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 (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: Shikhar Murty
- Course Manager: John Cho
- TAs: Many wonderful people – see website!
- Time: Tu/Th 4:30–5:50 Pacific time, Nvidia Aud. (livestreamed to PanOpto video)
- Email list: cs224n-spr2324-staff@lists.stanford.edu
- 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!**
    - Python/numpy and then PyTorch tutorials: First two Fridays (4/5, 4/12), 3:30–4:20, Gates B01
  - Slide PDFs uploaded before each lecture

Instructors

Chris Manning

Shikhar Murty
Head TA

Course Manager

John Cho

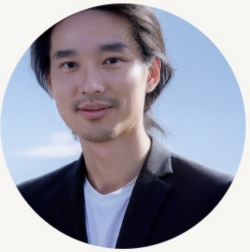
Teaching Assistants

Aditya Agrawal

Anna Goldie

Archit Sharma

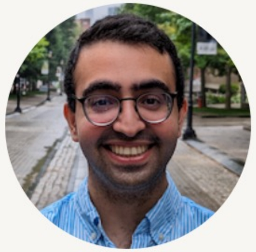
Arvind Mahankali



Chaofei Fan

Jingwen Wu

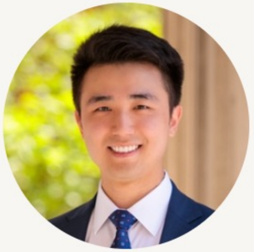
Johnny Chang



Kamyar Salahi

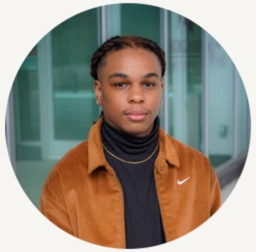
Kaylee Burns

Moussa Doumbouya



Neil Nie

Olivia Lee



Rashon Poole

Ryan Li

Shijia Yang

Soumya Chatterjee

Timothy Dai

Yuan Gao

Zhoujie Ding

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: Word vectors, feed-forward networks, recurrent networks, attention
   - Then key methods used in NLP in 2024: transformers, encoder-decoder models, pretraining, post-training (RLHF, SFT), efficient adaptation, model interpretability, language model agents, 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

- 4 x mainly 1.5-week Assignments: 6% + 3 x 14%: 48%
  - **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): 49%
  - Project proposal: 8%, milestone: 6%, poster: 3%, report: 32%
- Participation: 3%
  - Guest lecture reactions, Ed, course evals, karma – see website!
- Late day policy
  - 6 free late days; afterwards, 1% off total course grade per day late
  - Assignments not accepted more than 3 days late per assignment unless given permission in advance

Course work and grading policy

- 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
  - Students must independently submit their solutions to CS224N homeworks
- AI tools policy
  - Large language models are great (!), but we don't want ChatGPT's solutions to our assignments
  - Collaborative coding with AI tools is allowed; asking it to answer questions is strictly prohibited
  - Employing AI tools to substantially complete assignments will be considered a violation of the Honor Code (see Generative AI Policy Guidance here for more details)

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

- Ass 1 is hopefully an easy on ramp – a Jupyter/IPython Notebook
- Ass 2 expects you to do (multivariate) calculus, so you really understand the basics, introduces PyTorch, and you build a feed-forward network for dependency parsing
- Ass 3 and Ass 4 use PyTorch on a GPU (Google Cloud)
    - 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
        - You implement a BERT LLM and then fine-tune and adapt it for downstream tasks
        - 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)


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


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

Neural machine translation was an early big success of Neural NLP

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.

Swahili English

Google Translate

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.

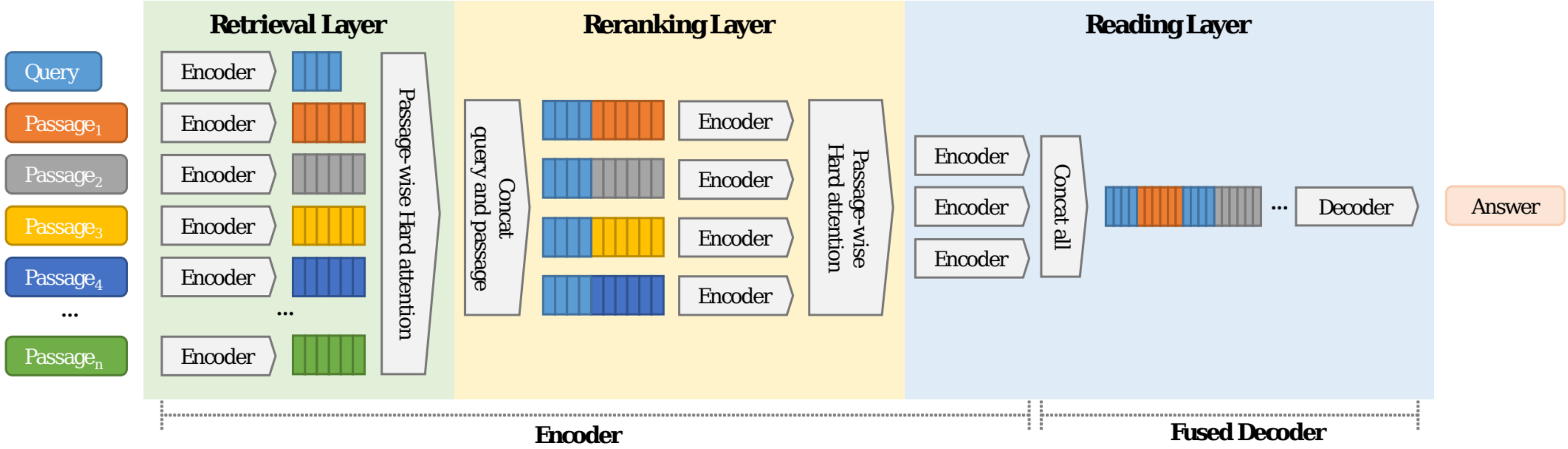
Free-text question answering: Next gen search

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, https://arxiv.org/pdf/2112.07381.pdf) uses a T5-Large model fine-tuned for QA 3 times to run entire QA pipeline



GPT-2 (2019): An early step towards foundation models

OpenAI

PROMPT (HUMAN-WRITTEN)	*A train carriage containing controlled nuclear materials was stolen in Cincinnati today. Its whereabouts are unknown.*
MODEL COMPLETION	The incident occurred on the downtown train line, which runs from Covington and Ashland stations. In an email to Ohio news outlets, the U.S. Department of Energy said it is working with the Federal Railroad Administration to find the thief. “The theft of this nuclear material will have significant negative consequences on public and environmental health, our workforce and the economy of our nation,” said Tom Hicks, the U.S. Energy Secretary, in a statement. “Our top priority is to secure the theft and ensure it doesn’t happen again.” The stolen material was taken from the University of Cincinnati’s Research Triangle Park nuclear research site, according to a news release from Department officials .

ChatGPT, GPT-4, and more

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]

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.

Pict
Ope

ge

A illustration from a graphic novel. A bustling city street under the shine of a full moon. The sidewalks bustling with pedestrians enjoying the nightlife. At the corner stall, a young woman with fiery red hair, dressed in a signature velvet cloak, is haggling with the grumpy old vendor. the grumpy vendor, a tall, sophisticated man is wearing a sharp suit, sports a noteworthy moustache is animatedly conversing on his steampunk telephone.

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

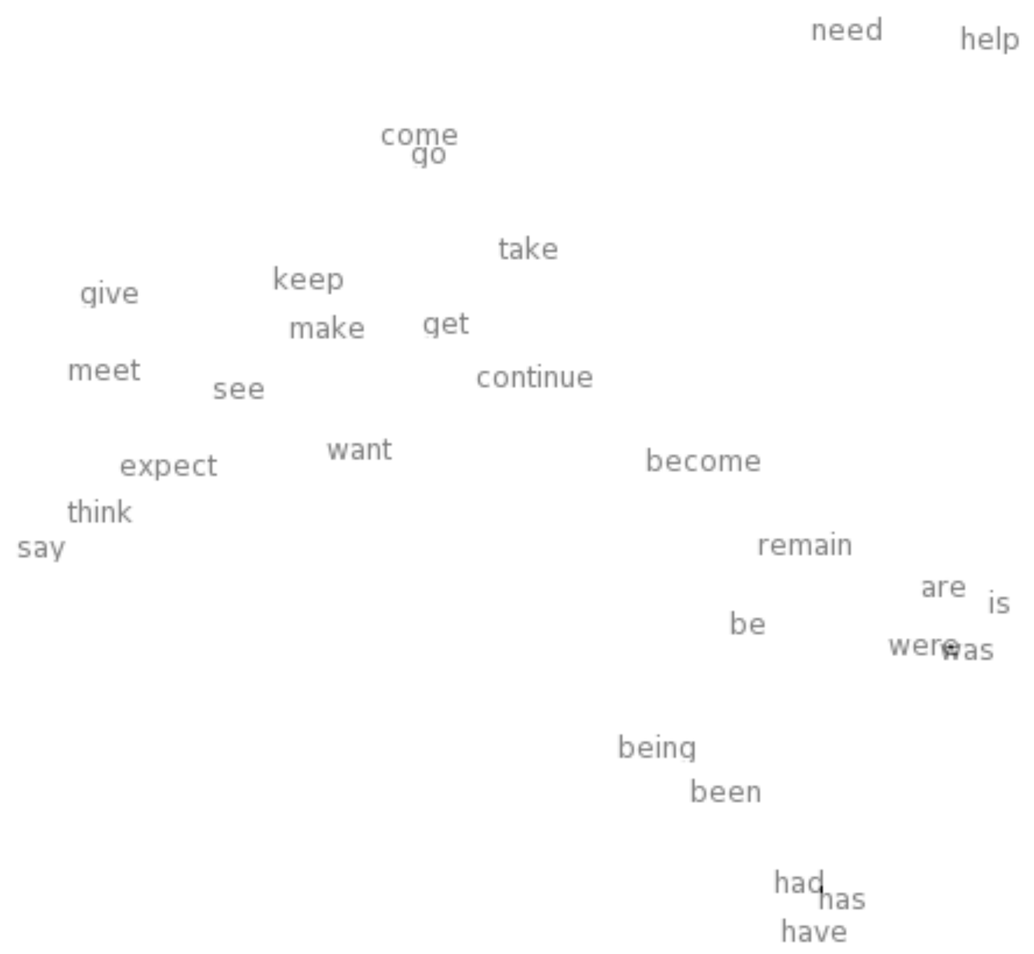
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector dot (scalar) product

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

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

Word meaning as a neural word vector – visualization

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

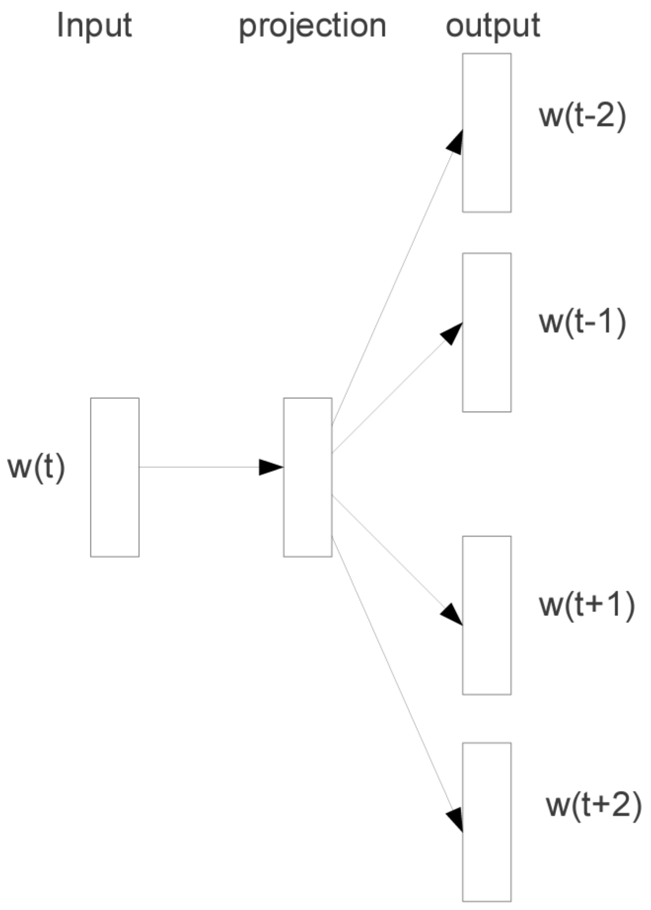


3. Word2vec: Overview

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

Idea:

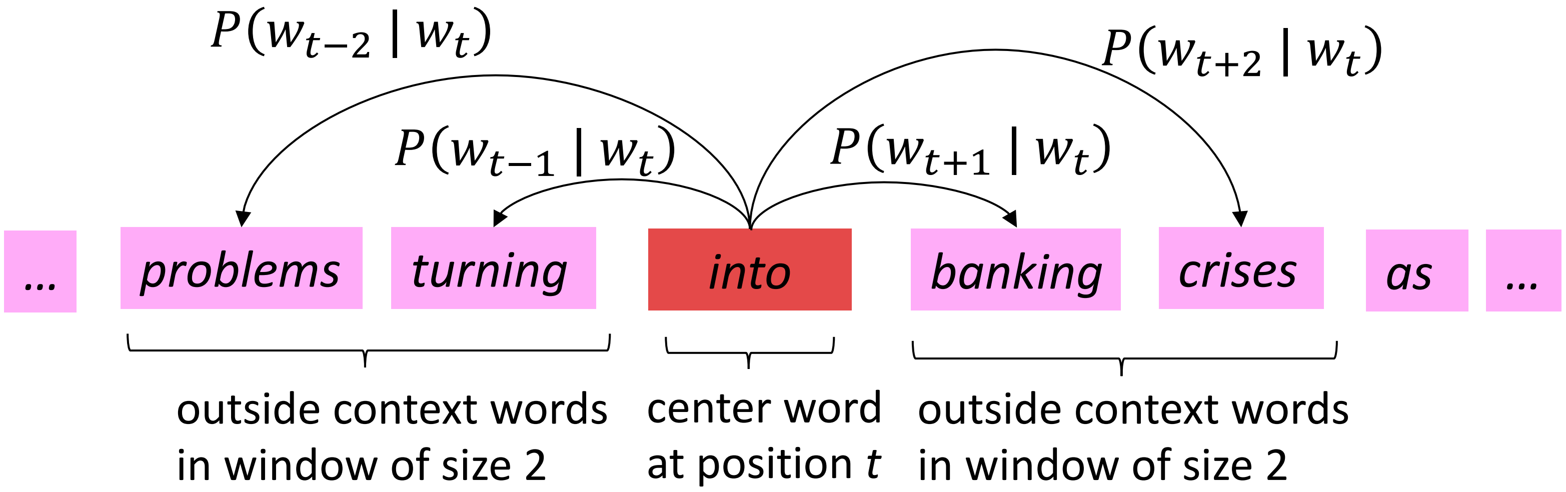
- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate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

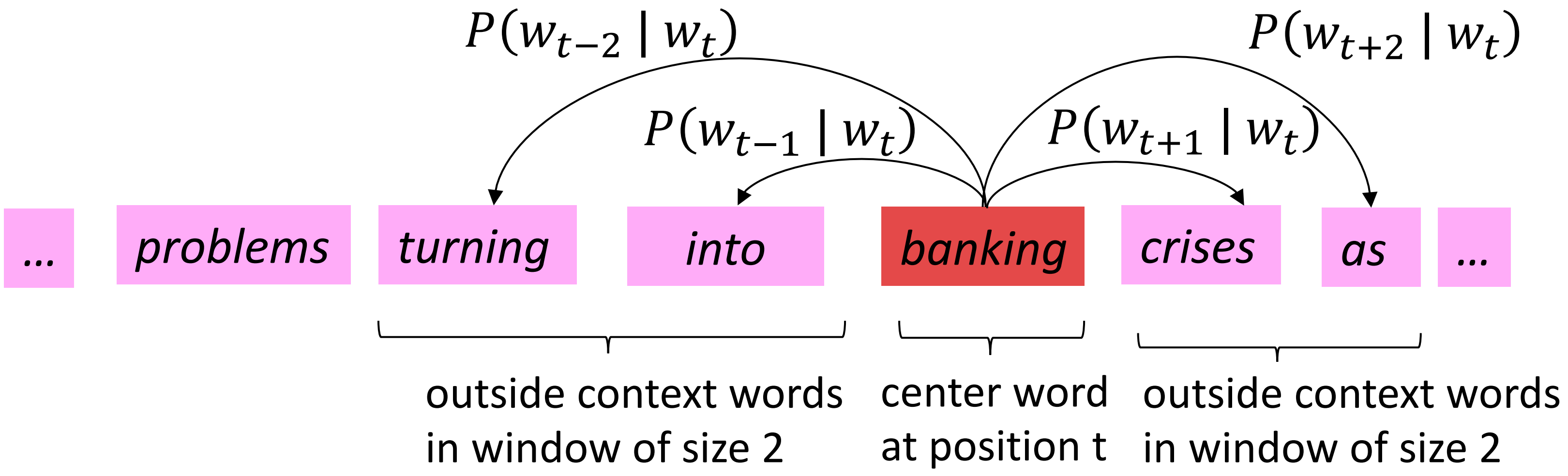
Word2Vec Overview

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



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:

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

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:

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

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

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

- **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

  } These word vectors are subparts of the big vector of all parameters θ
- 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: 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

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

③ Normalize over entire vocabulary to give probability distribution

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

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

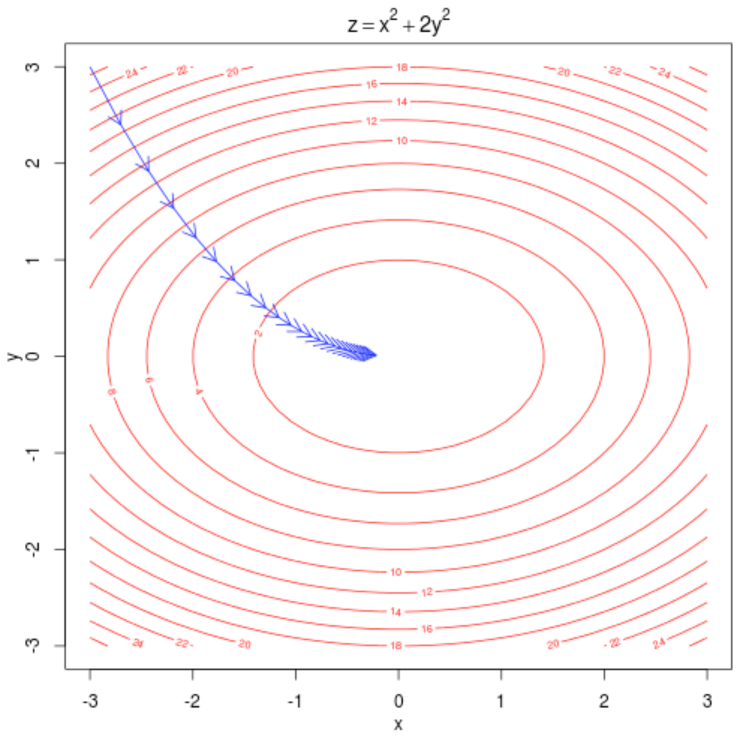
But sort of a weird name because it returns a distribution!

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!

Interactive Session!

- $L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P\left(w_{t+j} \mid w_t; \theta\right)$
- For a center word *c* and a context word *o*: $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$

4. Objective Function

Maximize $J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} p(w'_{t+j} \mid w_t; \theta)$

Or minimize ave. neg. log likelihood $J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log p(w'_{t+j} \mid w_t)$

[negate to minimize; log is monotone]

Text length

window size

where

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

word IDs

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

We now take derivatives to work out minimum

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

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

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

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.

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

$$= (u_o)_j$$

Each term is zero except when $i=j$

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

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

Important to change index

$$\frac{\partial}{\partial v_c} f(\overbrace{g(v_c)}^{z}) = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial v_c}$$

Use chain rule

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

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

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

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

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

Distribute term across sum

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

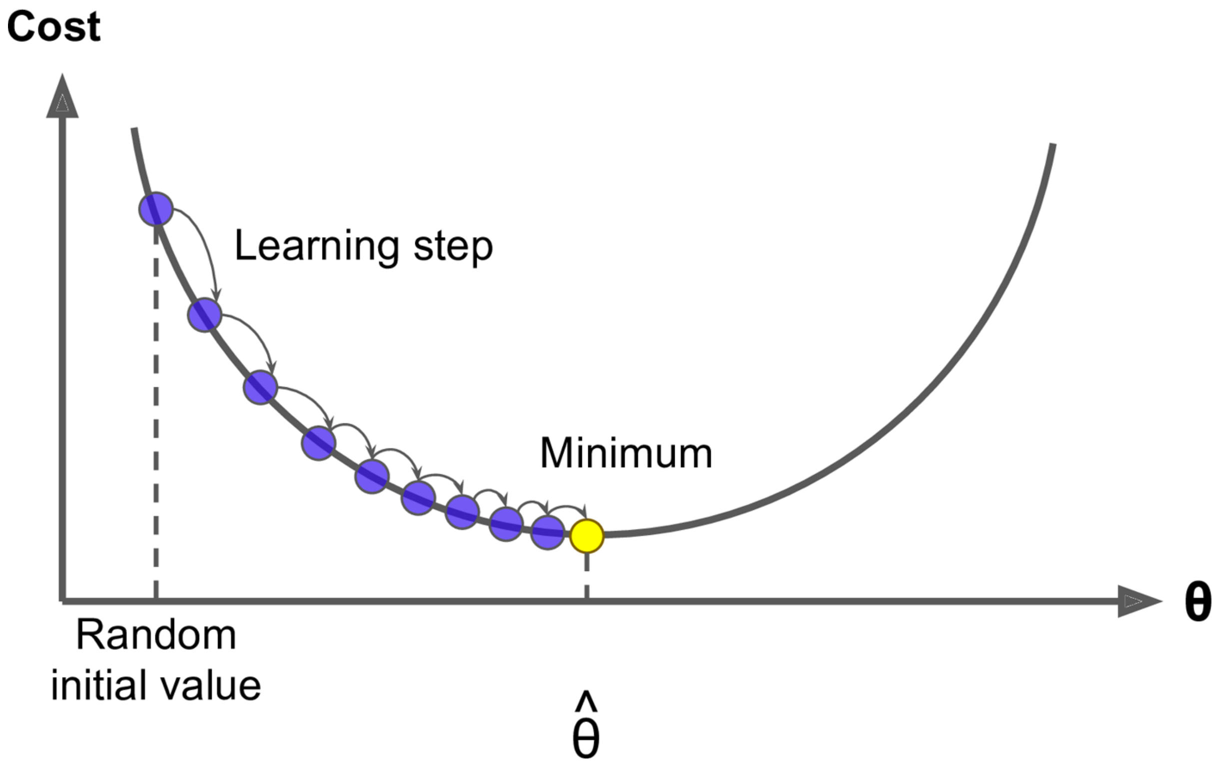
This an expectation: average over all context vectors weighted by their probability

= observed − expected

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:

Mini Batch Gradient Descent

```
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