Word Meaning

Word Meaning

- Word2Vec
- …
- ..

Word2Vec

WordNet

- **Synonym and hypernym(“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')]
```

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1204/slides/cs224n-2020-lecture01-wordvecs1.pdf

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+)

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

Solution: **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

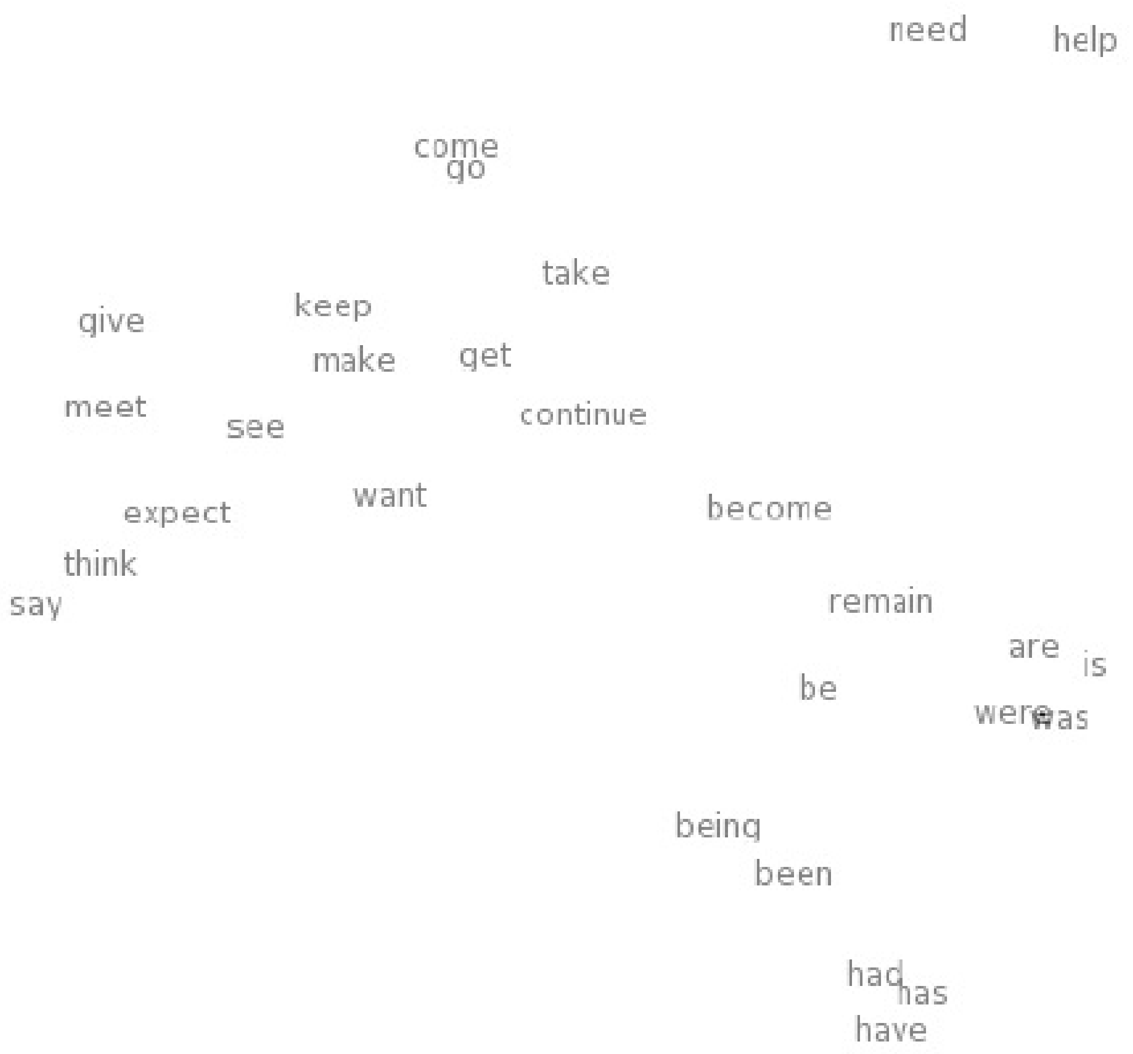
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector dot (scalar) product

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

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

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



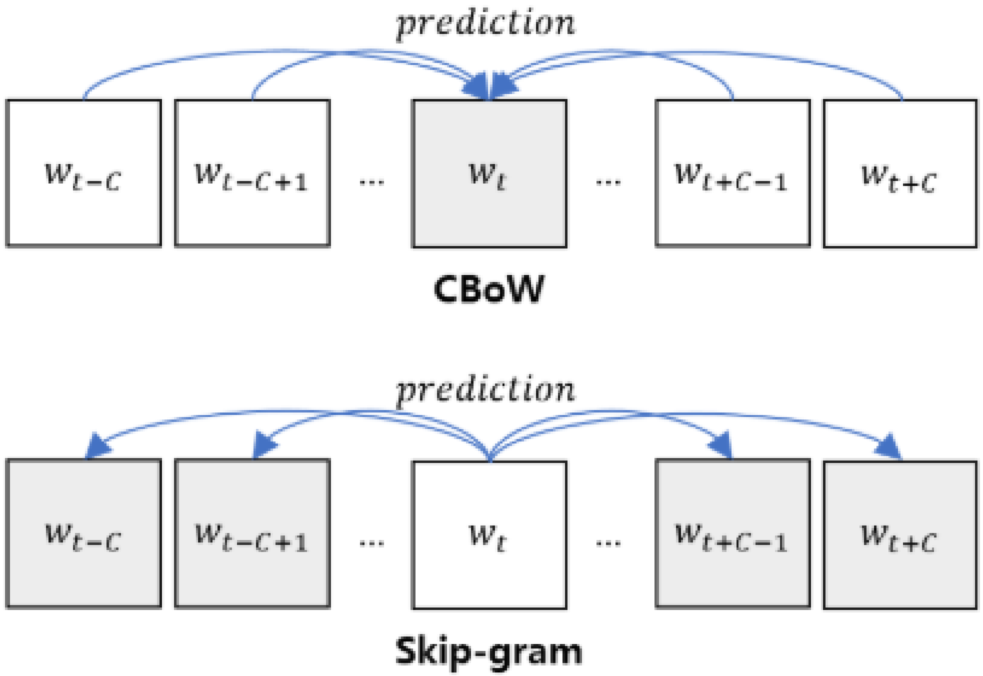
Word2Vec

Word2Vec: Overview

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

Word2Vec

- Skip-Gram
- CBOW(Continuous Bag of Words)

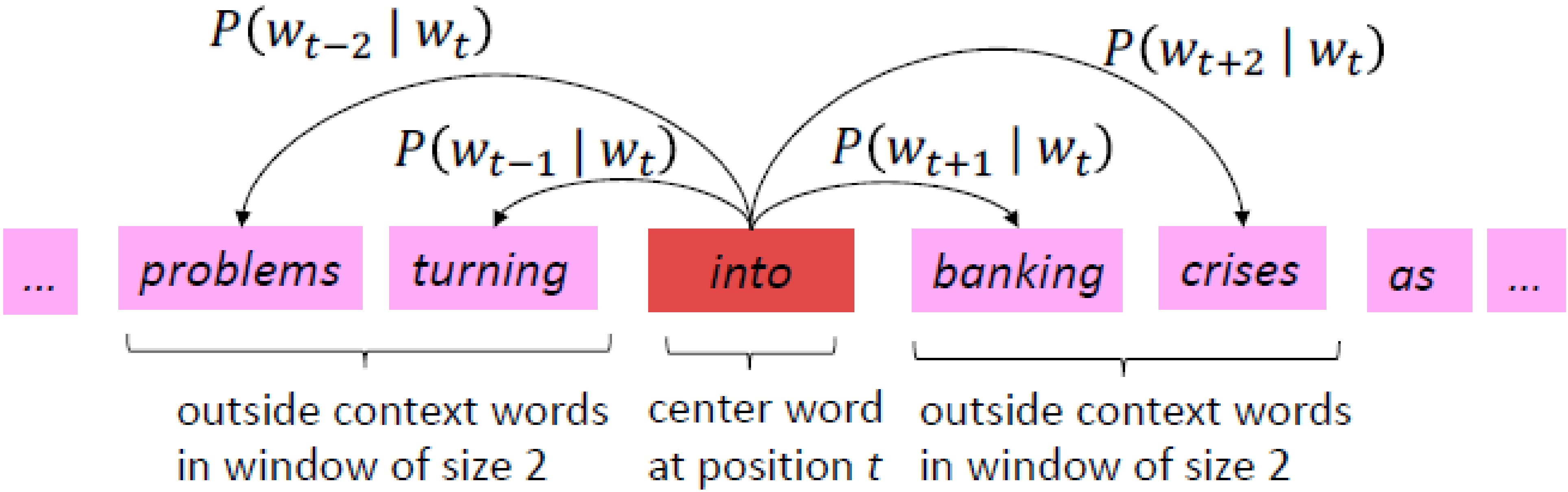


Predict center word based on neighbors

Predict neighbors based on center word

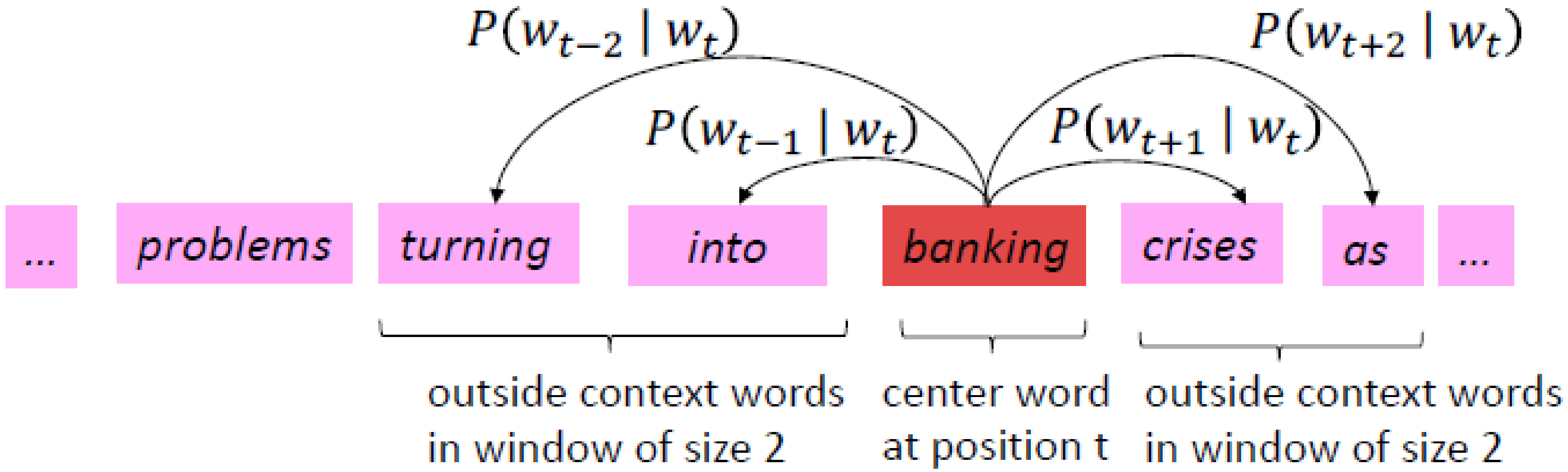
Word2Vec (skip-gram): Overview

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



Word2Vec: Skip-Gram Overview

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



Word2Vec: Skip-Gram Overview

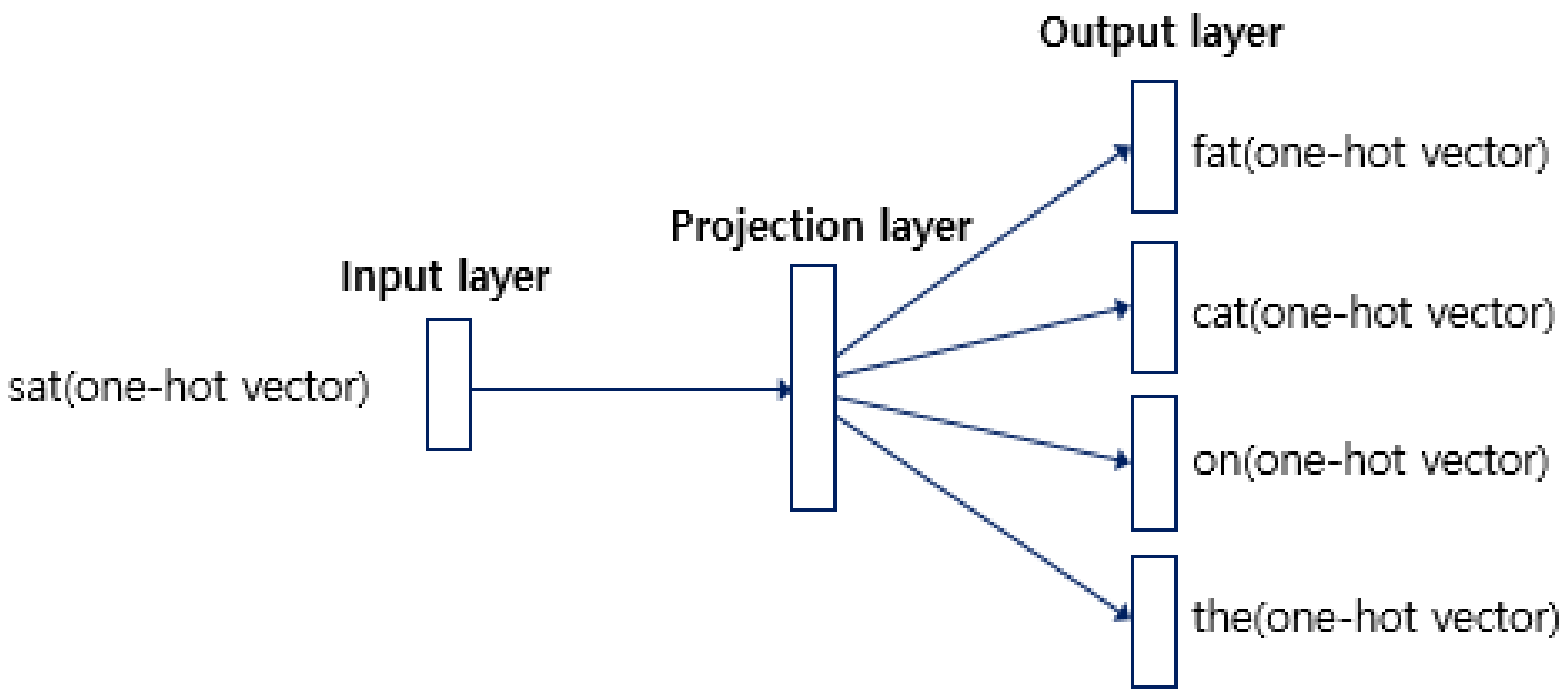
center word
context word

The fat cat sat on the mat

The fat cat sat on the mat

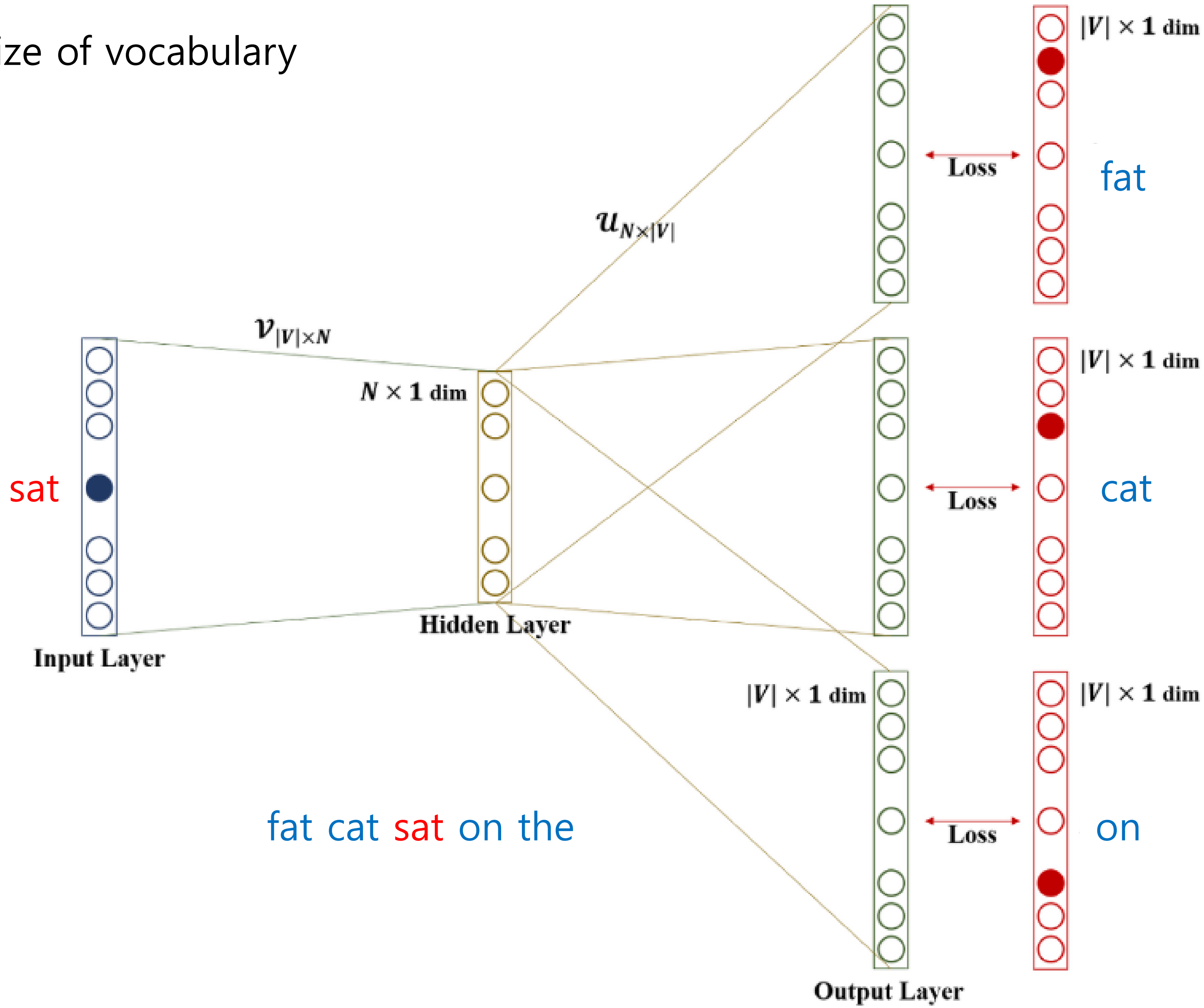
center word	context word
cat	The
cat	Fat
cat	sat
cat	on
sat	fat
sat	cat
sat	on
sat	the

Word2Vec: Skip-Gram Overview



Word2Vec: Skip-Gram Overview

$|V|$: size of vocabulary



fat cat sat on the

Word2Vec: Objective Function

For each position $t = 1, \ldots, T$, 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
- 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 with Vectors

② Exponentiation makes anything positive

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

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

③ Normalize over entire vocabulary to give probability distribution

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

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- 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

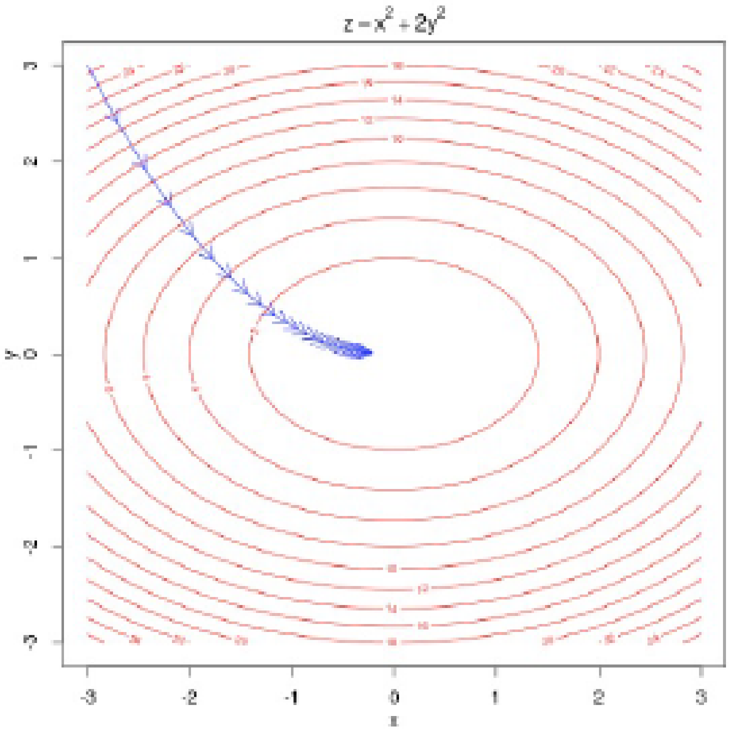
But sort of a weird name because it returns a distribution!

Training the Model: Optimize value of parameters

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!

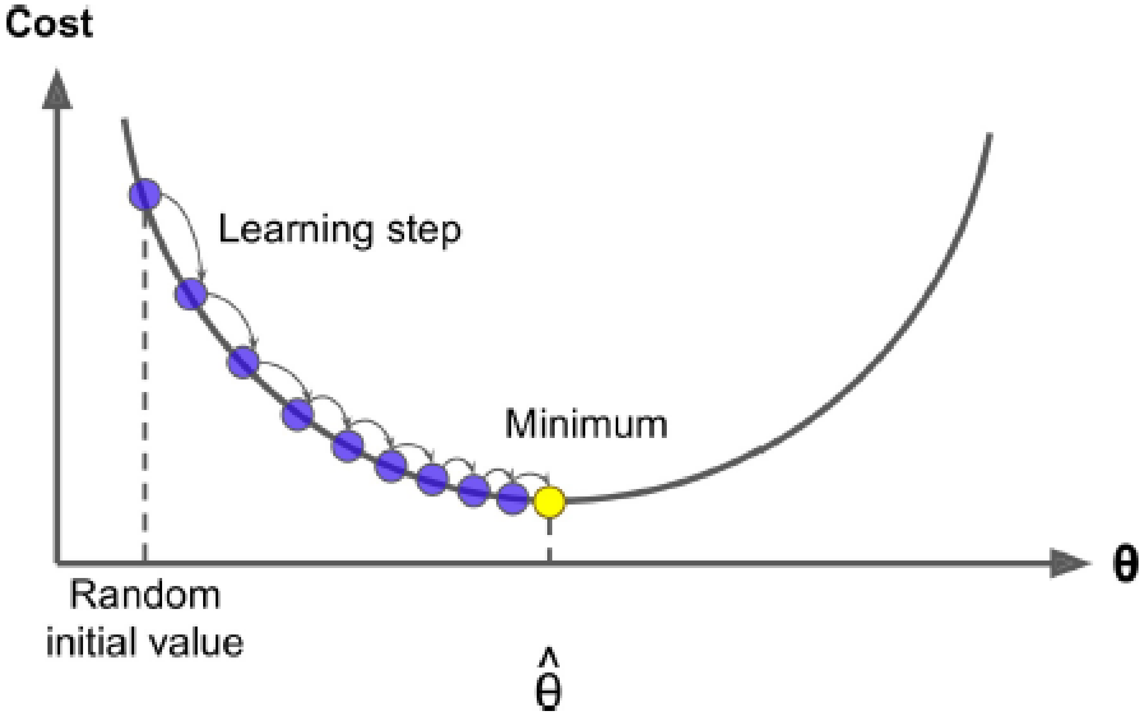
Exercise (not evaluated):

- Derive this

$$\frac{d\ p(o|c)}{d\ v_c}$$

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

Skip-gram with Negative Sampling

- The normalization term is computationally expensive

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

  ➔ Use Skip-gram model with Negative Sampling

- Main idea: train binary logistic regressions for a true pair (center word and a word in its context window) versus several "noise" pairs (the center word paired with a random word)

"Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al. 2013)

Skip-gram with Negative Sampling

The input is the central word, the model's prediction is the context word

Both the central word and the context words are input, and the probability of whether these two words are actually neighbors within the window size is predicted.

https://wikidocs.net/69141

Skip-gram with Negative Sampling

입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	sat	1
cat	on	1

Negative Sampling

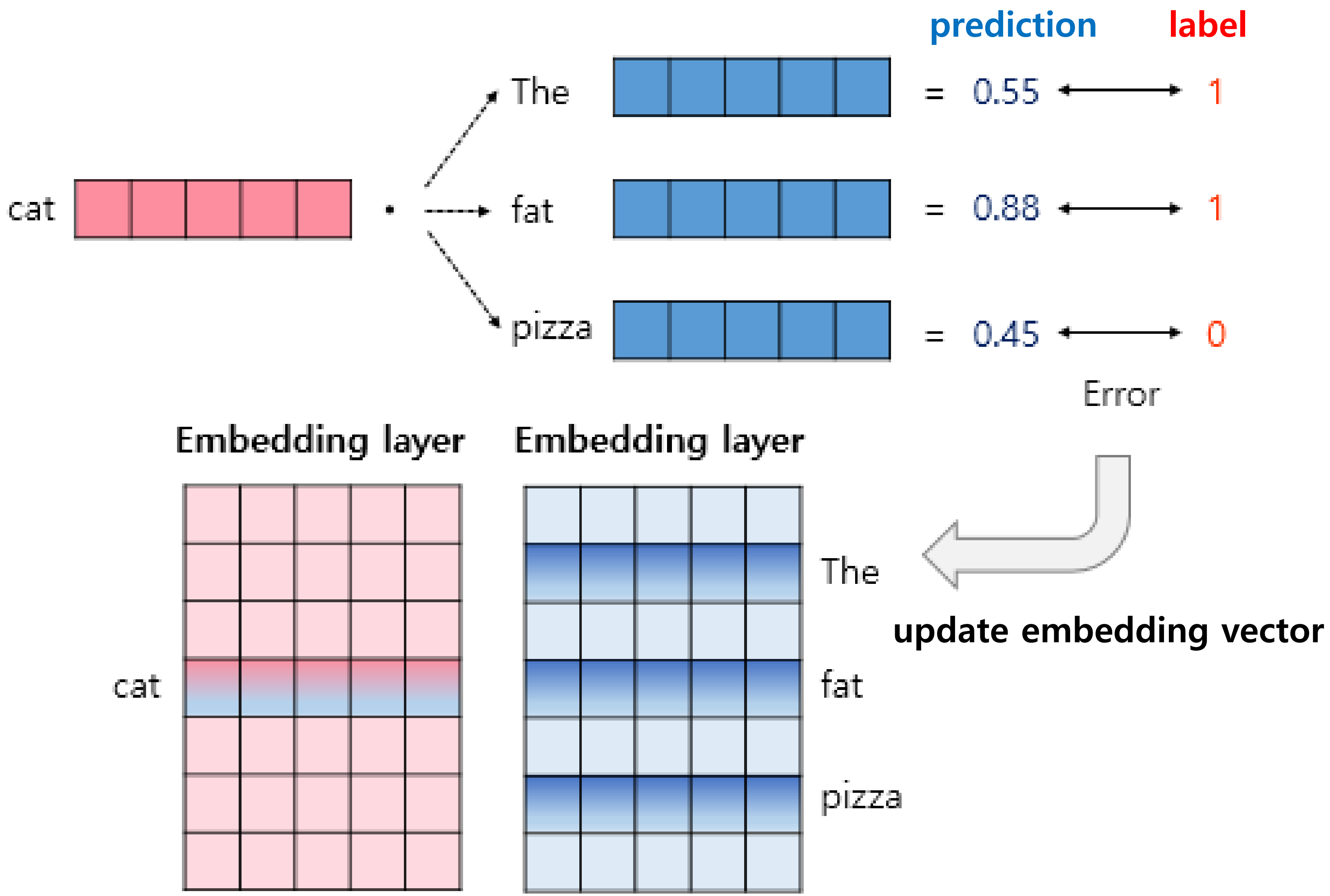
입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	**pizza**	**0**
cat	**computer**	**0**
cat	sat	1
cat	on	1

randomly sampled set of negative examples are taken for each word

Skip-gram with Negative Sampling

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the weights as the embeddings

Skip-gram with Negative Sampling



https://wikidocs.net/69141

Skip-gram with Negative Sampling

Objective Function (they maximize): $J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta)$

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

Maximize the probability of two words co-occurring in first log and minimize probability of noise words in second part

Skip-gram with Negative Sampling

A pair of words that appear near each other where w being a word and c its context

$$p(D = 1 \mid w, c)$$

the pair is not in the training data : $p(D = 0 \mid w, c) = 1 - p(D = 1 \mid w, c)$

→ We have to optimize this:

$$\arg\max_{\theta} \prod_{(w,c)\in D} p(D = 1|w, c; \theta) \prod_{(w,c)\in D'} p(D = 0|w, c; \theta) \qquad (2)$$

→ converting the max of products to max of sum of logarithms,

$$\arg\max_{\theta} \sum_{(w,c)\in D} \log p(D = 1|w, c; \theta) + \sum_{(w,c)\in D'} \log (1 - p(D = 1|w, c; \theta)) \qquad (3)$$

How to compute p(D=1|w,c)?

Intuition:

- Words are likely to appear near similar words
- Model similarity with dot-product!
- Similarity(t,c) $\propto$ t · c

Problem:

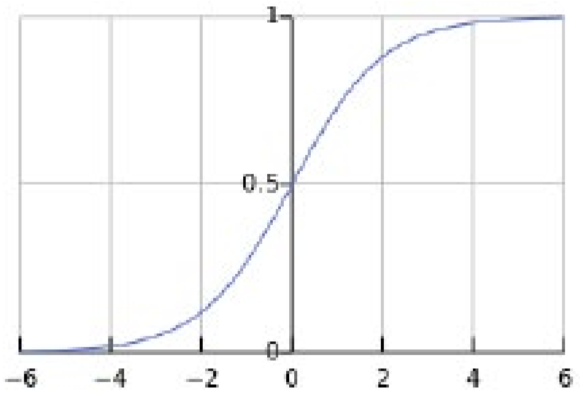
Dot product is not a probability!

(Neither is cosine)

Turning dot product into a probability:

Use the logistic/sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Skip-gram with Negative Sampling

We can compute $p(D = 1 \mid w, c\ ; \theta)$ using the sigmoid function, where v_w and v_c are representations of center and context words with the current θ

$$p(D = 1|w, c; \theta) = \sigma(v_c.v_w) = \frac{1}{1 + e^{-v_c.v_w}} \tag{4}$$

Formula (3) becomes:

$$\arg\max_{\theta} \sum_{(w,c)\in D} \log \sigma(v_c.v_w) + \sum_{(w,c)\in D'} \log\ (\sigma(-v_c.v_w)) \tag{5}$$

This is same as Formula (1) summed over entire corpus

Choosing noise words

- Could pick w according to their unigram frequency P(w)
- More common to choose w according to $p_\alpha(w)$

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w count(w)^\alpha}$$

- α= ¾ works well because it gives rare noise words slightly higher probability
- To show this, imagine two events p(a)=.99 and p(b) = .01:

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

윈도우 내에 등장하지 않은 어떤 단어(w)가
negative sample로 뽑힐 확률

maximizing positive pairs and minimizing negative pairs

Center word: drilling

Current context word: engineer

$K = 5$

		$iter = 1$	$iter = 2$	$iter = 3$	$iter = 4$
engineer	$p(D = 1 \| w_{drilling}, c_{engineer})$	0.432	0.653	0.853	0.998
minimized	$p(D = 1 \| w_{drilling}, c_{minimized})$	0.312	0.169	0.042	0.000
primary	$p(D = 1 \| w_{drilling}, c_{primary})$	0.565	0.221	0.083	0.001
concerns	$p(D = 1 \| w_{drilling}, c_{concerns})$	0.102	0.043	0.009	0.000
led	$p(D = 1 \| w_{drilling}, c_{led})$	0.853	0.532	0.115	0.001
page	$p(D = 1 \| w_{drilling}, c_{page})$	0.642	0.380	0.101	0.002

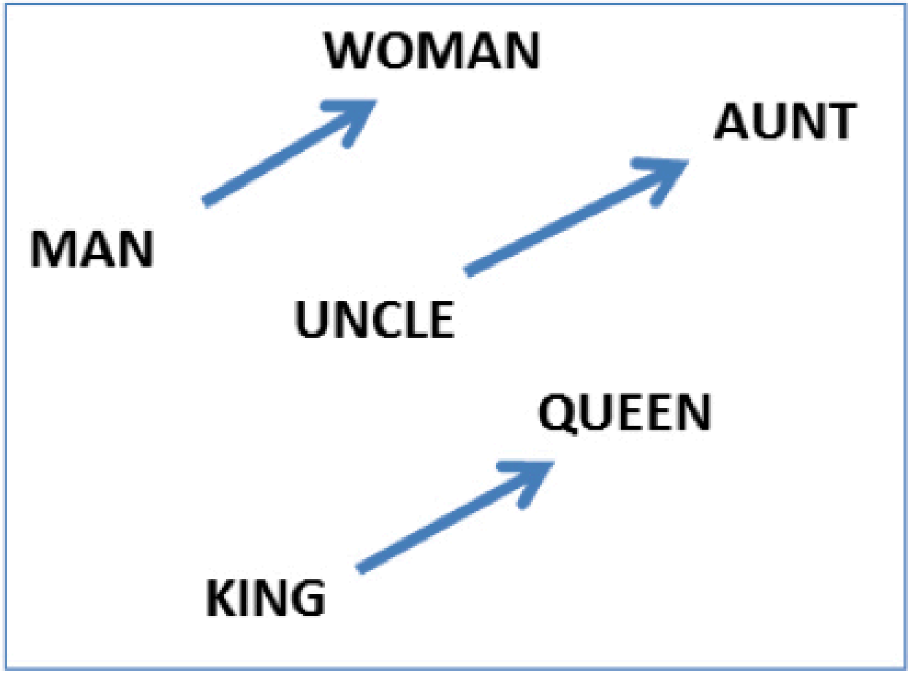
Update θ → Update θ → Update θ

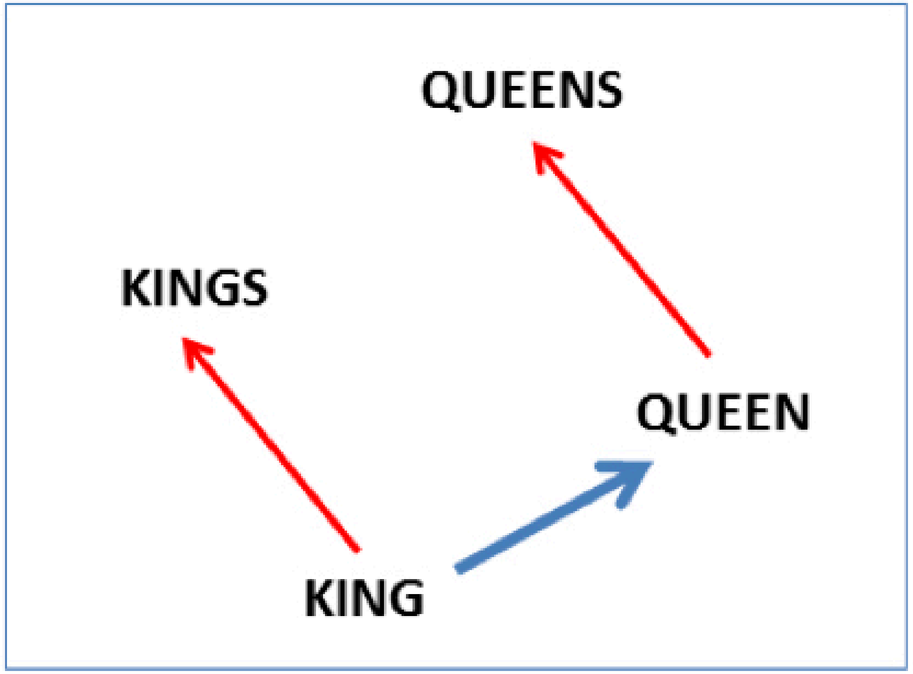
Analogy

- Word embedding → meaning

vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')





Word Representation

- **Word2Vec** takes texts as training data for a neural network. The resulting embedding captures whether words appear in similar contexts.
- **GloVe** focuses on words co-occurrences over the whole corpus. Its embeddings relate to the probabilities that two words appear together.
- **FastText** improves on Word2Vec by taking word parts (characters) into account, too. This trick enables training of embeddings on smaller datasets and generalization to unknown words.