Deep Learning for Natural Language Processing

Cornelia Caragea

Computer Science
University of Illinois at Chicago

Credits for slides: Manning, Socher

Word Vectors - word2vec

Today

Lecture Structure:

- Human language and word meaning
- Word2vec introduction
- Word2vec objective function gradients
- Optimization basics
- Word2vec efficiency
- Word2vec evaluation

Recommended reading:

- Efficient Estimation of Word Representations in Vector Space
- Distributed Representations of Words and Phrases and their Compositionality

Human language and word meaning

- Language is very complex!
- Word meaning may require context to understand!

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.

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

- Great as a resource but missing nuance
  - e.g. "estimable" 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
- Cannot compute accurate word similarity →

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel - a localist representation

$\boxed{\text{Means one 1, the rest 0s}}$

Words can be represented by one-hot vectors:

$$\begin{aligned} \text{motel} &= [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0] \\ \text{hotel} &= [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0] \end{aligned}$$

Vector dimension = number of words in vocabulary (e.g., 500,000)

Representing words as discrete symbols

Example: in web search, if user searches for "Seattle motel", we would like to match documents containing "Seattle hotel".

But:

$$\text{motel} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]$$
$$\text{hotel} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

These two vectors are orthogonal.
There is no natural notion of **similarity** for one-hot vectors!

Solution:

- Could 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**

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)
  - 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***

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

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

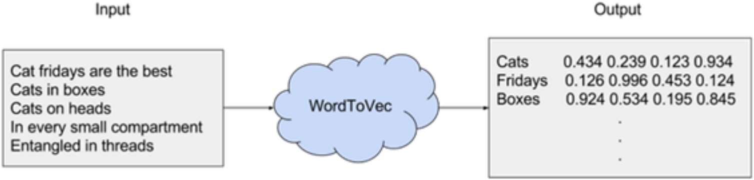
Note: word vectors are sometimes called word embeddings or word representations. They are a distributed representation.

Word meaning as a neural word vector - visualization

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

Word2vec: Overview

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

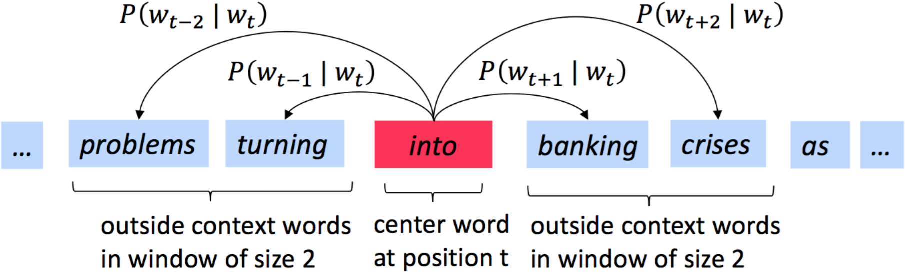


Idea:

- 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

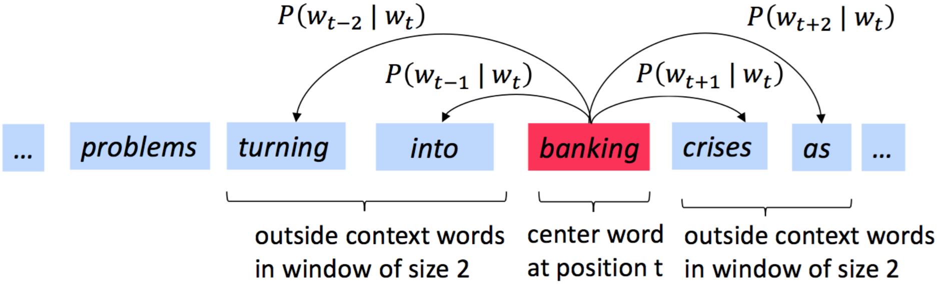
Word2vec: Overview

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



Word2vec: Overview

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



Word2vec: training objective

- Find word representations that are useful for predicting the surrounding words in a sentence or a document.
- Give high probability estimates for the words that occur in the context and low probability estimates for the words the do not occur in the context
  - E.g., if the center word is "bank", then "withdraw" gets high probability, whereas other words such as "neural" will get low probability.

Word2vec: objective function

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

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

θ is all variables to be optimized

sometimes called *cost* or *loss* function

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

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

Minimizing objective function $\Longleftrightarrow$ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

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

- Question: How to calculate $P(w_{t+j}|w_t;\theta)$?

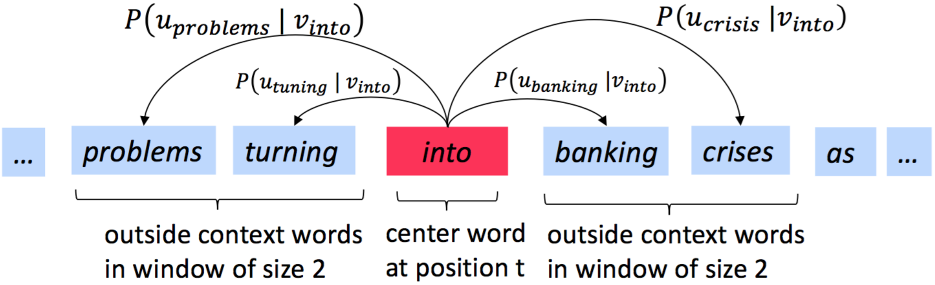
- Answer: We will use *two* vectors per word w:
  - v_w when w is a center word
  - u_w when w is a context word

- 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: overview with vectors

- Example windows and process for computing $P(w_{t+j}|w_t)$
- $P(u_{problems}|v_{into})$ short for $P(problems|into; u_{problems}, v_{into}, \theta)$



Word2vec: prediction function

Exponentiation makes anything positive

Dot product compares similarity of o and c.
$u^T v = < u, v > = u \cdot v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

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

Normalize over entire vocabulary to give probability distribution

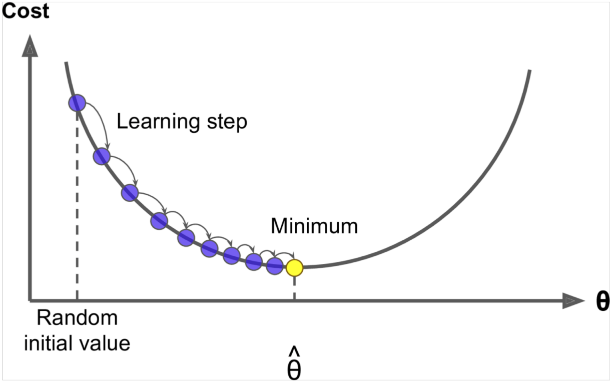
- This is an example of the **softmax function** $\mathbb{R}^n \to \mathbb{R}^n$

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

- 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

Optimization: Gradient Descent

- We have a cost function $J(\theta)$ we want to minimize
- To train a model, we adjust parameters to minimize a loss
- **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
```

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

Word2vec derivations of gradient

- The basic Lego piece
- Useful basics:

$$\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a} \tag{1}$$

- If in doubt: write out with indices
- Chain rule! If $y = f(u)$ and $u = g(x)$, i.e. $y = f(g(x))$, then:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} \tag{2}$$

Chain rule

- Chain rule! If $y = f(u)$ and $u = g(x)$, i.e. $y = f(g(x))$, then:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} = \frac{df(u)}{du}\frac{dg(x)}{dx} \tag{3}$$

- Simple example:

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

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

$$\frac{dy}{du} = 20u^3 \qquad \frac{du}{dx} = 3x^2 \tag{6}$$

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

Calculating all gradients!

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

Let's derive gradient for center word together
For **one example window** and **one example outside word**:

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

You then also need the gradient for context words (it's similar; left for homework). That's all of the parameters θ here.

$$\max J'(\theta) = \frac{1}{T} \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w'_{t+j} | w_t; \theta)$$

$$\min J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w'_{t+j} | w_t)$$

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

$$\max J'(\theta) = \frac{1}{T} \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w'_{t+j} | w_t; \theta)$$

$$\min J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w'_{t+j} | w_t)$$

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

Goal: Minimizing $J(\theta)$ by changing the parameters θ, which are the contents of the word vectors.

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

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

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

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

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

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

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

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

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

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

$$\frac{\partial}{\partial v_c} \log P(o|c) = u_o - \sum_{x \in V} P(x|c) \cdot u_x$$

That is, the difference between the observed (actual) context word and the expected context word according to our model.

$$\frac{\partial}{\partial v_c} \log P(o|c) = u_o - \sum_{x \in V} P(x|c) \cdot u_x$$

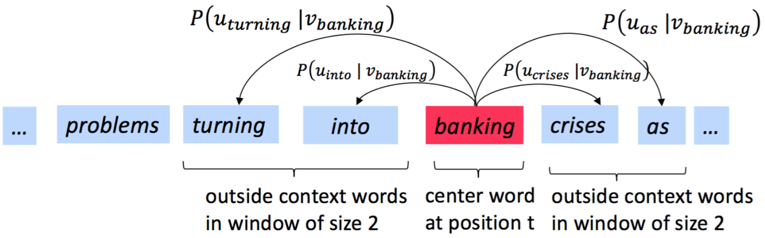
That is, the difference between the observed (actual) context word and the expected context word according to our model.

- This difference gives us the slope as to which direction we should be changing the representation in order to improve our model's ability to predict.

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!

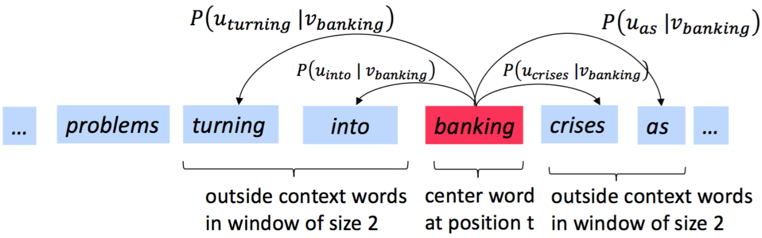
- Generally, in each window we will compute updates for all parameters that are being used in that window. For example:



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:



- Why two vectors? $\rightarrow$ Easier optimization. Average both vectors at the end.

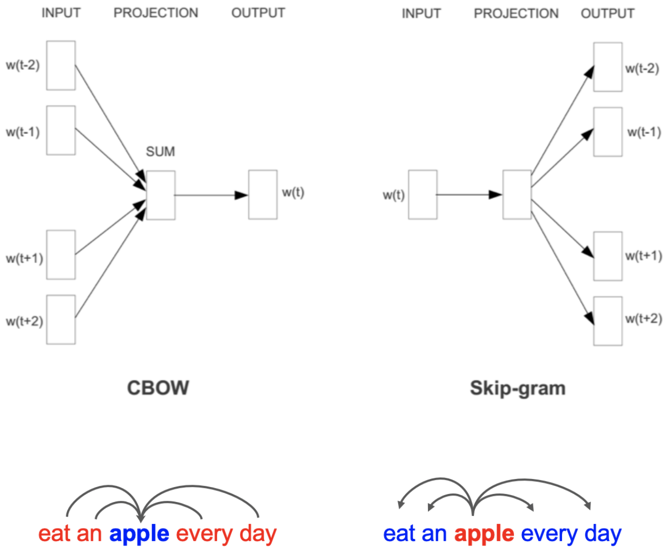
Word2vec: more details

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

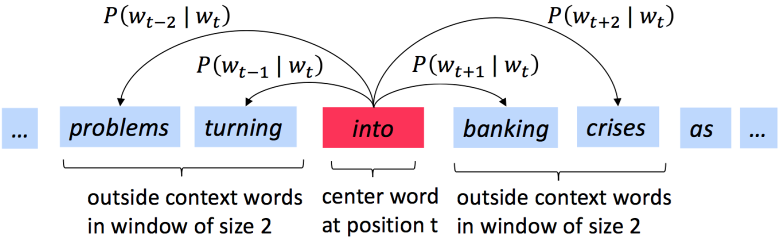
This lecture so far: **Skip-gram model**

Word2vec: neural network models



Calculating all gradients!

- Iterate through each word of the whole corpus
- Predict surrounding words using word vectors



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

- Update vectors so you can predict well

Stochastic Gradient Descent

- Problem with gradient descent: $J(\theta)$ 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 (or after a small batch)

Stochastic gradients with word vectors!

- Iteratively take gradients at each such window for SGD
- But in each window, we only have at most $2m+1$ words, so $\nabla_\theta J_t(\theta)$ is very sparse!

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

- We might only update the word vectors that actually appear in a window!

Skip-gram: additional efficiency

- **Skip-grams (SG)**
  Predict context ("outside") words (position independent) given center word
- So far: Focus on **softmax** (simpler, but expensive training method)
- The normalization factor is too computationally expensive.

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

Additional efficiency in training:

- Negative sampling

Skip-gram model with negative sampling

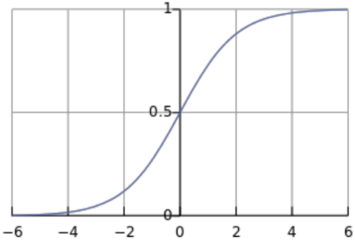
- In standard word2vec, the skip-gram model uses **negative sampling**
- Main idea: train binary logistic regressions for a true pair (center word and word in its context window) versus several noise pairs (the center word paired with a random word)

Skip-gram model with 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: $\sigma(x) = \frac{1}{1+e^{-x}}$
- So we maximize the probability of two words co-occurring in first log



Skip-gram model with negative sampling

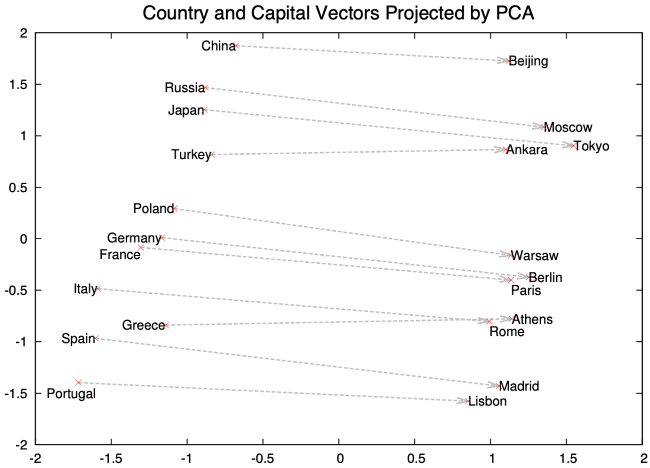
- Notation more similar to class:

$$J_{neg\_sample}(u_o, v_c, U) = -\log(\sigma(u_o^T v_c)) - \sum_{k=1}^{K} \log(\sigma(-u_k^T v_c))$$

- We take k negative samples (using word probabilities)
- Maximize probability that real outside word appears,
  minimize probability 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, Z is the normalizing factor.
- The power makes less frequent words be sampled more often (e.g., $0.9^{3/4} = 0.92$, $0.01^{3/4} = 0.032$).

Word Vectors Evaluation: Similarity based



Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities.

Task description for evaluation - Analogy

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Examples of five types of semantic and nine types of syntactic questions in the Semantic- Syntactic Word Relationship test set. Overall, there are 8869 semantic and 10675 syntactic questions.

Accuracy on the analogy task

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	**64.5**	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	**50.0**	55.9	**53.3**

Accuracy vs. training time on the analogy task

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Comparison of models trained for three epochs on the same data and models trained for one epoch. Accuracy is reported on the full Semantic-Syntactic data set.