CS546: Machine Learning in NLP (Spring 2020)
http://courses.engr.illinois.edu/cs546/

Lecture 4: Static word embeddings

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center
Office hours: Monday, 11am—12:30pm

(Static) Word Embeddings

A (static) word embedding is a function that maps each word type to a single vector

— these vectors are typically dense and have much lower dimensionality than the size of the vocabulary

— this mapping function typically ignores that the same string of letters may have different senses (dining table vs. a table of contents) or parts of speech (to table a motion vs. a table)

— this mapping function typically assumes a fixed size vocabulary (so an UNK token is still required)

Word2Vec Embeddings

Main idea:

Use a classifier to predict which words appear in the context of (i.e. near) a target word (or vice versa)

This classifier induces a dense vector representation of words (embedding)

Words that appear in similar contexts (that have high distributional similarity) will have very similar vector representations.

These models can be trained on large amounts of raw text (and pre-trained embeddings can be downloaded)

Word2Vec (Mikolov et al. 2013)

The first really influential dense word embeddings

Two ways to think about Word2Vec:

— a simplification of neural language models

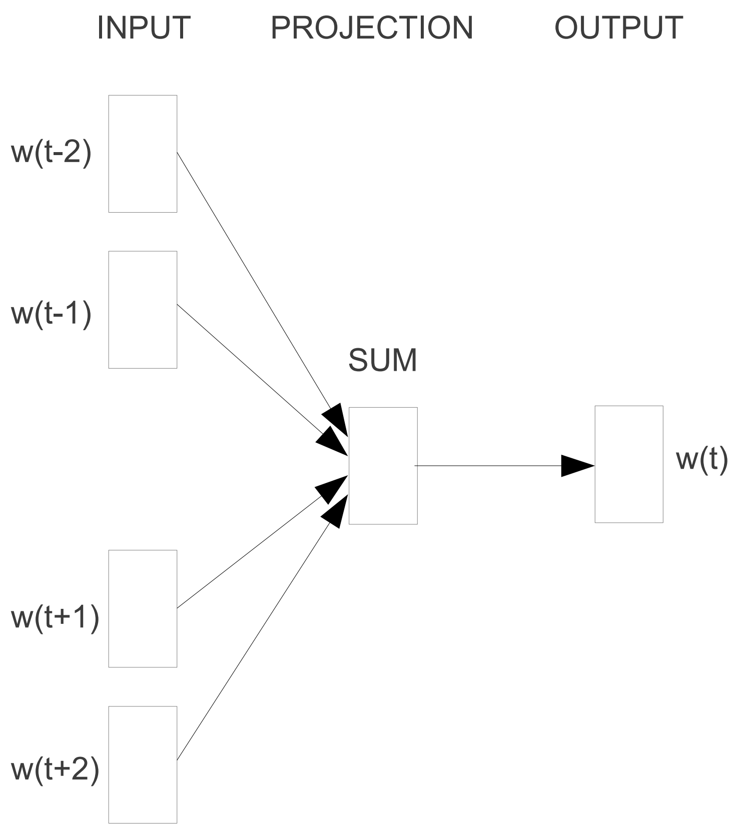
— a binary logistic regression classifier

Variants of Word2Vec

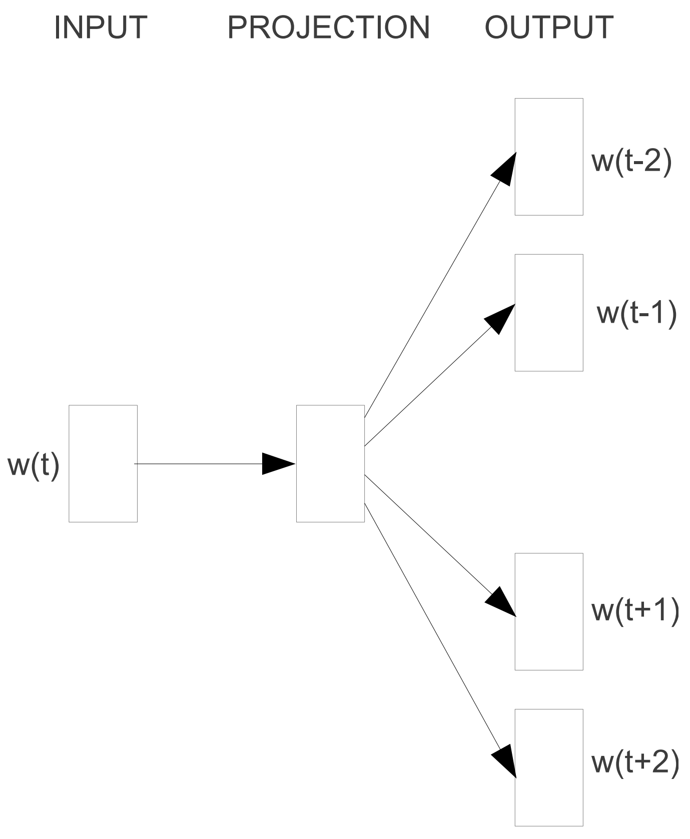
— Two different context representations: CBOW or Skip-Gram

— Two different optimization objectives:
Negative sampling (NS) or hierarchical softmax

Word2Vec architectures



CBOW



Skip-gram

CBOW: predict target from context

(CBOW=Continuous Bag of Words)

Training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 t c3 c4

Given the surrounding context words (tablespoon, of, jam, a), predict the target word (apricot).

Input: each context word is a one-hot vector
Projection layer: map each one-hot vector down to a dense D-dimensional vector, and average these vectors
Output: predict the target word with softmax

Skipgram: predict context from target

Training sentence:

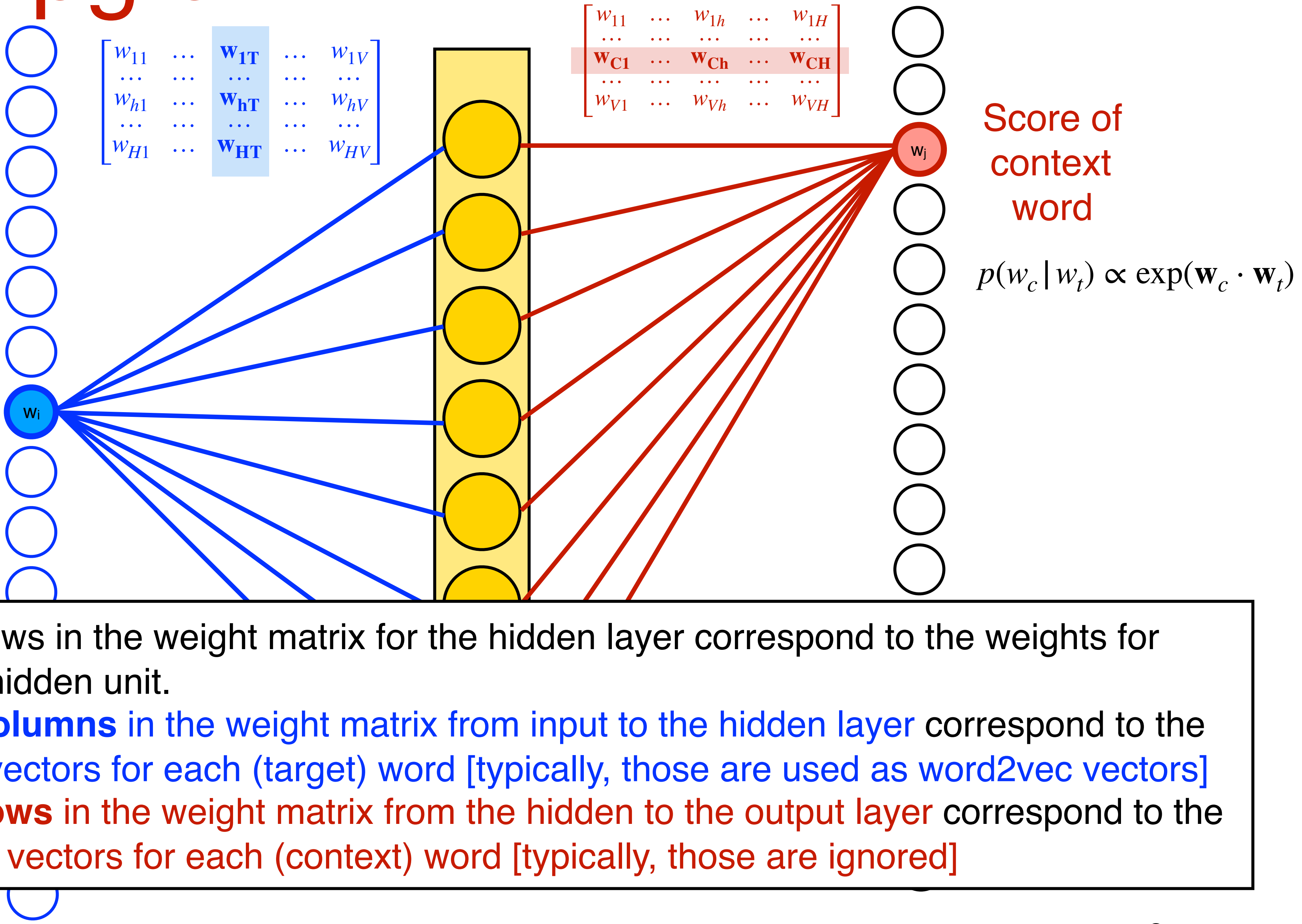
... lemon, a tablespoon of **apricot** jam a pinch ...

... lemon, a	tablespoon	of	apricot	jam	a	pinch ...
	c1	c2	t	c3	c4	

Given the target word (apricot), predict the surrounding context words (tablespoon, of, jam, a),

- **Input:** each target word is a one-hot vector
- **Projection layer:** map each one-hot vector down to a dense D-dimensional vector, and average these vectors
- **Output**: predict the context word with softmax

Skipgram



$\begin{bmatrix} w_{11} & \cdots & \mathbf{w_{1T}} & \cdots & w_{1V} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ w_{h1} & \cdots & \mathbf{w_{hT}} & \cdots & w_{hV} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ w_{H1} & \cdots & \mathbf{w_{HT}} & \cdots & w_{HV} \end{bmatrix}$
$\begin{bmatrix} w_{11} & \cdots & w_{1h} & \cdots & w_{1H} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{w_{C1}} & \cdots & \mathbf{w_{Ch}} & \cdots & \mathbf{w_{CH}} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ w_{V1} & \cdots & w_{Vh} & \cdots & w_{VH} \end{bmatrix}$
w_i
w_j

One-hot encoding of target word

Score of context word

$$p(w_c \mid w_t) \propto \exp(\mathbf{w}_c \cdot \mathbf{w}_t)$$

The rows in the weight matrix for the hidden layer correspond to the weights for each hidden unit.
The **columns** in the weight matrix from input to the hidden layer correspond to the input vectors for each (target) word [typically, those are used as word2vec vectors]
The **rows** in the weight matrix from the hidden to the output layer correspond to the output vectors for each (context) word [typically, those are ignored]

Negative sampling

Skipgram aims to optimize the avg log probability of the data:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j} \mid w_t) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log\left(\frac{\exp(w_{t+j}w_t)}{\sum_{k=1}^{V}\exp(w_k w_t)}\right)$$

But computing the partition function $\sum_{k=1}^{V}\exp(w_k w_t)$ is very expensive

— This can be mitigated by **hierarchical softmax**
(represent each w_{t+j} by Huffman encoding, and predict the sequence of nodes in the resulting binary tree via softmax).

— **Noise Contrastive Estimation** is an alternative to (hierarchical) softmax that aims to distinguish actual data points w_{t+j} from noise via logistic regression

— But we just want good word representations, so we do something simpler:

Negative Sampling instead aims to optimize

$$\log\sigma(w_T\cdot w_c) + \sum_{i=1}^{k} E_{w_i\sim P(w)}\Big[\log\sigma(-w_T w_i)\Big] \qquad \text{with } \sigma(x) = \frac{1}{1+\exp(-x)}$$

Skip-Gram Training data

Training sentence:

... lemon, a tablespoon of **apricot** jam   a   pinch ...

c1   c2   t   c3   c4

Training data: input/output pairs centering on *apricot*

Assume a +/- 2 word window (in reality: use +/- 10 words)

Positive examples:
(apricot, tablespoon), (apricot, of), (apricot, jam), (apricot, a)

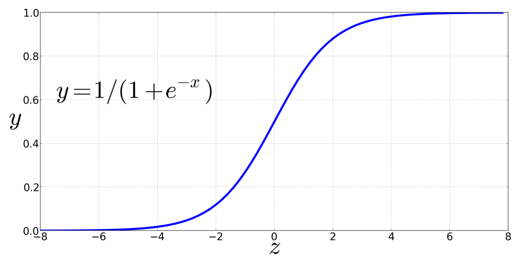
For each positive example, sample k **negative examples**, using noise words (according to [adjusted] unigram probability)

(apricot, aardvark), (apricot, puddle)…

P(Y | **X**) with Logistic Regression

The sigmoid lies between 0 and 1 and is used in (binary) logistic regression

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



Logistic regression for **binary** classification ($y \in \{0,1\}$):

$$P(\,Y = 1 \mid \mathbf{x}\,) = \sigma(\mathbf{wx} + b) = \frac{1}{1 + \exp(\,-(\mathbf{wx} + b))}$$

Parameters to learn: one feature weight vector **w** and one bias term *b*

Back to word2vec

Skipgram with negative sampling also uses the sigmoid,
but requires *two* sets of parameters that are multiplied together
(for target and context vectors)

$$\log \sigma(w_T \cdot w_c) + \sum_{i=1}^{k} E_{w_i \sim P(w)} \left[\log \sigma(-w_T w_i) \right]$$

We can view word2vec as training a binary classifier for the decision whether c is an actual context word for t.

The probability that *c* is a positive (real) context word for *t*:

$P(\mathrm{D} = + \mid t, c)$

The probability that *c* is a negative (sampled) context word for *t*:

$P(\mathrm{D} = - \mid t, c) = 1 - P(\mathrm{D} = + \mid t, c)$

Negative Sampling

$$\log \sigma(w_t \cdot w_c) + \sum_{i=1}^{k} E_{w_i \sim P(w)} \left[\log \sigma(-w_t \cdot w_i) \right]$$

$$= \log\left(\frac{1}{1+\exp(-w_t \cdot w_c)}\right) + \sum_{i=1}^{k} E_{w_i \sim P(w)} \left[\log\left(\frac{1}{1+\exp(w_t \cdot w_i)}\right)\right]$$

$$= \log \frac{1}{1+\exp(-w_t \cdot w_c)} + \sum_{i=1}^{k} E_{w_i \sim P(w)} \left[\log\left(1 - \frac{1}{1+\exp(-w_t \cdot w_i)}\right)\right]$$

$$= \log \boxed{P(D=+ \mid w_c, w_t)} + \sum_{i} E_{w_i \sim P(w)} \left[\log(1 - \boxed{P(D=+ \mid w_i, w_t)} \right]$$

Should be high for actual context words

Should be low for sampled context words

Negative Sampling

Basic idea:

— For each actual **(positive) target-context word pair**, sample ***k*** **negative examples** consisting of the target word and a randomly sampled word.

— Train a model to predict a **high conditional probability for the actual (positive)context words**, and a **low conditional probability for the sampled (negative) context words**.

This can be reformulated as (approximated by) predicting whether a word-context pair comes from the actual (positive) data, or from the sampled (negative) data:

$$\log \sigma(w_T \cdot w_c) + \sum_{i=1}^{k} E_{w_i \sim P(w)} \left[\log \sigma(-w_T w_i) \right]$$

Word2Vec: Negative Sampling

Distinguish "good" (correct) word-context pairs (D=1), from "bad" ones (D=0)

Probabilistic objective:

$P(D = 1 \mid \mathrm{t}, \mathrm{c})$ defined by sigmoid:

$$P(D=1|w,c) = \frac{1}{1+exp(-s(w,c))}$$

$P(D = 0 \mid \mathrm{t}, \mathrm{c}) = 1 - P(D = 0 \mid \mathrm{t}, \mathrm{c})$

$P(D = 1 \mid \mathrm{t}, \mathrm{c})$ should be high when $(\mathrm{t}, \mathrm{c}) \in \mathrm{D}+$, and low when $(\mathrm{t},\mathrm{c}) \in \mathrm{D}-$

Word2Vec: Negative Sampling

Training data: D+ ∪ D-

D+ = actual examples from training data

Where do we get D- from?

Word2Vec: for each good pair (w,c), sample k words and add each w_i as a negative example (w_i,c) to D'

(D' is k times as large as D)

Words can be sampled according to corpus frequency or according to smoothed variant where freq'(w) = $\text{freq}(w)^{0.75}$

(This gives more weight to rare words and performs better)

Word2Vec: Negative Sampling

Training objective:
Maximize log-likelihood of training data D+ ∪ D-:

$$\mathscr{L}(\Theta, D, D') = \sum_{(w,c)\in D} \log P(D=1|w,c) + \sum_{(w,c)\in D'} \log P(D=0|w,c)$$

Skip-Gram with negative sampling

Train a binary classifier that decides whether a target word *t* appears in the context of other words $c_{1..k}$

— **Context**: the set of k words near (surrounding) *t*

— Treat the target word *t* and any word that *actually* appears in its context in a real corpus as **positive** examples

— Treat the target word *t* and *randomly sampled* words that don't appear in its context as **negative** examples

— Train a (variant of a) **binary logistic regression** classifier with two sets of weights (target and context embeddings) to distinguish these cases

— The **weights** of this classifier depend on the **similarity** of t and the words in $c_{1..k}$

Use the target embeddings to represent *t*

The Skip-Gram classifier

Use logistic regression to predict whether the pair (*t*, *c*) (target word *t* and a context word c), is a positive or negative example:

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$

$$\begin{aligned} P(-|t,c) &= 1-P(+|t,c) \\ &= \frac{e^{-t\cdot c}}{1+e^{-t\cdot c}} \end{aligned}$$

Assume that t and c are represented as vectors,
so that their dot product *tc* captures their similarity
To capture the entire context window $c_{1..k}$, assume the words in $c_{1:k}$ are independent (multiply) and take the log:

$$\begin{aligned} P(+|t,c_{1:k}) &= \prod_{i=1}^{k} \frac{1}{1+e^{-t\cdot c_i}} \\ \log P(+|t,c_{1:k}) &= \sum_{i=1}^{k} \log \frac{1}{1+e^{-t\cdot c_i}} \end{aligned}$$

Where do we get vectors t, c from?

Iterative approach (gradient descent):
Assume an initial set of vectors, and then adjust them during training to maximize the probability of the training examples.

Summary: How to learn word2vec (skip-gram) embeddings

For a vocabulary of size V: Start with two sets of V random 300-dimensional vectors as initial embeddings

Train a logistic regression classifier to distinguish words that co-occur in corpus from those that don’t

- Pairs of words that co-occur are positive examples
- Pairs of words that don't co-occur are negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance

Throw away the classifier code and keep the embeddings.

Evaluating embeddings

Compare to human scores on word similarity-type tasks:

WordSim-353 (Finkelstein et al., 2002)
SimLex-999 (Hill et al., 2015)
Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)
TOEFL dataset: *Levied is closest in meaning to: imposed, believed, requested, correlated*

Properties of embeddings

Similarity depends on window size C

C = ±2 The nearest words to *Hogwarts*:

Sunnydale
Evernight

C = ±5 The nearest words to *Hogwarts*:

Dumbledore
Malfoy
halfblood

Vectors “capture concepts”

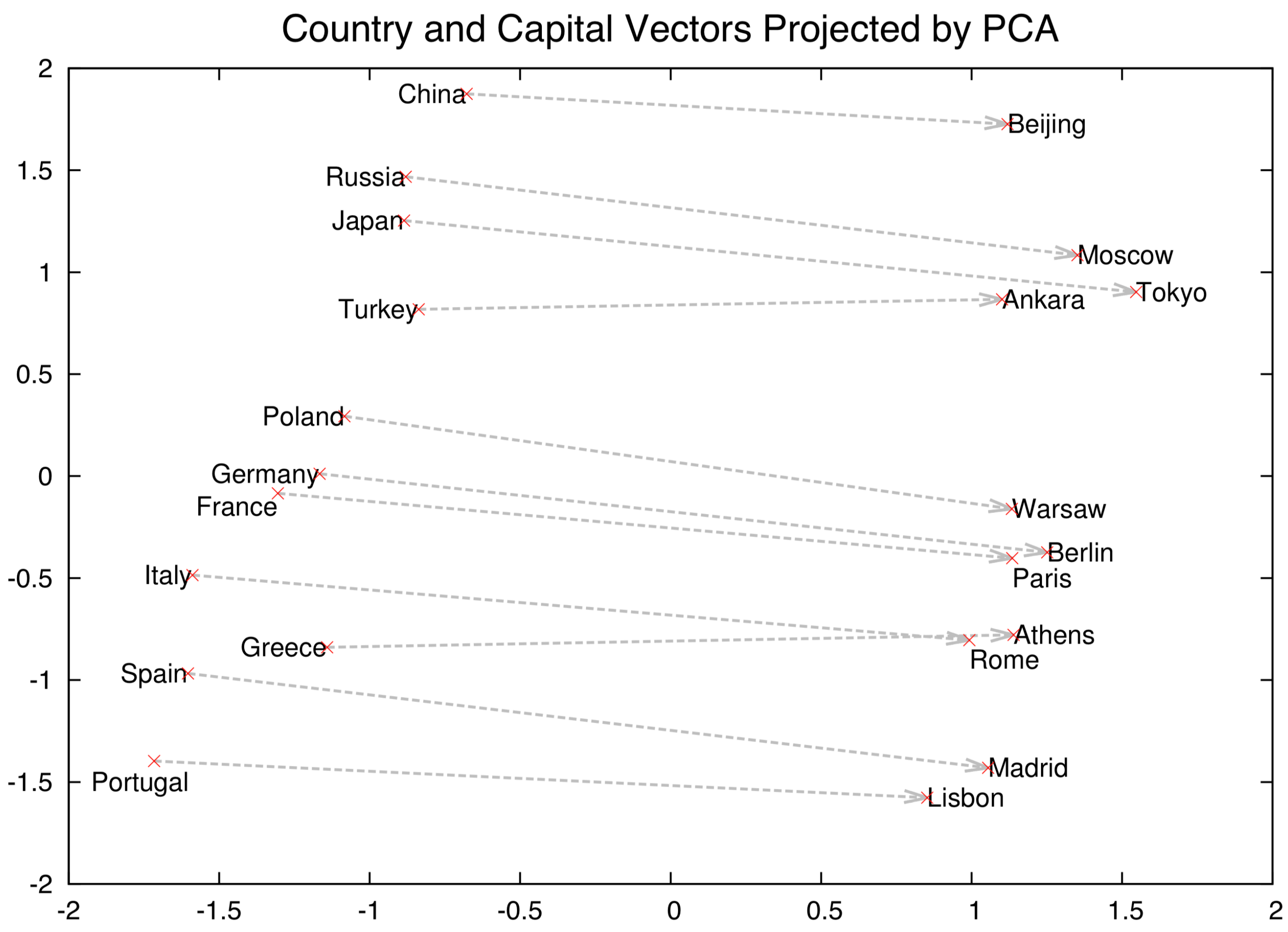


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

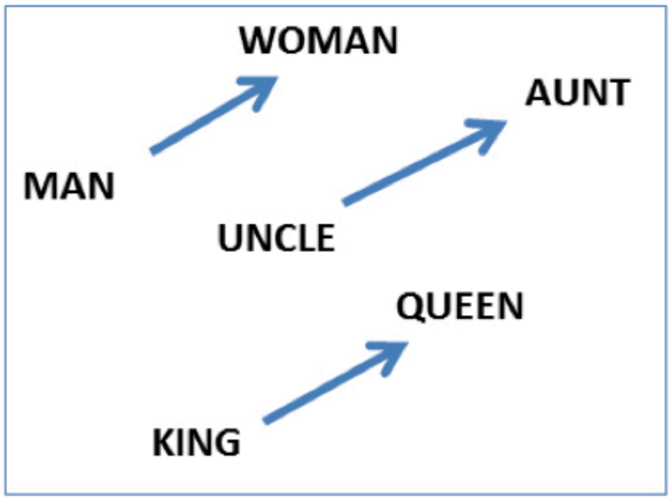
Analogy pairs

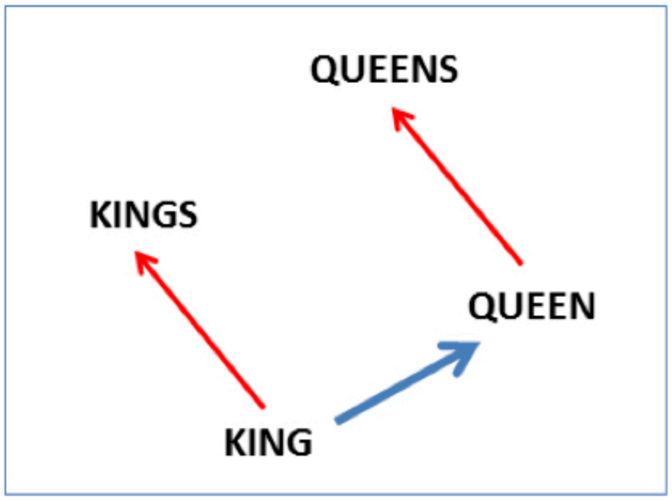
Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	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

Analogy: Embeddings capture relational meaning!

vector(‘*king*’) - vector(‘*man*’) + vector(‘*woman*’) = vector(‘queen’)

vector(‘*Paris*’) - vector(‘*France*’) + vector(‘*Italy*’) = vector(‘*Rome*’)





Word2vec results

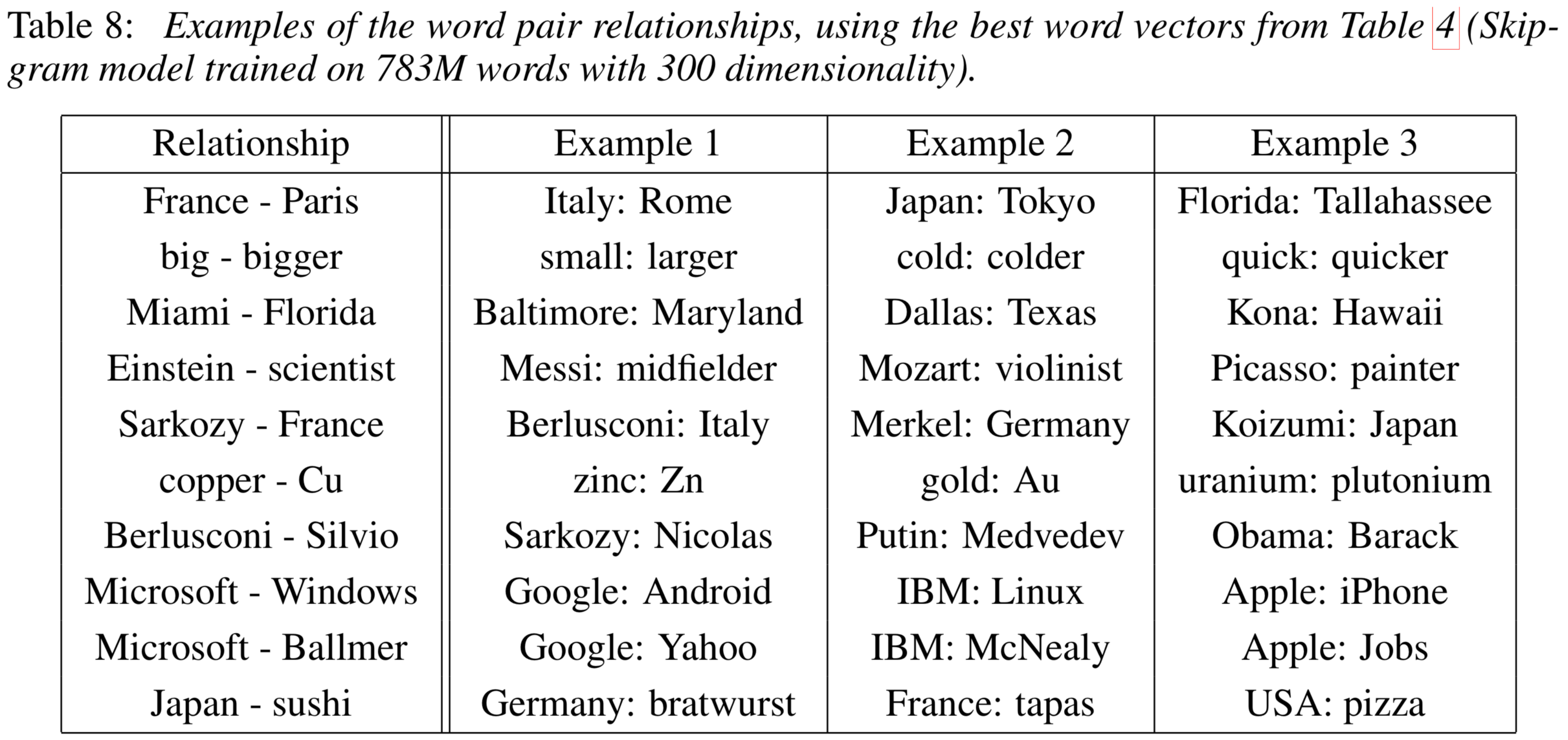


Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Word2Vec and distributional similarities

Why does the word2vec objective yield sensible results?

Levy and Goldberg (NIPS 2014):

Skipgram with negative sampling can be seen as a weighted factorization of a word-context PMI matrix.

=> It is actually very similar to traditional distributional approaches!

Levy, Goldberg and Dagan (TACL 2015) suggest tricks that can be applied to traditional approaches that yield similar results on these lexical tests.

Using Word Embeddings

Using pre-trained embeddings

Assume you have pre-trained embeddings E.
How do you use them in your model?

- Option 1: Adapt E during training

  Disadvantage: only words in training data will be affected.
- Option 2: Keep E fixed, but add another hidden layer that is learned for your task
- Option 3: Learn matrix $T \in \dim(emb) \times \dim(emb)$ and use rows of $E' = ET$ (adapts all embeddings, not specific words)
- Option 4: Keep E fixed, but learn matrix $\Delta \in R^{|V| \times \dim(emb)}$ and use $E' = E + \Delta$ or $E' = ET + \Delta$ (this learns to adapt specific words)

More on embeddings

Embeddings aren’t just for words!

- You can take any discrete input feature (with a fixed number of K outcomes, e.g. POS tags, etc.) and learn an embedding matrix for that feature.

Where do we get the input embeddings from?

- We can learn the embedding matrix during training.
- Initialization matters: use random weights, but in special range (e.g. [-1/(2d), +(1/2d)] for d-dimensional embeddings), or use Xavier initialization
- We can also use pre-trained embeddings
- LM-based embeddings are useful for many NLP task

Dense embeddings you can download!

Word2vec (Mikolov et al.)
https://code.google.com/archive/p/word2vec/
Fasttext http://www.fasttext.cc/

Glove (Pennington, Socher, Manning)
http://nlp.stanford.edu/projects/glove/

Traditional Distributional similarities and PMI

Distributional similarities

Distributional similarities use the set of contexts in which words appear to measure their similarity.

They represent each word *w* as a vector $\mathbf{w}$

$$\mathbf{w} = (w_1, \ldots, w_N) \in \mathbf{R}^N$$

in an N-dimensional vector space.

- Each dimension corresponds to a particular context c_n
- Each element w_n of $\mathbf{w}$ captures the degree to which the word *w* is associated with the context c_n.
- w_n depends on the co-occurrence counts of *w* and c_n

The similarity of words *w* and *u* is given by the similarity of their vectors $\mathbf{w}$ and $\mathbf{u}$

Using nearby words as contexts

- Decide on a fixed vocabulary of N context words $c_1..c_N$
  Context words should occur frequently enough in your corpus that you get reliable co-occurrence counts, but you should ignore words that are too common ('stop words': *a*, *the, on, in, and, or, is, have,* etc.)

- Define what 'nearby' means
  For example: *w* appears near *c if* c appears within ±5 words of *w*

- Get co-occurrence counts of words *w* and contexts *c*

- Define how to transform co-occurrence counts of words *w* and contexts *c* into vector elements w_n
  For example: compute (positive) PMI of words and contexts

- Define how to compute the similarity of word vectors
  For example: use the cosine of their angles.

Defining and counting co-occurrence

Defining co-occurrences:

- **Within a fixed window**: v_i occurs within ±n words of w
- **Within the same sentence:** requires sentence boundaries
- **By grammatical relations**:
  v_i occurs as a subject/object/modifier/… of verb w
  (requires parsing - and separate features for each relation)

Counting co-occurrences:

- f_i as **binary features** (1,0): w does/does not occur with v_i
- f_i as **frequencies**: w occurs n times with v_i
- f_i as **probabilities**:
  e.g. f_i is the probability that v_i is the subject of w.

Getting co-occurrence counts

Co-occurrence as a binary feature:

Does word w ever appear in the context c? (1 = yes/0 = no)

	arts	boil	data	function	large	sugar	water
apricot	0	1	0	0	1	1	1
pineapple	0	1	0	0	1	1	1
digital	0	0	1	1	1	0	0
information	0	0	1	1	1	0	0

Co-occurrence as a frequency count:

How often does word w appear in the context c? (0…n times)

	arts	boil	data	function	large	sugar	water
apricot	0	1	0	0	5	2	7
pineapple	0	2	0	0	10	8	5
digital	0	0	31	8	20	0	0
information	0	0	35	23	5	0	0

Typically: 10K-100K dimensions (contexts), very sparse vectors

Counts vs PMI

Sometimes, low co-occurrences counts are very informative, and high co-occurrence counts are not:

- Any word is going to have relatively high co-occurrence counts with very common contexts (e.g. “it”, “anything”, “is”, etc.), but this won’t tell us much about what that word means.
- We need to identify when co-occurrence counts are more likely than we would expect by chance.

We therefore want to use PMI values instead of raw frequency counts:

$$PMI(w, c) = \log \frac{p(w, c)}{p(w)p(c)}$$

But this requires us to define $p(\mathrm{w}, \mathrm{c})$, $p(\mathrm{w})$ and $p(\mathrm{c})$

Pointwise mutual information (PMI)

Recall that two **events** x, y are **independent** if their joint probability is equal to the product of their individual probabilities:

x,y are independent iff $p(x,y) = p(x)p(y)$

x,y are independent iff $p(x,y) / p(x)p(y) = 1$

In NLP, we often use the pointwise mutual information (PMI) of two outcomes/events (e.g. words):

$$PMI(x,y) = \log \frac{p(X=x, Y=y)}{p(X=x)p(Y=y)}$$

Positive Pointwise Mutual Information

PMI is negative when words co-occur less than expected by chance.

This is unreliable without huge corpora:

With $P(w_1) \approx P(w_2) \approx 10^{-6}$, we can't estimate whether $P(w_1,w_2)$ is significantly different from 10^{-12}

We often just use positive PMI values,
and replace all PMI values < 0 with 0:

Positive Pointwise Mutual Information (PPMI):

$$\mathrm{PPMI}(w,c) = \begin{cases} \mathrm{PMI}(w,c) & \text{if } \mathrm{PMI}(w,c) > 0 \\ 0 & \text{if } \mathrm{PMI}(w,c) \leq 0 \end{cases}$$

Frequencies vs. PMI

Objects of ‘drink’ *(Lin, 1998)*

	Count	**PMI**
bunch beer	2	12.34
tea	2	11.75
liquid	2	10.53
champagne	4	11.75
anything	3	5.15
it	3	1.25