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 (→ 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

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

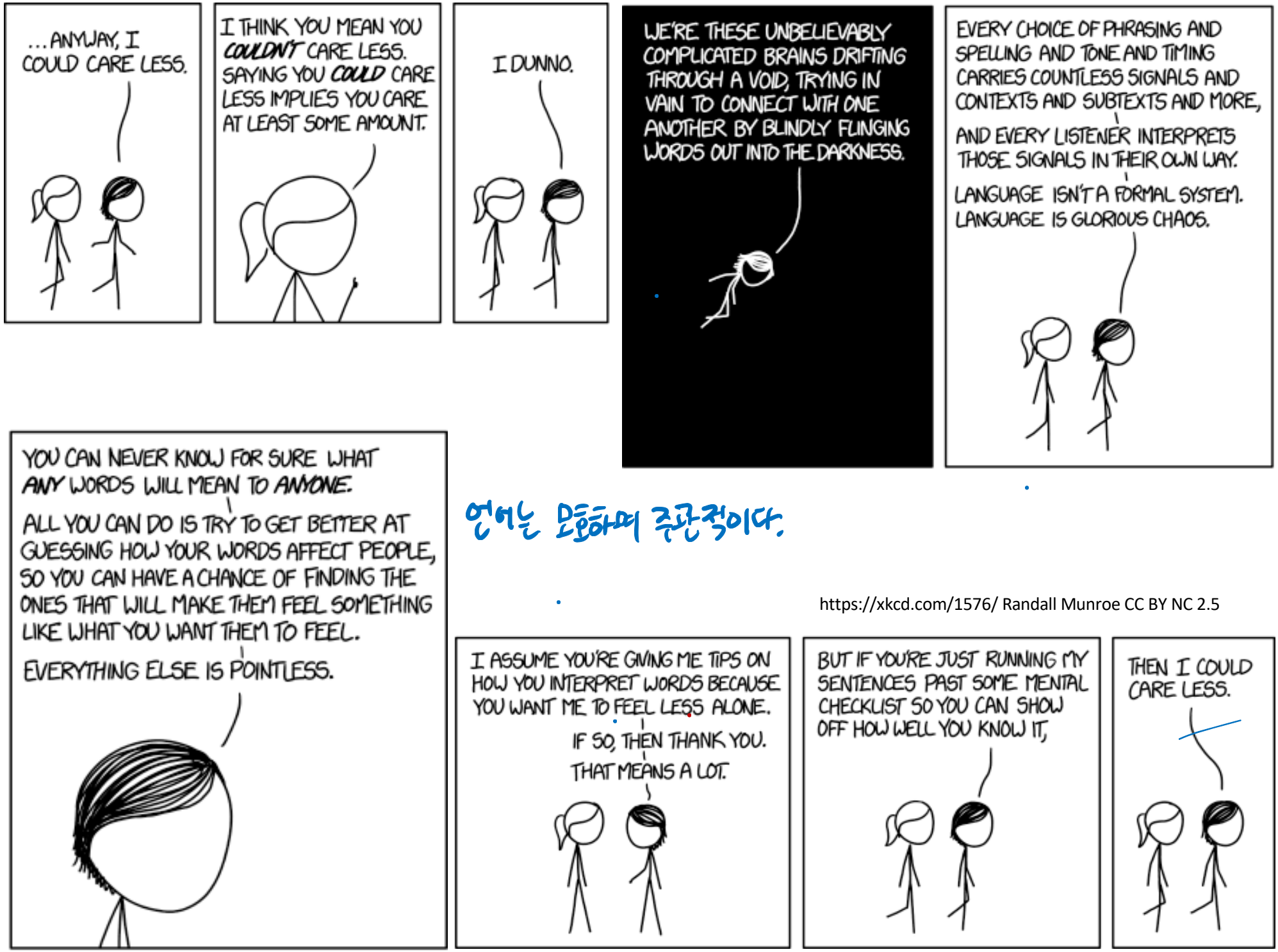
- 5 x 1-week Assignments: 6% + 4 x 12%: 54%
  - HW1 is released today! Due next Tuesday! At 4:30 p.m.
  - Please use @stanford.edu email for your Gradescope account
- Final Default or Custom Course Project (1–3 people): 43%
  - Project proposal: 5%, milestone: 5%, poster: 3%, report: 30%
  - Final poster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



언어는 모호하며 주관적이다.


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

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

명시적 의미

대표 단어들과의 관계를 통해 단어를 나타낼 수 있다 (좋은가?)

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

뉘앙스

- 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 주관적
- Requires human labor to create and adapt
  - 많은 사람이 일일이 작성
- Can't compute accurate word similarity →
  - fixed synonym set → 실제는 공통점이 있어도 set이 없으면 반영이 안 됨

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

희소한 상은 단어가 존재함 → 매우 고차원

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. 직교

There is no natural notion of **similarity** for one-hot vectors!

유사도가 없다!!

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**

벡터들이 스스로 유사도를 학습하도록

Representing words by their context

중요한건 같이 쓰이는 단어들! (J. R. Firth)

- 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***

주변 단어들이 중심 단어를 나타낼수 있다.

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 =

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

실제로는 더 높은 차원을 사용

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

Word meaning as a neural word vector – visualization

100차원을 그림으로 나타낸 것

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

① 가까운 단어는 유사도가 있다는 정도의 정보만 의미함.
하지만 더 많은 정보를 얻을 수도 있음(나중에 배움)

3. Word2vec: Overview

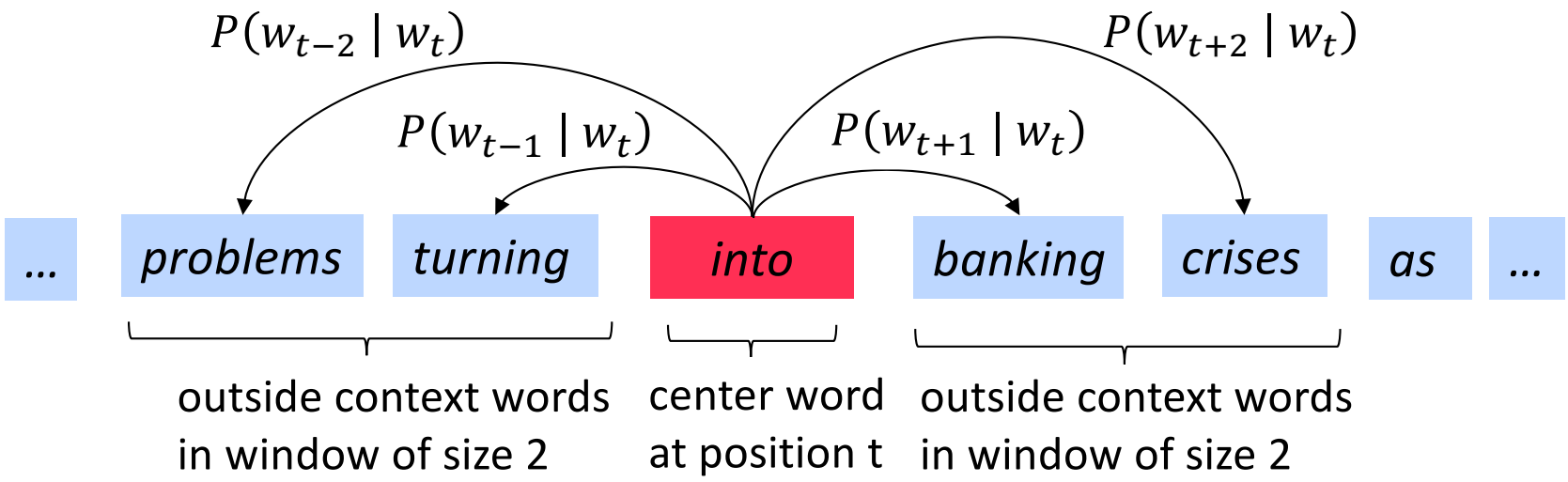
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

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

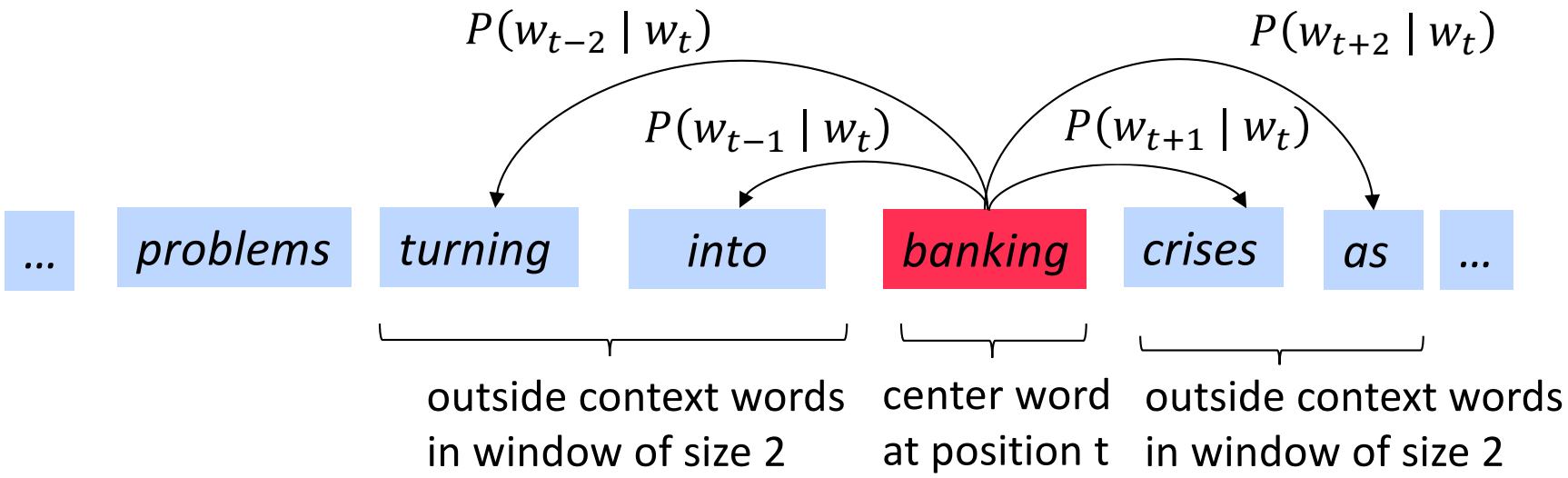
- Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2Vec Overview

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

주변 단어들을 이용해서 단어의 벡터를 계산한다.



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

θ is all variables to be optimized

only parameter: vector representation

sometimes called *cost* or *loss* function

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

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

average

$\log$ × → +

Minimizing objective function ⟺ 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
- Then for a center word *c* and a context word *o*:

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

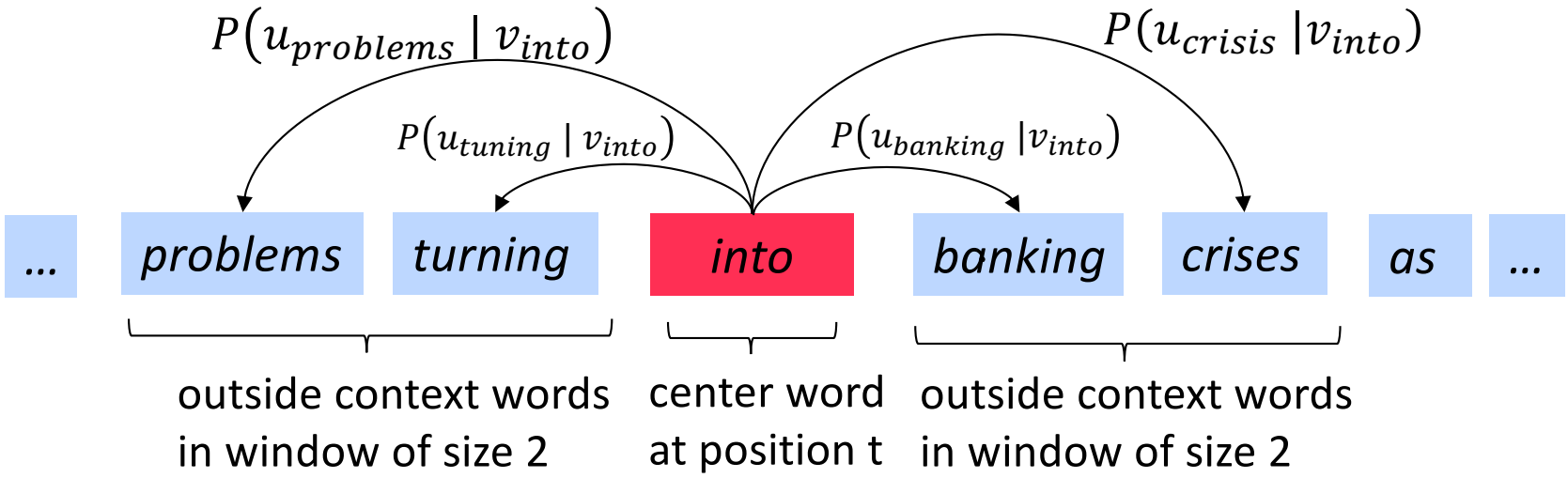
그현 단어와 중심 단어의 코사인 유사도

V = 입력층과 은닉층을 잇는 W의 행벡터
U = 은닉층과 출력층을 잇는 W'의 열벡터

그현 단어와 등장하지 않은 단어와 중심 단어의 내적(코사인 유사도)

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

Exponentiation makes anything positive

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

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

$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

Normalize over entire vocabulary to give probability distribution

V_w = w is center

U_w = w is context

대응되는 단어의 내적의 합으로 나눔 (≒ softmax)

- 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

remind

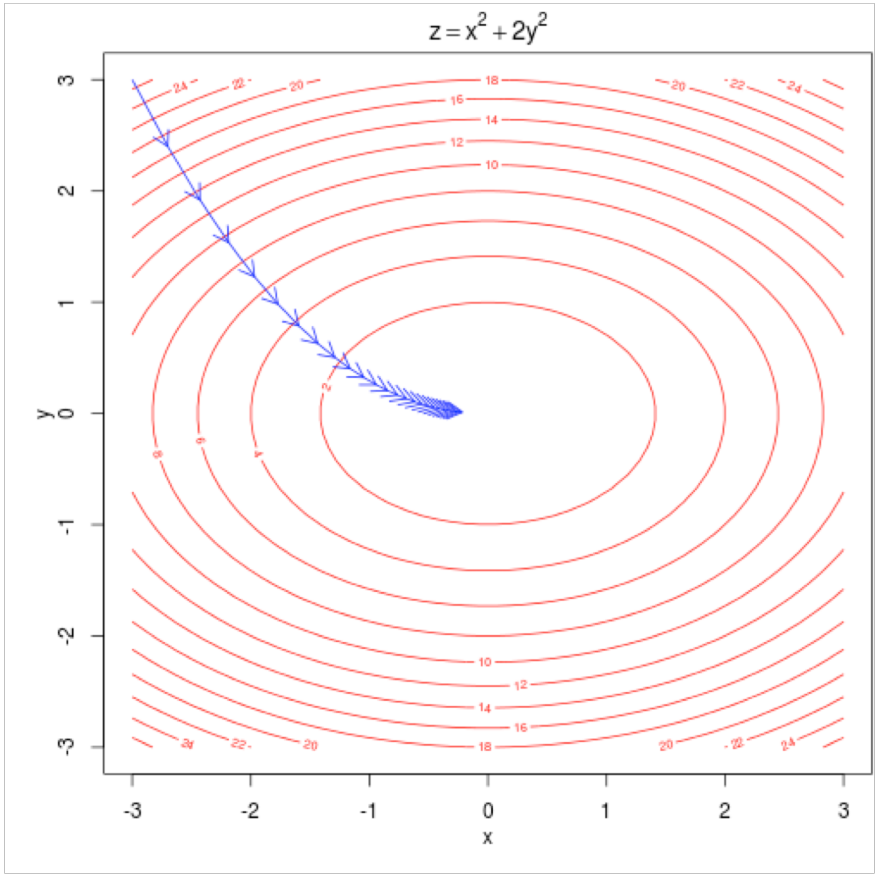
그래서 목적은 → 중앙의 말는 단어가 나온들

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



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

$$\max J'\theta = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} p(w'_{tj} | w_t; \theta)$$

$$J\theta = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 1}} \log p(w'_{tj} | w_t)$$

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

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

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

분모

$$\frac{\partial}{\partial v_c} u_o^T v_c$$

$\hookrightarrow u_o$

분자

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

f $\quad$ z(v_c)

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

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

f $\quad$ z(v_c)

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

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

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

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

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

$p(x|c)$

각 단어의 확률로 가중합한 값!!

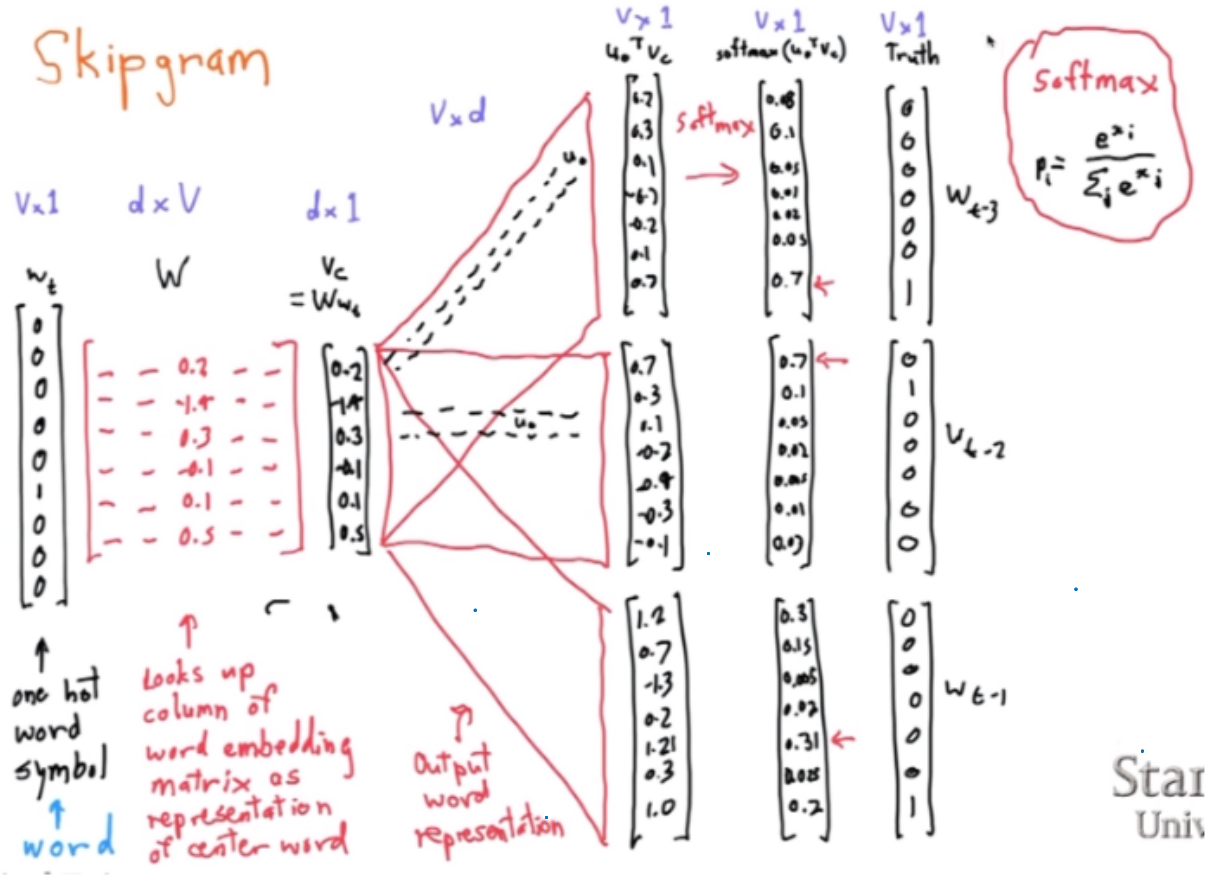
"크로스 엔트로피"

Corss-Entropy loss = **$-\log(y_i')$**, where $y_i' = softmax(u_o^t v_c)$

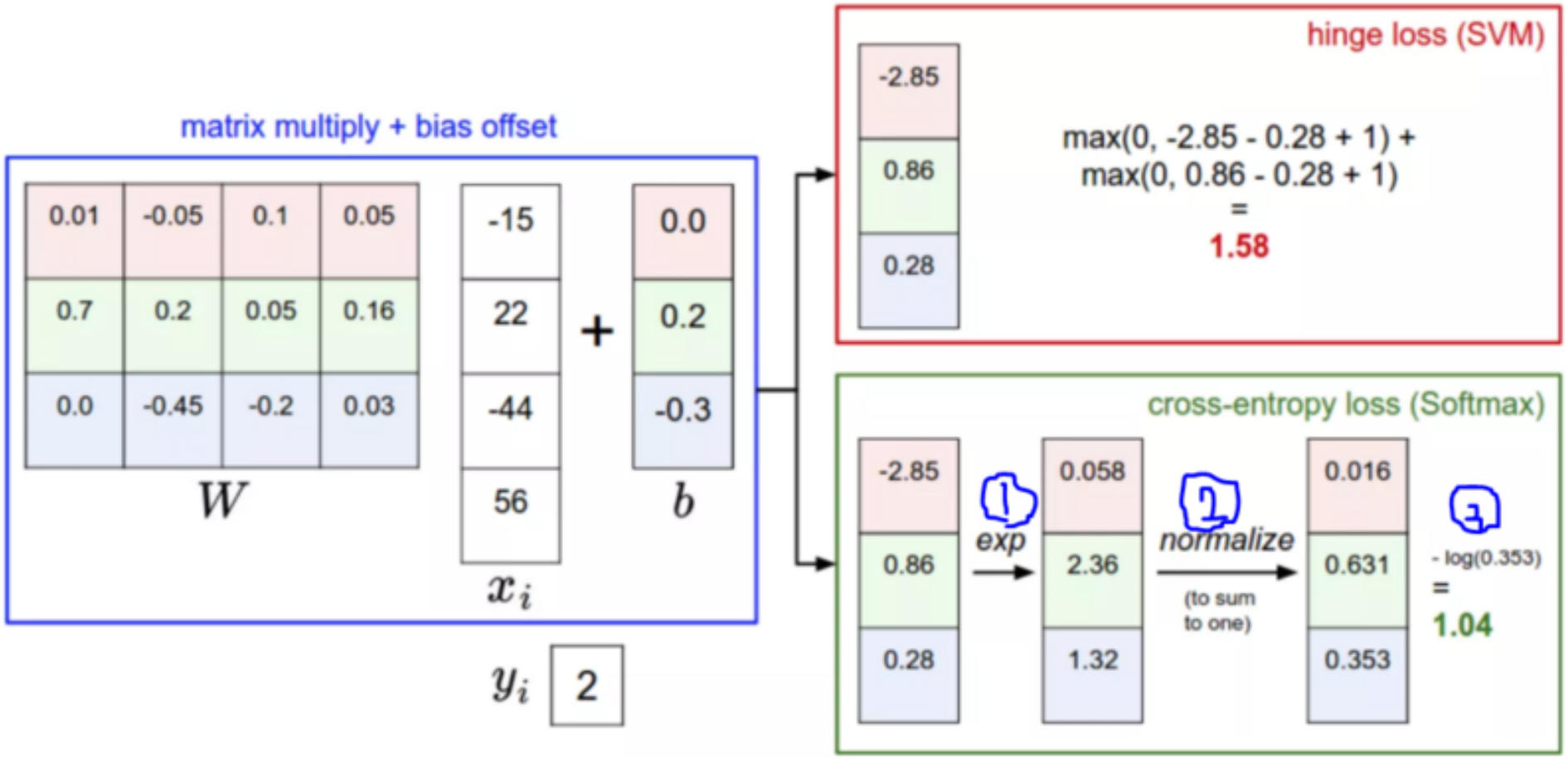
a Loss(θ) = **$-\log (\exp(u_o^t v_c) / \sum \exp(u_w^t v_c))$** : **context 한개 loss**

$J'(\theta) = -\sum_{-m<=j<=m} (\log (\exp(u_{o+j}^t v_c) / \sum \exp(u_w^t v_c)))$: context 전체 loss

$J(\theta) = -(1/T) \sum_{c=1}^{C} \sum_{-m<=j<=m} (\log (\exp(u_{o+j}^t v_c) / \sum \exp(u_w^t v_c)))$: sentence 전체 loss



1. V×1 = 단어 갯수 V의 one hot vector
2. d×V = center word의 word embedding 차원의 크기로 (d = embedding size)
3. V×d 의 행렬곱은 scoring 작업 진행 후 output word representation
4. softmax
5. cross entropy


matrix multiply + bias offset
0.01
-0.05
0.1
0.05
0.7
0.2
0.05
0.16
0.0
-0.45
-0.2
0.03
W
-15
22
-44
56
x_i
+
0.0
0.2
-0.3
b
y_i
2
hinge loss (SVM)
-2.85
0.86
0.28
max(0, -2.85 - 0.28 + 1) +
max(0, 0.86 - 0.28 + 1)
=
1.58
cross-entropy loss (Softmax)
-2.85
0.86
0.28
1
exp
0.058
2.36
1.32
2
normalize
(to sum
to one)
0.016
0.631
0.353
3
- log(0.353)
=
1.04

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:

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

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

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

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

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

Interactive Whiteboard Session!

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

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.

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

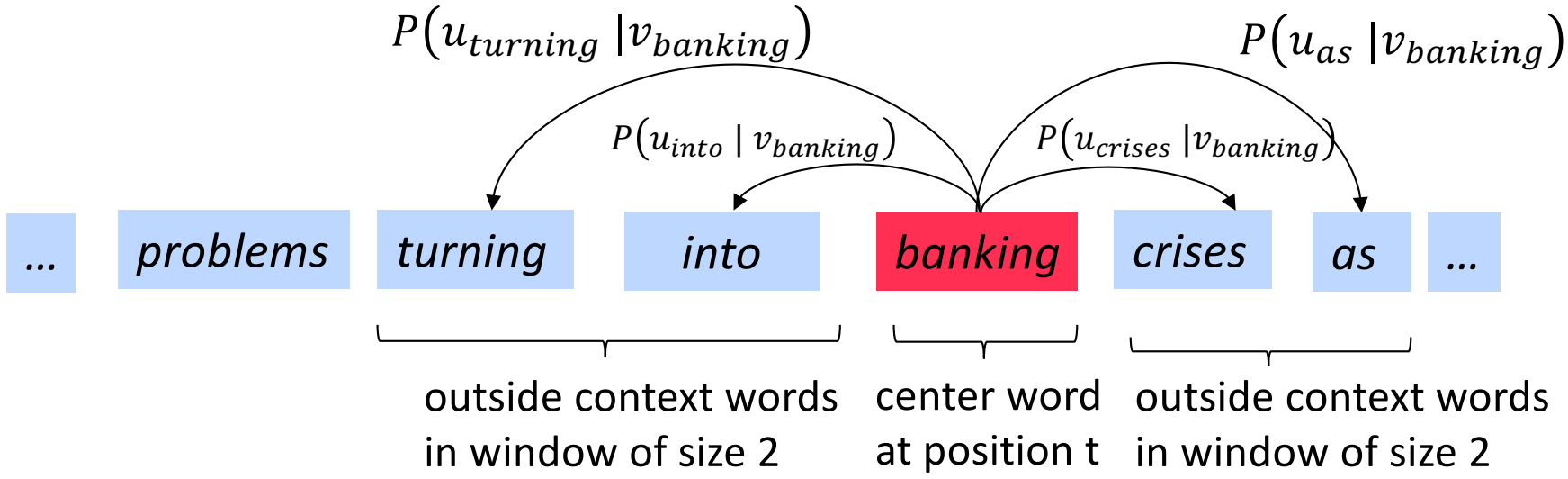
$p(x|c)$

차근차근

observed

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:



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**

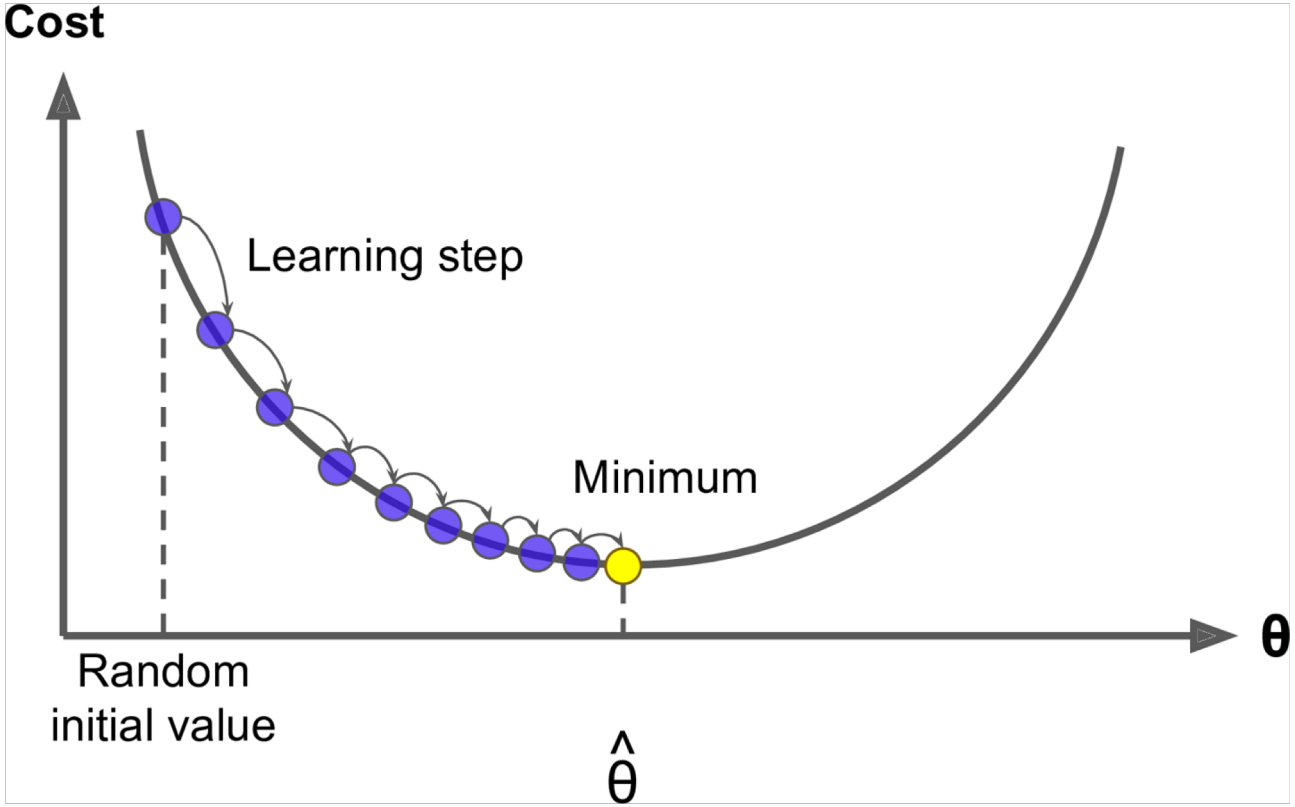
Additional efficiency in training:

1. Negative sampling

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

5. Optimization: Gradient Descent

- We have a cost function $J(\theta)$ 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 :(

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)