COS 484/584

(Advanced) Natural Language Processing

L6: Word Embeddings (II)

(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 corpora, V, d
    - V: a pre-defined vocabulary
    - d: dimension of word vectors (e.g. 300)
    - Text corpora:
      - Wikipedia + Gigaword 5: 6B tokens
      - Twitter: 27B tokens
      - Common Crawl: 840B tokens
  - Output: $f : V \to \mathbb{R}^d$

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

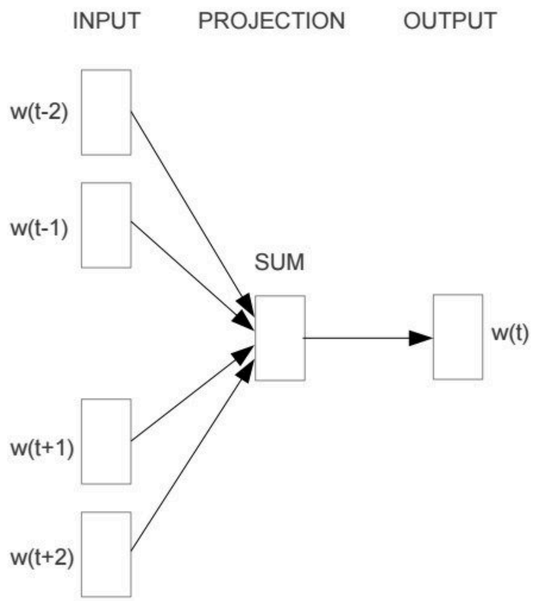
$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

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

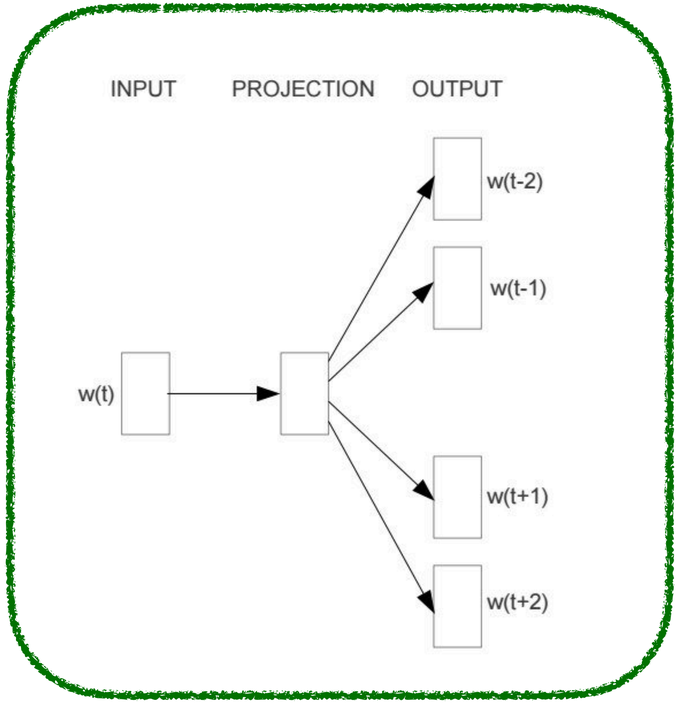
Each coordinate/dimension of the vector doesn't have a particular interpretation

word2vec

- (Mikolov et al 2013a): Efficient Estimation of Word Representations in Vector Space
  - Original word2vec formulation
- (Mikolov et al 2013b): Distributed Representations of Words and Phrases and their Compositionality
  - Negative sampling



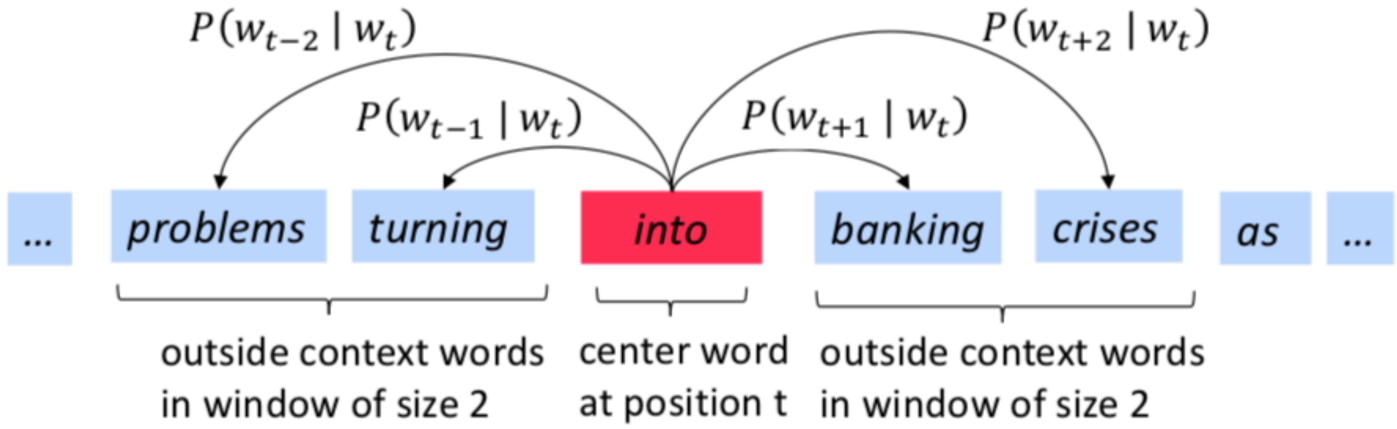
Continuous Bag of Words (CBOW)



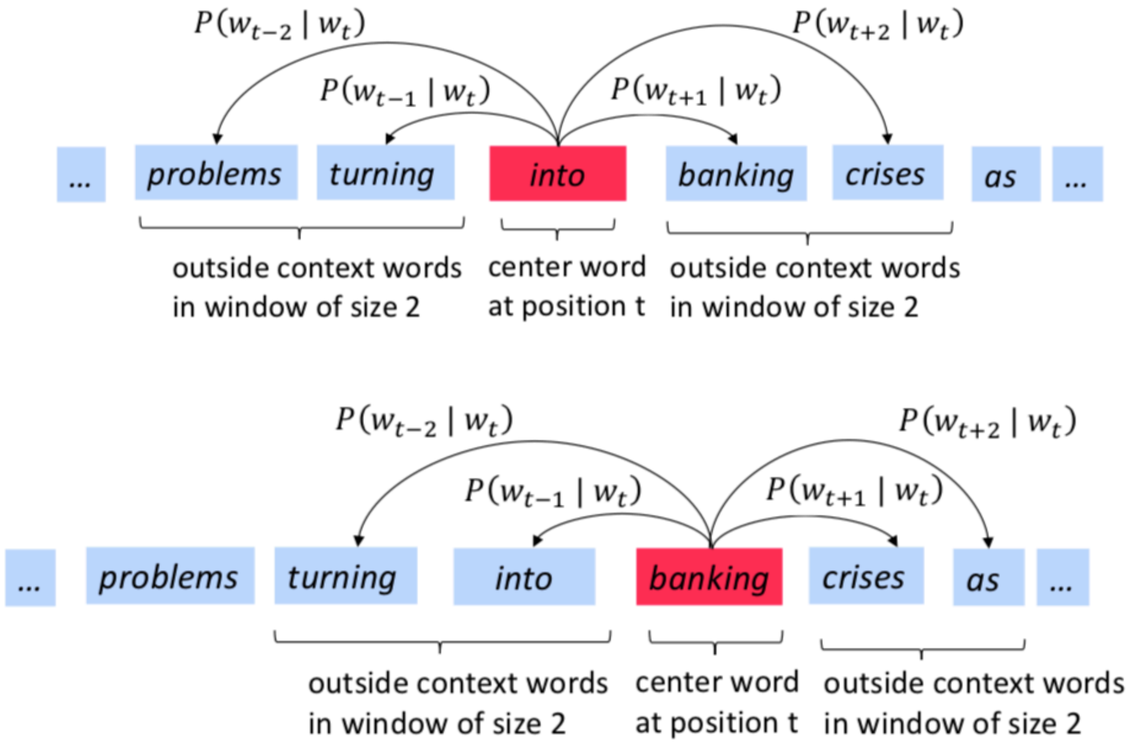
Skip-gram

Skip-gram

- The idea: we want to use words to **predict** their context words
- 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 $P(w_{t+j} \mid w_t; \theta)$?

- We have two sets of vectors for each word in the vocabulary

$$\mathbf{u}_k \in \mathbb{R}^d \text{ : embedding for center word } k, \forall k \in V$$

$$\mathbf{v}_k \in \mathbb{R}^d \text{ : embedding for context word } k, \forall k \in V$$

- Use inner product $\mathbf{u}_x \cdot \mathbf{v}_c$ to measure how likely word $x \in V$ appears with context word $c \in V$, the larger the better

Softmax we learned in multinomial logistic regression!

$$P(c \mid x) = \frac{\exp(\mathbf{u}_x \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_x \cdot \mathbf{v}_k)}$$

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

… vs multinominal logistic regression

Multinomial Logistic Regression

- What if we have more than 2 classes? (e.g. Part of speech tagging, named entity recognition)
- Need to model $P(y = c \mid x) \quad \forall c \in C$

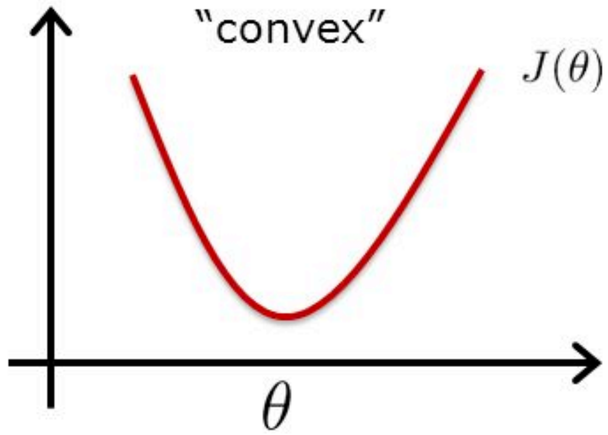
For multinomial LR,

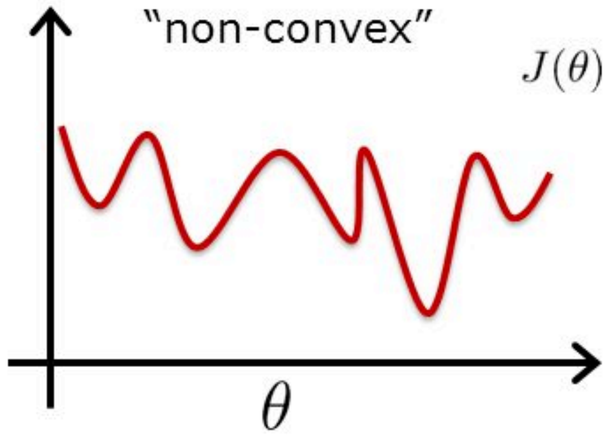
$$P(y = c \mid x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

$$P(c \mid x) = \frac{\exp(\mathbf{u}_x \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_x \cdot \mathbf{v}_k)}$$

- Think about it as a |V|-way classification problem
- If we fix $\mathbf{u}_x$, it is reduced to a multinominal 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.
- As we will see, we still use stochastic gradient descent to optimize θ:

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

Zoom 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 (size of θ)?

(a) $d|V|$

(b) $2d|V|$

(c) $2m|V|$

(d) $2md|V|$

Zoom 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 (size of θ)?

(a) $d|V|$

(b) $2d|V|$

(c) $2m|V|$

(d) $2md|V|$

The answer is (b). $\theta = \{\{\mathbf{u}_k\}, \{\mathbf{v}_k\}\}$ are all the parameters in this model!

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?

A: “dog” as a context word is not to be considered the same as the “dog” as a center word.

It is not very likely that one word appears in its own context.

If we use the same set of vectors, P(w | w) is high because of dot product.

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

In the word2vec paper, $\mathbf{u}_w$ is used (most common).

(Pennington et al., 2014) proposed using $\mathbf{u}_w + \mathbf{v}_w$.
(Levy et al., 2015) gave an interpretation for this.

word2vec formulation

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

- Another important thing to have in mind—we don't care about the classification task itself as we do for the logistic regression model we learned.

- 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)!

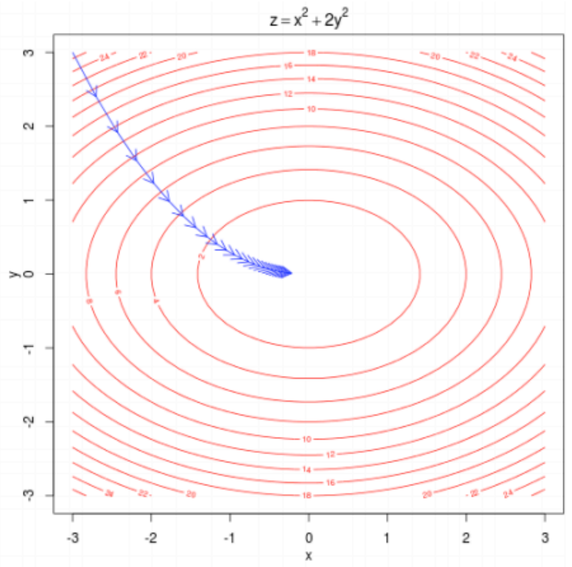
How to train this model?

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

- To train such a model, we need to compute the vector gradient $\nabla_\theta J(\theta) = ?$

- Again, θ represents all $2d|V|$ model parameters, in one vector.

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



Warm-up

$$f(x) = \exp(x) \qquad \frac{df}{dx} = \exp(x)$$

$$f(x) = \log(x) \qquad \frac{df}{dx} = \frac{1}{x}$$

$$f(x) = f_1(x) + f_2(x) \qquad \frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{d}f_1(x)}{\mathrm{d}x} + \frac{\mathrm{d}f_2(x)}{\mathrm{d}x}$$

chain rule:

$$f(x) = f_1(f_2(x)) \qquad \frac{df}{dx} = \frac{df_1(z)}{dz}\frac{df_2(x)}{dx} \qquad z = f_2(x)$$

Vectorized gradients

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

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

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

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

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

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

Vectorized gradients

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

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

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

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

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

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

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

Zoom 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) $\mathbf{x}$

Zoom 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 $\frac{\partial f}{\partial \mathbf{x}}$?

(a) $\mathbf{W}$
(b) $\mathbf{W}^\intercal$
(c) $\mathbf{W}\mathbf{x}$
(d) $\mathbf{x}$

The answer is (a).

$$\frac{\partial f_i}{\partial x_j} = \frac{\partial \sum_{k=1}^{n} W_{i,k} x_k}{\partial x_j}$$
$$= \frac{\partial W_{i,j} x_j}{\partial x_j} = W_{i,j}$$
$$\implies \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \mathbf{W}$$

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

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

Let's compute gradients for word2vec

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

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

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

Recall that

$$P(c \mid x) = \frac{\exp(\mathbf{u}_x \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_x \cdot \mathbf{v}_k)}$$

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

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

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

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

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

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

Let's compute gradients for word2vec

How about context vectors?

$$\frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k \mid x) - 1)\,\mathbf{u}_x & k = c \\ P(k \mid x)\mathbf{u}_x & k \neq c \end{cases}$$

See the assignment :)

Putting it together

- Input: text corpus, context size m, embedding size d, vocabulary V
- Initialize $\mathbf{u}_k, \mathbf{v}_k$ randomly, $\forall k \in V$
- Walk through the training corpus and collect training data (x, c):
  - Update $\mathbf{u}_x \leftarrow \mathbf{u}_x - \eta \frac{\partial y}{\partial \mathbf{u}_x}$
  - Update $\mathbf{v}_k \leftarrow \mathbf{v}_k - \eta \frac{\partial y}{\partial \mathbf{v}_k}, \forall k \in V$

Q: Can you think of any issue in this algorithm?

Skip-gram with negative sampling (SGNS)

Problem: every time you get one pair of (*x*, *c*), you need to iterate through all the words in the vocabulary *V*! It is very computationally expensive.

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

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

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

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

Skip-gram with negative sampling (SGNS)

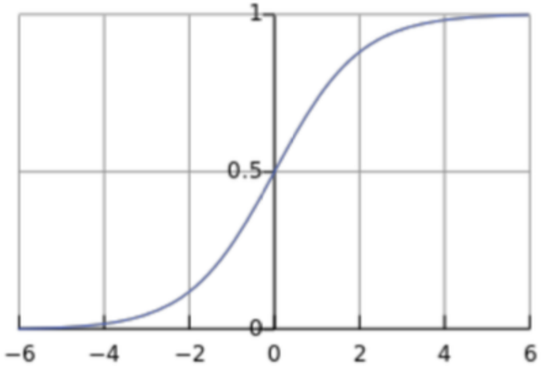
Key idea: we convert the $|V|$-way classification into a binary classification task.

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

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

positive examples +	
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -			
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if



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

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

$$P(y = 1 \mid x, c) = \sigma(\mathbf{u}_x \cdot \mathbf{v}_c) \qquad P(y = 0 \mid x, c') = 1 - \sigma(\mathbf{u}_x \cdot \mathbf{v}_{c'}) = \sigma(-\mathbf{u}_x \cdot \mathbf{v}_{c'})$$

Zoom poll

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

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

(a) Kd

(b) $2Kd$

(c) $(K+1)d$

(d) $(K+2)d$

Zoom poll

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

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

(a) Kd

(b) $2Kd$

(c) $(K+1)d$

(d) $(K+2)d$

The answer is (d). We have $(K+1)$ $\mathbf{v}$ vectors and one $\mathbf{u}$ vector to optimize every time.

Skip-gram with negative sampling (SGNS)

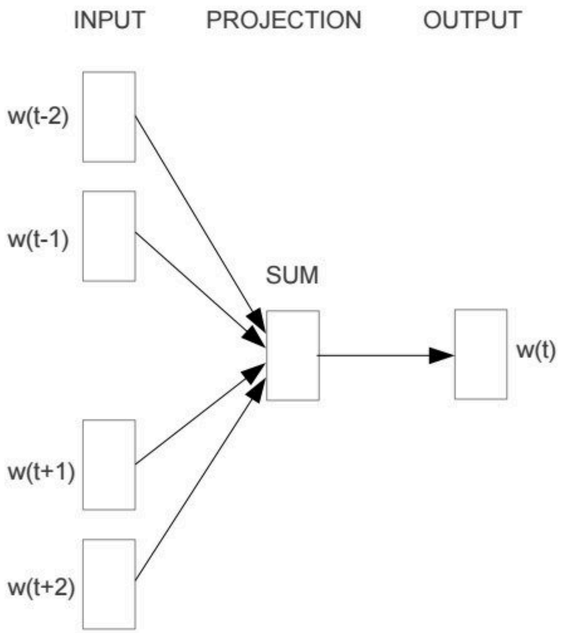
$$y = -\log\left(\sigma(\mathbf{u}_x \cdot \mathbf{v}_c)\right) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log\left(\sigma(-\mathbf{u}_x \cdot \mathbf{v}_j)\right)$$

- 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, they find that $\alpha = 0.75$ gives the best performance because it gives rare words slightly higher probability

Continuous Bag of Words (CBOW)



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

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

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

Zoom poll

Let's make a comparison between skip-gram and 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

Zoom poll

Let's make a comparison between skip-gram and 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

The answer is (d). See the next slide.

Here is my oversimplified and rather naive understanding of the difference:

As we know, **CBOW** is learning to predict the word by the context. Or maximize the probability of the target word by looking at the context. And this happens to be a problem for rare words. For example, given the context `yesterday was really [...] day` CBOW model will tell you that most probably the word is `beautiful` or `nice`. Words like `delightful` will get much less attention of the model, because it is designed to predict the most probable word. Rare words will be smoothed over a lot of examples with more frequent words.

On the other hand, the **skip-gram** is designed to predict the context. Given the word `delightful` it must understand it and tell us, that there is huge probability, the context is `yesterday was really [...] day`, or some other relevant context. With **skip-gram** the word `delightful` will not try to compete with word `beautiful` but instead, `delightful+context` pairs will be treated as new observations. Because of this, **skip-gram** will need more data so it will learn to understand even rare words.

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 a simpler 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

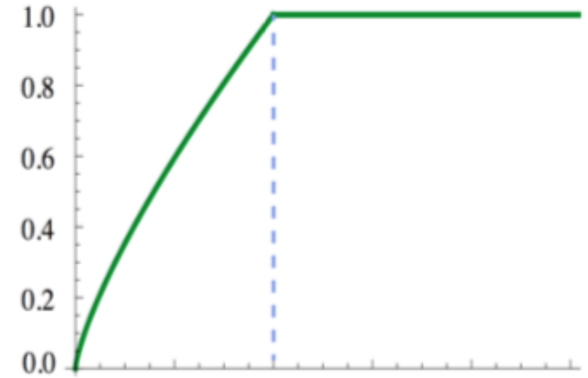
- Key idea: let's approximate $\mathbf{u}_i \cdot \mathbf{v}_j$ using their co-occurrence counts directly.
- Take the global co-occurrence statistics: $X_{i,j}$

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

- 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 as 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 $\mathbf{u}_x \cdot \mathbf{v}_c$ by $\sum_{g \in n\text{-grams}(x)} \mathbf{u}_g \cdot \mathbf{v}_c$

- 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']
```

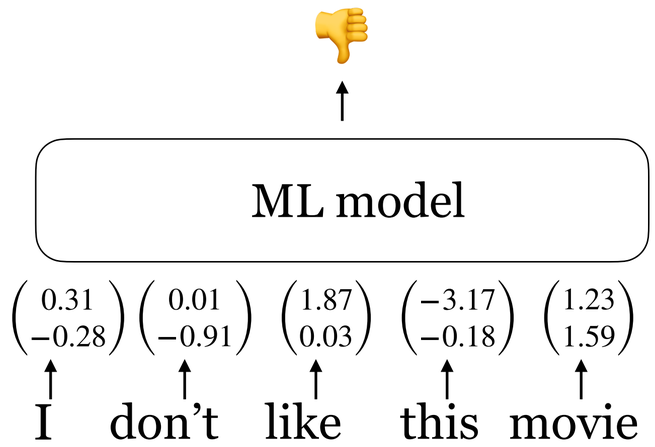
```
In [17]: model.similarity('straightforward', 'easy')
Out[17]: 0.5717043285477517

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

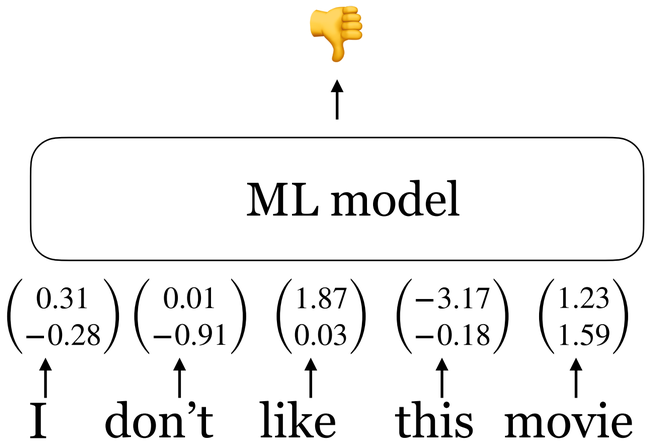
- Let's plug these word embeddings into a real NLP system and see whether this improves performance
- Could take a long time but still the most important evaluation metric



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†	6B	57.2	65.6	68.2	57.0	32.5
SG†	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

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**