Natural Language Processing
Lecture 03 Word Embeddings

Qun Liu, Valentin Malykh
Huawei Noah's Ark Lab

Autumn 2020
A course delivered at MIPT, Moscow

Content

Content

Word representations

- In rule-based approaches, i.e., grammars, automata, etc., words are represented as symbols.
- However, if we want to apply machine learning algorithms, we should represents linguistic units (words, phrases, etc.) as numerical vectors.
- Then the questions is, how can we represent words as numerical vectors?

Representing words by their context

"You shall know a word by the company it keeps"

(J. R. Firth, 1957)

- Distributional hypothesis:
  Linguistic items with similar distributions have similar meanings.

  $\Rightarrow$ **Distributional Semantics**

Distributional semantics

- *Distributional semantics* is a research area that develops and studies theories and methods for quantifying and categorizing semantic similarities between linguistic items based on their distributional properties in large samples of language data. (Wikipedia)
- Idea: Collect distributional information in high-dimensional vectors, and to define distributional/semantic similarity in terms of vector similarity.

when the door opened and	Doctor	Livesey came in on
visit to my father Oh	doctor	we cried what shall
s end ! said the	doctor	No more wounded than
back with the basin the	doctor	had already ripped up
great spirit Prophetic said the	doctor	touching this picture with
him First he recognized the	doctor	with an unmistakable frown

A sample of concordance

Distributional semantic models

- Distributional semantic models differ primarily with respect to the following parameters:
  - Context type (text regions vs. linguistic items)
  - Context window (size, extension, etc.)
  - Frequency weighting (e.g. entropy, pointwise mutual information, etc.)
  - Dimension reduction (e.g. random indexing, singular value decomposition, etc.)
  - Similarity measure (e.g. cosine similarity, Minkowski distance, etc.)

Example: Window based co-occurrence matrix

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Window based co-occurrence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Problems with simple co-occurrence vectors

Increase in size with vocabulary

Very high dimensional: requires a lot of storage

Subsequent classification models have sparsity issues

→ Models are less robust

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Solution: Low dimensional vectors

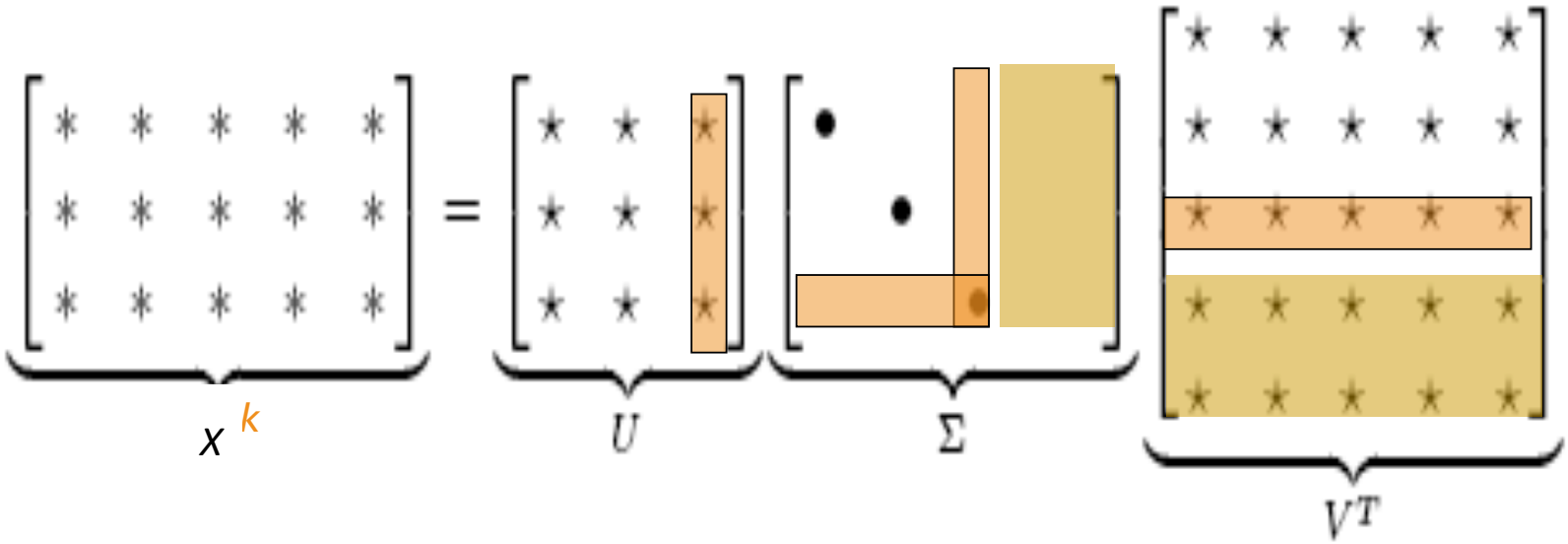
- Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
- Usually 25–1000 dimensions, similar to word2vec
- How to reduce the dimensionality?

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Method: Dimensionality Reduction on X (HW1)

Singular Value Decomposition of co-occurrence matrix X

Factorizes X into $U\Sigma V^T$, where U and V are orthonormal



Retain only k singular values, in order to generalize.

$\hat{X}$ is the best rank k approximation to X , in terms of least squares.

Classic linear algebra result. Expensive to compute for large matrices.

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Simple SVD word vectors in Python

Corpus:
I like deep learning. I like NLP. I enjoy flying.

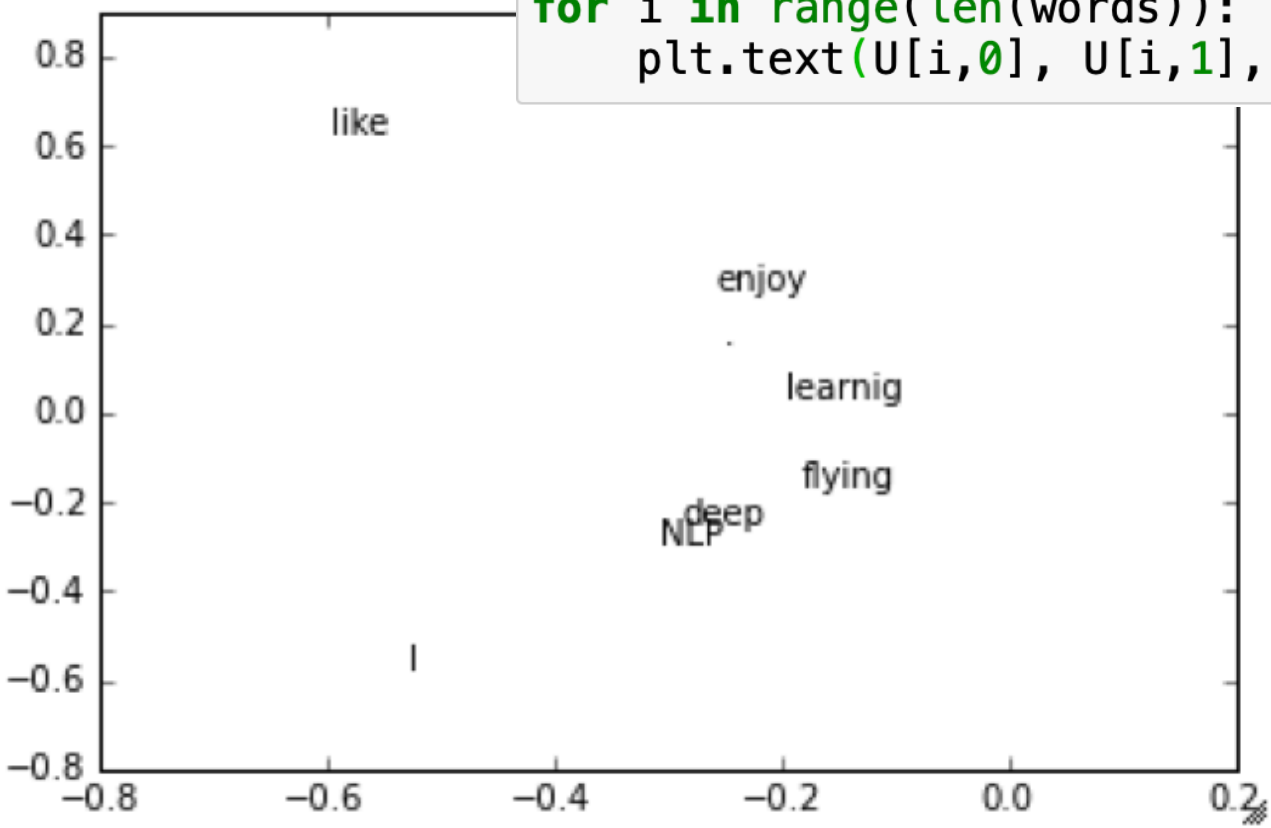
```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Simple SVD word vectors in Python

Corpus: I like deep learning. I like NLP. I enjoy flying.
Printing first two columns of U corresponding to the 2 biggest singular values

```
for i in range(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

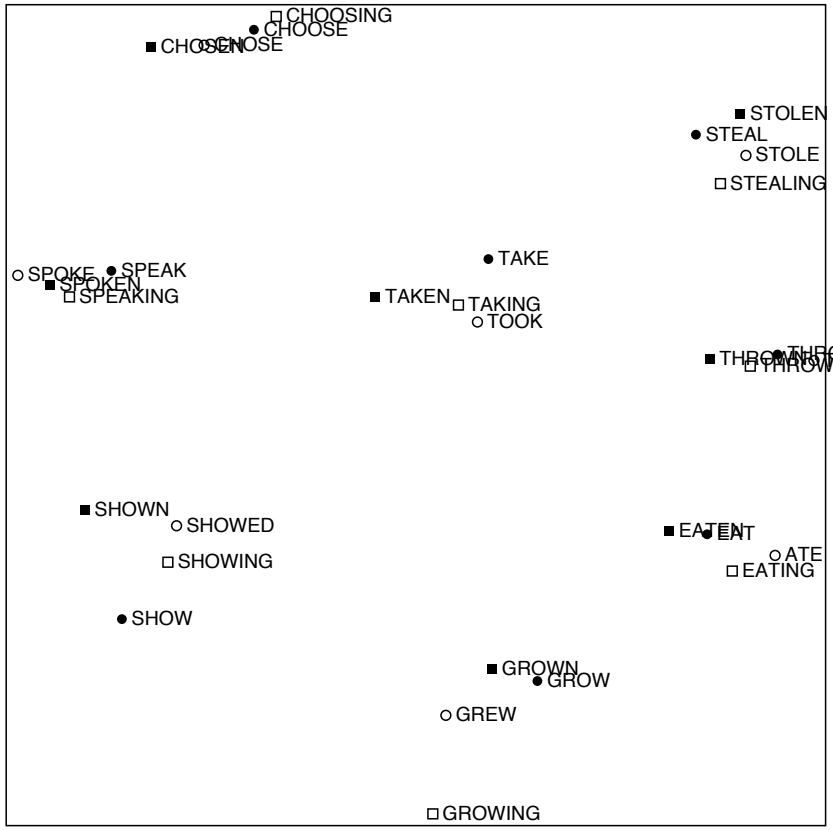
Hacks to X (several used in Rohde et al. 2005)

Scaling the counts in the cells can help ***a lot***

- Problem: function words (*the, he, has*) are too frequent → syntax has too much impact. Some fixes:
  - min(X,t), with t ≈ 100
  - Ignore them all
- Ramped windows that count closer words more
- Use Pearson correlations instead of counts, then set negative values to 0
- Etc.

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

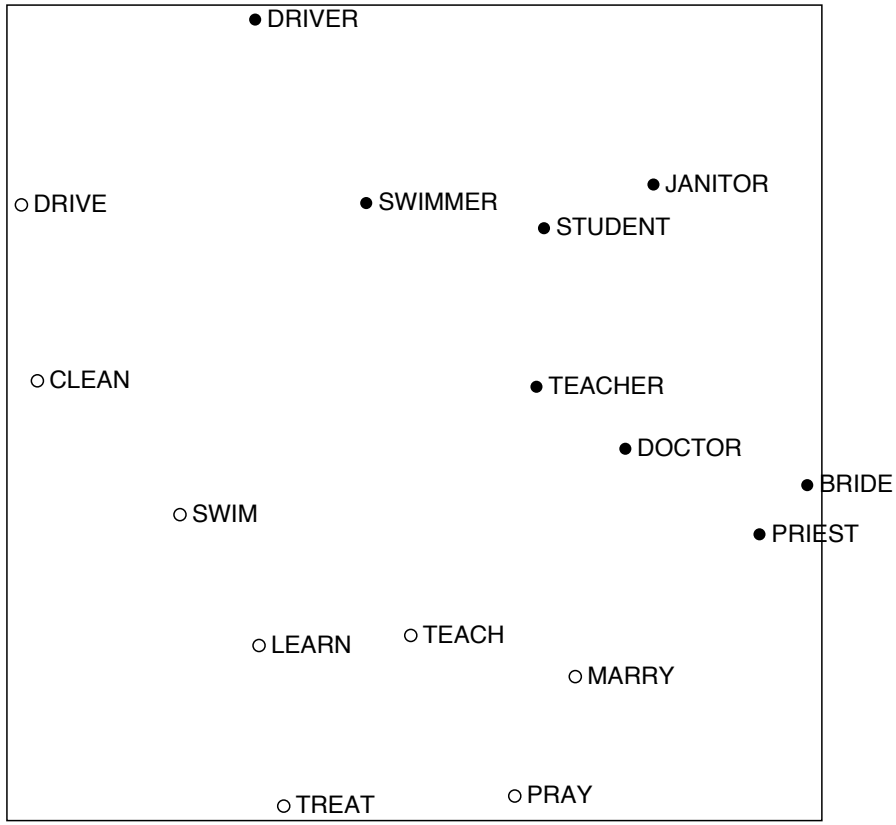
Interesting syntactic patterns emerge in the vectors



COALS model from
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. ms., 2005

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Interesting semantic patterns emerge in the vectors



COALS model from
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. ms., 2005

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

Count-based representations

- The vectors used in distributional semantics are based on frequencies, which are also called count-based representations.
- Count-based representations were successful in many similarity-related tasks, however, their usage was not able to extended to other NLP tasks.
- In order to take full advantages of machine learning / deep learning approaches, it is necessity to represent words solely in vectors. In another word, we must use vectors to replace words completely, and get rid of symbols in computing, except for the output layer.

Prediction-based representations

- If a word can be predicted by the vectors of its content words, then we can expect we can use the vectors to replace the words in any NLP tasks.
- Prediction-based word representations:
  - Randomly assign a vector to each word in the vocabulary;
  - Prepare a corpus;
  - For each of the word (referred as the current word) in the corpus, repeat:
    - Calculate the probability of all content words given the current word (or vice versa);
    - Adjust the word vectors to maximize the above probability.

Word embeddings

- Various predict-based *word representations*, or *word embeddings*, are developed, including:
  - Word2Vec, GloVe, FastText, etc.
- Word embeddings are the first step towards deep learning (neural network) based NLP
- In some cases, the pretrained word embeddings (like Word2Vec) can be directly used to solve NLP problems.
- However, this is not always the case.
- Instead, word embeddings are defined as components of the whole NLP system, whose parameters are tuned together with all other parameters.

Content

Word2Vec

- T Mikolov, I Sutskever, K Chen, GS Corrado, J Dean, Distributed representations of words and phrases and their compositionality, NIPS 2013
  - Word2Vec (include Skip-gram (SG) and Continuous Bag of Word (CBOG))
- Y Goldberg, O Levy. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv:1402.3722
- A Joulin, É Grave, P Bojanowski, T Mikolov, Bag of Tricks for Efficient Text Classification. EACL 2017.
  - FastText

Tomas Mikolov

Content

3. Word2vec: Overview

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

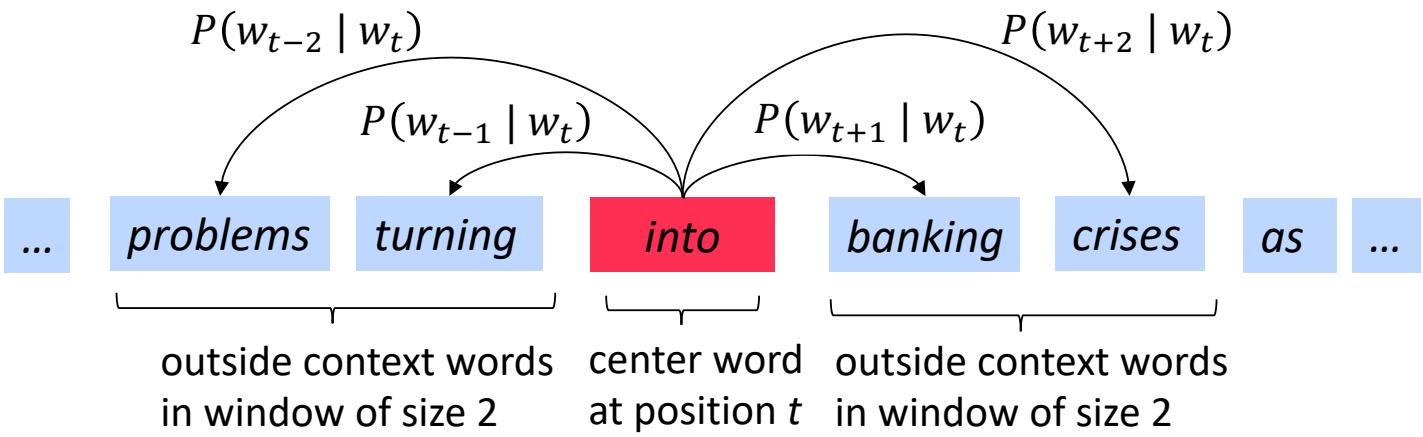
Idea:

- 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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Word2Vec Overview

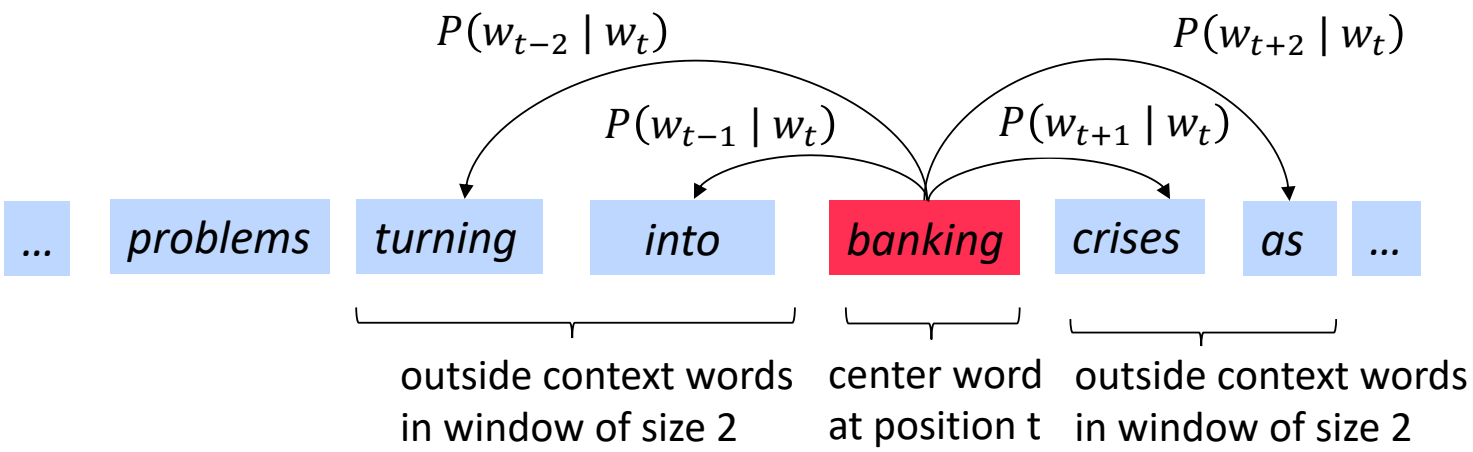
- Example windows and process for computing $P(w_{t+j} \mid w_t)$



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Word2Vec Overview

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



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

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.

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

θ is all variables to be optimized

sometimes called *cost* or *loss* function

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

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

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Cross-entropy loss function

- Negative Log Likelihood Loss:

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

  is also called Cross-Entropy Loss.
- Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events.

Cross-entropy loss function

- Consider two probability distributions p and q defined on a set of events $X = \{x_1, x_2, ..., x_n\}$, then cross-entropy between p and q is:

$$H(q, p) = -\sum_{x \in X} q(x) \log p(x)$$

- Assume X is the vocabulary, $p(x)$ is the model generated probabilities over the vocabulary, $q(x)$ is the actual distribution of the content word at $t + j$:

$$q(x) = \begin{cases} 1, & \text{for } x = w_{t+j} \\ 0, & \textit{otherwise} \end{cases}$$

then:

$$H(q, p) = -\log p(w_{t+j})$$

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*:

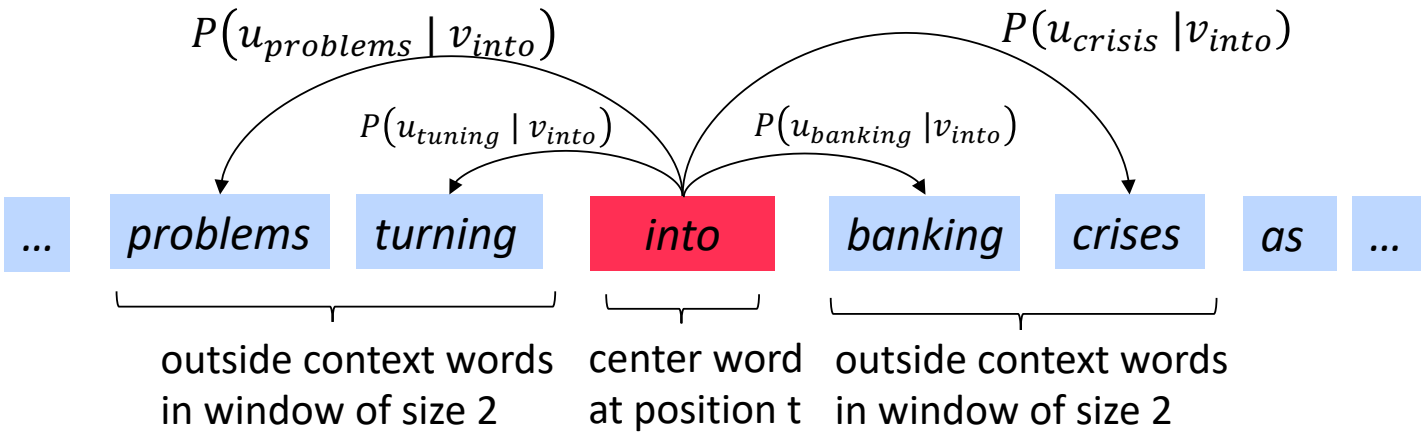
$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

22

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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 $\mathrm{P}(problems \mid into\ ; u_{problems}, v_{into}, \theta)$



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

Word2vec: prediction function

② Exponentiation makes anything positive

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

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

③ Normalize over entire vocabulary to give probability distribution

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

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

Open region

- 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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Softmax

- In mathematics, the *softmax* function, also known as *softargmax* or *normalized exponential function*, is a function that takes as input a vector of K real numbers, and normalizes it. We could interpret this as a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers.
- The standard (unit) softmax function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ is defined by the formula:

$$y_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

Softmax

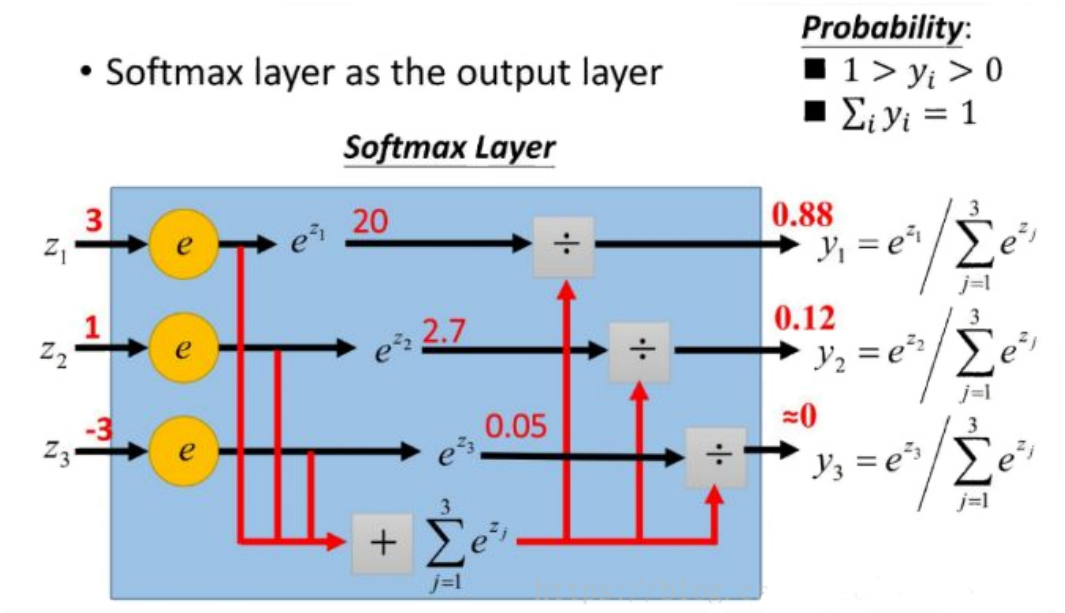
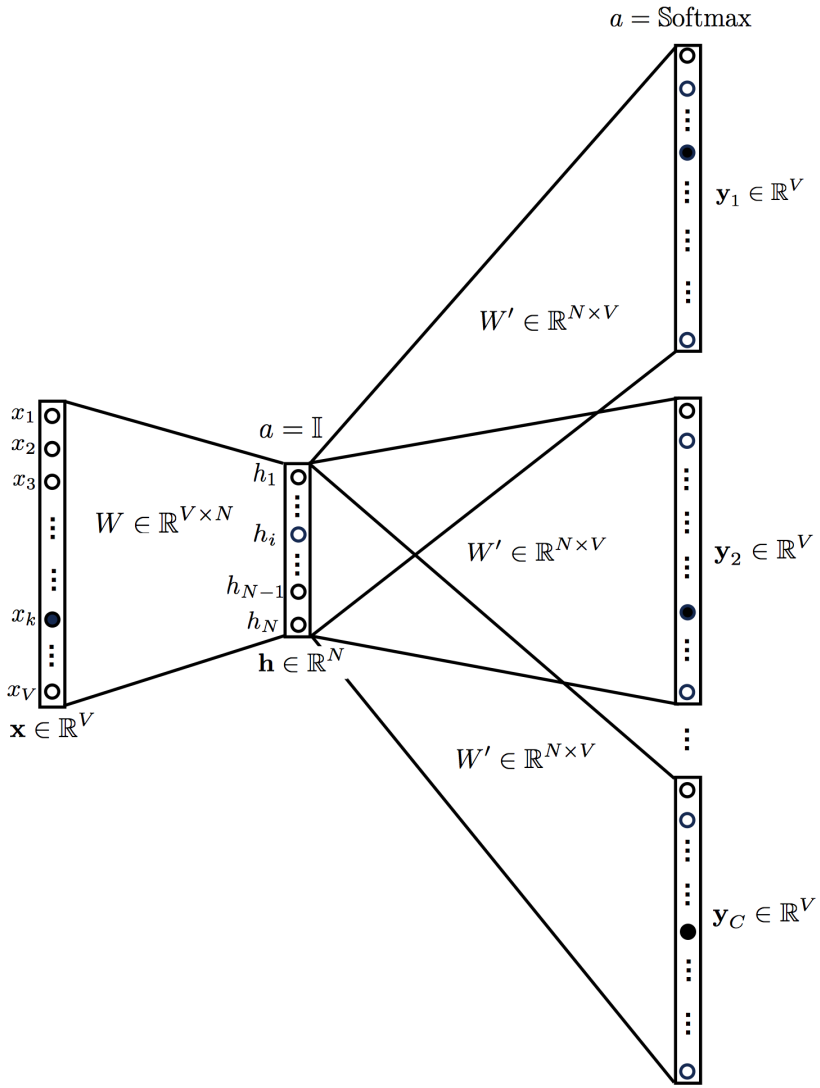


Figure source:https://blog.csdn.net/xg123321123/article/details/80781611

- In Word2Vec, because the softmax function is calculated over all words in the vocabulary, it is quite expensive computationally.

Content

Word2Vec: Skip-gram Model



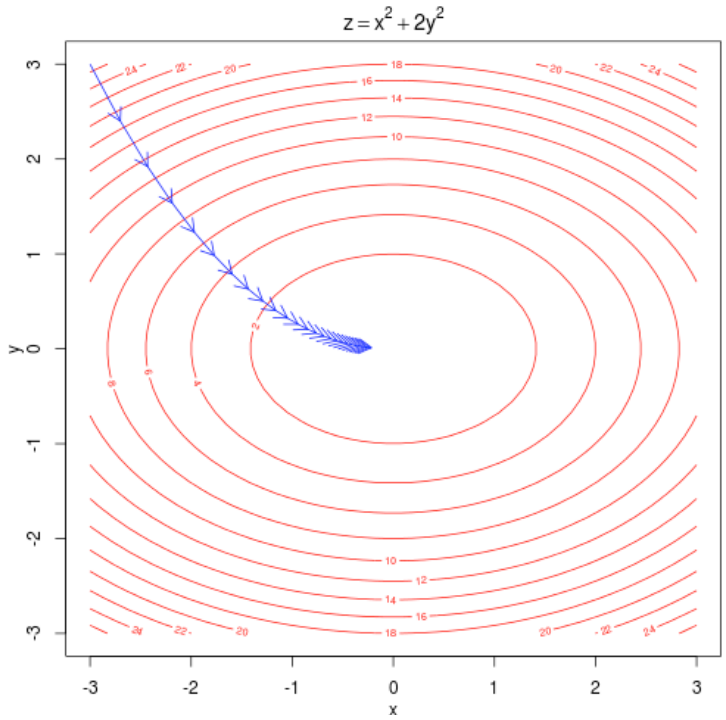
Content

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



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

Derivation of gradients for Word2Vec model

$$
\begin{aligned}
J(\theta) &= -\frac{1}{T}\log L(\theta) \\
&= -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}}\log p(w_{t+j}|w_t;\theta) \\
&= -\frac{1}{T}\sum_{o\in context(c)}\sum_{c\in corpus}\log p(o|c;u,v) \\
\frac{\partial}{\partial\theta}J(\theta) &= -\frac{1}{T}\sum_{o\in context(c)}\sum_{c\in corpus}\frac{\partial}{\partial\theta}\log p(o|c;u,v)
\end{aligned}
$$

Derivation of gradients for Word2Vec model

$$
\begin{aligned}
\frac{\partial}{\partial v_c} \log p(o|c;u,v) &= \frac{\partial}{\partial v_c} \log \frac{\exp({u_o}^T v_c)}{\sum_{w=1}^{V} \exp({u_w}^T v_c)} \\
&= \frac{\partial}{\partial v_c}({u_o}^T v_c) - \frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp({u_w}^T v_c) \\
&= u_o - \frac{1}{\sum_{w=1}^{V} \exp({u_w}^T v_c)} \frac{\partial}{\partial v_c} \sum_{w=1}^{V} \exp({u_w}^T v_c) \\
&= u_o - \frac{1}{\sum_{w=1}^{V} \exp({u_w}^T v_c)} \sum_{w=1}^{V} \frac{\partial}{\partial v_c} \exp({u_w}^T v_c) \\
&= u_o - \frac{1}{\sum_{w=1}^{V} \exp({u_w}^T v_c)} \sum_{w=1}^{V} \exp({u_w}^T v_c) u_w \\
&= u_o - \sum_{x=1}^{V} \frac{\exp({u_x}^T v_c)}{\sum_{w=1}^{V} \exp({u_w}^T v_c)} u_x \\
&= u_o - \sum_{x=1}^{V} p(x|c) u_x
\end{aligned}
$$

Derivation of gradients for Word2Vec model

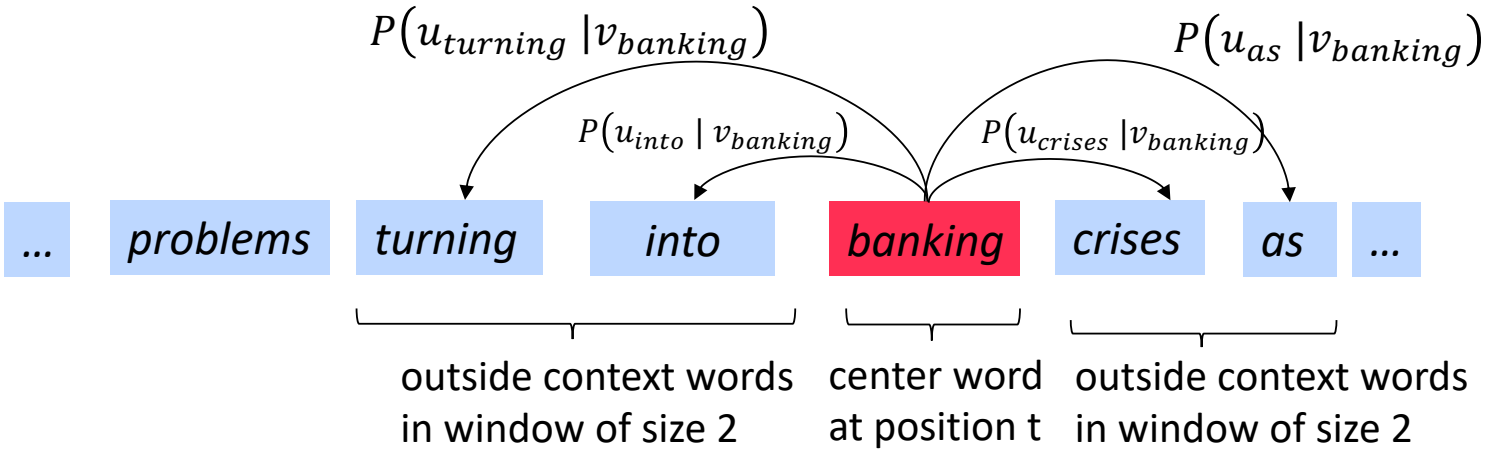
$$\frac{\partial}{\partial \theta} J(\theta) = -\frac{1}{T} \sum_{o \in context(c)} \sum_{c \in corpus} \frac{\partial}{\partial \theta} \log p(o|c; u, v)$$

$$\frac{\partial}{\partial v_c} J(\theta) = -\frac{1}{T} \sum_{o \in context(c)} \sum_{c \in corpus} \left[u_o - \sum_{x=1}^{V} p(x|c) u_x \right]$$

$$\frac{\partial}{\partial u_o} J(\theta) = -\frac{1}{T} \sum_{o \in context(c)} \sum_{c \in corpus} \left[v_c - \sum_{x=1}^{V} p(o|x) v_x \right]$$

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:

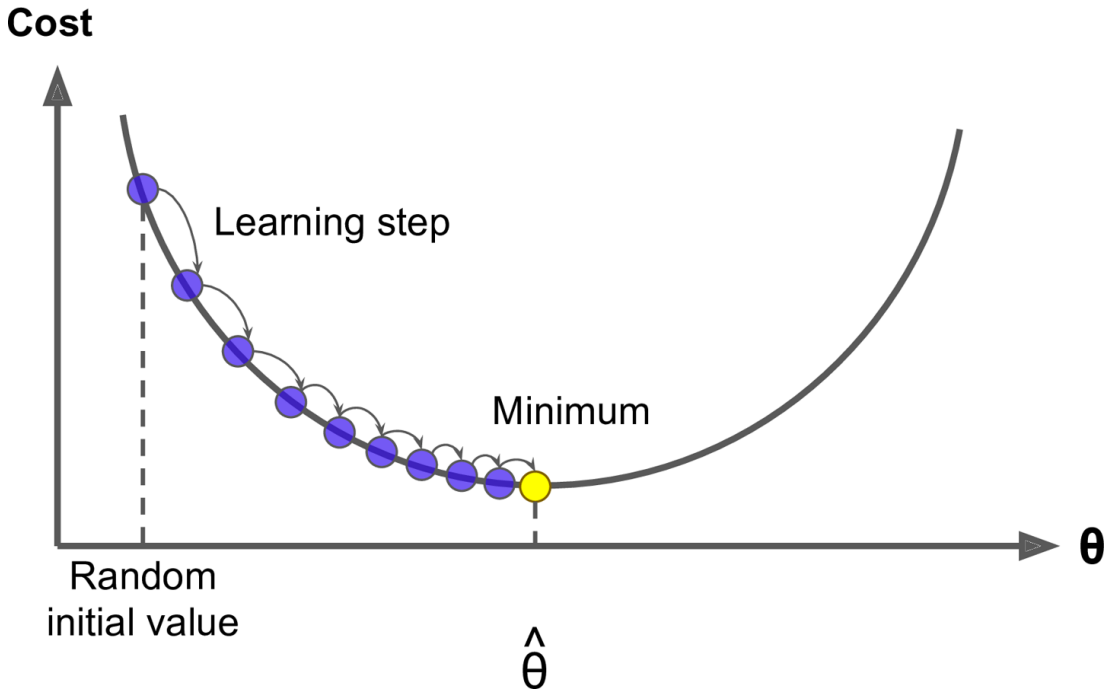


Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

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 :(

32

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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
```

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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
```

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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!

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

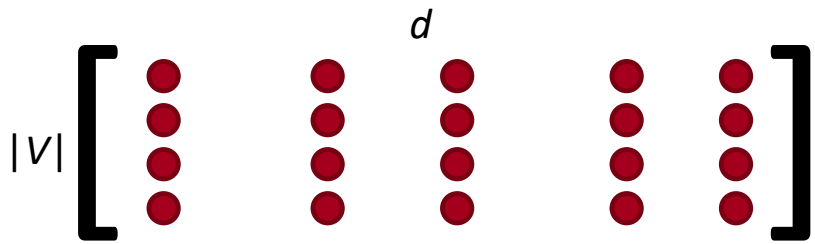
9

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Stochastic gradients with word vectors!

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

- Solution: either you need sparse matrix update operations to only update certain rows of full embedding matrices *U* and *V,* or you need to keep around a hash for word vectors



- If you have millions of word vectors and do distributed computing, it is important to not have to send gigantic updates around!

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

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**

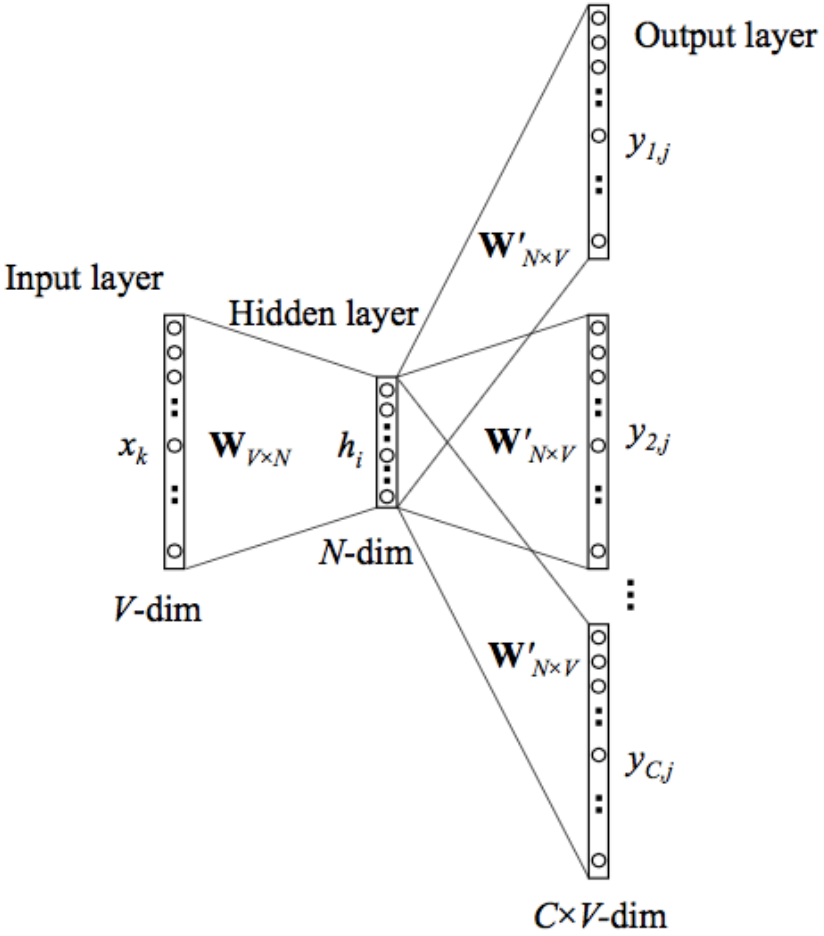
Additional efficiency in training:

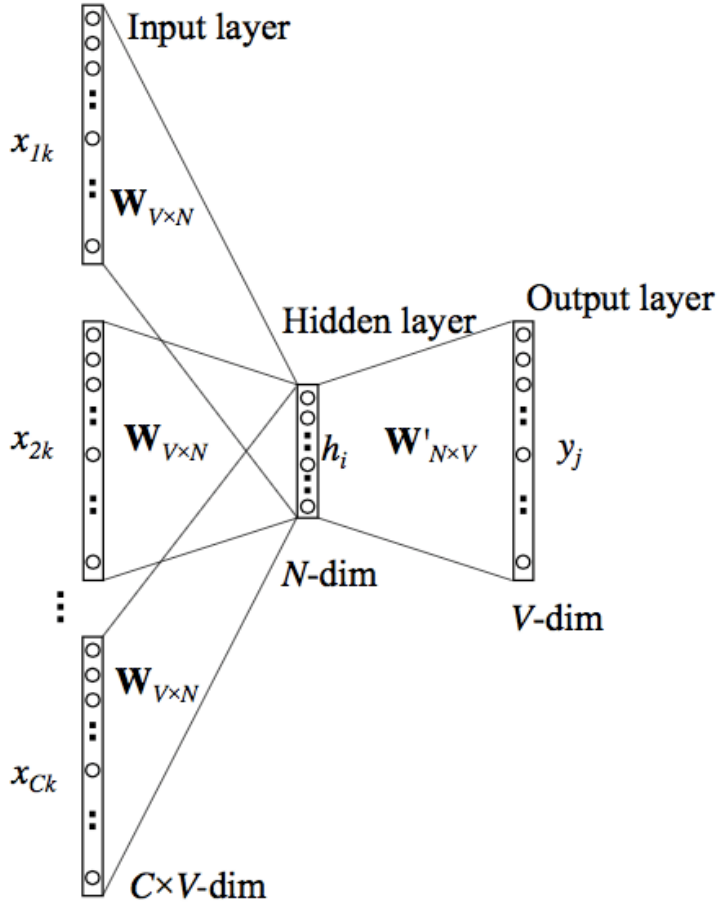
1. Negative sampling

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Skip-gram Model vs. CBOW Model





Negative Sampling

$$J(u_o, C) = \sum_{w \in C} exp(u_o^T u_w) + \sum_{w \notin C} exp(-u_o^T u_w)$$

- C - is a context (set of words),
- first part is *positive* samples,
- second part is *negative* samples.

Content

4. Towards GloVe: Count based vs. direct prediction

- LSA, HAL (Lund & Burgess),
- COALS, Hellinger-PCA (Rohde et al, Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

- Skip-gram/CBOW (Mikolov et al)
- NNLM, HLBL, RNN (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Encoding meaning in vector differences

[Pennington, Socher, and Manning, EMNLP 2014]

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Encoding meaning in vector differences

[Pennington, Socher, and Manning, EMNLP 2014]

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

	x = solid	x = gas	x = water	x = fashion
$P(x\|\text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(x\|\text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	8.9	8.5×10^{-2}	1.36	0.96

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Encoding meaning in vector differences

Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

A: Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$

with vector differences $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Encoding meaning in vector differences

Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

A: Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$

with vector differences $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

---

Note: $P(i|j) \neq P(j|i)$, w and $\tilde{w}$ should be defined separately!

Correction: Log-bilinear model: $w_i \cdot \tilde{w}_j = \log P(i|j)$

with vector differences: $w_x \cdot (\tilde{w}_a - \tilde{w}_b) = \log \frac{P(x|a)}{P(x|b)}$

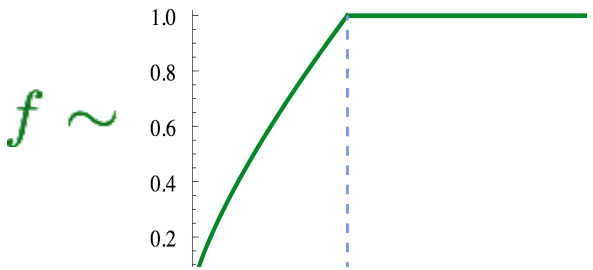
Combining the best of both worlds

GloVe [Pennington et al., EMNLP 2014]

$$w_i \cdot w_j = \log P(i|j)$$

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

GloVe results

Nearest words to frog:

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

litoria

leptodactylidae

rana

eleutherodactylus

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

5. How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs. extrinsic
- Intrinsic:
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
  - Evaluation on a real task
  - Can take a long time to compute accuracy
  - Unclear if the subsystem is the problem or its interaction or other subsystems
  - If replacing exactly one subsystem with another improves accuracy → Winning!

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Intrinsic word vector evaluation

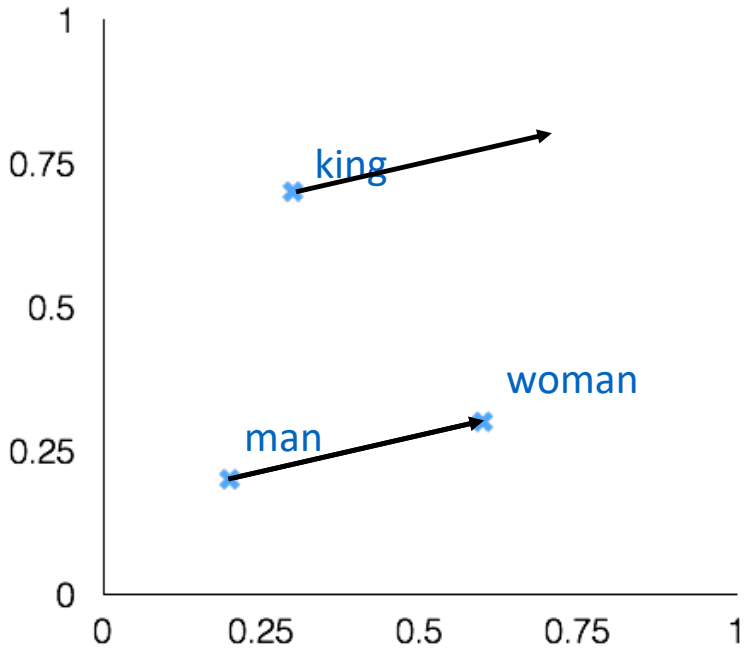
- Word Vector Analogies

a:b :: c:?

man:woman :: king:?

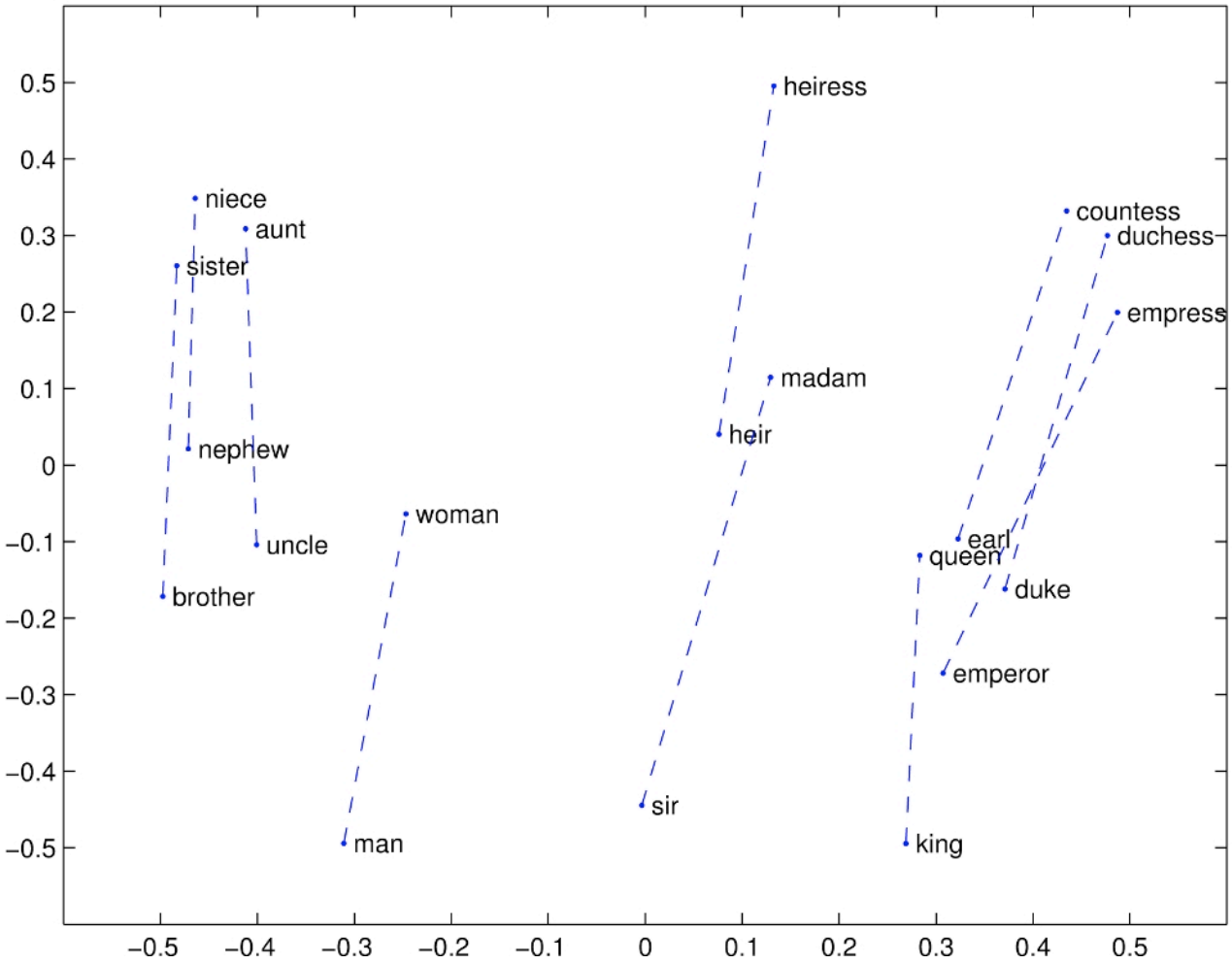
$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$

- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?



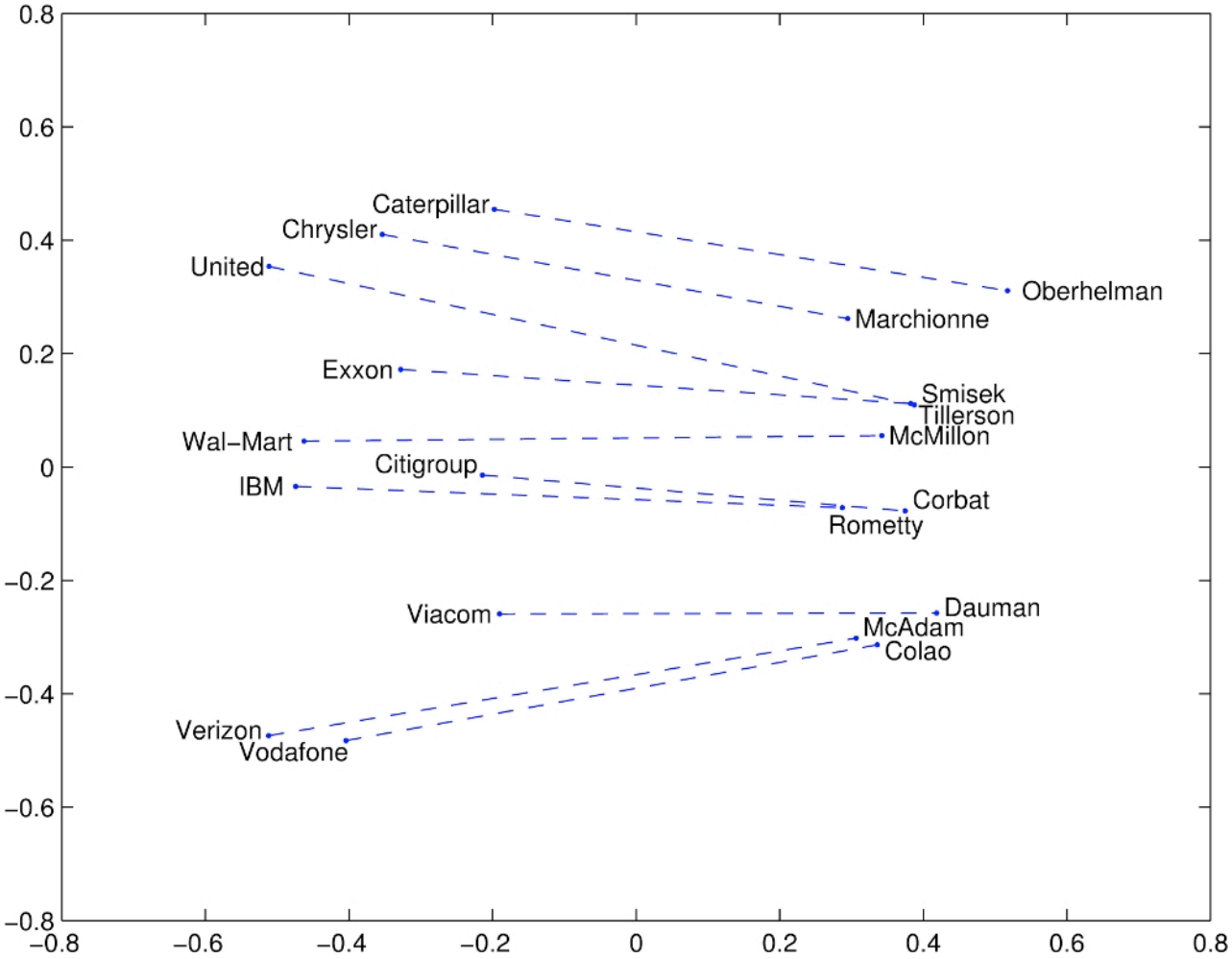
Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Glove Visualizations



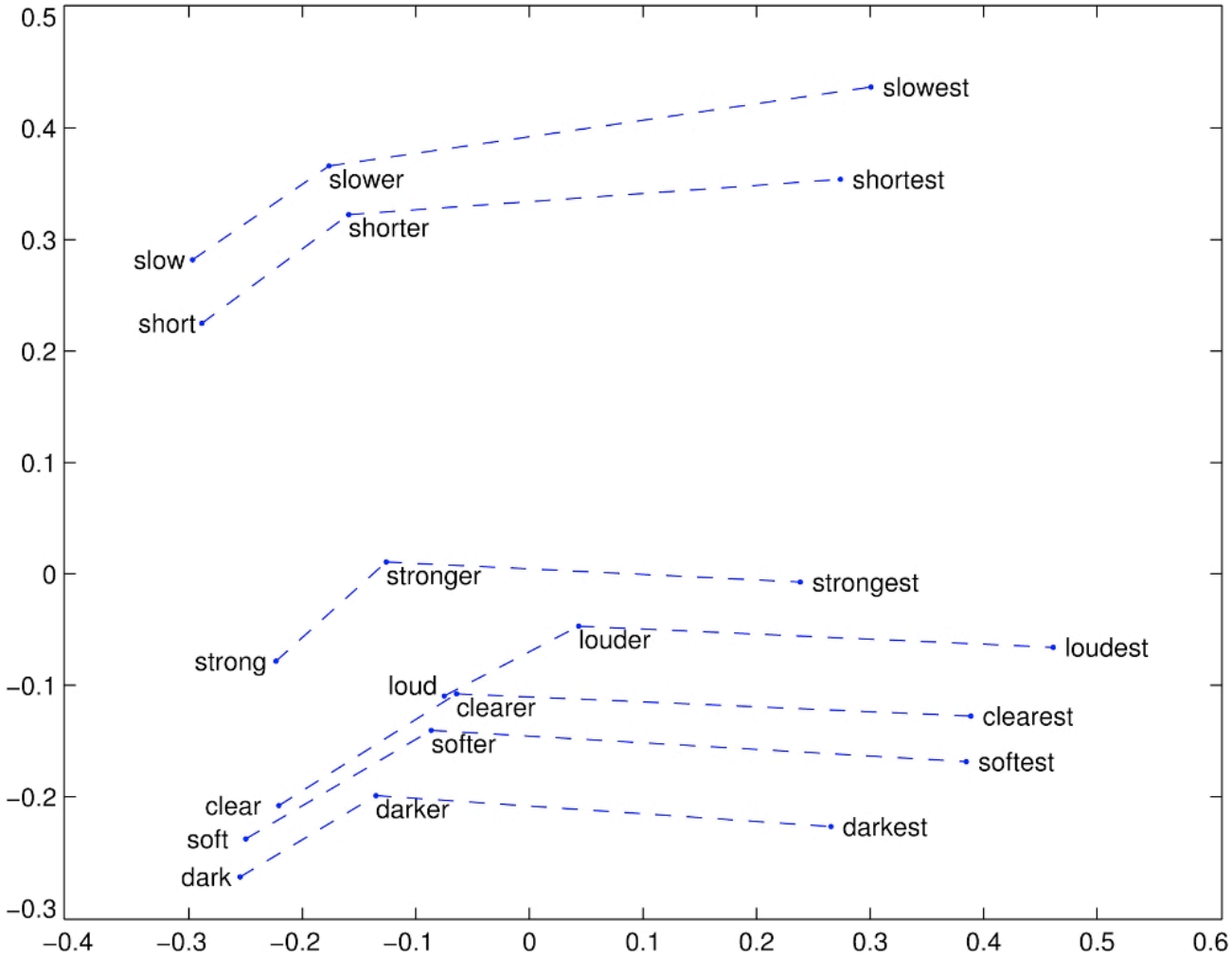
Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Glove Visualizations: Company - CEO



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Glove Visualizations: Superlatives



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Analogy evaluation and hyperparameters

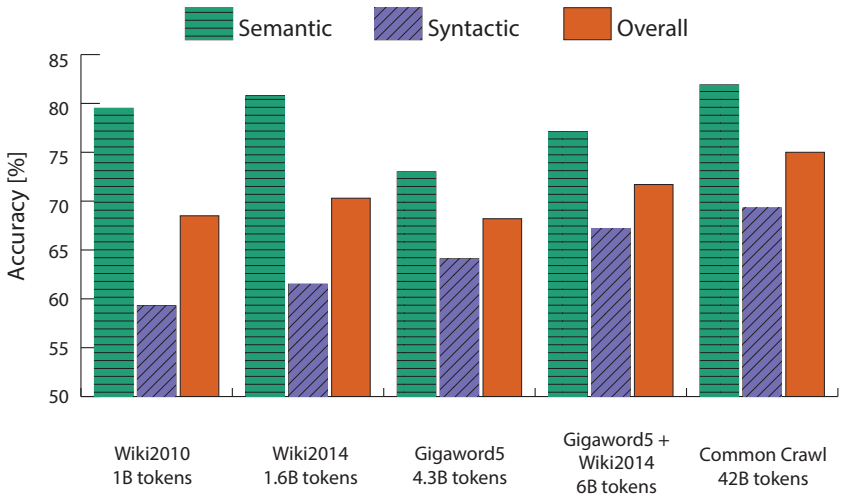
Glove word vectors evaluation

Model	Dim.	Size	Sem.	Syn.	Tot.
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

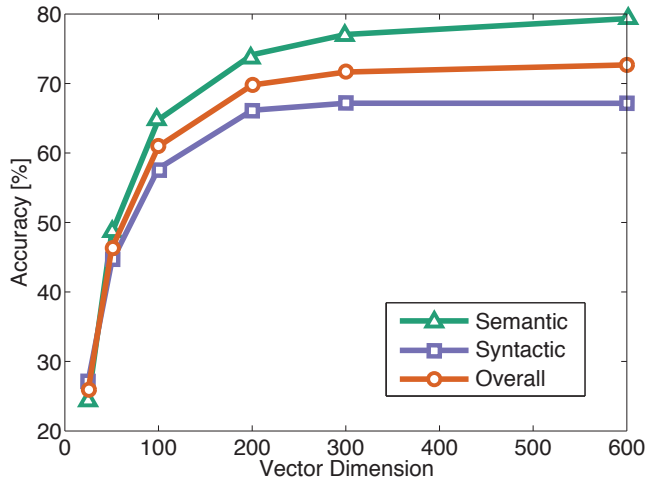
Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Analogy evaluation and hyperparameters

- More data helps
- Wikipedia is better than news text!



- Dimensionality
- Good dimension is ~300



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Another intrinsic word vector evaluation

- Word vector distances and their correlation with human judgments
- Example dataset: WordSim353
  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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Correlation evaluation

- Word vector distances and their correlation with human judgments

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

- Some ideas from Glove paper have been shown to improve skip-gram (SG) model also (e.g. sum both vectors)

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Extrinsic word vector evaluation

- Extrinsic evaluation of word vectors: All subsequent tasks in this class

- One example where good word vectors should help directly: named entity recognition: finding a person, organization or location

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**

- Next: How to use word vectors in neural net models!

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

6. Word senses and word sense ambiguity

- Most words have lots of meanings!
  - Especially common words
  - Especially words that have existed for a long time

- Example: **pike**

- Does one vector capture all these meanings or do we have a mess?

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

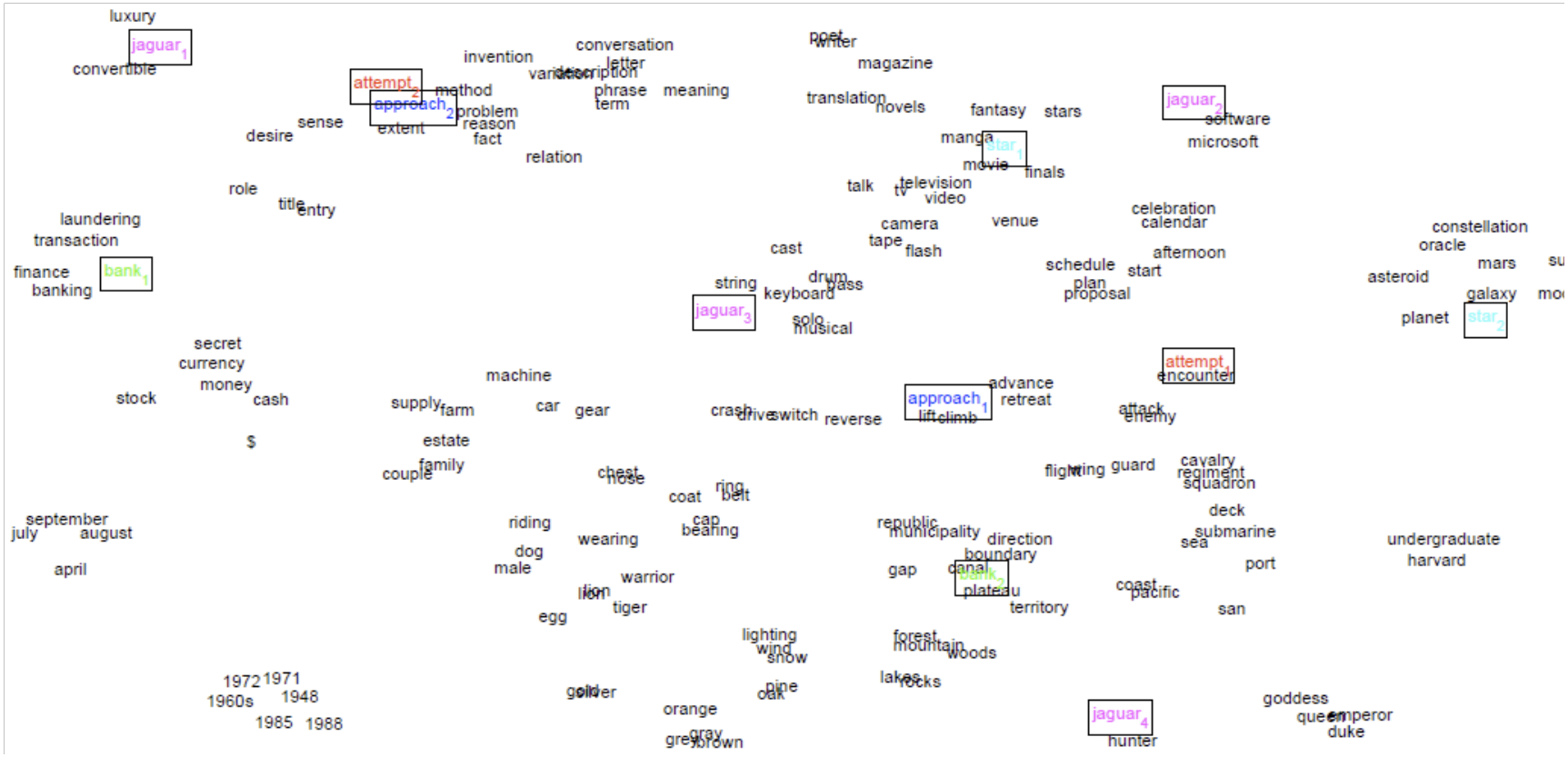
pike

- A sharp point or staff
- A type of elongated fish
- A railroad line or system
- A type of road
- The future (coming down the pike)
- A type of body position (as in diving)
- To kill or pierce with a pike
- To make one's way (pike along)
- In Australian English, pike means to pull out from doing something: *I reckon he could have climbed that cliff, but he piked!*

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Improving Word Representations Via Global Context And Multiple Word Prototypes (Huang et al. 2012)

- Idea: Cluster word windows around words, retrain with each word assigned to multiple different clusters $bank_1$, $bank_2$, etc



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Linear Algebraic Structure of Word Senses, with Applications to Polysemy (Arora, …, Ma, …, TACL 2018)

- Different senses of a word reside in a linear superposition (weighted sum) in standard word embeddings like word2vec
- $v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$
- Where $\alpha_1 = \frac{f_1}{f_1+f_2+f_3}$, etc., for frequency f
- Surprising result:
  - Because of ideas from *sparse coding* you can actually separate out the senses (providing they are relatively common)

tie				
trousers	season	scoreline	wires	operatic
blouse	teams	goalless	cables	soprano
waistcoat	winning	equaliser	wiring	mezzo
skirt	league	clinching	electrical	contralto
sleeved	finished	scoreless	wire	baritone
pants	championship	replay	cable	coloratura

45

Content

Limitation of Skip-Gram

- It is difficult for good representations of rare words to be learned with traditional word2vec.
- There could be words in the NLP task that were not present in the word2vec training corpus

- This limitation is more pronounced in case of morphologically rich languages

EX) In French or Spanish, most verbs have more than forty different inflected froms. While the Finnish languages has fifteen cases for nouns

-> It is possible to improve vector representations for Morphologically rich languages by using **character level** information.

Piotr Bonjanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching Word Vectors with Subword Information, slides

Example

- German verb : 'sein' (English verb : 'be')

Indikativ

Indikativ Präsens

ich	bin
du	bist
er/sie/es	ist
wir	sind
ihr	seid
sie/Sie	sind

Indikativ Perfekt

ich	bin gewesen
du	bist gewesen
er/sie/es	ist gewesen
wir	sind gewesen
ihr	seid gewesen
sie/Sie	sind gewesen

Indikativ Futur I

ich	werde sein
du	wirst sein
er/sie/es	wird sein
wir	werden sein
ihr	werdet sein
sie/Sie	werden sein

Konjuktiv

Konjunktiv I Präsens

ich	sei
du	seiest; seist
er/sie/es	sei
wir	seien
ihr	seiet
sie/Sie	seien

Konjunktiv I Perfekt

ich	sei gewesen
du	seiest gewesen; seist gewesen
er/sie/es	sei gewesen
wir	seien gewesen
ihr	seiet gewesen
sie/Sie	seien gewesen

Konjunktiv I Futur I

ich	werde sein
du	werdest sein
er/sie/es	werde sein
wir	werden sein
ihr	werdet sein
sie/Sie	werden sein

Konjunktiv I Futur II

ich	werde gewesen sein
du	werdest gewesen sein
er/sie/es	werde gewesen sein
wir	werden gewesen sein
ihr	werdet gewesen sein
sie/Sie	werden gewesen sein

Konjunktiv II Präteritum

ich	wäre
du	wärest
er/sie/es	wäre
wir	wären
ihr	wäret
sie/Sie	wären

Konjunktiv II Plusquamperfekt

Konjunktiv II Futur I

ich	würde sein
du	würdest sein
er/sie/es	würde sein
wir	würden sein
ihr	würdet sein
sie/Sie	würden sein

Imperativ

Imperativ

du	sei
ihr	seid

Partizip

Partizip Präsens	Partizip Perfekt
seiend	gewesen

Piotr Bonjanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching Word Vectors with Subword Information, slides

Character n-gram based model

- The basic skip-gram model described above ignores the internal structure of the word.
- However, character n-gram based model incorporates information about the structure in terms of character n-gram embeddings.
- This paper suppose that each word w is represented as a bag of character n-gram.

EX) where / n = 3, it will be represented by the character n-grams :

<wh / whe / her / ere / re>

And the special sequence

<where>

word *<her>* is different from the tri-gram *her* from the word where *where*

Piotr Bonjanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching Word Vectors with Subword Information, slides

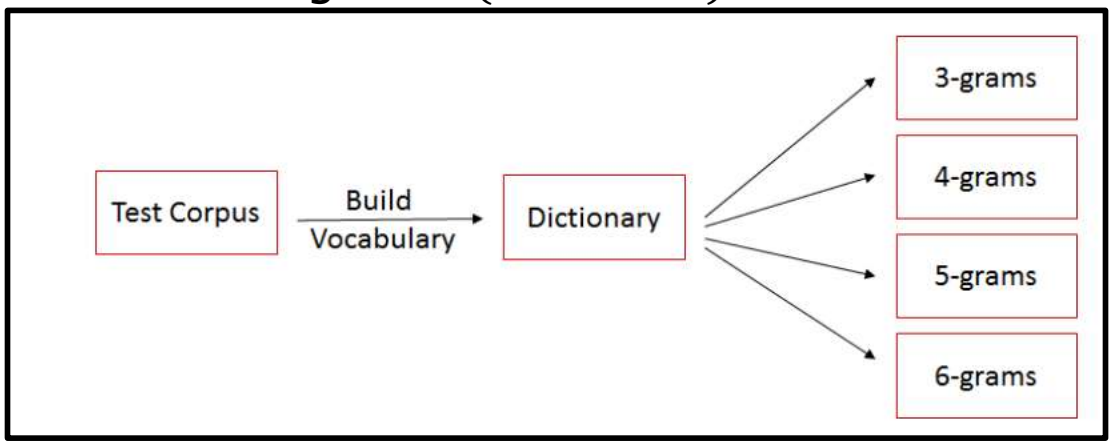
Character n-gram based model (cont'd)

- We represent a word by the sum of the vector representations of its n-gram. We thus obtain the scoring function :

$$s(w,c) = \sum_{g \in \varsigma_w} z_g^T v_c$$

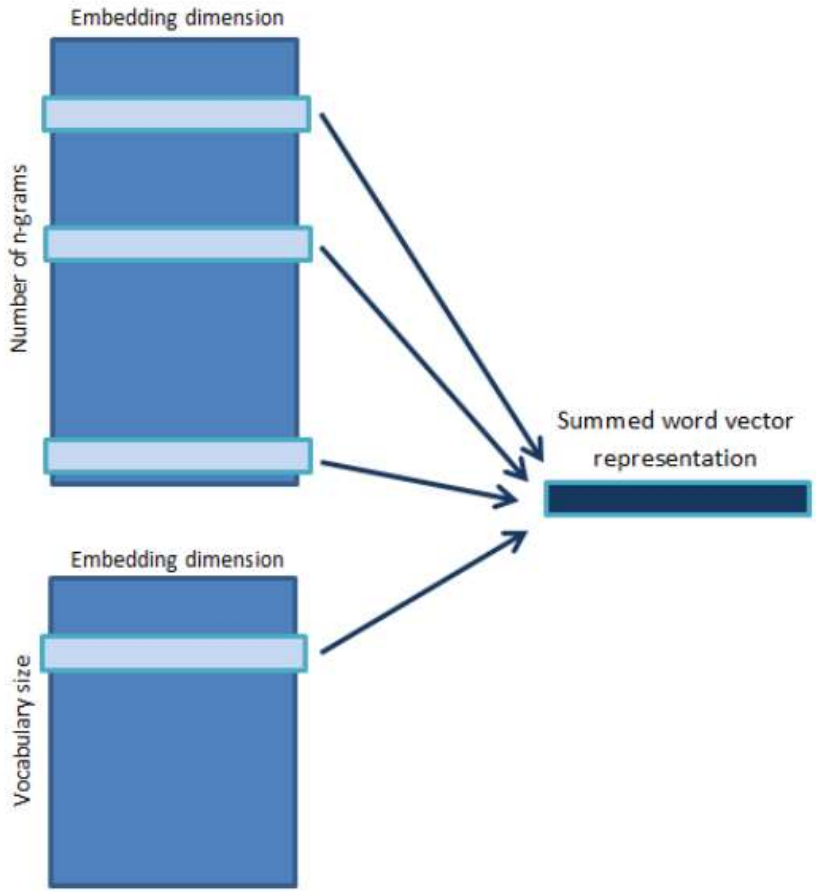
w : *given word*
ς_w : the set of n − grams appearing in word w
z_g : *vector representation to each* $n-grams$
v_c : *the word vector of the center word* C

- We extract all the n -grams. $(3 \leq n \leq 6)$



Piotr Bonjanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching Word Vectors with Subword Information, slides

Computing word vector representation



Piotr Bonjanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching Word Vectors with Subword Information, slides

Experiments Settings

- Target Languages
  - German / English / French / Spanish / Arabic / Romanian / Russian / Czech

- Kind of tasks
  1. Human similarity judgement
  2. Word analogy tasks
  3. Comparison with morphological representations
  4. Effect of the size of the training data
  5. Effect of the size of n-grams

Piotr Bonjanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching Word Vectors with Subword Information, slides

1. Human similarity judgement

- Correlation between human judgement and similarity scores on word similarity datasets.

		sg	cbow	sisg-	sisg
AR	WS353	51	52	54	**55**
DE	GUR350	61	62	64	**70**
	GUR65	78	78	**81**	**81**
	ZG222	35	38	41	**44**
EN	RW	43	43	46	**47**
	WS353	72	**73**	71	71
ES	WS353	57	58	58	**59**
FR	RG65	70	69	**75**	**75**
RO	WS353	48	52	51	**54**
RU	HJ	59	60	60	**66**

RW : Rare Words dataset

sg : Skip-Gram
cbow : continuous bag of words
sisg- : Subword Information Skip-Gram
(Treat unseen words as a null vector)
sisg : Subword Information Skip-Gram
(Treat unseen words by summing the n-gram vectors)

Piotr Bonjanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching Word Vectors with Subword Information, slides

2. Word analogy tasks

- Accuracy of our model and baselines on word analogy tasks for Czech, German, English and Italian

		sg	cbow	sisg
CS	Semantic	25.7	27.6	27.5
	Syntactic	52.8	55.0	77.8
DE	Semantic	66.5	66.8	62.3
	Syntactic	44.5	45.0	56.4
EN	Semantic	78.5	78.2	77.8
	Syntactic	70.1	69.9	74.9
IT	Semantic	52.3	54.7	52.3
	Syntactic	51.5	51.8	62.7

* It is observed that morphological information significantly improves the syntactic tasks; our approach outperforms the baselines. In contrast, it does not help for semantic questions, and even degrades the performance for German and Italian.

Piotr Bonjanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching Word Vectors with Subword Information, slides

3. Comparison with morphological representations

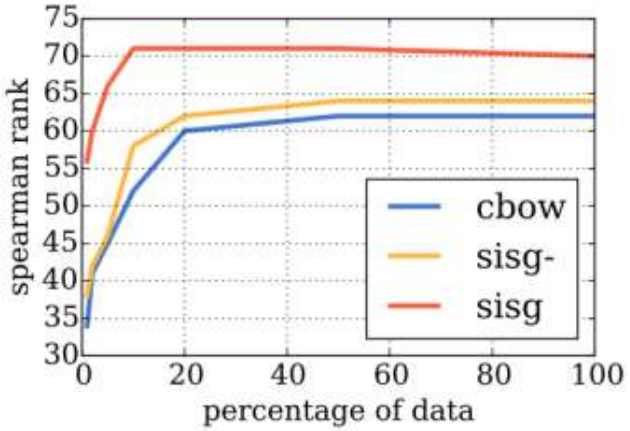
- Spearman's rank correlation coefficient between human judgement and model scores for different methods using morphology to learn word representations.

	DE		EN		ES	FR
	GUR350	ZG222	WS353	RW	WS353	RG65
Luong et al. (2013)	-	-	64	34	-	-
Qiu et al. (2014)	-	-	65	33	-	-
Soricut and Och (2015)	64	22	71	42	47	67
sisg	73	43	73	48	54	69

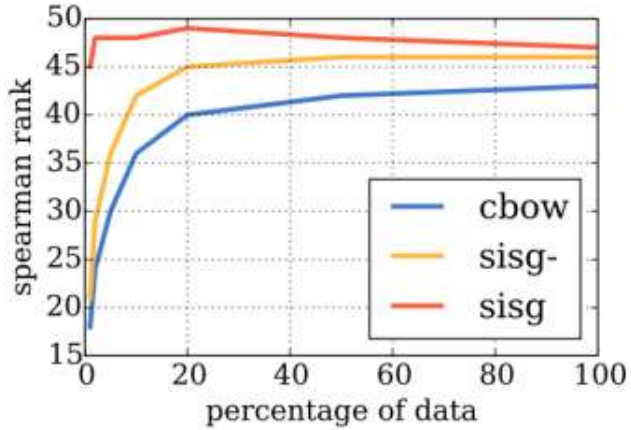
Piotr Bonjanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching Word Vectors with Subword Information, slides

4. Effect of the size of the training data

- Influence of size of the training data on performance
  (Data : full Wikipedia dump / Task : 1. similarity task)



(a) DE-GUR350



(b) EN-RW

- Sisg model is more robust to the size of the training data.
- However, the performance of the baseline cbow model gets better as more and more data is available. Sisg model, on the other hand, seems to quickly saturate and adding more data does not always lead to improved result.
- It is observed the performance sisg with very small dataset better than the performance

Piotr Bonjanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching Word Vectors with Subword Information, slides

5. Effect of the size of n-grams

Maximum value of n

Minimum value of n

	2	3	4	5	6
2	57	64	67	69	69
3		65	68	70	70
4			70	70	**71**
5				69	**71**
6					70

(a) DE-GUR350

	2	3	4	5	6
2	59	55	56	59	60
3		60	58	60	62
4			62	62	63
5				64	64
6					**65**

(b) DE Semantic

	2	3	4	5	6
2	45	50	53	54	55
3		51	55	55	**56**
4			54	**56**	**56**
5				**56**	**56**
6					54

(c) DE Syntactic

	2	3	4	5	6
2	41	42	46	47	**48**
3		44	46	**48**	**48**
4			47	**48**	**48**
5				**48**	**48**
6					**48**

(d) EN-RW

	2	3	4	5	6
2	78	76	75	76	76
3		78	77	78	77
4			79	79	79
5				**80**	79
6					**80**

(e) EN Semantic

	2	3	4	5	6
2	70	71	73	74	73
3		72	74	**75**	74
4			74	**75**	**75**
5				74	74
6					72

(f) EN Syntactic

- The choice of *n* boundary is observed to be language and task dependent.
- Results are always improved by taking $n \geq 3$ rather than $n \geq 2$, which shows that character 2-games arenot informative for that task

Piotr Bonjanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching Word Vectors with Subword Information, slides

Conclusion

- General Skip-Gram model has some limitations. (Ex: OOV)
- But, This can be overcome by using subword information (character n-grams).
- This model is simple. Because of simplicity, this model trains fast and does not require any preprocessing or supervision.
- It works better for certain languages. (Ex: German)

Piotr Bonjanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching Word Vectors with Subword Information, slides

Content