

EE-608: Deep Learning For Natural Language Processing

James Henderson



DLNLP, Lecture 1

Outline

Course Introduction

Course Overview

Word Embeddings

Outline

Course Introduction

Course Overview

Word Embeddings

EE-608: Deep Learning For Natural Language Processing

Lecturer: James Henderson

TAs: Melika Behjati, Andrei Coman, and Fabio Fehr

Emails: *first.last@idiap.ch*

Lectures: Wednesdays 10:15–12:00, DIA003

Slides will be uploaded before each lecture.

Exercises: Wednesdays 13:15–15:00, DIA003

(starting next week)

Exercises will be distributed at least a week before they are discussed.

Course website:

<https://moodle.epfl.ch/course/view.php?id=16062>

Exercises

No Exercise session today. An exercise will be distributed today for next week.

- ▶ We will be using PyTorch
- ▶ Exercise sessions for an introduction to PyTorch
- ▶ Exercise sessions for training and running existing models for important NLP tasks
- ▶ Exercise sessions for help with projects

Evaluation

We will do evaluation with a course project

- ▶ Started around midway through the course
- ▶ Teams of 1-3 people
- ▶ Written report on the project (60% of grade)
- ▶ Oral presentation and answers to questions (40% of grade)

What do we plan to teach?

Provide an overview of neural network methods applied to text

- ▶ An understanding of the basic properties of human language
 - ▶ structured, unbounded
 - ▶ categorical and continuous, large vocabulary
- ▶ An understanding of the deep learning methods commonly used for language
 - ▶ word embeddings
 - ▶ RNN, LSTM, Seg2Seq
 - ▶ attention, Transformers
 - ▶ multi-task learning, pretraining, BERT, chatGPT
- ▶ The ability to build systems for some NLP tasks (in PyTorch)
 - ▶ Language modelling
 - ▶ Machine translation
 - ▶ Syntactic parsing
 - ▶ Natural Language Inference

This course is continually changing

- ▶ Based on the Stanford course “Natural Language Processing with Deep Learning” by C. Manning
- ▶ See <http://web.stanford.edu/class/cs224n/> for further reading and some alternative topics
- ▶ We **need student feedback** to help develop the course

Expected Student Background

- ▶ What are you studying?
- ▶ What topics do you already know?
 - ▶ Calculus, linear algebra
 - ▶ Probability and statistics
 - ▶ Basic machine learning
 - ▶ Neural networks
 - ▶ Natural language processing
- ▶ What programming experience do you have?
 - ▶ Python, NumPy
 - ▶ PyTorch

Outline

Course Introduction

Course Overview

Word Embeddings

Deep Learning is Representation Learning

The most important property of deep learning models is their ability to learn their own latent representations.

- ▶ vector representations embed features and categories (and possibly more) in a low-dimensional continuous vector space
- ▶ bag-of-vectors representations can also embed relationships between vectors, and can be arbitrarily large
- ▶ sequence-of-vectors representations add an explicit ordering on vectors
- ▶ graphs-of-vectors add explicit graph relations between vectors

Deep Learning is Representation Learning

Backpropagation training allows both the encoding into a representation and the decoding from a representation to be trained jointly. Any encoding and decoding functions can be used, provided they are differentiable.

- ▶ multi-layered perceptrons
- ▶ resnets
- ▶ attention functions
- ▶ ...

When the decoding for one learned representation is the encoding of another, it is a *deep* model.

Deep Learning For Natural Language Processing

Models	Properties	NLP Tasks
word embeddings	continuous space, semantic similarity	information access
RNN, LSTM	unbounded sequences, induced state	language modelling
BiLSTM with pooling	unbounded input	text classification
Seq2Seq	unbounded output, end-to-end training	machine translation
attention	unbounded memory, content-based access	machine translation
Transformer	bag-of-vector representa- tions, deep	machine translation

Deep Learning For Natural Language Processing

Models	Properties	NLP Tasks
NN transition-based parsing	structured prediction, unfactorised	syntactic parsing
NN graph-based parsing	structured prediction, conditionally factorised	syntactic parsing
language-model pretraining	representation learning	many tasks
BERT, GPTn, chatGPT	transfer learning	many tasks
...

Outline

Course Introduction

Course Overview

Word Embeddings

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.

Commonest linguistic way of thinking of meaning:

signifier (symbol) \Leftrightarrow signified (idea or thing)

= denotational semantics

tree $\Leftrightarrow \{ \text{$

How do we have usable meaning in a computer?

Previously commonest NLP solution: Use, e.g., **WordNet**, a thesaurus containing lists of **synonym sets** and **hypercnyms** (“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., *hypercnyms of “panda”*:

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(pandaclosure(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

- A useful resource but missing nuance:
 - e.g., “proficient” is listed as a synonym for “good”
This is only correct in some contexts
 - Also, WordNet lists offensive synonyms in some synonym sets without any coverage of the connotations or appropriateness of words
- 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
- Can’t be used to accurately compute word similarity (see following slides)

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel – a **localist** representation

Means one 1, the rest 0s

Such symbols for words can be represented by **one-hot** vectors:

motel = [0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0]

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

Problem with words as discrete symbols

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

But:

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

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: 11)
 - 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).
- We 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**

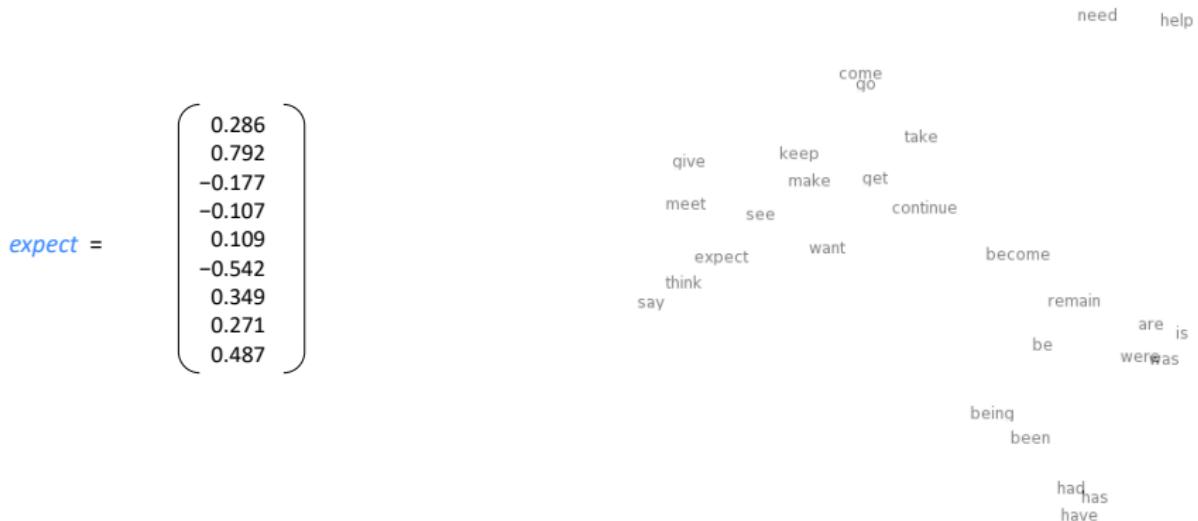
Word vectors

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector **dot** (scalar) **product**

$$\begin{aligned} \text{banking} &= \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} & \text{monetary} &= \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix} \end{aligned}$$

Note: word vectors are also called (word) embeddings or (neural) word representations
They are a distributed representation

Word meaning as a neural word vector – visualization



3. Word2vec: Overview

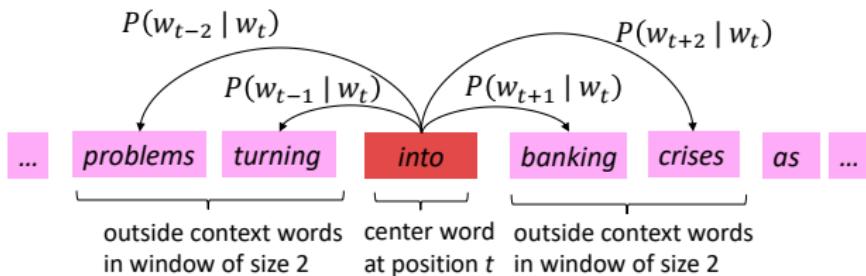
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus (“body”) of text: a long list of words
- 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

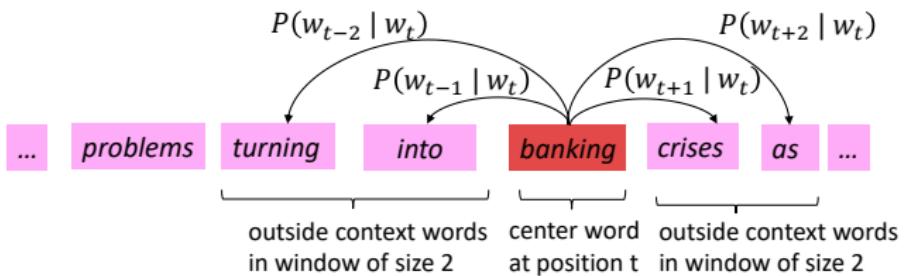
Word2Vec Overview

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: objective function

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

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

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

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

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

Minimizing objective function \Leftrightarrow Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

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

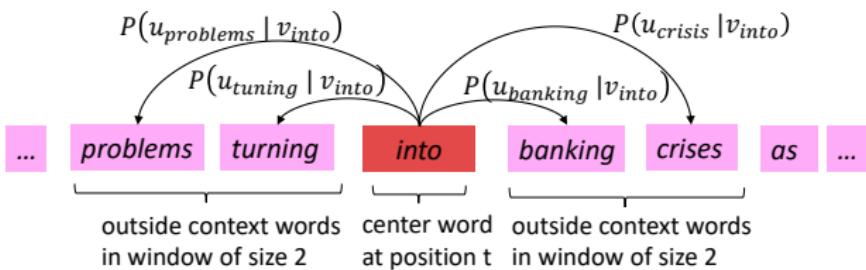
- **Question:** How to calculate $P(w_{t+j} | w_t; \theta)$?
- **Answer:** We will use two vectors per word w :
 - v_w when w is a center word
 - u_w when w is a context word
- Then for a center word c and a context word o :

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

Word2Vec with Vectors

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

All words vectors θ
appear in denominator



Word2vec: prediction function

② Exponentiation makes anything positive

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

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

③ Normalize over entire vocabulary to give probability distribution

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

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

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

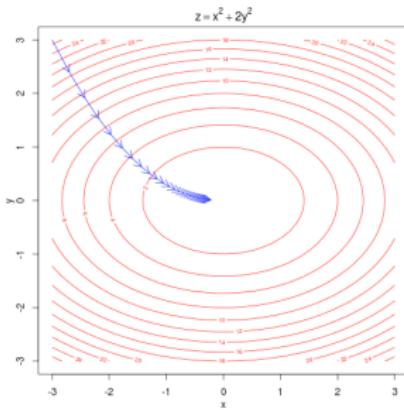
But sort of a weird name because it returns a distribution!

To train the model: Optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with d -dimensional vectors and V -many words, we have \rightarrow
- Remember: every word has two vectors

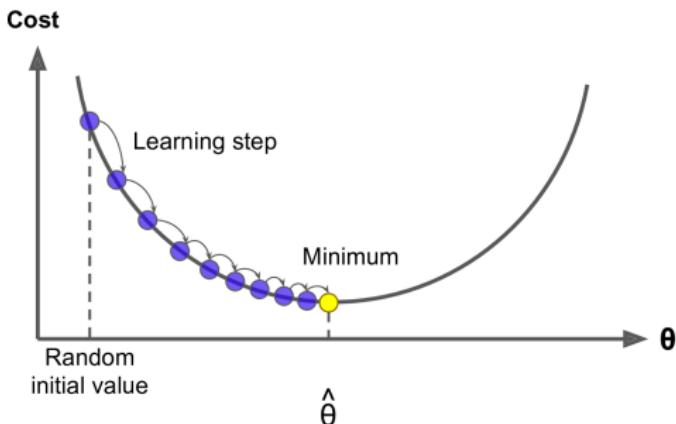
$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute **all** vector gradients!

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 ☹

But life turns out to be okay ☺

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:

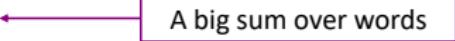
```
while True:  
    theta_grad = evaluate_gradient(J,corpus,theta)  
    theta = theta - alpha * theta_grad
```

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
```

The skip-gram model with negative sampling (HW2)

- The normalization term is computationally expensive (when many output classes):
- $$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$


A big sum over words
- Hence, in standard word2vec and HW2 you implement the skip-gram model with **negative sampling**
- Main idea: train binary logistic regressions to differentiate a true pair (center word and a word in its context window) versus several “noise” pairs (the center word paired with a random word)

Word2vec algorithm family (Mikolov et al. 2013): More details

Why two vectors? → Easier optimization. Average both at the end

- But can implement the algorithm with just one vector per word ... and it helps a bit

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

We presented: **Skip-gram model**

Loss functions for training:

1. Naïve softmax (simple but expensive loss function, when many output classes)
2. More optimized variants like hierarchical softmax
3. Negative sampling

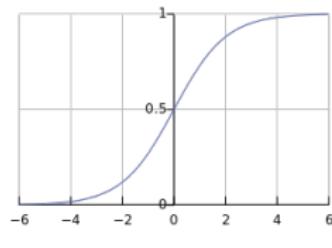
So far, we explained **naïve softmax**

The skip-gram model with negative sampling (HW2)

- Introduced in: “Distributed Representations of Words and Phrases and their Compositionality” (Mikolov et al. 2013)
- Overall objective function (they maximize):
$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J_t(\theta)$$

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$$

- The logistic/sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$
(we'll become good friends soon)
- We maximize the probability of two words co-occurring in first log and minimize probability of noise words in second part

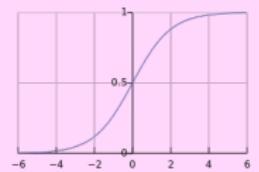


The skip-gram model with negative sampling (HW2)

- Using notation consistent with this class and HW2:

$$J_{\text{neg-sample}}(\mathbf{u}_o, \mathbf{v}_c, U) = -\log \sigma(\mathbf{u}_o^T \mathbf{v}_c) - \sum_{k \in \{K \text{ sampled indices}\}} \log \sigma(\mathbf{u}_k^T \mathbf{v}_c)$$

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



- We take k negative samples (using word probabilities)
- Maximize probability that real outside word appears;
minimize probability that random words appear around center word
- Sample with $P(w) = U(w)^{3/4}/Z$, the unigram distribution $U(w)$ raised to the $3/4$ power
(We provide this function in the starter code).
- The power makes less frequent words be sampled more often

Stochastic gradients with negative sampling [aside]

- We iteratively take gradients at each window for SGD
- In each window, we only have at most $2m + 1$ words plus $2km$ negative words with negative sampling, so $\nabla_{\theta}J_t(\theta)$ is very sparse!

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

Stochastic gradients with negative sampling [aside]

- 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

Rows not columns
in actual DL
packages!

$$|V| \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}^d$$

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

This is also a particular issue with more advanced optimization methods in the Adagrad family

3. Why not capture co-occurrence counts directly?

There's something weird about iterating through the whole corpus (perhaps many times); why don't we just accumulate all the statistics of what words appear near each other?!?

Building a co-occurrence matrix X

- 2 options: windows vs. full document
- Window: Similar to word2vec, use window around each word → captures some syntactic and semantic information ("word space")
- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to "Latent Semantic Analysis" ("document space")

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

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

Co-occurrence vectors

- Simple count co-occurrence vectors
 - Vectors increase in size with vocabulary
 - Very high dimensional: require a lot of storage (though sparse)
 - Subsequent classification models have sparsity issues → Models are less robust
- 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?

Classic 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

$$\underbrace{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}}_{X^k} = \underbrace{\begin{bmatrix} * & * \\ * & * \\ * & * \end{bmatrix}}_U \underbrace{\begin{bmatrix} \bullet & & \\ & \bullet & \\ & & \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}}_{V^T}$$

Retain only k singular values, in order to generalize.

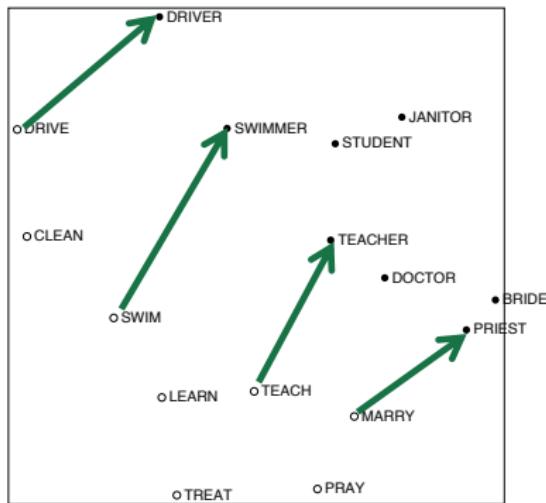
\hat{X} is the best rank k approximation to X , in terms of least squares.

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

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

- Running an SVD on raw counts doesn't work well!!!
- 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:
 - log the frequencies
 - $\min(X, t)$, with $t \approx 100$
 - Ignore the function words
- Ramped windows that count closer words more than further away words
- Use Pearson correlations instead of counts, then set negative values to 0
- Etc.

Interesting semantic patterns emerge in the scaled vectors



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

GloVe [Pennington, Socher, and Manning, EMNLP 2014]: Encoding meaning components in vector differences



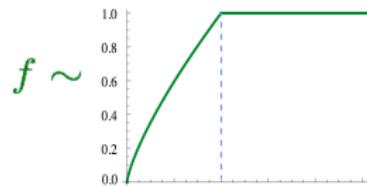
Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

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

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

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

- Fast training
- Scalable to huge corpora



4. 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!

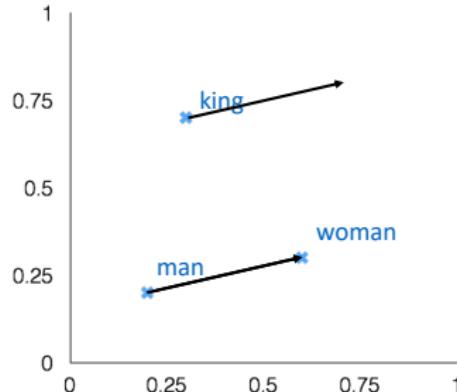
Intrinsic word vector evaluation

- Word Vector Analogies

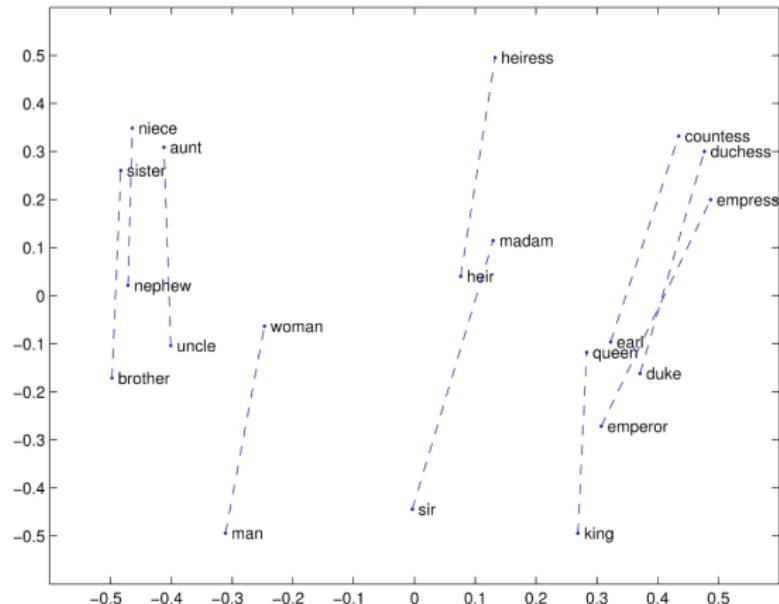
$$\boxed{a:b :: c: ?} \quad \longrightarrow \quad \boxed{\text{man:woman :: king:?}}$$

- 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?

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



GloVe Visualization



Meaning similarity: 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

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	<u>72.7</u>	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	<u>58.1</u>	37.2
GloVe	6B	<u>65.8</u>	<u>72.7</u>	<u>77.8</u>	53.9	<u>38.1</u>
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., average both vectors)

Extrinsic word vector evaluation

- One example where good word vectors should help directly: **named entity recognition**: identifying references to a person, organization or location: [Chris Manning](#) lives in [Palo Alto](#).

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

- Subsequent NLP tasks in this class are other examples. So, more examples soon.

5. Word senses and word sense ambiguity

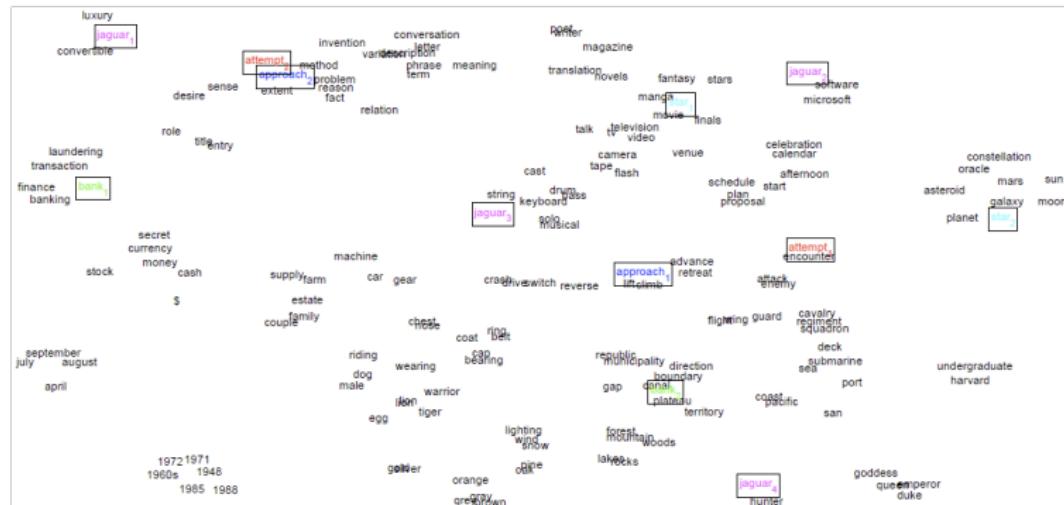
- Most words have lots of meanings!
 - Especially common words
 - Especially words that have existed for a long time
- Example: **pike**
- Does one vector capture all these meanings or do we have a mess?

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!*

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.



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

Summary of Word Embeddings (part1)

- ▶ Similarity of word meanings is a continuous, multi-dimensional phenomenon which is hard to define
- ▶ This is addressed by embedding words in a continuous vector space, called “word embeddings”, and using the dot product to predict word similarity
- ▶ Word embeddings can be learned automatically by training them to predict word cooccurrence in (very) large corpora
- ▶ Words are similar if they have similar distributions of context words

Summary of Word Embeddings (part 2)

- ▶ In fact, word embeddings are low dimensional factorisations of a word cooccurrence matrix
- ▶ Different transforms of the matrix and different optimisers give different word embeddings, but they all perform similarly
- ▶ Intrinsic evaluations measure useful properties in word embeddings (e.g. similarity as dot product, analogy as vector difference)
- ▶ Extrinsic evaluations test embeddings for their usefulness in other tasks
- ▶ Some aspects of word ambiguity are better modelled as discrete