Natural Language Processing with Deep Learning
CS224N/Ling284

Richard Socher

Lecture 2: Word Vectors

Organization

- PSet 1 is released. Coding Session 1/22: (Monday, PA1 due Thursday)

- Some of the questions from Piazza:

- sharing the choose-your-own final project with another class seems fine--> Yes*

- But how about the default final project? Can that also be used as a final project for a different course?--> Yes*

- Are we allowing students to bring one sheet of notes for the midterm?--> Yes

- Azure computing resources for Projects/PSet4. Part of milestone

Lecture Plan

1. Word meaning (15 mins)
2. Word2vec introduction (20 mins)
3. Word2vec objective function gradients (25 mins)
4. Optimization refresher (10 mins)

1/11/18

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) $\iff$ signified (idea or thing)

= denotation

1/11/18

How do we have usable meaning in a computer?

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

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
for synset in wn.synsets("good"):
    print "(%s)" % synset.pos(),
    print ", ".join([l.name() for l in synset.lemmas()])
```

```
(adj) full, good
(adj) estimable, good, honorable, respectable
(adj) beneficial, good
(adj) good, just, upright
(adj) adept, expert, good, practiced,
proficient, skillful
(adj) dear, good, near
(adj) good, right, ripe
…
(adv) well, good
(adv) thoroughly, soundly, good
(n) good, goodness
(n) commodity, trade good, 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')]
```

1/11/18

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

- Hard to compute accurate word similarity →

1/11/18

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel

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)

1/11/18

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 rely on WordNet's list of synonyms to get similarity?
- **Instead: learn to encode similarity in the vectors themselves**

Representing words by their context

- Core idea: **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**

1/11/18

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.

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

Note: word vectors are sometimes called word embeddings or word representations.
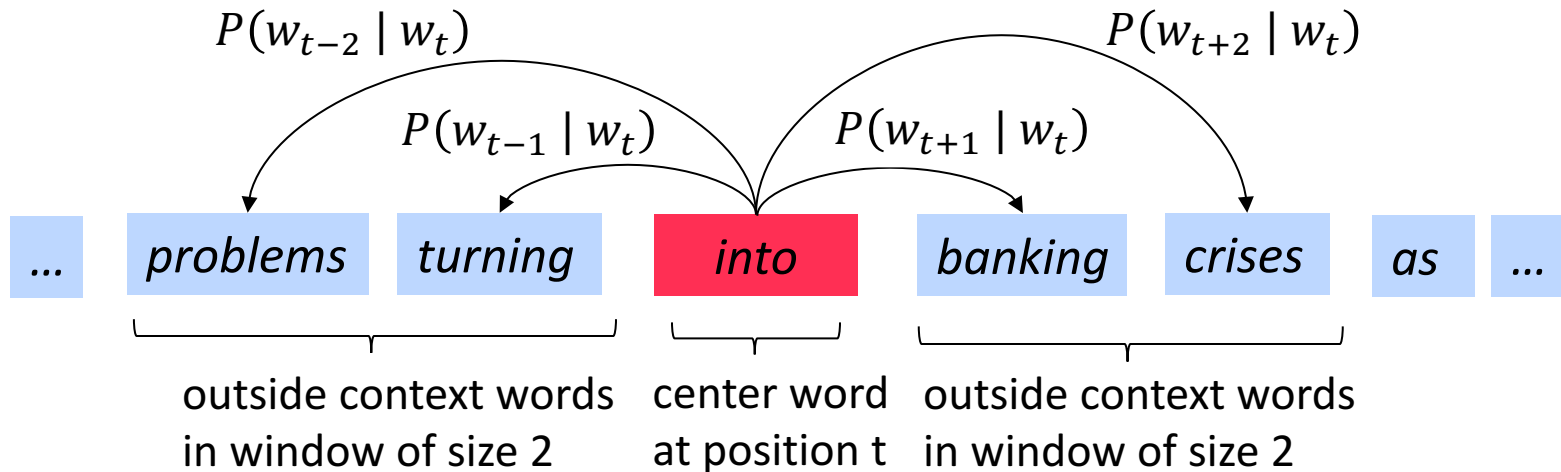
1/11/18

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

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

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

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

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

| ... | *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 Overview

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

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

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

| … | *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
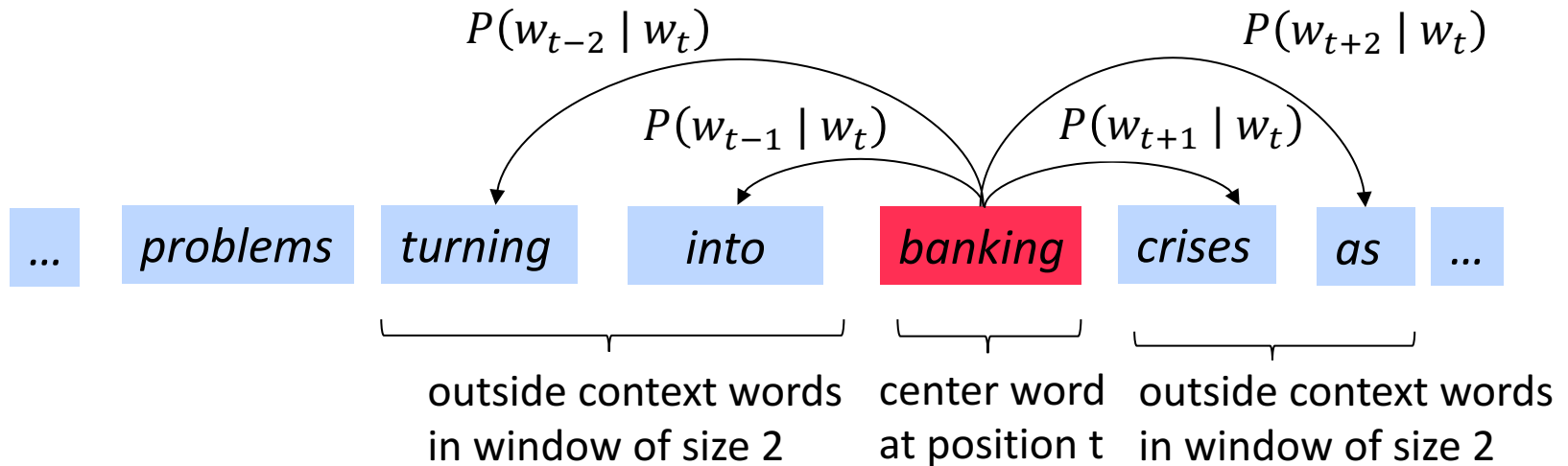
Word2vec: objective function

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

Likelihood =

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

θ is all variables to be optimized

sometimes called *cost* or *loss* function

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

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

Minimizing objective function $\Longleftrightarrow$ Maximizing predictive accuracy

1/11/18

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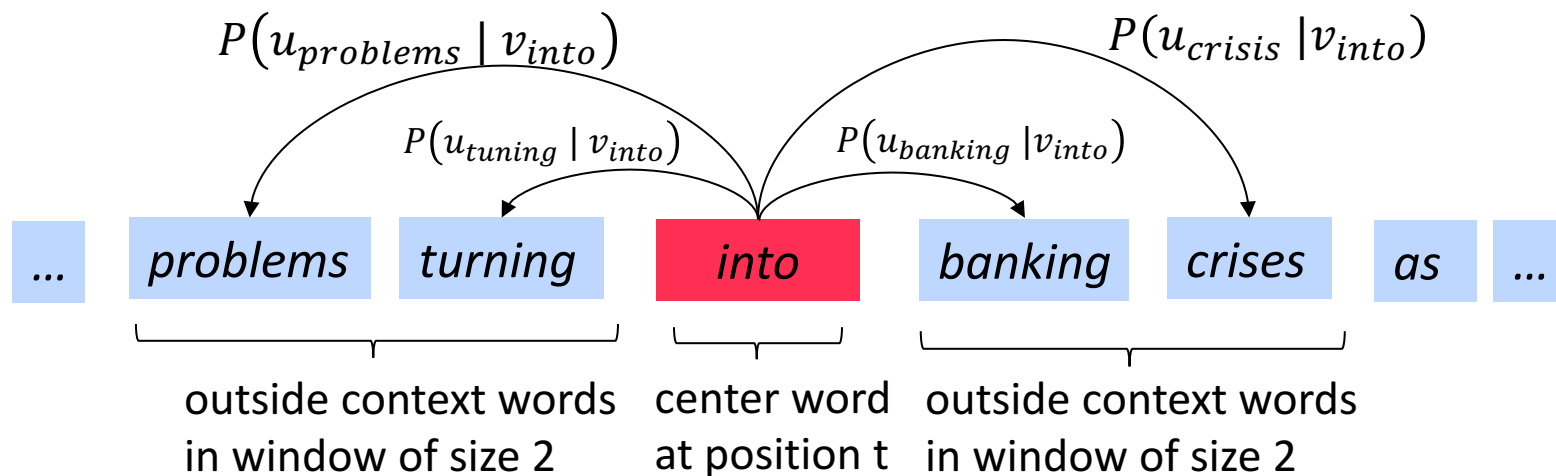
- We want to minimize the objective function:

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

- <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*:

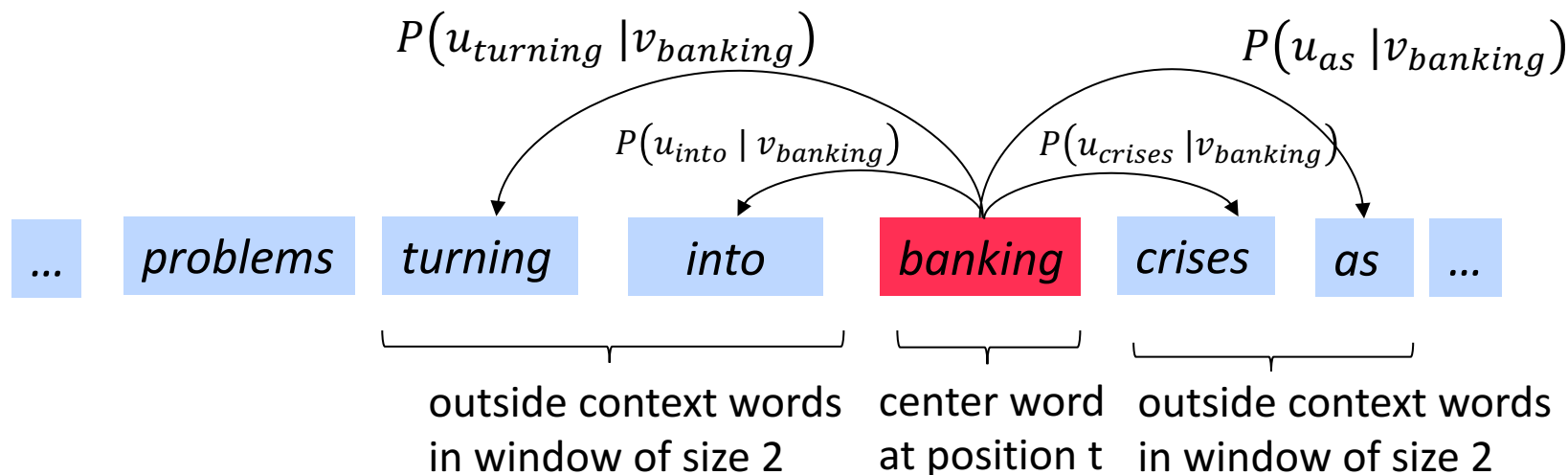$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

1/11/18

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

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

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

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

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

| … | *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 Overview with Vectors

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

$$P\left(u_{turning} \mid v_{banking}\right) \qquad P\left(u_{as} \mid v_{banking}\right)$$

$$P\left(u_{into} \mid v_{banking}\right) \qquad P\left(u_{crises} \mid v_{banking}\right)$$

| … | *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

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

Dot product compares similarity of *o* and *c*. Larger dot product = larger probability

After taking exponent, normalize over entire vocabulary

- 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

18   1/11/18

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 then optimize these parameters

3. 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:

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

1/11/18

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

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

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

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

Interactive Whiteboard Session!

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

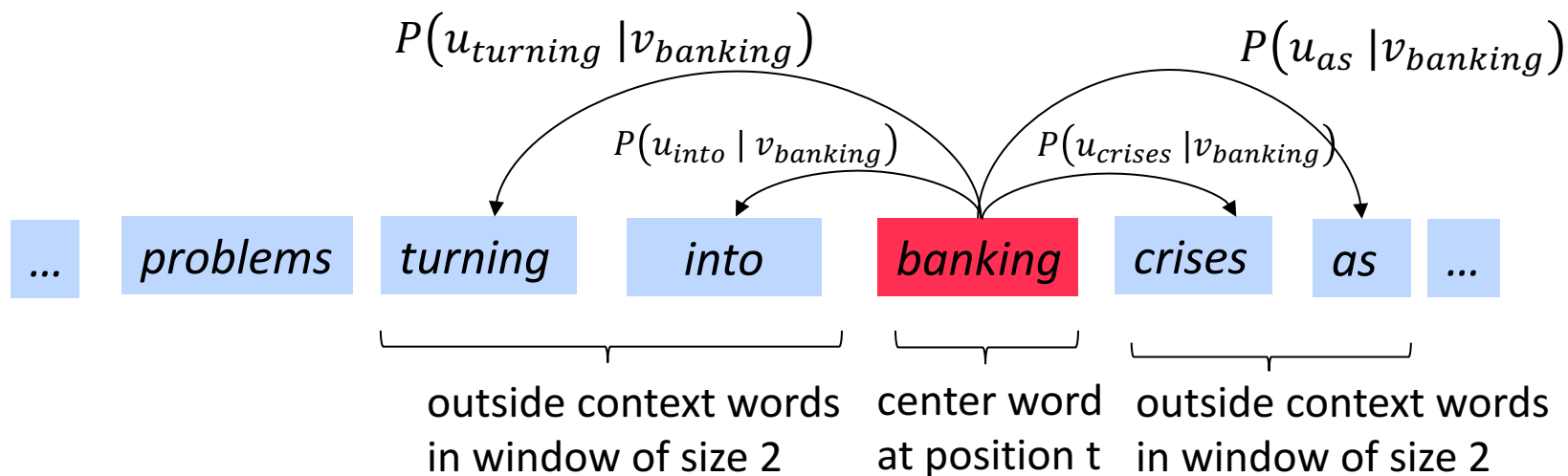Let's derive gradient for center word together
For <u>one example window</u> and <u>one example outside word</u>:

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

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:

$$P(u_{turning} \mid v_{banking}) \qquad P(u_{as} \mid v_{banking})$$

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

| ... | *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)

1/11/18

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.

Cost

Learning step

Minimum

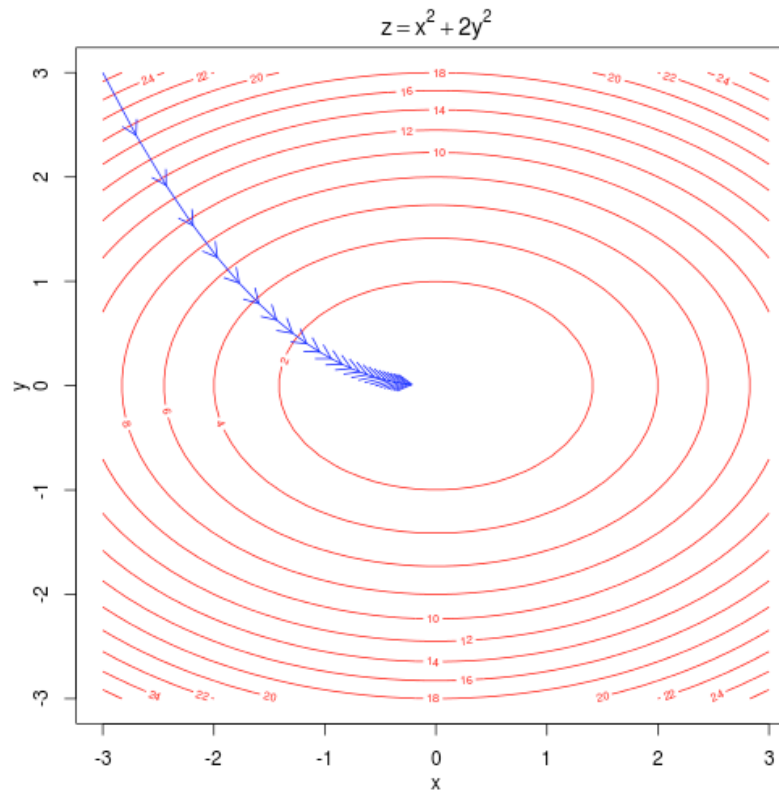Random initial value

$\hat{\theta}$

θ
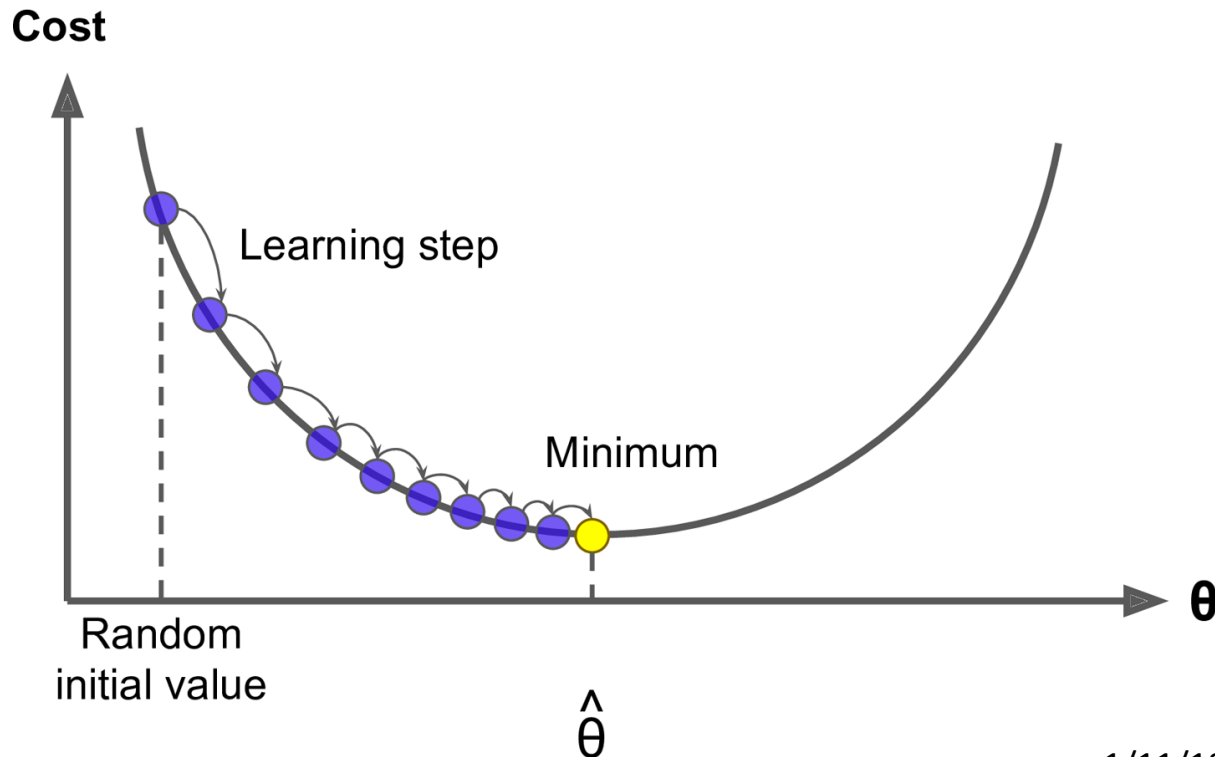
Note: Our objectives are not convex like this :(

Intuition

For a simple convex function over two parameters.

Contour lines show levels of objective function

$$z = x^2 + 2y^2$$

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
```

1/11/18

Stochastic Gradient Descent

- <u>Problem</u>: $J(\theta)$ 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!
- <u>Solution</u>: **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
```

1/11/18

PSet1: The skip-gram model and negative sampling

- From paper: "Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al. 2013)
- Overall objective function (they maximize): $J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta)$

$$J_t(\theta) = \log \sigma \left(u_o^T v_c \right) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)} \left[\log \sigma \left(-u_j^T v_c \right) \right]$$

- The sigmoid function! $\sigma(x) = \frac{1}{1+e^{-x}}$ (we'll become good friends soon)
- So we maximize the probability of two words co-occurring in first log →
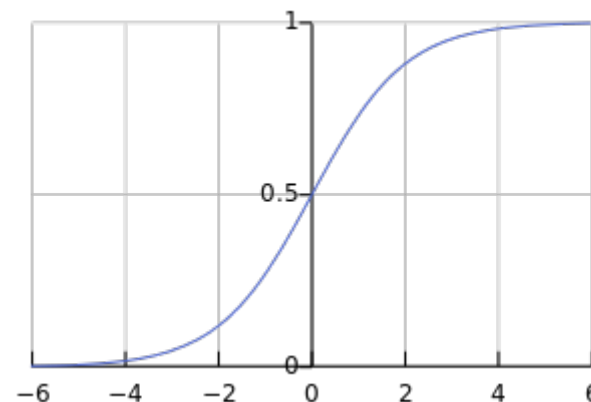
PSet1: The skip-gram model and negative sampling

- Simpler notation, more similar to class and PSet:

$$J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))$$

- We take k negative samples.

- Maximize probability that real outside word appears, minimize prob. that random words appear around center word

- P(w)=U(w)$^{3/4}$/Z,
  the unigram distribution U(w) raised to the 3/4 power
  (We provide this function in the starter code).

- The power makes less frequent words be sampled more often

1/11/18

PSet1: The continuous bag of words model

- Main idea for continuous bag of words (CBOW): Predict center word from sum of surrounding word vectors instead of predicting surrounding single words from center word as in skip-gram model

- To make assignment slightly easier:

  Implementation of the CBOW model is not required (you can do it for a couple of bonus points!), but you do have to do the theory problem on CBOW.

1/11/18