Word Vectors [word2vec]

LING 575K Deep Learning for NLP
Shane Steinert-Threlkeld
April 4 2022

Announcements

- Thanks for all the discussion on Canvas :)
- File / env permissions: thanks to everyone for pointing them out! Let me know if any others persist (user error on recursive application)
- max_size: not including special tokens like <unk> (cf lines 43-44 in vocabulary.py)
- Python 3.9:
  - The code makes heavy use of type hinting
    - Improves readability / future-proofing
    - Works well with code completers, static type checkers like mypy
  - Including *native* type hinting for many data structures, which is new to 3.9
  - So be sure to run in the environment we provide, which includes 3.9

Announcements

- Developing / testing locally vs on patas:
  - We grade on patas, so your code *must* run there (and you should check that it does before submitting)
  - Developing on patas:
    - Using vim, emacs, …
    - Some editors allow remote editing, e.g. SSH Plugin for VSCode
  - Developing locally:
    - Download `environment.yml` from our dropbox, and `conda env create --file environment.yml`
    - This should give you a local environment that matches the one we provide on patas
    - NB: we (teaching staff) have not tested this, so consider it alpha/beta :)

Beware of Frequency!

source

Today's Plan

- Last time:
  - Loss minimization
  - Gradient descent
  - Why word vectors
- Today:
  - Count-based word vectors [briefly]
  - Prediction-based word vectors
  - In particular: *skip-gram with negative sampling*
  - Two tasks:
    - Language modeling
    - Sentiment analysis

Prediction-Based Models [Word2Vec]

Prediction-based Embeddings

- ***Skip-gram*** and ***Continuous Bag of Words*** (CBOW) models

Prediction-based Embeddings

- ***Skip-gram*** and ***Continuous Bag of Words*** (CBOW) models
- Intuition:
  - Words with similar meanings share similar contexts
  - Instead of counting:
    - Train models to learn to *predict* context words
  - Models train embeddings that make current word more like nearby words and less like distance words

Embeddings: Skip-Gram vs. Continuous Bag of Words

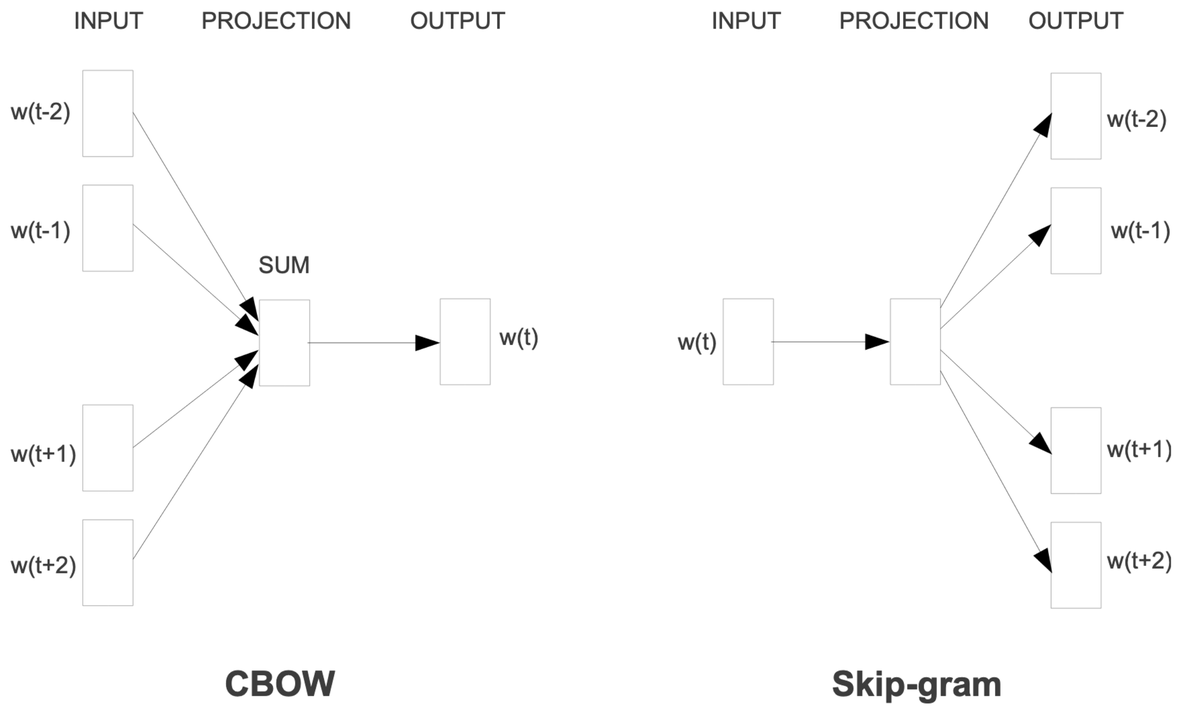
- Continuous Bag of Words (CBOW):
  - $P(word|context)$
  - Input: $(w_{t-1},\ w_{t-2},\ w_{t+1},\ wt_{+2}\ ...)$
  - Output: $p(w_t)$

Embeddings: Skip-Gram vs. Continuous Bag of Words

- Continuous Bag of Words (CBOW):
  - $P(word|context)$
  - Input: $(w_{t-1},\ w_{t-2},\ w_{t+1},\ wt_{+2}\ ...)$
  - Output: $p(w_t)$
- **Skip-gram:**
  - $P(context|word)$
  - Input: w_t
  - Output: $p(w_{t-1},\ w_{t-2},\ w_{t+1},\ w_{t+2}\ ...)$

Embeddings: Skip-Gram vs. Continuous Bag of Words

- Continuous Bag of Words (CBOW):
  - $P(word \mid context)$
  - Input: $(w_{t-1},\ w_{t-2},\ w_{t+1},\ wt_{+2}\ \ldots)$
  - Output: $p(w_t)$
- **Skip-gram:**
  - $P(context \mid word)$
  - Input: w_t
  - Output: $p(w_{t-1},\ w_{t-2},\ w_{t+1},\ w_{t+2}\ \ldots)$



Mikolov et al 2013a (the OG `word2vec` paper)

Skip-Gram Model

- Learns two embeddings
  - W : word, matrix of shape [vocab_size, embedding_dimension]
  - C : context embedding, matrix of same shape

$$p(w_k \mid w_j) = \frac{e^{\mathbf{C}_k \cdot \mathbf{W}_j}}{\sum_i e^{\mathbf{C}_i \cdot \mathbf{W}_j}}$$

Skip-Gram Model

- Learns two embeddings
  - W : word, matrix of shape [vocab_size, embedding_dimension]
  - C : context embedding, matrix of same shape
- Prediction task:
  - Given a word, predict each neighbor word in window
  - Compute $p(w_k|w_j)$ as proportional to $\boldsymbol{c_k} \cdot \boldsymbol{w_j}$
    - For each context position
  - Convert to probability via softmax

$$p(w_k \mid w_j) = \frac{e^{\mathbf{C}_k \cdot \mathbf{W}_j}}{\sum_i e^{\mathbf{C}_i \cdot \mathbf{W}_j}}$$

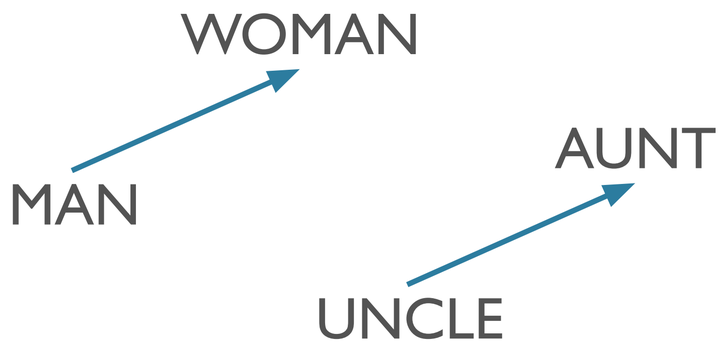
Parameters and Hyper-parameters

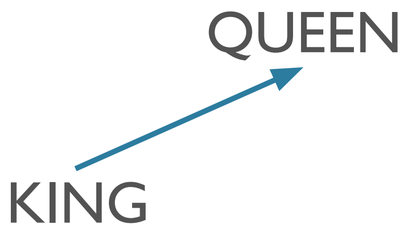
- The embedding dimension is a *hyper-parameter*
  - Chosen by the modeler / practitioner
  - Not updated during the course of learning / training
  - Other examples we've seen so far:
    - Learning rate for SGD
  - Will talk more about how to choose hyper-parameters later
- Parameters: parts of the model that are updated by the learning algorithm

Power of Prediction-based Embeddings

- Count-based embeddings:
  - Very high-dimensional (|V|)
  - Sparse
  - Pro: features are interpretable [“occurred with word W N times in corpus”]
- Prediction-based embeddings:
  - “Low”-dimensional (typically ~300-1200)
  - Dense
  - Con: features are not immediately interpretable
    - i.e. what does “dimension 36 has value -9.63” mean?

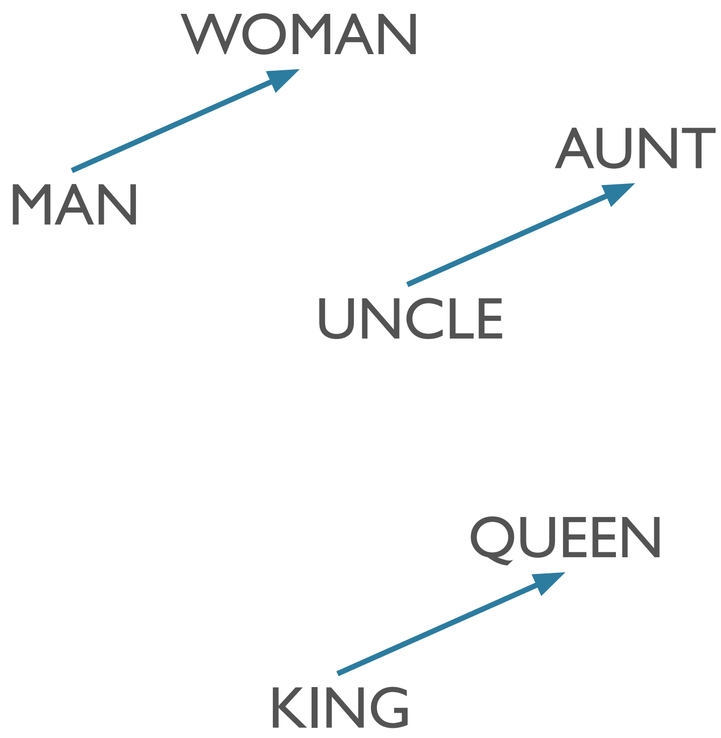
Relationships via Offsets

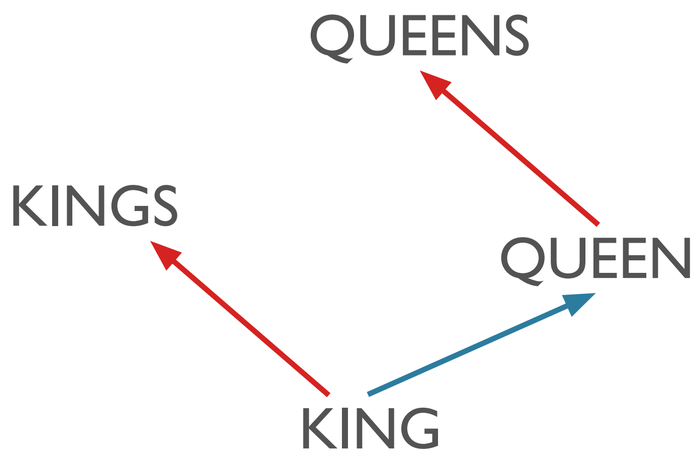




Mikolov et al 2013b

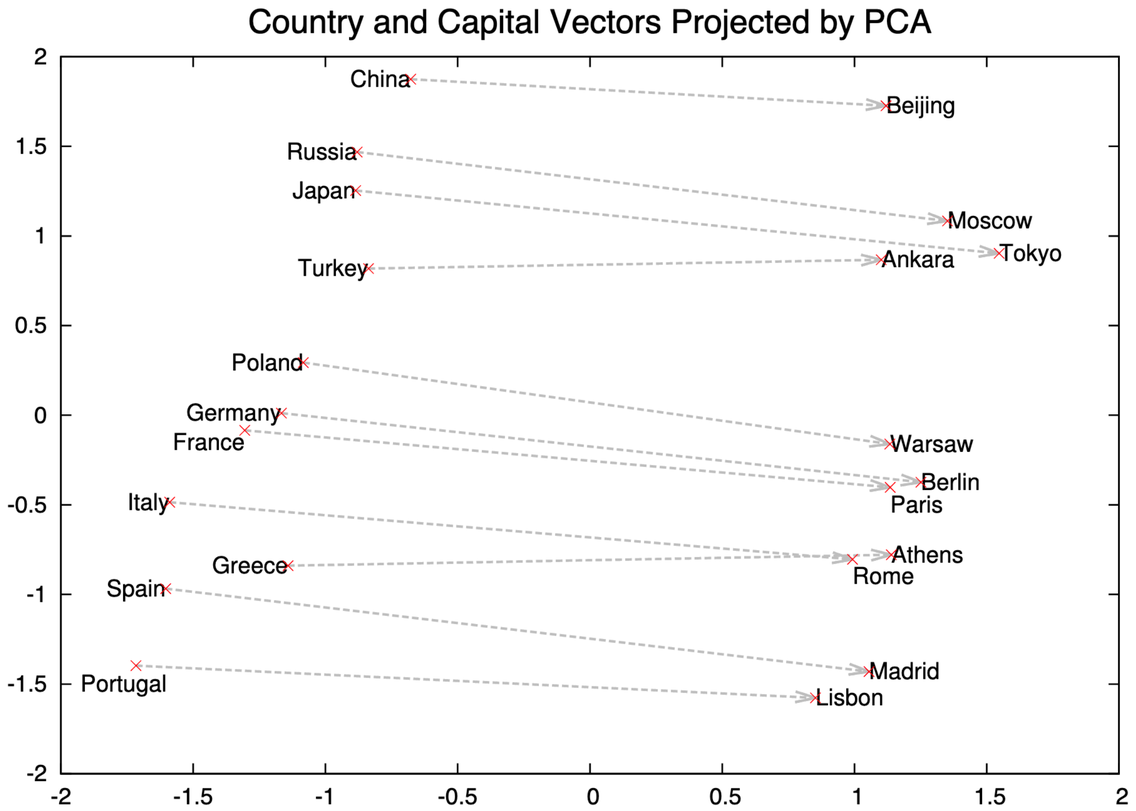
Relationships via Offsets





Mikolov et al 2013b

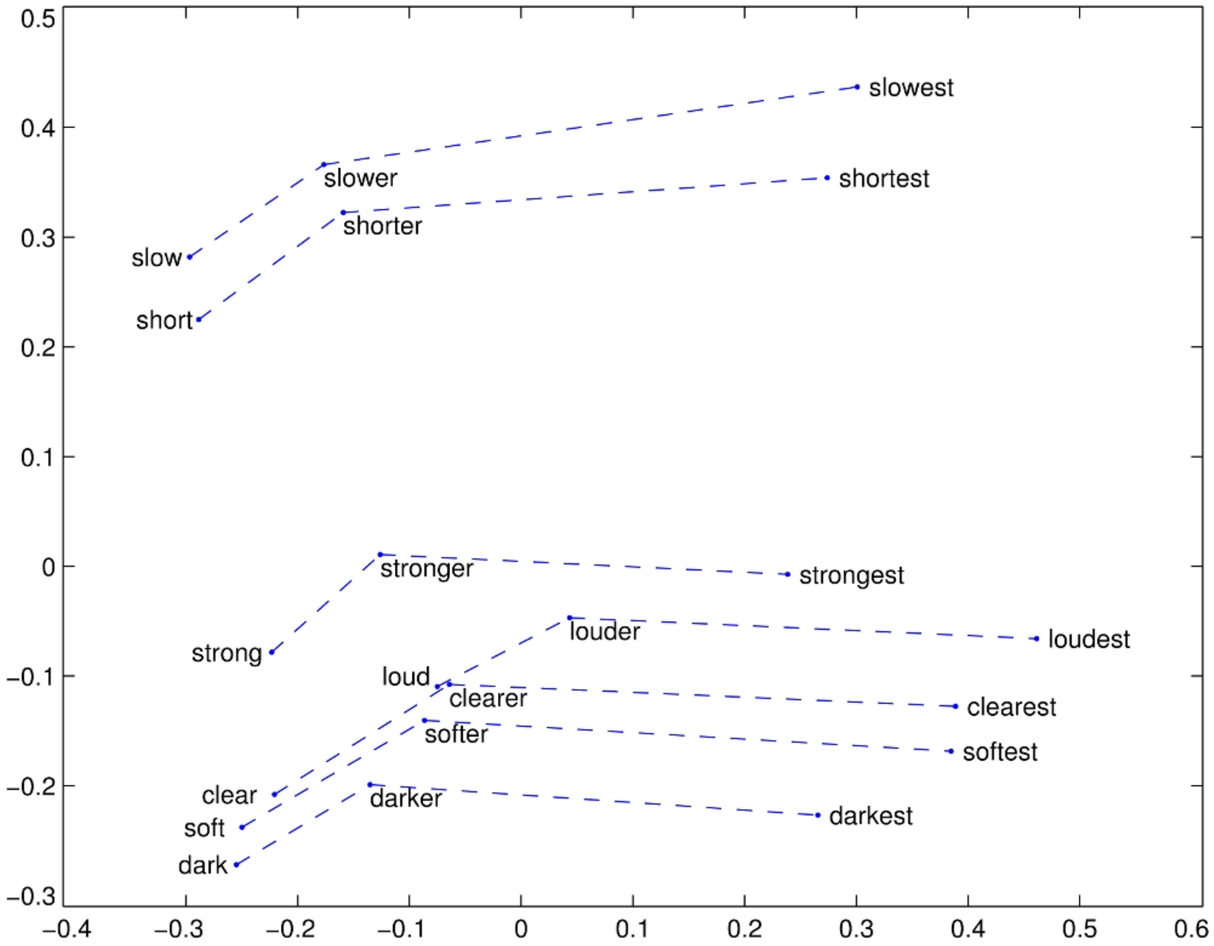
One More Example



Mikolov et al 2013c

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.

One More Example



Caveat Emptor

Issues in evaluating semantic spaces using word analogies

Tal Linzen
LSCP & IJN
École Normale Supérieure
PSL Research University
tal.linzen@ens.fr

Abstract

The offset method for solving word analogies has become a standard evaluation tool for vector-space semantic models: it is considered desirable for a space to represent semantic relations as consistent vector offsets. We show that the method's reliance on cosine similarity conflates offset consistency with largely irrelevant neighborhood structure, and propose simple baselines that should be used to improve the utility of the method in vector space evaluation.

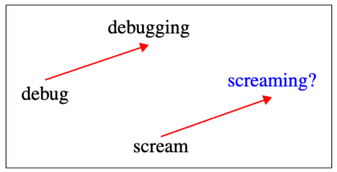


Figure 1: Using the vector offset method to solve the analogy task (Mikolov et al., 2013c).

cosine similarity to the landing point. Formally, if the analogy is given by

$$a : a^* :: b : \_\_\_ \quad (1)$$

Linzen 2016, a.o.

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Tolga Bolukbasi[1], **Kai-Wei Chang**[2], **James Zou**[2], **Venkatesh Saligrama**[1,2], **Adam Kalai**[2]
[1]Boston University, 8 Saint Mary's Street, Boston, MA
[2]Microsoft Research New England, 1 Memorial Drive, Cambridge, MA
tolgab@bu.edu, kw@kwchang.net, jamesyzou@gmail.com, srv@bu.edu, adam.kalai@microsoft.com

Abstract

The blind application of machine learning runs the risk of amplifying biases present in data. Such a danger is facing us with *word embedding*, a popular framework to represent text data as vectors which has been used in many machine learning and natural language processing tasks. We show that even word embeddings trained on Google News articles exhibit female/male gender stereotypes to a disturbing extent.

Bolukbasi et al 2016

Skip-Gram with Negative Sampling (SGNS)

Training The Skip-Gram Model

- Issue:
  - Denominator computation is very expensive

$$p(w_k \mid w_j) = \frac{\mathbf{C}_k \cdot \mathbf{W}_j}{\sum_i \mathbf{C}_i \cdot \mathbf{W}_j}$$

- Strategy:
  - Approximate by *negative sampling* (efficient approximation to Noise Contrastive Estimation):
    - + example: true context word
    - – example: k other words, randomly sampled

Negative Sampling, Idea

- Skip-Gram:
  - $P(w_k \mid w_j)$: what is the probability that w_k occurred in the context of w_j
  - Classifier with |V| classes
- Negative sampling:
  - $P(+ \mid w_k, w_j)$: what is the probability that (w_k, w_j) was a true co-occurrence?
  - $P(- \mid w_k, w_j) = 1 - P(+ \mid w_k, w_j)$
    - Probability that (w_k, w_j) was *not* a true co-occurrence
    - Examples of "fake" co-occurrences = *negative samples*
  - Binary classifier

Generating Positive Examples

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1         c2    w     c3       c4
```

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Generating Positive Examples

- Iterate through the corpus. For each word: add all words within a *window_size* of the current word as a positive pair.

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1         c2    w     c3       c4
```

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Generating Positive Examples

- Iterate through the corpus. For each word: add all words within a *window_size* of the current word as a positive pair.
  - NB: *window_size* is a hyper-parameter

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1         c2    w     c3       c4
```

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Negative Samples

- For each positive (w, c) sample, generate *num_negatives* samples
  - (w, c'), where c' is different from c
  - NB: *num_negatives* is another hyper-parameter

negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Negative Samples, up-weighting

- It's also common to "upsample" less frequent words
- Instead of sampling from raw frequencies from the corpus, raise them to a power to "flatten" the distribution

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

The Data, Summary

- X = pairs of words
- Y = {0, 1}
  - 1 = + (positive example), 0 = - (negative example)
- Example (x, y) pairs:
  - (("apricot", "tablespoon"), 1)
  - (("apricot", "jam"), 1)
  - (("apricot", "aardvark"), 0)
  - (("apricot", "my"), 0)

The Model

- So what is $P(1 \mid w, c)$ (more specifically, $P(1 \mid w, c; \theta)$)?
- As before, learns two embeddings
  - E: word, matrix of shape [vocab_size, embedding_dimension]
    - E_w: embedding for word *w* [row of the matrix]
  - C: context embedding, matrix of same shape

The Model

$$P(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

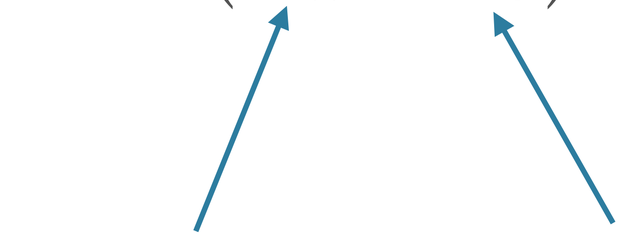
The Model

$$P(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

Target word embedding

The Model

$$P(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

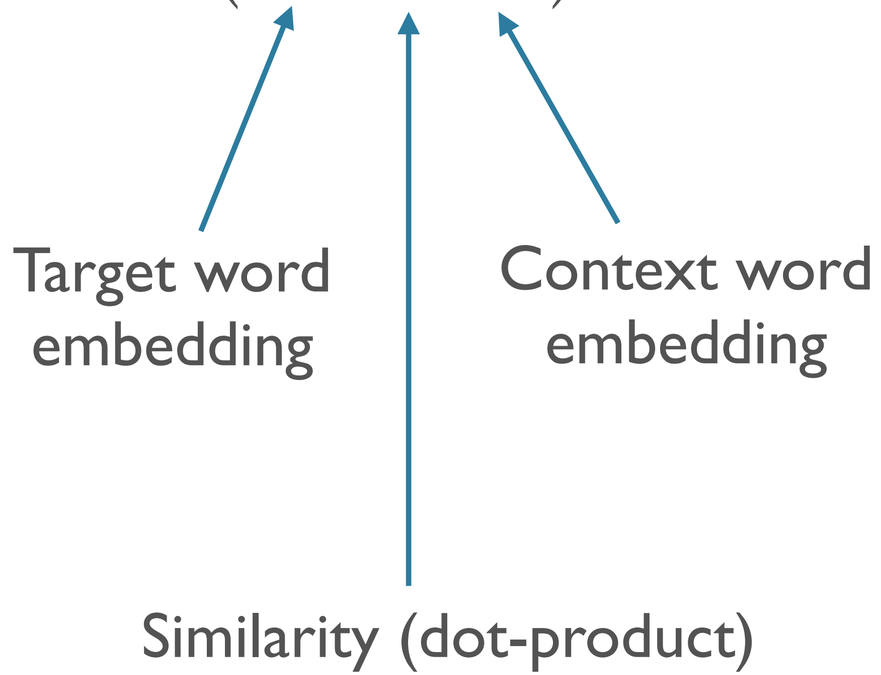


Target word embedding

Context word embedding

The Model

$$P(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

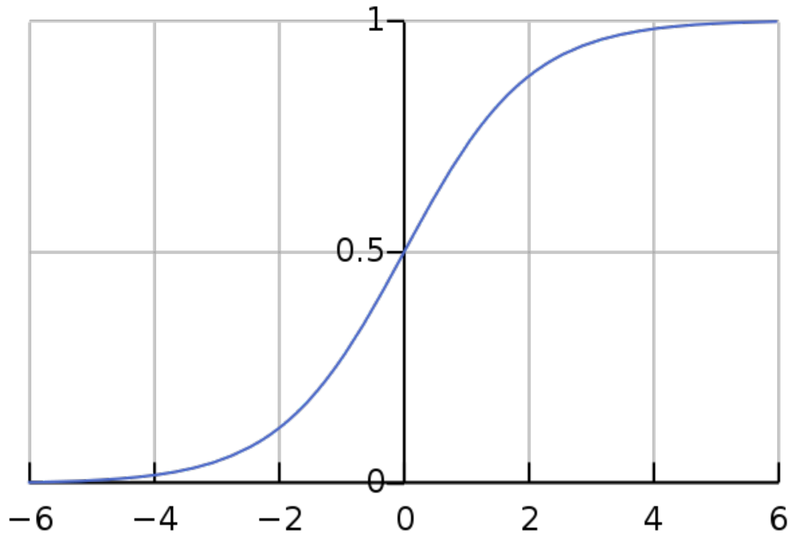


The Model

$$P(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

sigmoid

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



Target word embedding

Context word embedding

Similarity (dot-product)

The Model

$$P(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

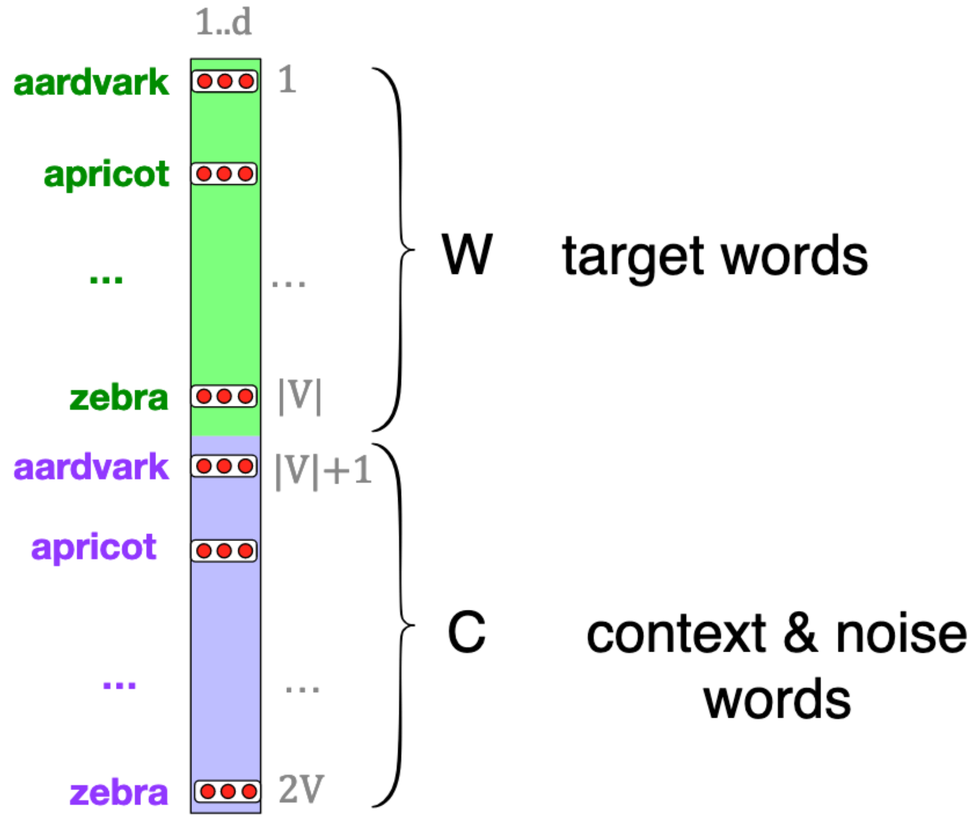
- Target and context words that are *more similar* to each other (have more similar embeddings) have a *higher probability* of being a positive example.

Learning

- What are the parameters?
- What is the loss?

Learning: Parameters

$\boldsymbol{\theta} =$



Learning: Loss

- We want our model to:
  - Assign high $P(1 \mid w, c_+)$ [c+ is a positive context word]
  - Assign low $P(1 \mid w, c\_)$ [c is a positive context word]
    - Equivalently: assign high $P(0 \mid w, c\_)$

Loss: Binary Cross-Entropy

$$\ell_{BCE}(\hat{y}, y) := -y \log \hat{y} - (1 - y)\log(1 - \hat{y})$$

- y = 1: $-\log(\hat{y}) = -\log P(1 \mid w, c)$
- y = 0: $-\log(1 - \hat{y}) = -\log P(0 \mid w, c)$
- So: negative log probability that the model assigns to the true label.

- Exercise: show that this is a special case of cross-entropy between two probability distributions:

$$H(p, q) = -\sum_i p_i \log q_i$$

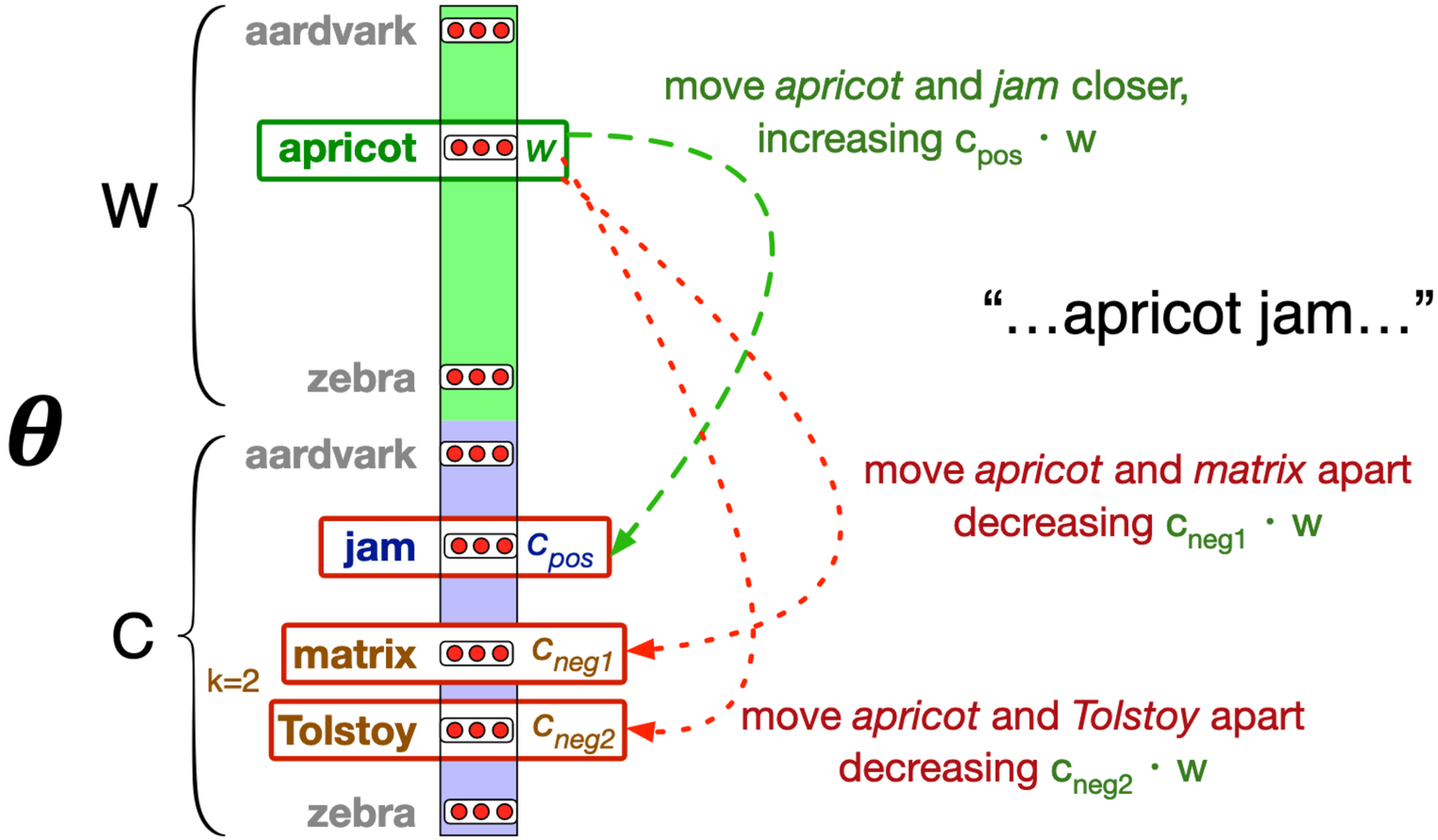
Training Loop w/ Negative Samples

```
initialize parameters / build model

for each epoch:

  positives = shuffle(positives)

  for each example in positives:

    positive_output = model(example)
    generate k negative samples
    negative_outputs = [model(negatives)]
    compute gradients
    update parameters
```

Combo Loss

$$
\begin{aligned}
L_{CE} &= -\log P(1,0,0,\ldots,0 \mid w, c_+, c_{-1}, c_{-2}, \ldots, c_{-k}) \\
&= -\log P(1 \mid w, c_+) \prod_{i=1}^{k} P(0 \mid w, c_{-i}) \\
&= -\log P(1 \mid w, c_+) - \sum_{i=1}^{k} \log P(0 \mid w, c_{-i})
\end{aligned}
$$

Learning: Intuitively



Tasks: Text Classification (Sentiment Analysis), Language Modeling

Text Classification

- Many different specific tasks
  - Input: text of some kind
  - Output: finite number of categories (usually fairly few)

Examples

- Spam detection:
  - Input: e-mail
  - Output: spam vs. not spam
- Intent classification:
  - Input: message from user to chatbot
  - Output: domain-specific intents
    - e.g. place new order, ask for hours, update cart, unknown

Sentiment Analysis

- Input: text
- Output: sentiment labels
  - e.g. negative, positive
  - e.g. very negative, somewhat negative, neutral, somewhat positive, very positive
  - e.g. # of stars
- Example inputs:
  - Product reviews
  - Movie reviews
  - Social media posts

Stanford Sentiment Treebank

- For many assignments in this class, we will use the Stanford Sentiment Treebank
  - Input: movie reviews from Rotten Tomatoes
  - Output: discrete ratings (0-4) of the sentiment from very negative to very positive
  - Simple/cleaned version available in `/dropbox/21-22/575k/data/sst/`

Stanford Sentiment Treebank

- 11,855 sentences
  - originally 10,662, but a parser split some into more than one
  - [full dataset includes annotations for every node of a parse tree]
  - Train = 8544; dev = 1101; test = 2210
- Annotation on Mechanical Turk:
  - 25 positions for a slider
  - 3 annotations per sentence
  - Avg score in [0, 1], mapped to 5 discrete labels

SST Examples

- grenier is terrific , bringing an unforced , rapid-fire delivery to toback 's heidegger - and nietzsche-referencing dialogue .
  - 4
- made me unintentionally famous -- as the queasy-stomached critic who staggered from the theater and blacked out in the lobby .
  - 1
- a fascinating , dark thriller that keeps you hooked on the delicious pulpiness of its lurid fiction .
  - 3
- beresford nicely mixes in as much humor as pathos to take us on his sentimental journey of the heart .
  - 3

Language Modeling

- A language model parametrized by θ computes $P_\theta(w_1, \ldots, w_n)$

- Typically (though we'll see variations): $P_\theta(w_1, \ldots, w_n) = \prod_i P_\theta(w_i \mid w_1, \ldots, w_{i-1})$

- E.g. of labeled data: "Today is the third day of 575k." —>
  - (<s>, Today)
  - (<s> Today, is)
  - (<s> Today is, the)
  - (<s> Today is the, third)

Language Modeling

Language Modeling

- A good language model should produce good *general-purpose* and *transferable* representations

Language Modeling

- A good language model should produce good *general-purpose* and *transferable* representations

- Linguistic knowledge:
  - The bicycles, even though old, were in good shape because ____ …
  - The bicycle, even though old, was in good shape because ____ …

Language Modeling

- A good language model should produce good *general-purpose* and *transferable* representations

- Linguistic knowledge:
  - The bicycles, even though old, were in good shape because ____ …
  - The bicycle, even though old, was in good shape because ____ …

- World knowledge:
  - The University of Washington was founded in _____
  - Seattle had a huge population boom as a launching point for expeditions to _____

Data for LM is cheap

Data for LM is cheap

Data for LM is cheap

Text

Language Model Pre-training

- A currently powerful paradigm for training models for NLP tasks:
  - *Pre-train* a large language model on a large amount of raw text
  - *Fine-tune* a small model on top of the LM for the task you care about
    - [or use the LM as a general feature extractor]
- More on this use case later in the course

LMs for Generation

- By iteratively sampling from the distribution of an LM, one can generate text [in the style of the training data]
- Samples from a character-level recurrent LM on (i) Shakespeare and (ii) Linux source code:

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.
```

```c
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
```

source

Costs of LMs

On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜

Emily M. Bender*
ebender@uw.edu
University of Washington
Seattle, WA, USA

Timnit Gebru*
timnit@blackinai.org
Black in AI
Palo Alto, CA, USA

Angelina McMillan-Major
aymm@uw.edu
University of Washington
Seattle, WA, USA

Shmargaret Shmitchell
shmargaret.shmitchell@gmail.com
The Aether

ABSTRACT

The past 3 years of work in NLP have been characterized by the development and deployment of ever larger language models, especially for English. BERT, its variants, GPT-2/3, and others, most recently Switch-C, have pushed the boundaries of the possible both through architectural innovations and through sheer size. Using these pretrained models and the methodology of fine-tuning them for specific tasks, researchers have extended the state of the art alone, we have seen the emergence of BERT and its variants [39, 70, 74, 113, 146], GPT-2 [106], T-NLG [112], GPT-3 [25], and most recently Switch-C [43], with institutions seemingly competing to produce ever larger LMs. While investigating properties of LMs and how they change with size holds scientific interest, and large LMs have shown improvements on various tasks (§2), we ask whether enough thought has been put into the potential risks associated with developing them and strategies to mitigate these risks.

- For more on the reactions to this paper: https://faculty.washington.edu/ebender/stochasticparrots.html
- More later this quarter as well

Classifiers and LMs in This Class

- For the next several weeks, we will develop different neural architectures for both classification and language modeling
  - For LM, using the text of reviews in SST as our raw texts

- Conceptually, LM training is just |V|-way classification
  - But very different at inference time
  - Some different modeling assumptions will arise as well

Next Time

- Introduction to Neural Networks
  - Feed-forward architecture
  - Basic notation, expressive power
  - Parameters and hyper-parameters
  - Some more info on training