COS 484

Natural Language Processing

# L5: Word Embeddings (II)

Spring 2022

(Some slides adapted from Chris Manning, Dan Jurafsky)

# Approaches for representing words

**Count-based approaches**

- Used since the 90s
- Sparse word-context PPMI matrix
- Decomposed with SVD

**Prediction-based approaches (word embeddings)**

- Formulated as a machine learning problem
- Word2vec (Mikolov et al., 2013)
- GloVe (Pennington et al., 2014)

Underlying theory: The Distributional Hypothesis *(Firth, '57)*
"Similar words occur in similar contexts"

# Word embeddings

- **Learned** vectors from text for representing words
    - Input: a large text corpus, vocabulary $V$, vector dimension $d$
        - Text corpora:
            - Wikipedia + Gigaword 5: 6B tokens
            - Twitter: 27B tokens
            - Common Crawl: 840B tokens
    - Output: $f : V \rightarrow \mathbb{R}^d$

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

Each word is represented by a low-dimensional (e.g., d = 300), real-valued vector

Each coordinate/dimension of the vector doesn't have a particular interpretation

# Word embeddings

- Basic property: similar words have similar vectors

word = "sweden"

| Word | Cosine distance |
|---|---|
| norway | 0.760124 |
| denmark | 0.715460 |
| finland | 0.620022 |
| switzerland | 0.588132 |
| belgium | 0.585835 |
| netherlands | 0.574631 |
| iceland | 0.562368 |
| estonia | 0.547621 |
| slovenia | 0.531408 |

$\cos(u, v)$ ranges between -1 and 1

# Word embeddings

- Basic property: similar words have similar vectors

Nearest words to
frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

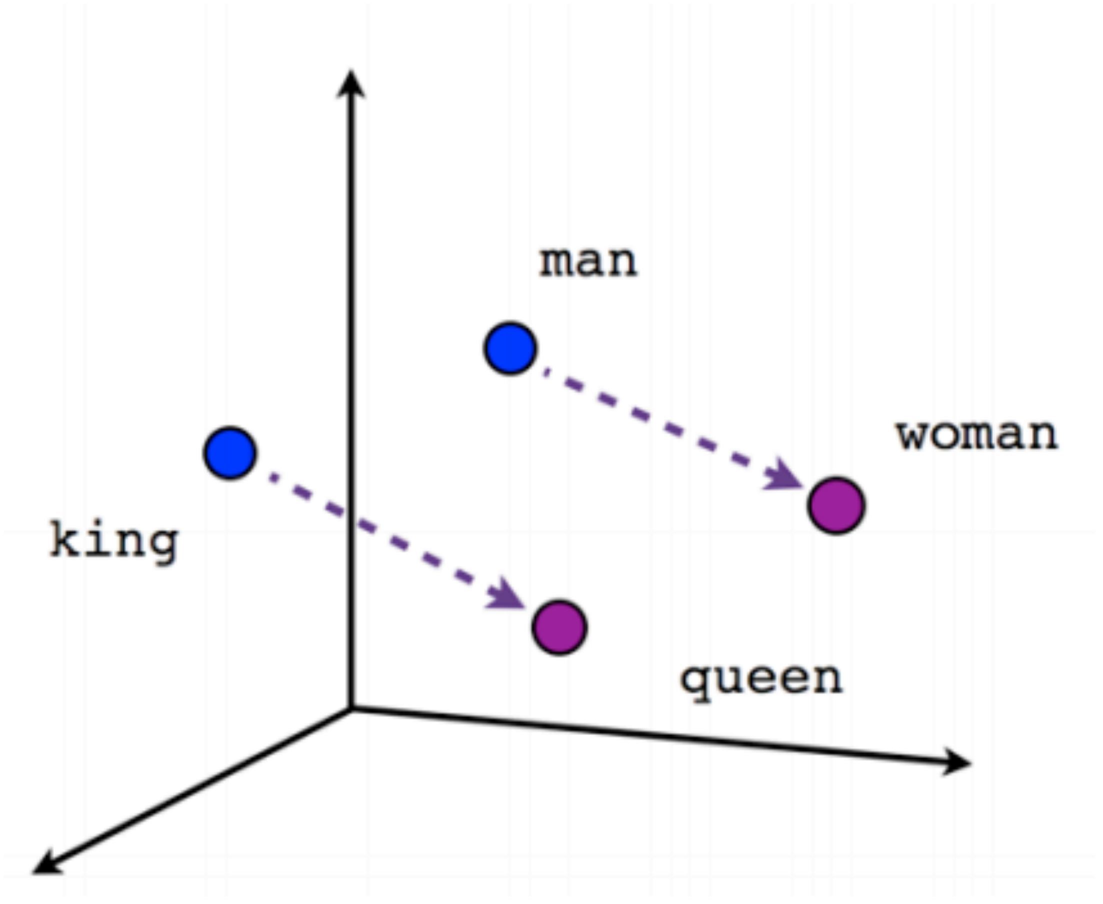
rana

eleutherodactylus

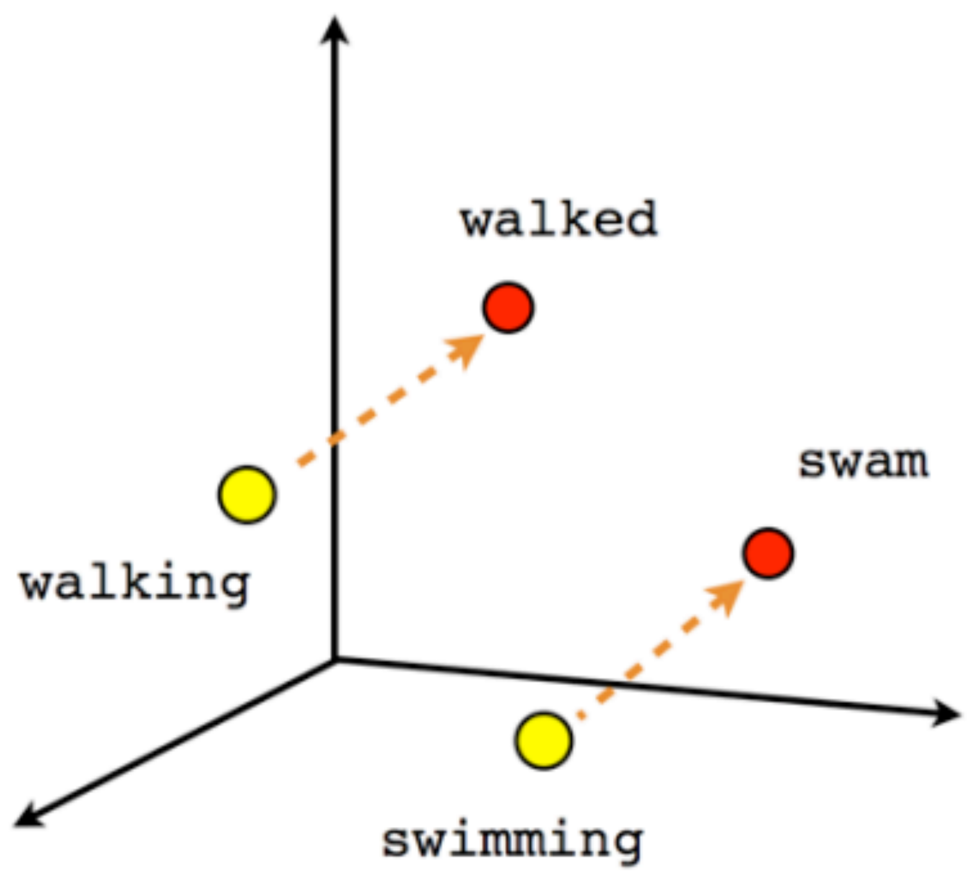
(Pennington et al, 2014): GloVe: Global Vectors for Word Representation

# Word embeddings

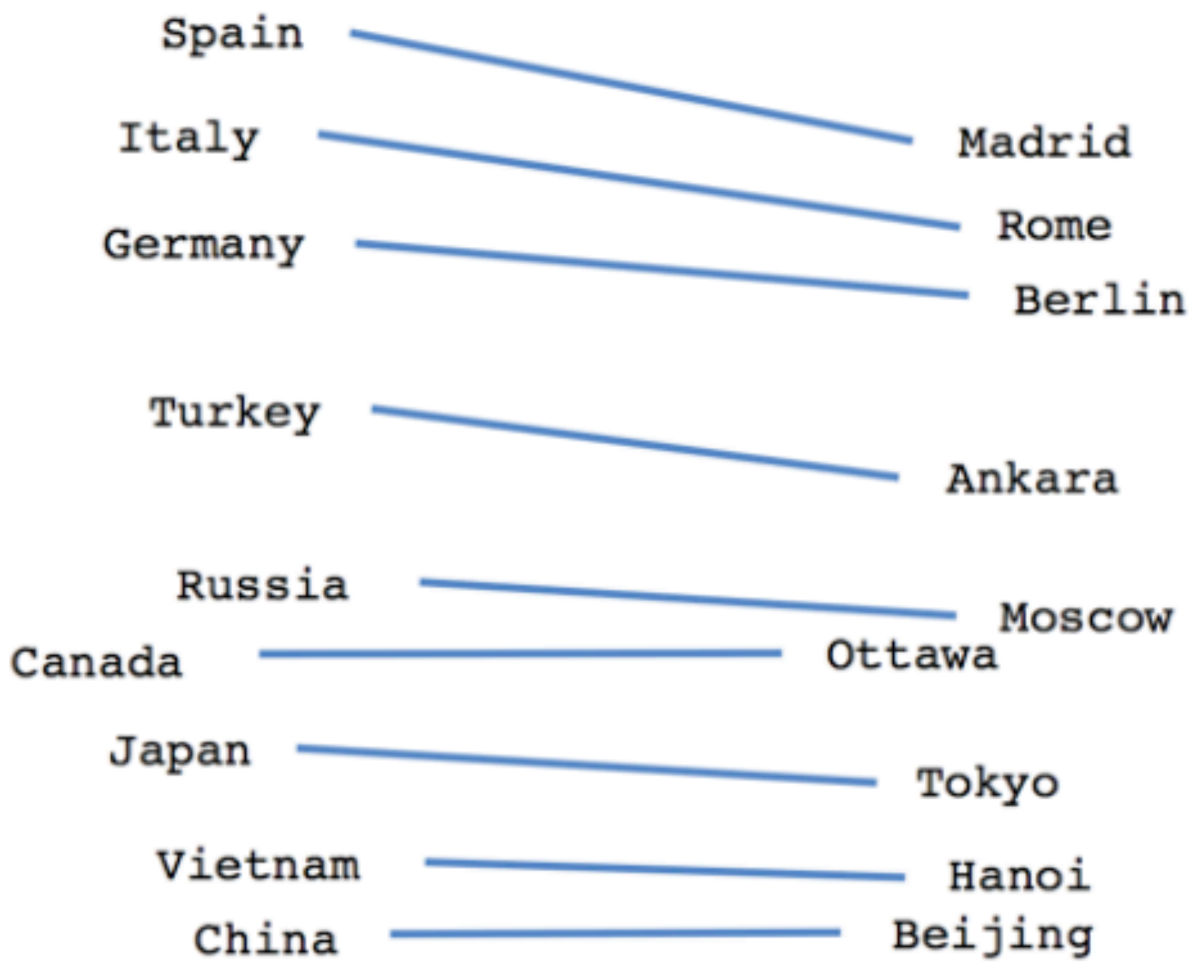
- They have some other nice properties too!




Male-Female




Verb tense



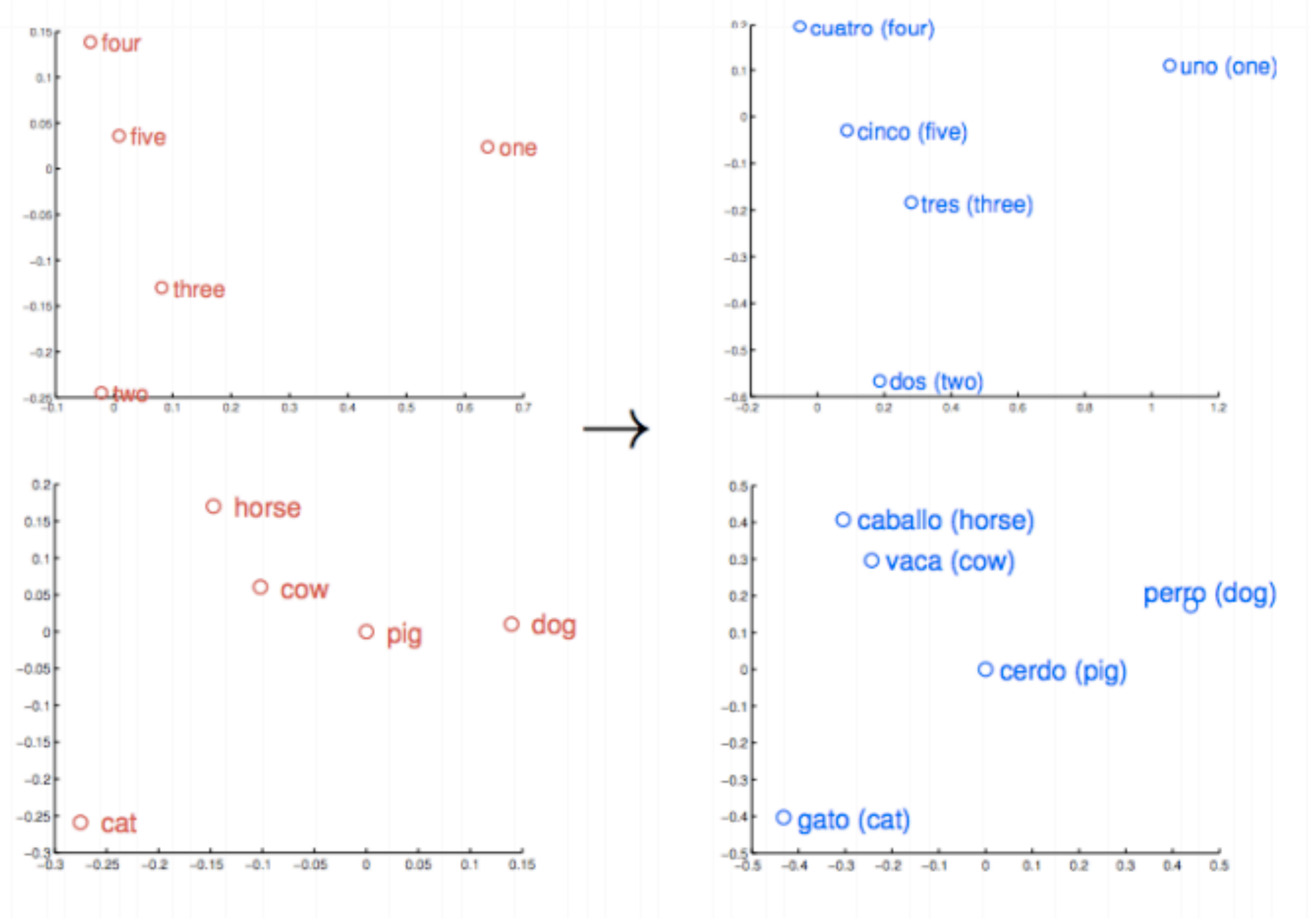

Country-Capital

$$v_{\text{man}} - v_{\text{woman}} \approx v_{\text{king}} - v_{\text{queen}}$$

# Word embeddings

- They have some other nice properties too!

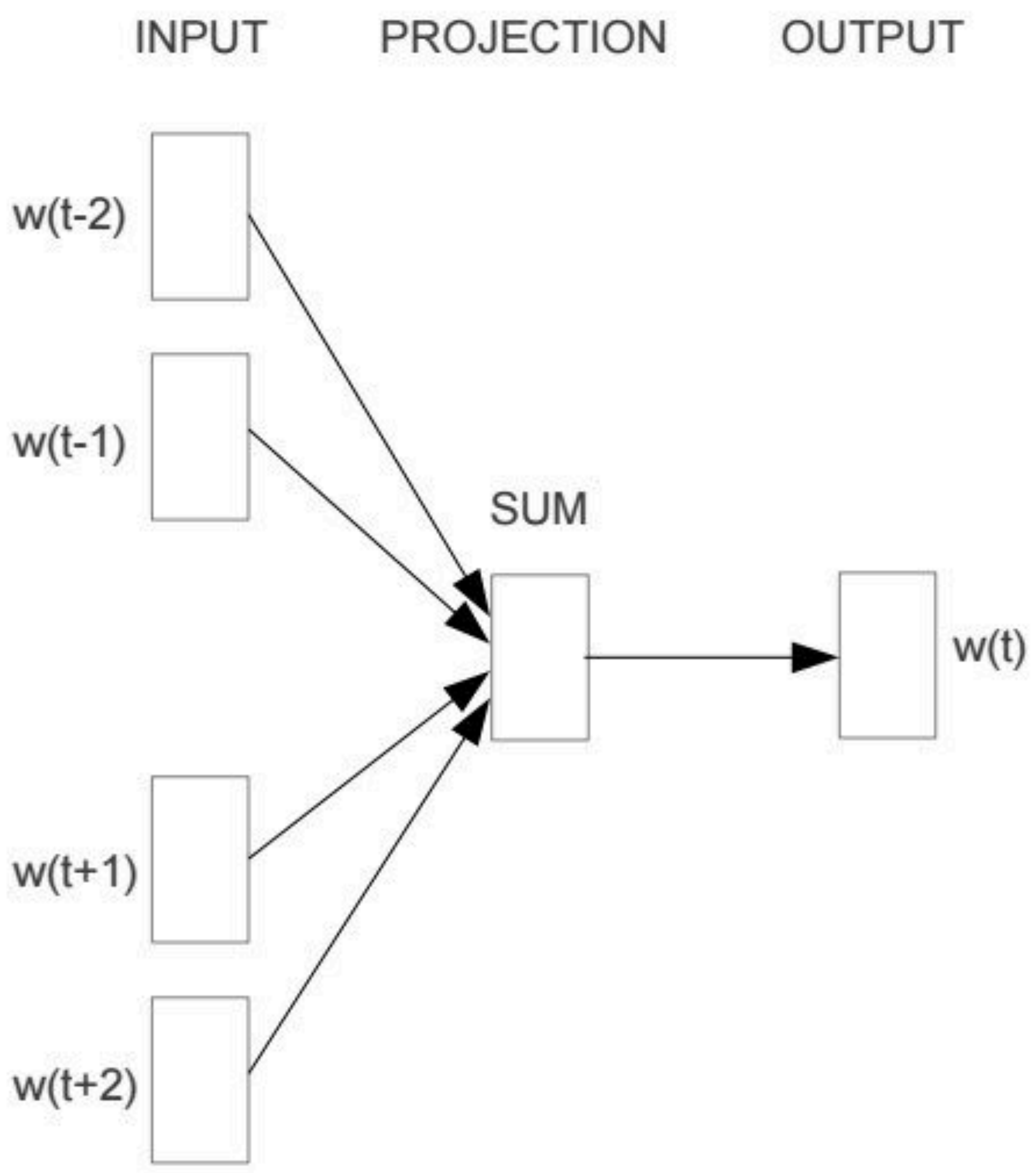
$$v(\text{cuatro}) \approx Wv(\text{four})$$



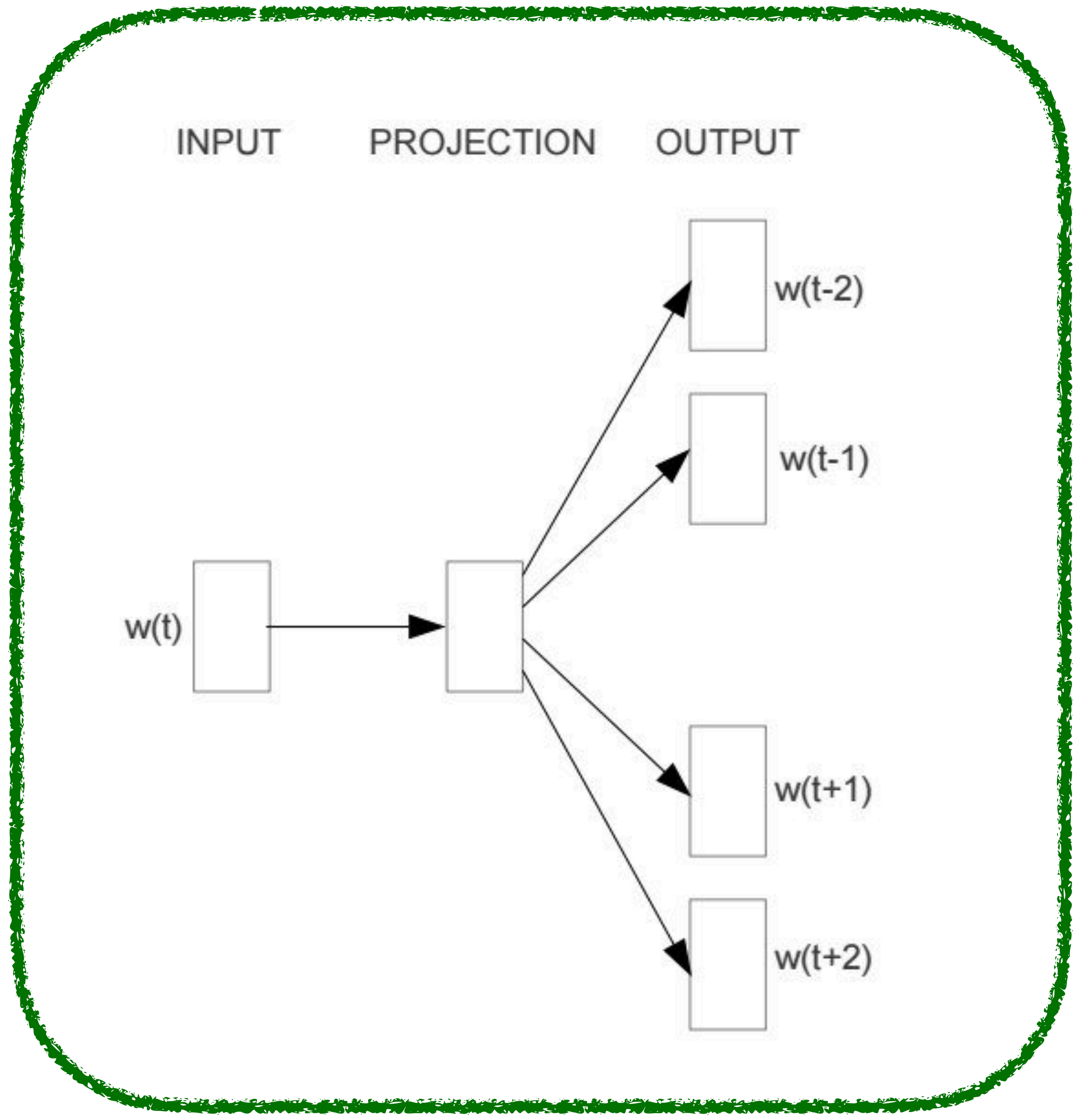

(Mikolov et al, 2013): Exploiting Similarities among Languages for Machine Translation

# word2vec

- (Mikolov et al 2013a): Efficient Estimation of Word Representations in Vector Space
- (Mikolov et al 2013b): Distributed Representations of Words and Phrases and their Compositionality




Continuous Bag of Words (CBOW)

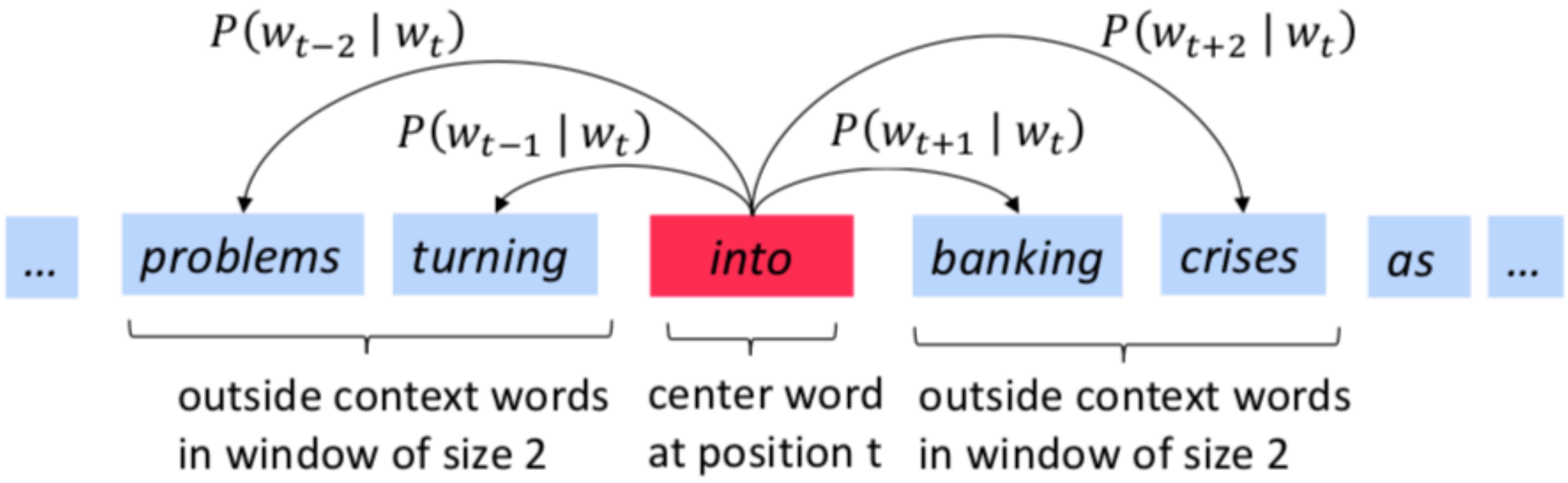



Skip-gram

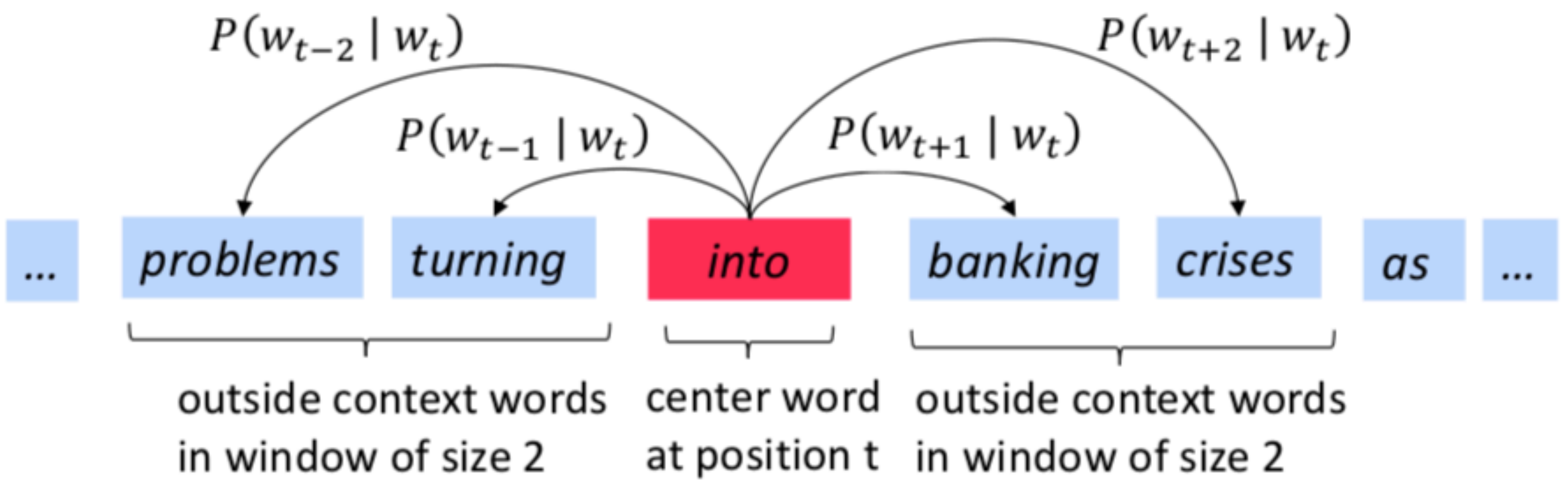
# Skip-gram

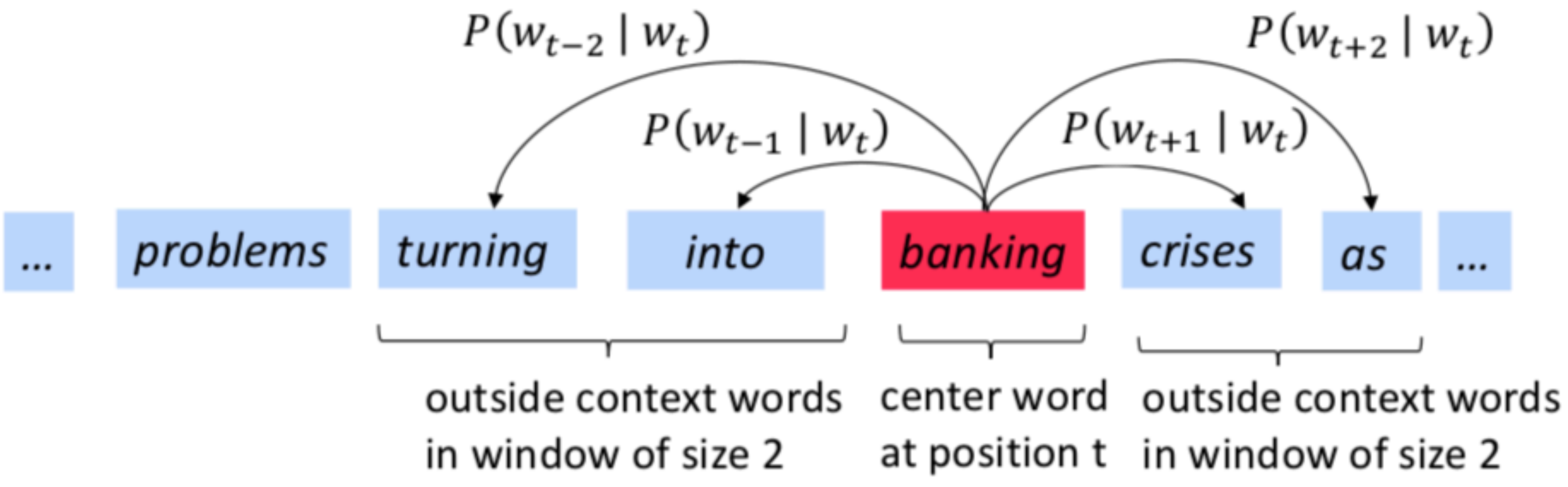
A classification problem!

- **Key idea:** Use each word to **predict** other words in its context
- Assume that we have a large corpus $w_1, w_2, \ldots, w_T \in V$
- Context: a fixed window of size $2m$ (m = 2 in the example)



# Skip-gram




Our goal is to find parameters that can maximize

$P(\text{problems} \mid \text{into}) \times P(\text{turning} \mid \text{into}) \times P(\text{banking} \mid \text{into}) \times P(\text{crises} \mid \text{into}) \times P(\text{turning} \mid \text{banking}) \times P(\text{into} \mid \text{banking}) \times P(\text{crises} \mid \text{banking}) \times P(\text{as} \mid \text{banking})\ldots$

# Skip-gram: objective function

- For each position $t = 1,2,\ldots T$, predict context words within context size m, given center word $w_t$:

all the parameters to be optimized

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

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

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

# How to define $P(w_{t+j} \mid w_t; \theta)$?

- Use two sets of vectors for each word in the vocabulary

$\mathbf{u}_i \in \mathbb{R}^d$ : embedding for center word $i$, $\forall i \in V$

$\mathbf{v}_{i'} \in \mathbb{R}^d$ : embedding for context word $i'$, $\forall i' \in V$

- Use inner product $\mathbf{u}_i \cdot \mathbf{v}_{i'}$ to measure how likely word $i$ appears with context word $i'$

Softmax we learned in multinomial logistic regression!

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

# … vs multinominal logistic regression

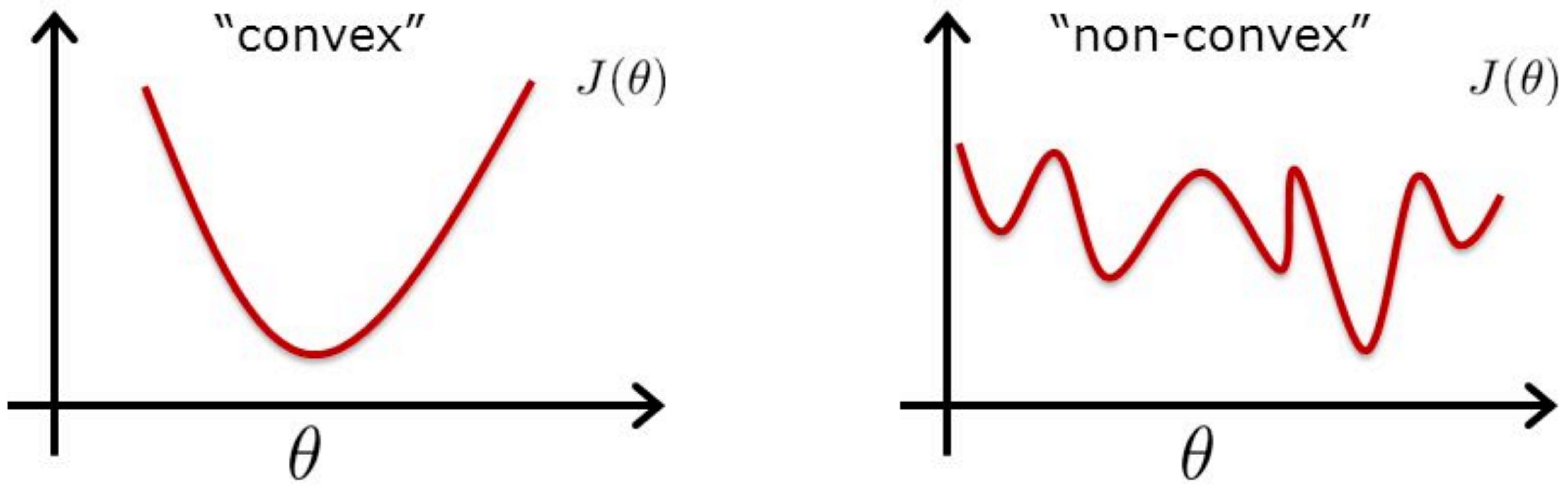
For multinomial LR,

$$P(y = c \mid x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

- Essentially a |V|-way classification problem
- If we fix $\mathbf{u}_{w_t}$, it is reduced to a multinomial logistic regression problem.
- However, since we have to learn both $\mathbf{u}$ and $\mathbf{v}$ together, the training objective is non-convex.

# … vs multinominal logistic regression




- It is hard to find a global minimum.
- But can still use stochastic gradient descent to optimize $\theta$:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta J(\theta)$$

# Poll

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

How many parameters does this model have (i.e. what is size of $\theta$)?

(a) $d|V|$

(b) $2d|V|$

(c) $2m|V|$

(d) $2md|V|$

[d = dimension of each vector]

# word2vec formulation

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

Q: Why do we need two vectors for each word?

Q: Which set of vectors are used as word embeddings?

# word2vec formulation

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

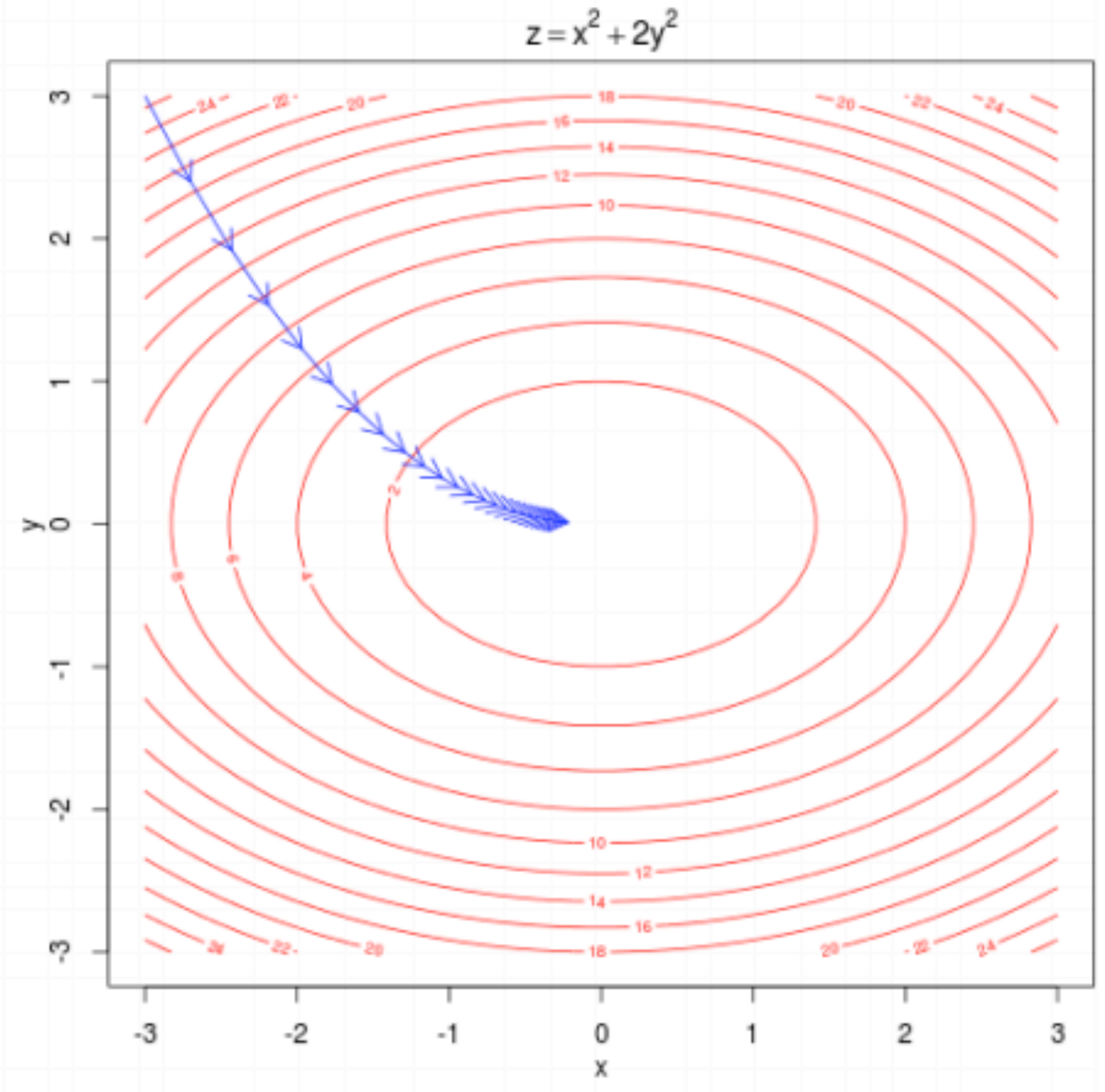
- In this formulation, we don't care about the classification task itself like we do for the logistic regression model we saw previously.
- The key point is that the *parameters* used to optimize this training objective—when the training corpus is large enough—can give us very good representations of words (following the principle of distributional hypothesis)!

# How to train this model?

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

- To train such a model, we need to compute the vector gradient $\nabla_\theta J(\theta) = ?$
- Again, $\theta$ represents all $2d|V|$ model parameters, in one vector.

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix}$$



# Warmup: Vectorized gradients

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{a}$$

$$\mathbf{x}, \mathbf{a} \in \mathbb{R}^n$$

$$\frac{\partial f}{\partial \mathbf{x}} = \quad \mathbf{a}$$

$$f = x_1 a_1 + x_2 a_2 + \ldots + x_n a_n$$

$$\frac{\partial f}{\partial \mathbf{x}} = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}]$$

# Vectorized gradients

Next, we are going to compute gradients with respect to many variables together and write them in vector/matrix notations.

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$$

$$\boldsymbol{f}(\boldsymbol{x}) = [f_1(x_1, ..., x_n), f_2(x_1, ..., x_n), ..., f_m(x_1, ..., x_n)]$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$f(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^n$$

$$\frac{\partial f}{\partial \mathbf{x}} = I_n$$

$$\frac{\partial f_i}{\partial x_j} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

# Poll

Let $f(\mathbf{x}) = \mathbf{W}\mathbf{x}$, $\mathbf{W} \in \mathbb{R}^{m \times n}, \mathbf{x} \in \mathbb{R}^n$, what is the value of $\dfrac{\partial f}{\partial \mathbf{x}}$ ?

(a) $\mathbf{W}$

(b) $\mathbf{W}^\intercal$

(c) $\mathbf{W}\mathbf{x}$

(d) $\mathbf{x}$

# Let's compute gradients for word2vec

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

Consider one pair of center/context words $(t, c)$: $y = -\log\left(\frac{\exp(\mathbf{u}_t\cdot\mathbf{v}_c)}{\sum_{k\in V}\exp(\mathbf{u}_t\cdot\mathbf{v}_k)}\right)$

We need to compute the gradient of $y$ with respect to $\mathbf{u}_t$ and $\mathbf{v}_k, \forall k \in V$

# Let's compute gradients for word2vec

$$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

$$y = -\log(\exp(\mathbf{u}_t \cdot \mathbf{v}_c)) + \log(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))$$

$$= -\mathbf{u}_t \cdot \mathbf{v}_c + \log(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))$$

Recall that

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

$$\frac{\partial y}{\partial \mathbf{u}_t} = \frac{\partial(-\mathbf{u}_t \cdot \mathbf{v}_c)}{\partial \mathbf{u}_t} + \frac{\partial(\log \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))}{\partial \mathbf{u}_t}$$

$$= -\mathbf{v}_c + \frac{\frac{\partial \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\partial \mathbf{u}_t}}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$

$$= -\mathbf{v}_c + \frac{\sum_{k \in V} \frac{\partial \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\partial \mathbf{u}_t}}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$

$$= -\mathbf{v}_c + \frac{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k) \cdot \mathbf{v}_k}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$

$$= -\mathbf{v}_c + \sum_{k \in V} \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\sum_{k' \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_{k'})} \mathbf{v}_k$$

$$= -\mathbf{v}_c + \sum_{k \in V} P(k \mid t) \mathbf{v}_k$$

# Gradients for word2vec

What about context vectors?

See assignment 1 :)

# Overall algorithm

- Input: text corpus, context size m, embedding size $d$, vocabulary $V$
- Initialize $\mathbf{u}_i$, $\mathbf{v}_i$ randomly
- Run through the training corpus and for each training instance $(t, c)$:
  - Update $\mathbf{u}_t \leftarrow \mathbf{u}_t - \eta \frac{\partial y}{\partial \mathbf{u_t}}$
  - Update $\mathbf{v}_k \leftarrow \mathbf{v}_k - \eta \frac{\partial y}{\partial \mathbf{v_k}}, \forall k \in V$

$$\frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k \mid v)\mathbf{v}_k$$

Q: Can you think of any issues with this algorithm?

# Skip-gram with negative sampling (SGNS)

Problem: every time you get one pair of $(t, c)$, you need to update $\mathbf{v}_k$ with all the words in the vocabulary! This is very expensive computationally.

$$\frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k \mid t)\mathbf{v}_k \qquad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k \mid t) - 1)\,\mathbf{u}_t & k = c \\ P(k \mid t)\mathbf{u}_t & k \neq c \end{cases}$$

**Negative sampling**: instead of considering all the words in V, let's randomly sample $K$ (5-20) negative examples.

softmax: $$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

NS: $$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

# Skip-gram with negative sampling (SGNS)

Key idea: Convert the $|V|$-way classification into a set of binary classification tasks.

Every time we get a pair of words (*t*, *c*), we don't predict *c* among all the words in the vocabulary. Instead, we predict (*t*, *c*) is a positive pair, and (*t*, *c'*) is a negative pair for a small number of sampled *c'*.

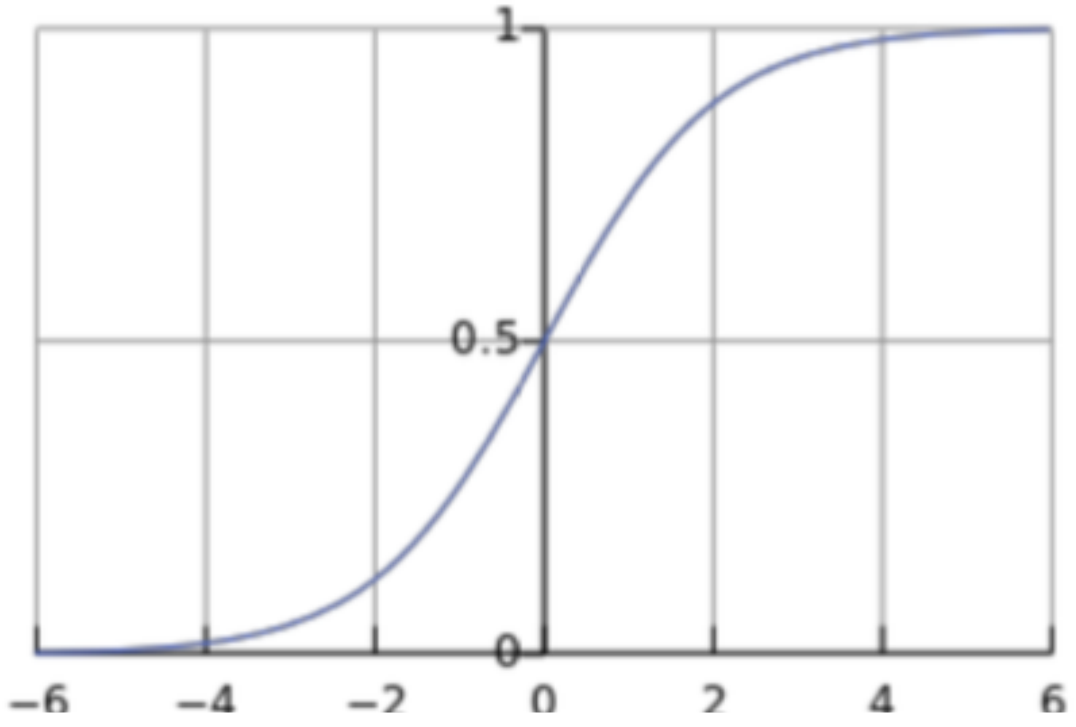
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

**positive examples +**

| t | c |
|---|---|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

**negative examples -**

| t | c | t | c |
|---|---|---|---|
| apricot | aardvark | apricot | seven |
| apricot | my | apricot | forever |
| apricot | where | apricot | dear |
| apricot | coaxial | apricot | if |




Similar to **binary logistic regression**, but we need to optimize u and v together.

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

$$P(y = 1 \mid t, c) = \sigma(\mathbf{u}_t \cdot \mathbf{v}_c) \qquad P(y = 0|t, c') = \sigma(\mathbf{u}_t \cdot \mathbf{v}_{c'})$$

# Poll

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

In skip-gram with negative sampling (SGNS), how many parameters need to be updated in $\theta$ for every $(t, c)$ pair?

(a) $Kd$

(b) $2Kd$

(c) $(K+1)d$

(d) $(K+2)d$

# Skip-gram with negative sampling (SGNS)

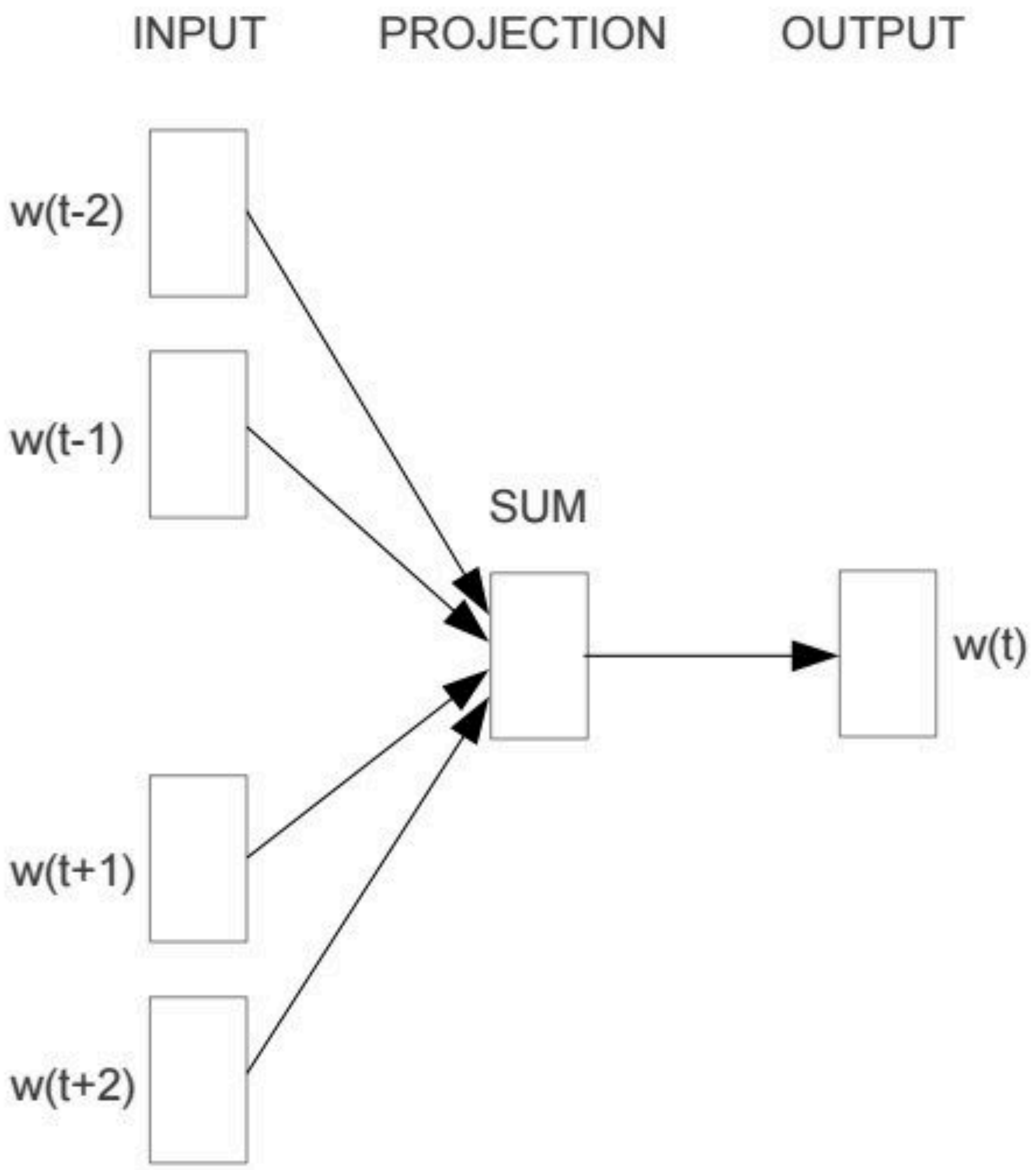
$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

- The gradients can be computed in a similar way but much cheaper!
- P(w): sampling according to the frequency of words

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

In practice, $\alpha \approx 0.75$ gives the best performance because it gives rare words slightly higher probability

# Continuous Bag of Words (CBOW)




$$L(\theta) = \prod_{t=1}^{T} P\left(w_t \mid \{w_{t+j}\}, -m \leq j \leq m, j \neq 0\right)$$

$$\bar{\mathbf{v}}_t = \frac{1}{2m} \sum_{-m \leq j \leq m, j \neq 0} \mathbf{v}_{t+j}$$

$$P(w_t \mid \{w_{t+j}\}) = \frac{\exp(\mathbf{u}_{w_t} \cdot \bar{\mathbf{v}}_t)}{\sum_{k \in V} \exp(\mathbf{u}_k \cdot \bar{\mathbf{v}}_t)}$$

# Poll

Let’s compare skip-gram with CBOW. Which of the following is correct?

(a) Skip-gram is a simpler task compared to CBOW

(b) Skip-gram is faster to train than CBOW

(c) Skip-gram handles frequent words better

(d) Skip-gram handles infrequent words better

# Skip-gram vs CBOW

- CBOW is comparatively faster to train than skip-gram and better for frequently occurring words

- Skip-gram is slower but works well for smaller amount of data and works well for less frequently occurring words

- CBOW is an easier classification problem than Skip-gram because in CBOW we just need to predict the one center word given many context words.

# GloVe: **Glo**bal **Ve**ctors

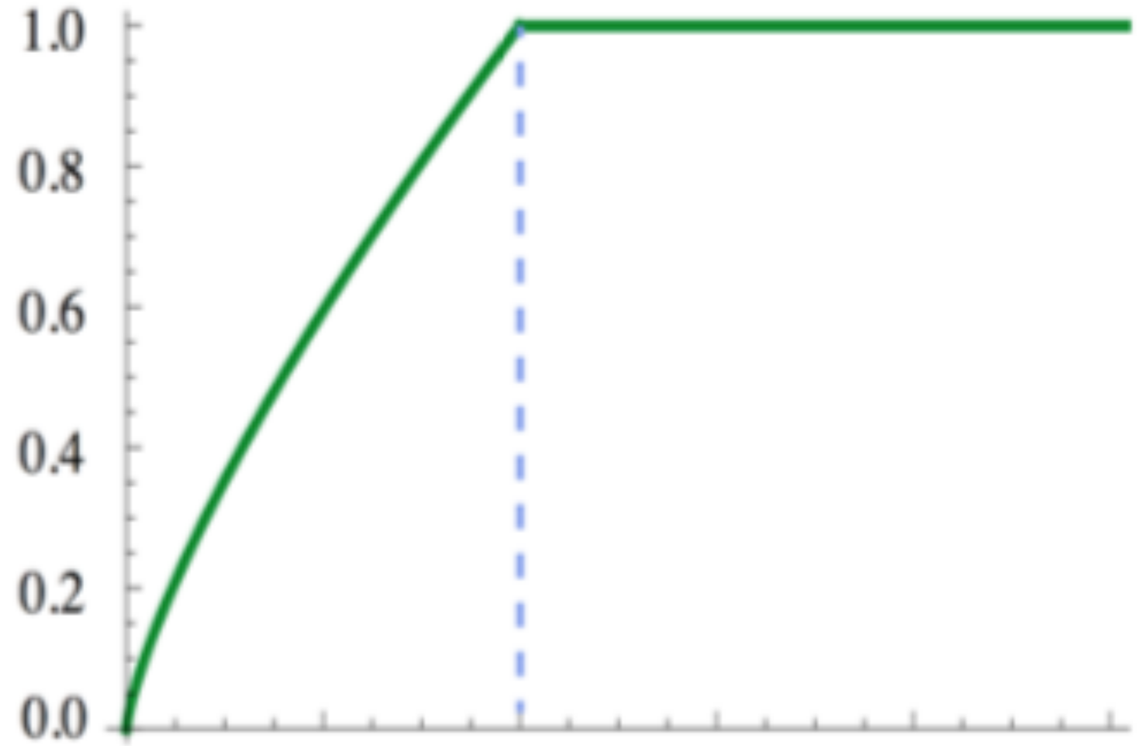
- Key idea: let's approximate $\mathbf{u}_i \cdot \mathbf{v}_j$ using their co-occurrence counts directly.
- Take the global co-occurrence statistics: $X_{i,j}$

$$J(\theta) = \sum_{i,j \in V} f(X_{i,j}) \left( \mathbf{u}_i \cdot \mathbf{v}_j + b_i + \tilde{b}_j - \log X_{i,j} \right)^2$$

- Training faster
- Scalable to very large corpora

Q: Why?

$f \sim$




(Pennington et al, 2014): GloVe: Global Vectors for Word Representation

# FastText: Sub-Word Embeddings

- Similar to Skip-gram, but break words into n-grams with n = 3 to 6

where: 3-grams: <wh, whe, her, ere, re>

4-grams: <whe, wher, here, ere>

5-grams: <wher, where, here>

6-grams: <where, where>

All the embeddings that we have learned are also called "**static word embeddings**": there is one fixed vector for every word in the vocabulary.

- Replace $\mathbf{u}_i \cdot \mathbf{v}_j$ by $\sum_{g \in n\text{-grams}(w_i)} \mathbf{u}_g \cdot \mathbf{v}_j$

- More to come! Contextualized word embeddings

(Bojanowski et al, 2017): Enriching Word Vectors with Subword Information

# Trained word embeddings available

- word2vec: https://code.google.com/archive/p/word2vec/
- GloVe: https://nlp.stanford.edu/projects/glove/
- FastText: https://fasttext.cc/

**Download pre-trained word vectors**

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip
- Ruby script for preprocessing Twitter data

Differ in algorithms, text corpora, dimensions, cased/uncased...

Applied to many other languages

# Easy to use!

```python
from gensim.models import KeyedVectors
# Load vectors directly from the file
model = KeyedVectors.load_word2vec_format('data/GoogleGoogleNews-vectors-negative300.bin', binary=True)
# Access vectors for specific words with a keyed lookup:
vector = model['easy']
```

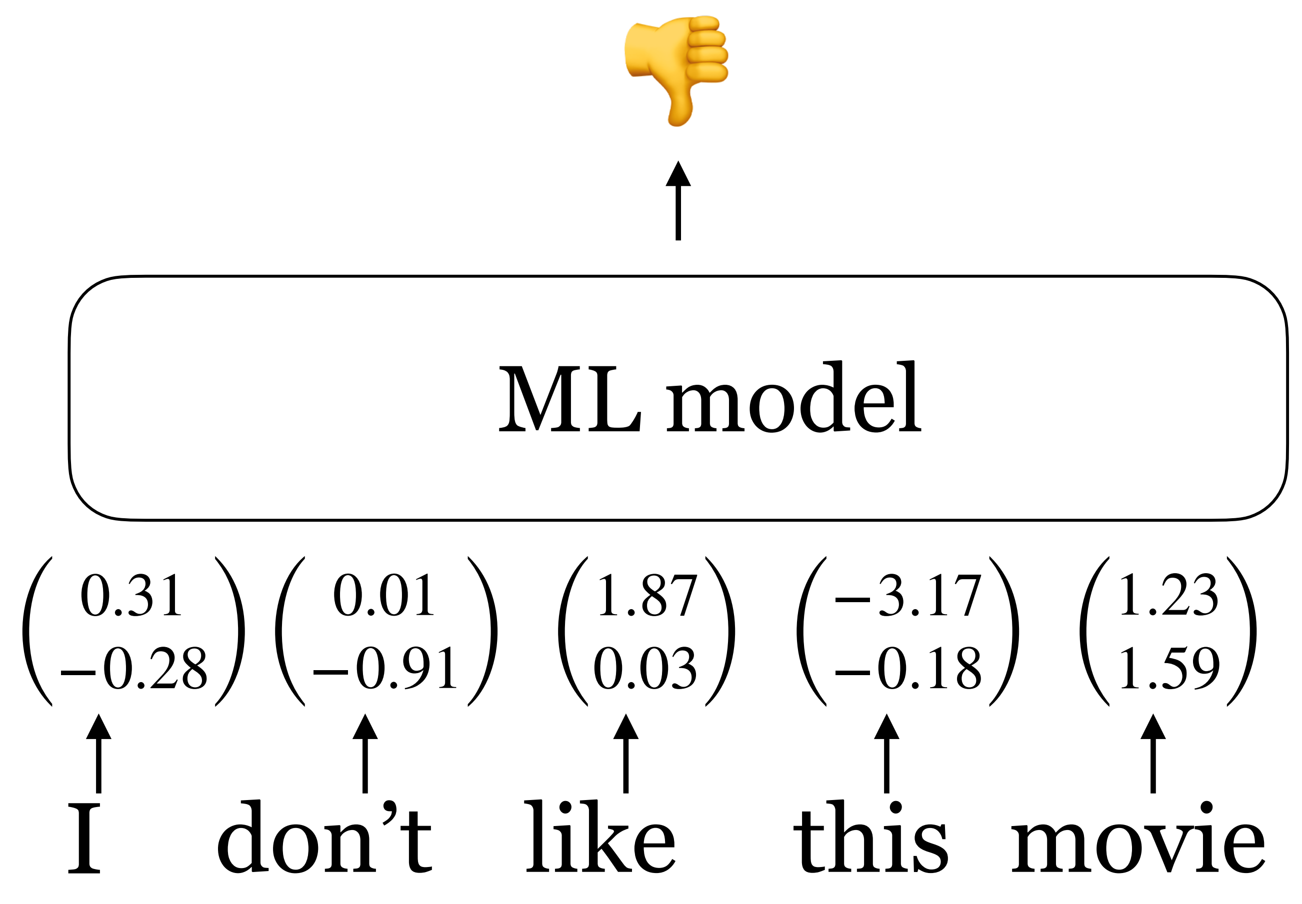
```
In [17]: model.similarity('straightforward', 'easy')
Out[17]: 0.5717043285477517

In [18]: model.similarity('simple', 'impossible')
Out[18]: 0.29156160264633707

In [19]: model.most_similar('simple')
Out[19]: [('straightforward', 0.7460169196128845),
          ('Simple', 0.7108174562454224),
          ('uncomplicated', 0.6297484636306763),
          ('simplest', 0.6171397566795349),
          ('easy', 0.5990299582481384),
          ('fairly_straightforward', 0.5893306732177734),
          ('deceptively_simple', 0.5743066072463989),
          ('simpler', 0.5537199378013611),
          ('simplistic', 0.5516539216041565),
          ('disarmingly_simple', 0.5365327000617981)]
```

# Evaluating Word Embeddings

# Extrinsic vs intrinsic evaluation

Extrinsic evaluation

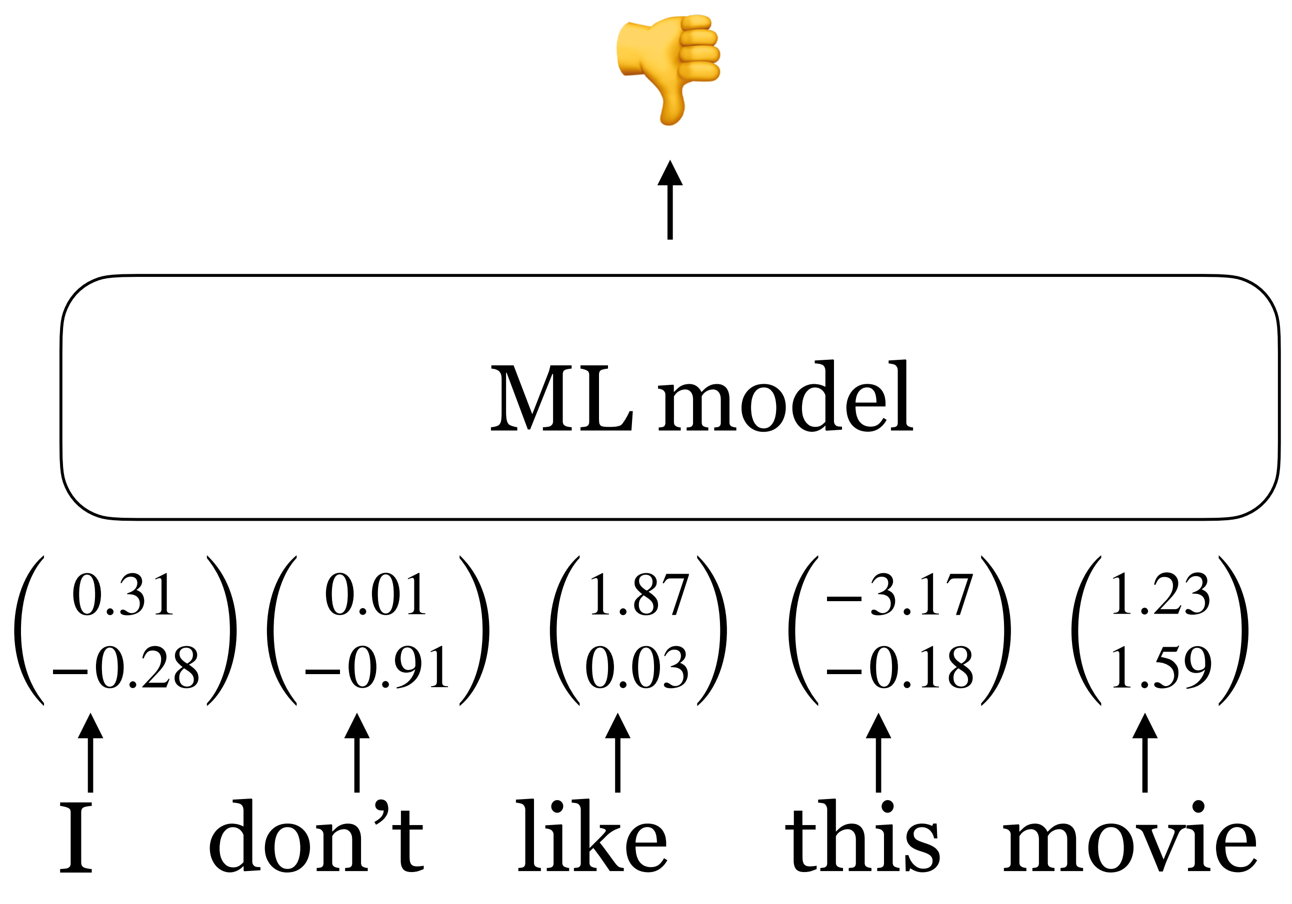
- Let's plug these word embeddings into a real NLP system and see whether this improves performance
- Could take a long time but still the most important evaluation metric




Intrinsic evaluation

- Evaluate on a specific/intermediate subtask
- Fast to compute
- Not clear if it really helps downstream tasks

# Extrinsic evaluation




A straightforward solution: given an input sentence $x_1, x_2, \ldots, x_n$

Instead of using a bag-of-words model, we can compute $vec(x) = \mathbf{e}(x_1) + \mathbf{e}(x_2) + \ldots + \mathbf{e}(x_n)$

And then train a logistic regression classifier on $vec(x)$ as we did before!

# Intrinsic evaluation: word similarity

**Word similarity**

Example dataset: wordsim-353
353 pairs of words with human judgement
http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

| Word 1 | Word 2 | Human (mean) |
|---|---|---|
| tiger | cat | 7.35 |
| tiger | tiger | 10 |
| book | paper | 7.46 |
| computer | internet | 7.58 |
| plane | car | 5.77 |
| professor | doctor | 6.62 |
| stock | phone | 1.62 |
| stock | CD | 1.31 |
| stock | jaguar | 0.92 |

Cosine similarity:

$$\cos(\boldsymbol{u}_i, \boldsymbol{u}_j) = \frac{\boldsymbol{u}_i \cdot \boldsymbol{u}_j}{||\boldsymbol{u}_i||_2 \times ||\boldsymbol{u}_j||_2}.$$

Metric: Spearman rank correlation

# Intrinsic evaluation: word similarity

| Model | Size | WS353 | MC | RG | SCWS | RW |
|---|---|---|---|---|---|---|
| SVD | 6B | 35.3 | 35.1 | 42.5 | 38.3 | 25.6 |
| SVD-S | 6B | 56.5 | 71.5 | 71.0 | 53.6 | 34.7 |
| SVD-L | 6B | 65.7 | 72.7 | 75.1 | 56.5 | 37.0 |
| CBOW$^{\dagger}$ | 6B | 57.2 | 65.6 | 68.2 | 57.0 | 32.5 |
| SG$^{\dagger}$ | 6B | 62.8 | 65.2 | 69.7 | 58.1 | 37.2 |
| GloVe | 6B | 65.8 | 72.7 | 77.8 | 53.9 | 38.1 |
| SVD-L | 42B | 74.0 | 76.4 | 74.1 | 58.3 | 39.9 |
| GloVe | 42B | **75.9** | **83.6** | **82.9** | **59.6** | **47.8** |
| CBOW$^{*}$ | 100B | 68.4 | 79.6 | 75.4 | 59.4 | 45.5 |

SG: Skip-gram

# Intrinsic evaluation: word analogy

**Word analogy**

man: woman ≈ king: ?

$$\arg\max_i \left(\cos(\mathbf{u}_i, \mathbf{u}_b - \mathbf{u}_a + \mathbf{u}_c)\right)$$

| semantic | syntactic |
|---|---|
| Chicago:Illinois≈Philadelphia: ? | bad:worst ≈ cool: ? |

More examples at

http://download.tensorflow.org/data/questions-words.txt

# Intrinsic evaluation: word analogy

| Model | Dim. | Size | Sem. | Syn. | Tot. |
|---|---|---|---|---|---|
| ivLBL | 100 | 1.5B | 55.9 | 50.1 | 53.2 |
| HPCA | 100 | 1.6B | 4.2 | 16.4 | 10.8 |
| GloVe | 100 | 1.6B | 67.5 | 54.3 | 60.3 |
| SG | 300 | 1B | 61 | 61 | 61 |
| CBOW | 300 | 1.6B | 16.1 | 52.6 | 36.1 |
| vLBL | 300 | 1.5B | 54.2 | 64.8 | 60.0 |
| ivLBL | 300 | 1.5B | 65.2 | 63.0 | 64.0 |
| GloVe | 300 | 1.6B | 80.8 | 61.5 | 70.3 |
| SVD | 300 | 6B | 6.3 | 8.1 | 7.3 |
| SVD-S | 300 | 6B | 36.7 | 46.6 | 42.1 |
| SVD-L | 300 | 6B | 56.6 | 63.0 | 60.1 |
| CBOW† | 300 | 6B | 63.6 | 67.4 | 65.7 |
| SG† | 300 | 6B | 73.0 | 66.0 | 69.1 |
| GloVe | 300 | 6B | 77.4 | 67.0 | 71.7 |
| CBOW | 1000 | 6B | 57.3 | 68.9 | 63.7 |
| SG | 1000 | 6B | 66.1 | 65.1 | 65.6 |
| SVD-L | 300 | 42B | 38.4 | 58.2 | 49.2 |
| GloVe | 300 | 42B | **81.9** | **69.3** | **75.0** |