} \mid w_t)$
- $P(u_{problems} \mid v_{into})$ short for P$(problems \mid into \,; u_{problems}, v_{into},)$
  U



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

… 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

$$P(o|c) = \frac{\exp(\boxed{u_o^T v})}{\boxed{\sum_{0\,\in V} \exp(u_0^T v\ )}}$$

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

$u^T v = u.v = \sum_{=?\ @}^{n} u_= v_=$

Larger dot product = larger probability

Normalize over entire vocabulary to give probability distribution

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

$$\text{softmax}(x_=) = \frac{\exp(x_=)}{\sum_{>?\ @}^{n} \exp(x_>)} = p_=$$

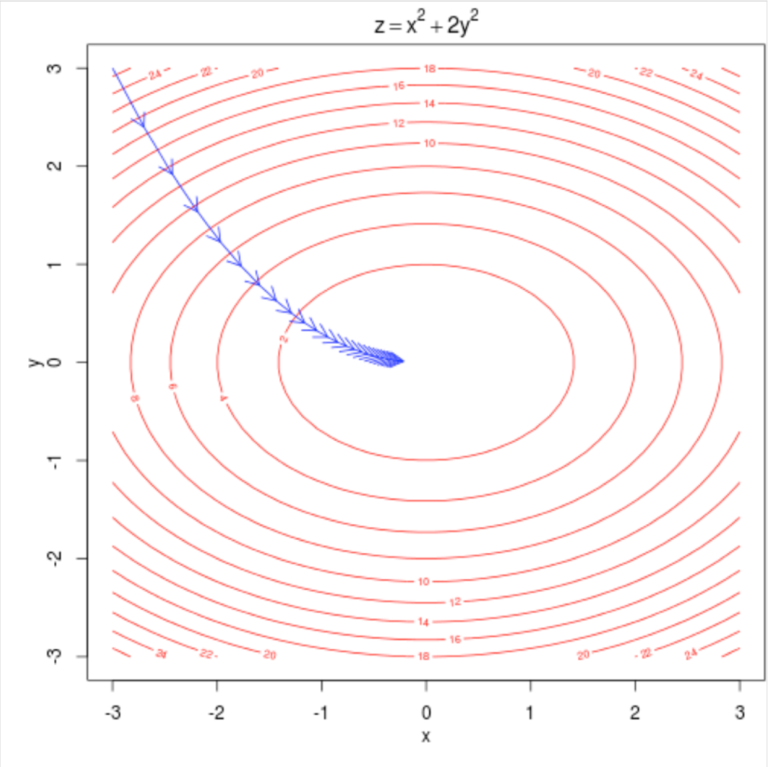
- The softmax function maps arbitrary values $x_=$ to a probability distribution $p_=$
  - "max" because amplifies probability of largest $x_=$
  - "soft" because still assigns some probability to smaller $x_=$
  - Frequently used in Deep Learning

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



4. Word2vec derivations of gradient

- Whiteboard – see video if you're not in class ;)
- The basic Lego piece
- Useful basics: $\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$
- If in doubt: write out with indices

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

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

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

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

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

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

Interactive Whiteboard Session!

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

Let's derive gradient for center word together

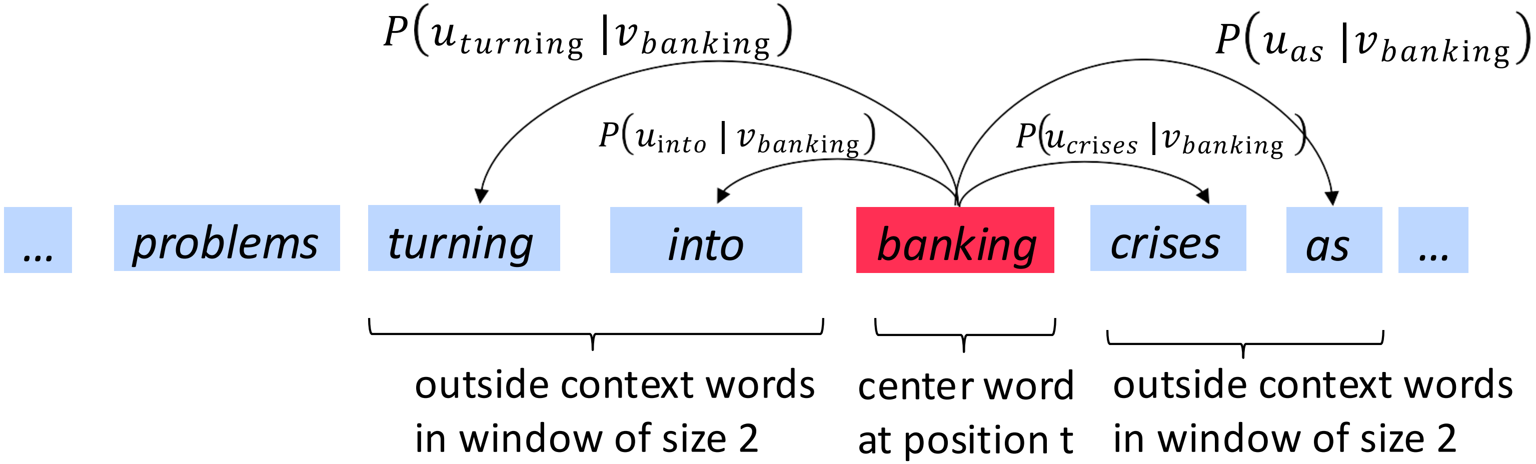
For <u>one example window</u> and <u>one example outside word</u>:

$$\log p(o|c) = \log \frac{\exp({u_o}^T v_c)}{\sum_{4\,5\,6}^{V} \exp({u_4}^T v_c)}$$

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



$P(u_{turning} \mid v_{banking})$
$P(u_{into} \mid v_{banking})$
$P(u_{crises} \mid v_{banking})$
$P(u_{as} \mid v_{banking})$
…
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**

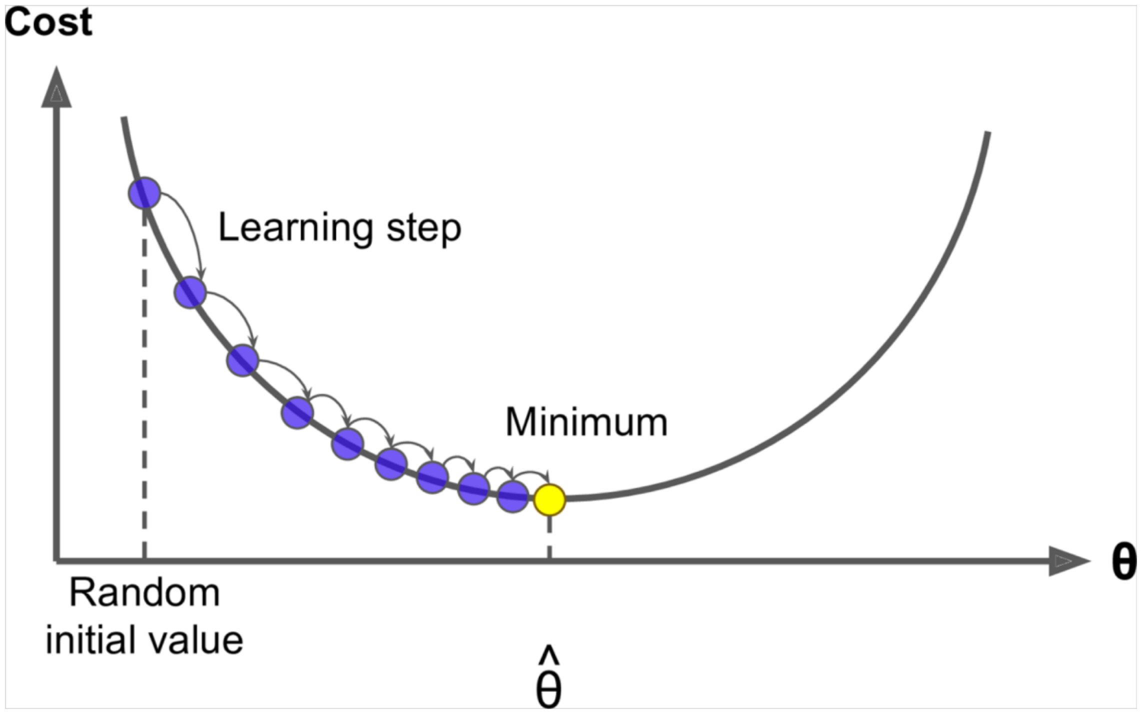
Additional efficiency in training:

1. Negative sampling

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

5. Optimization: Gradient Descent

- We have a cost function *J* (") we want to minimize
- **Gradient Descent** is an algorithm to minimize *J* (")
- Idea: for current value of " , calculate gradient of *J* ("), 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):

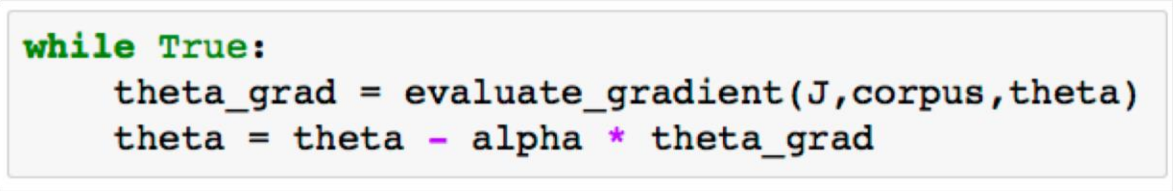
$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$$

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:

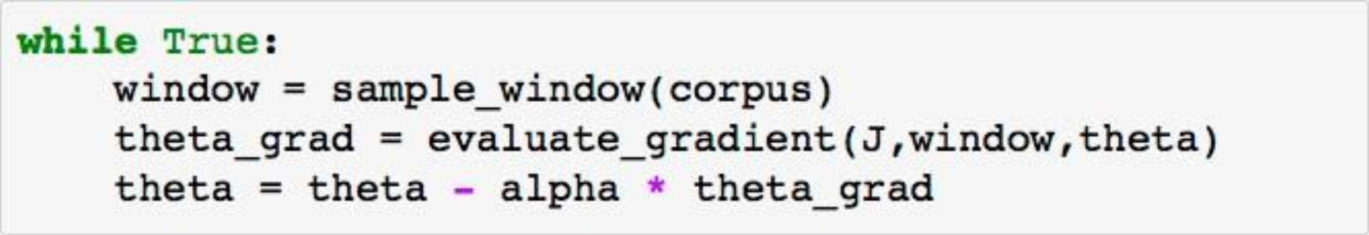


```
while True:
    theta_grad = evaluate_gradient(J,corpus,theta)
    theta = theta - alpha * theta_grad
```

Stochastic Gradient Descent

- Problem: $J(")$ 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
```

Class Projects /NLP

- Group size allowed (4-5) /UG, 30-33 teams
- Pick a project , (1+1) mid evaluations, Final project presentation
- TAs will float a sheet to fill your project team name/Topic
- Real-world problems (Look around for NLP problems)

  Do not restrict yourselves to sentiment analysis, recommender systems, searching, etc.

  Project topic and Team – Deadline – 12th Sept, 2024 11:59 PM

Class Homework

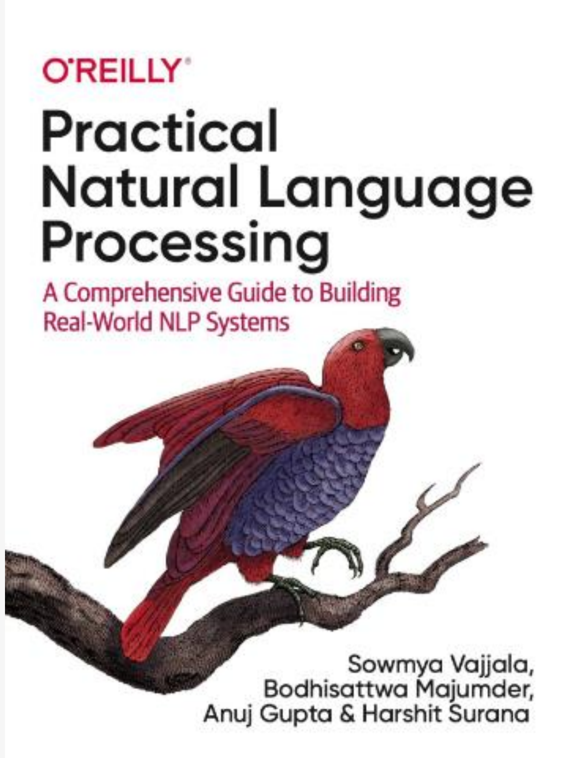
Implement Word2vec in python

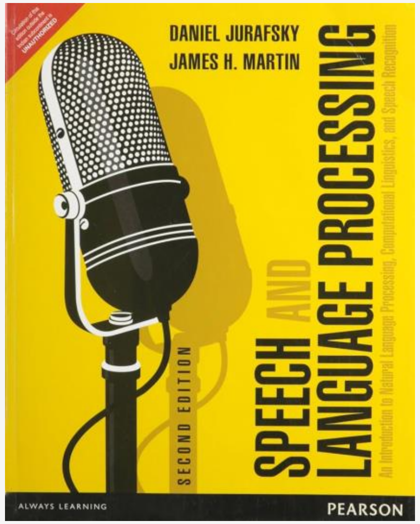
Reference materials

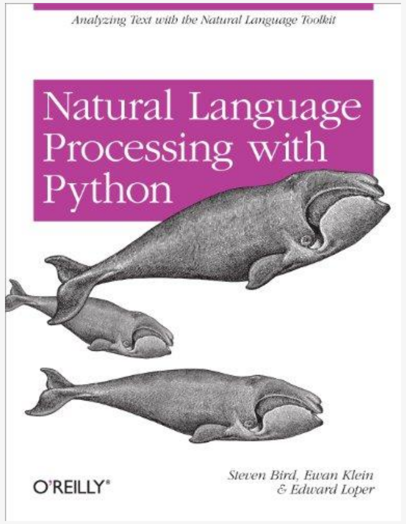
- https://vlanc-lab.github.io/mu-nlp-course/

- Lecture notes

- (A) Speech and Language Processing by Daniel Jurafsky and James H. Martin
- (B) Natural Language Processing with Python. (updated edition based on Python 3 and NLTK 3) Steven Bird et al. O’Reilly Media


O'REILLY
Practical Natural Language Processing
A Comprehensive Guide to Building Real-World NLP Systems
Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta & Harshit Surana


DANIEL JURAFSKY
JAMES H. MARTIN
SECOND EDITION
SPEECH AND LANGUAGE PROCESSING
An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition
ALWAYS LEARNING
PEARSON


Analyzing Text with the Natural Language Toolkit
Natural Language Processing with Python
O'REILLY
Steven Bird, Ewan Klein & Edward Loper