Word Vectors (word2vec)

Ling 282/482: Deep Learning for Computational Linguistics

C.M. Downey

Fall 2024

Beware of Frequency!

source

Today’s Plan

Today's Plan

- Last time:
  - Loss minimization
  - Gradient descent
  - Why word vectors

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*
  - Classification tasks

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 **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

Embeddings:
Skip-Gram vs. Continuous Bag of Words

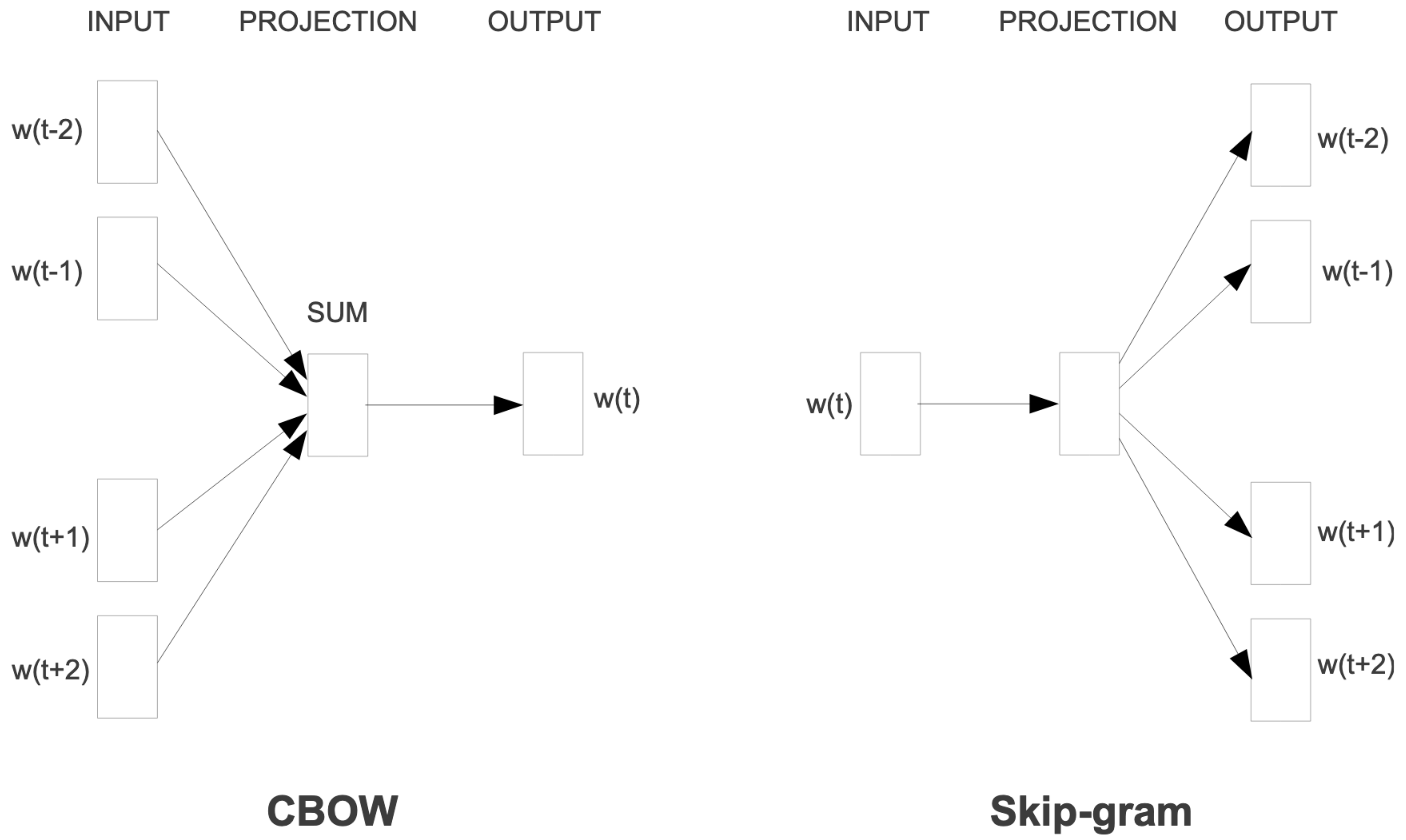
- Continuous Bag of Words (CBOW):
  - ***P*(*word*|*context*)**
  - Input: $(w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2} \ldots)$
  - Output: $p(\boldsymbol{w_t})$

Embeddings:
Skip-Gram vs. Continuous Bag of Words

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

Embeddings:
Skip-Gram vs. Continuous Bag of Words

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



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

Skip-Gram Model

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

Skip-Gram Model

- Learns two embedding matrices
  - 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 embedding matrices
  - 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

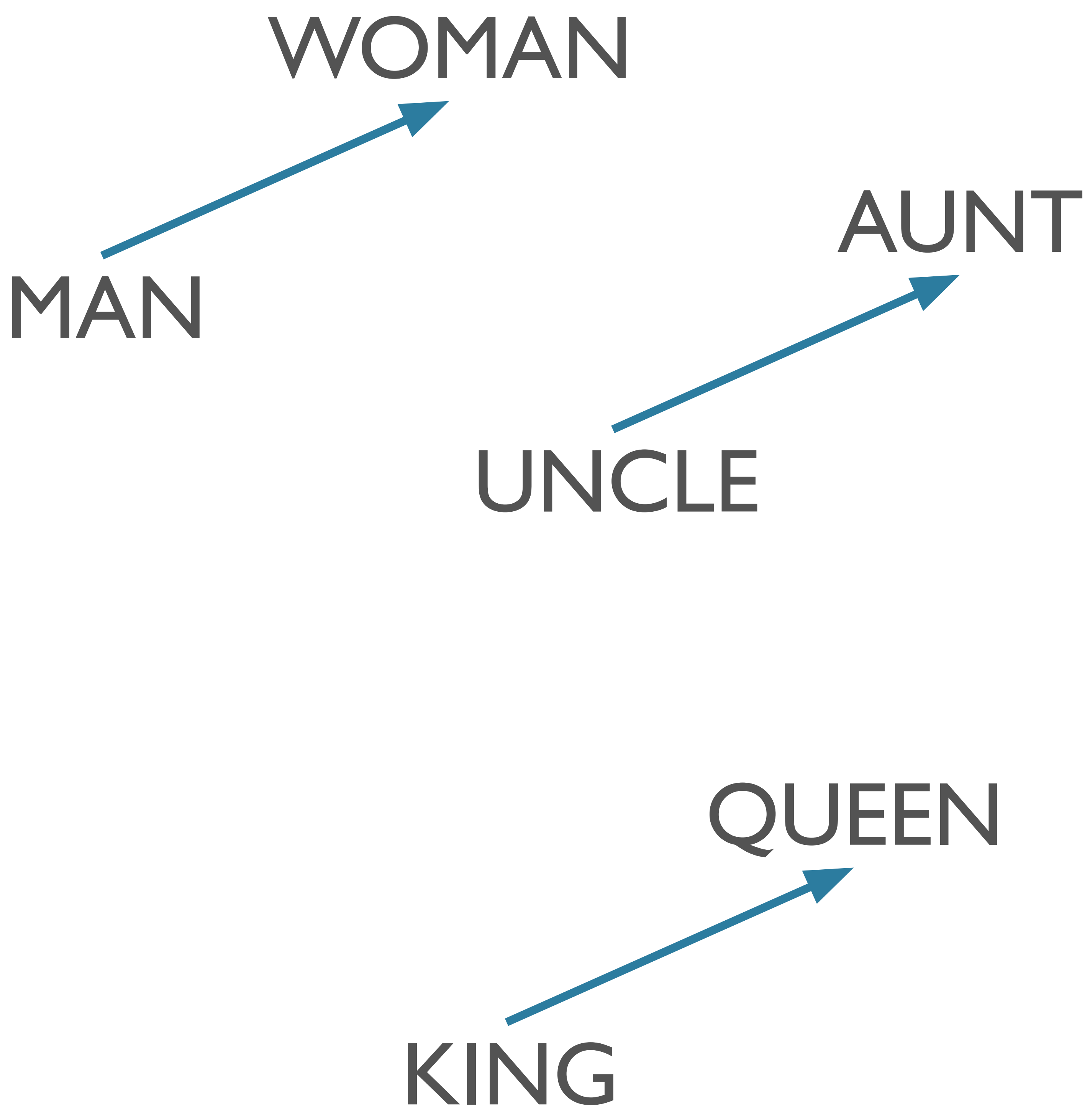
Power of Prediction-based Embeddings

- Count-based embeddings:
  - Very high-dimensional (|V|)
  - Sparse
  - Pro: features are interpretable [“occurred with word W N times in corpus”]

Power of Prediction-based Embeddings

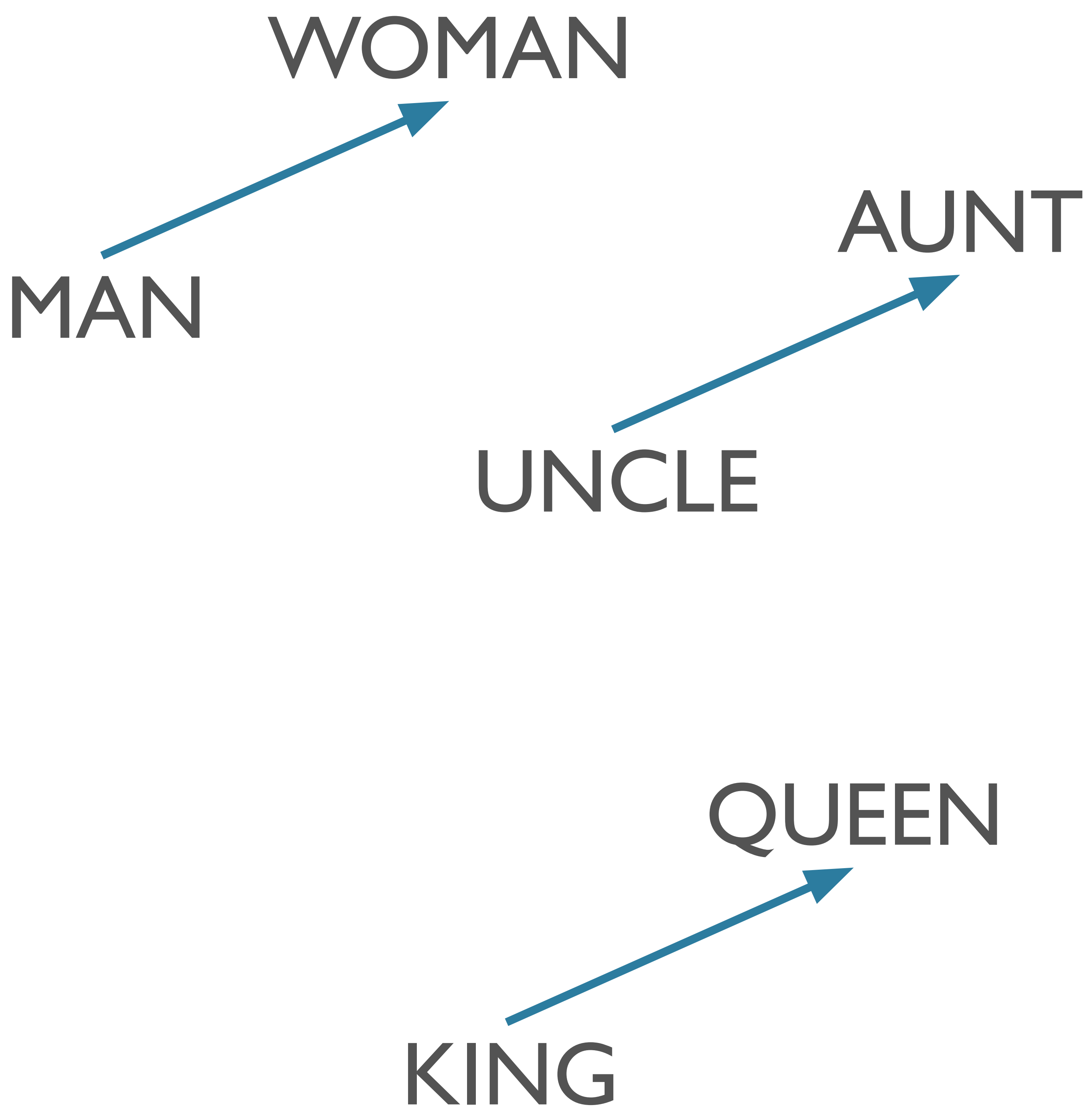
- Count-based embeddings:
  - Very high-dimensional (|V|)
  - Sparse
  - Pro: features are interpretable [“occurred with word W N times in corpus”]
- Prediction-based embeddings:
  - “Low”-dimensional (typically ~300-1200)
  - Dense
  - Con: features are not immediately interpretable
    - i.e. what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

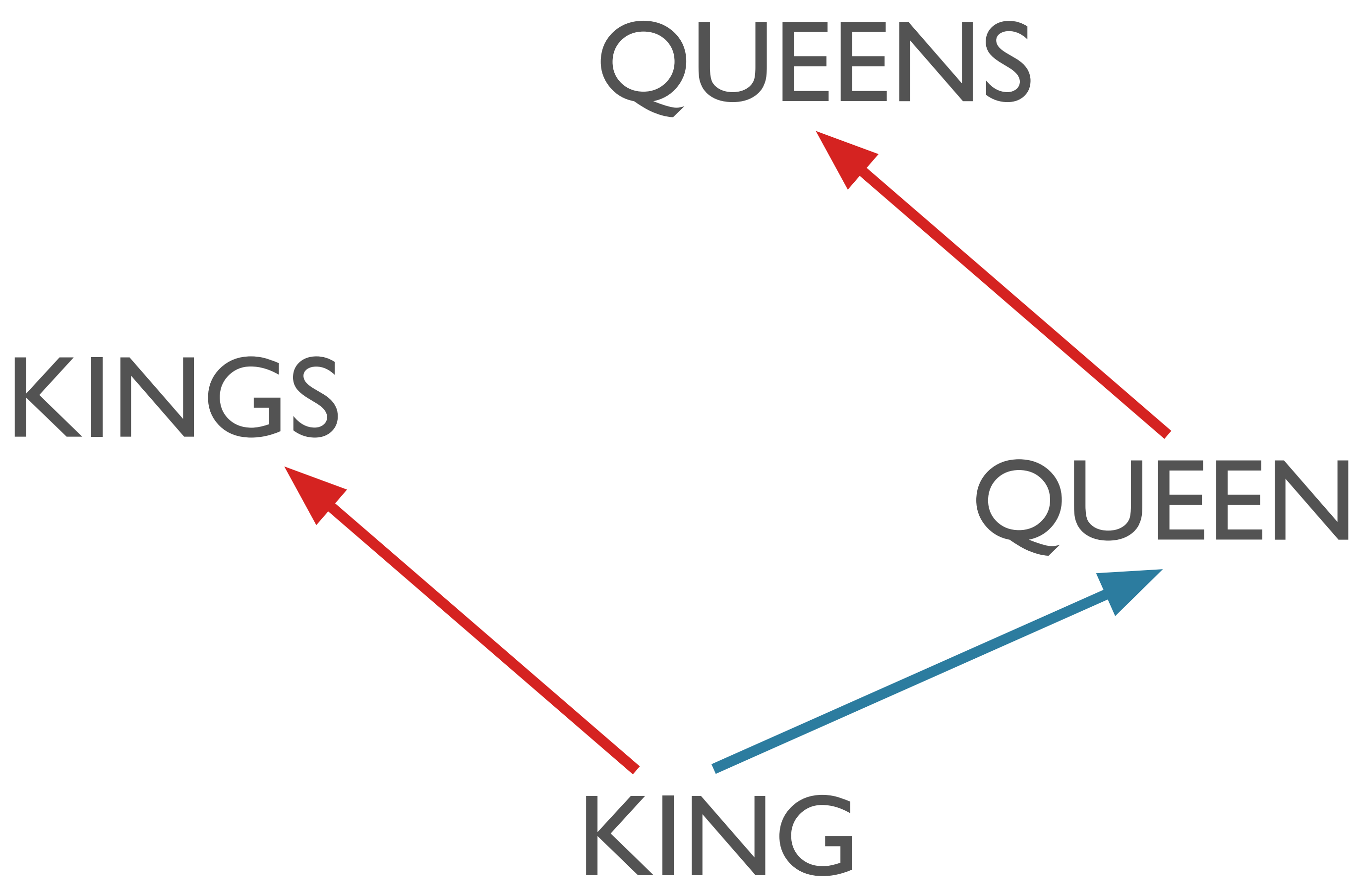
Relationships via Offsets



Mikolov et al 2013b

Relationships via Offsets





Mikolov et al 2013b

One More Example

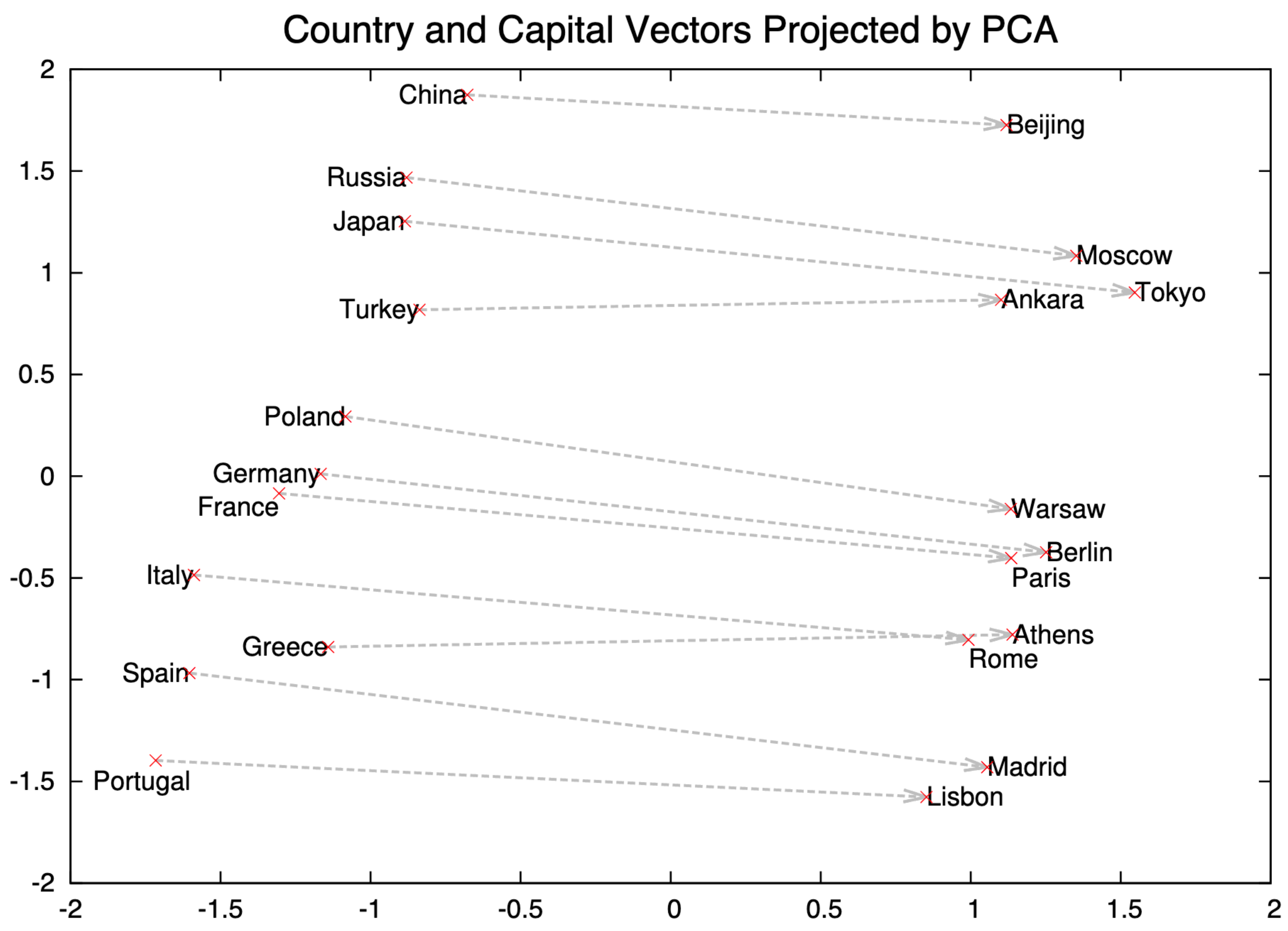
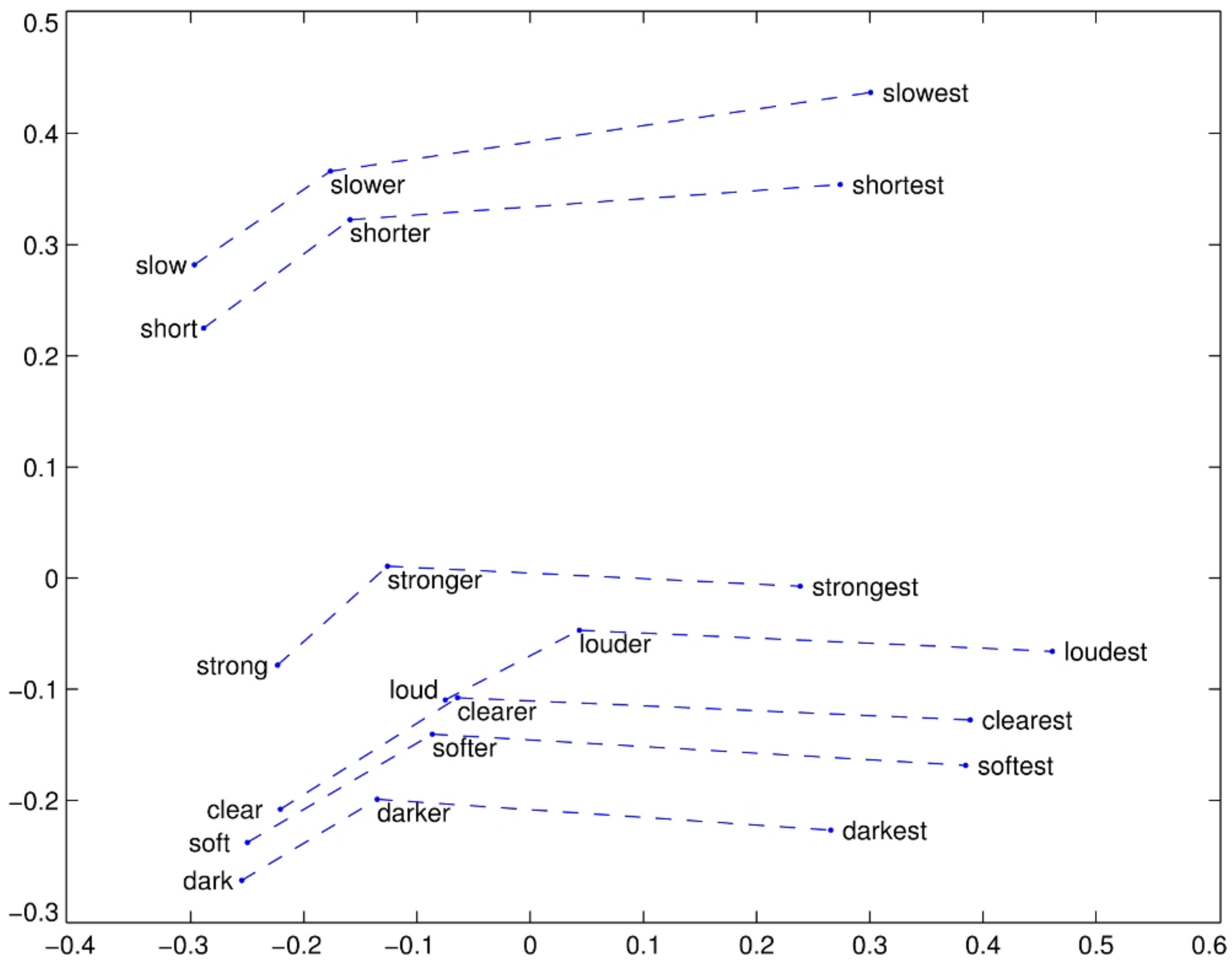


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised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

Mikolov et al 2013c

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.

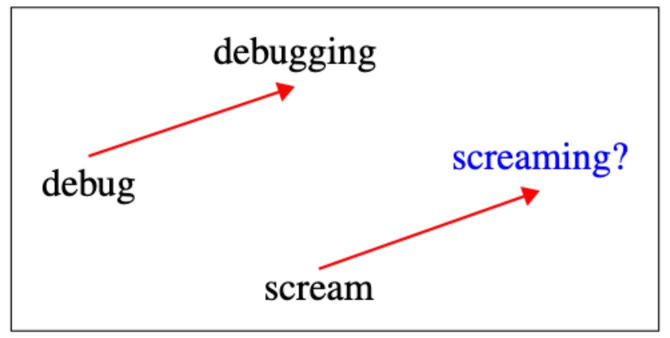


Figure 1: Using the vector offset method to solve the analogy task (Mikolov et al., 2013c).

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

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

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

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

Negative Sampling, Idea

Negative Sampling, Idea

- Skip-Gram:
  - $P(w_k | w_j)$: what is the probability that w_k occurred in the context of w_j
  - Classifier with |V| classes

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

Generating Positive Examples

- Iterate through the corpus

Generating Positive Examples

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

The Data

The Data

- X = pairs of words

The Data

- X = pairs of words
- Y = {0, 1}
  - 1 = + (positive example), 0 = - (negative example)

The Data

- 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

The Model

- So what is $P(1 \mid w, c)$?

The Model

- So what is $P(1 \mid w, c)$?
  - More specifically: $P(1 \mid w, c; \theta)$

The Model

- So what is $P(1 \mid w, c)$?
  - More specifically: $P(1 \mid w, c; \theta)$
- As before, learns **two embedding matrices**

The Model

- So what is $P(1 \mid w, c)$?
  - More specifically: $P(1 \mid w, c; \theta)$
- As before, learns **two embedding matrices**
  - ***E*: word embeddings**, matrix of shape [vocab_size, embedding_dimension]

The Model

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

The Model

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

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

Target word embedding

Context word embedding

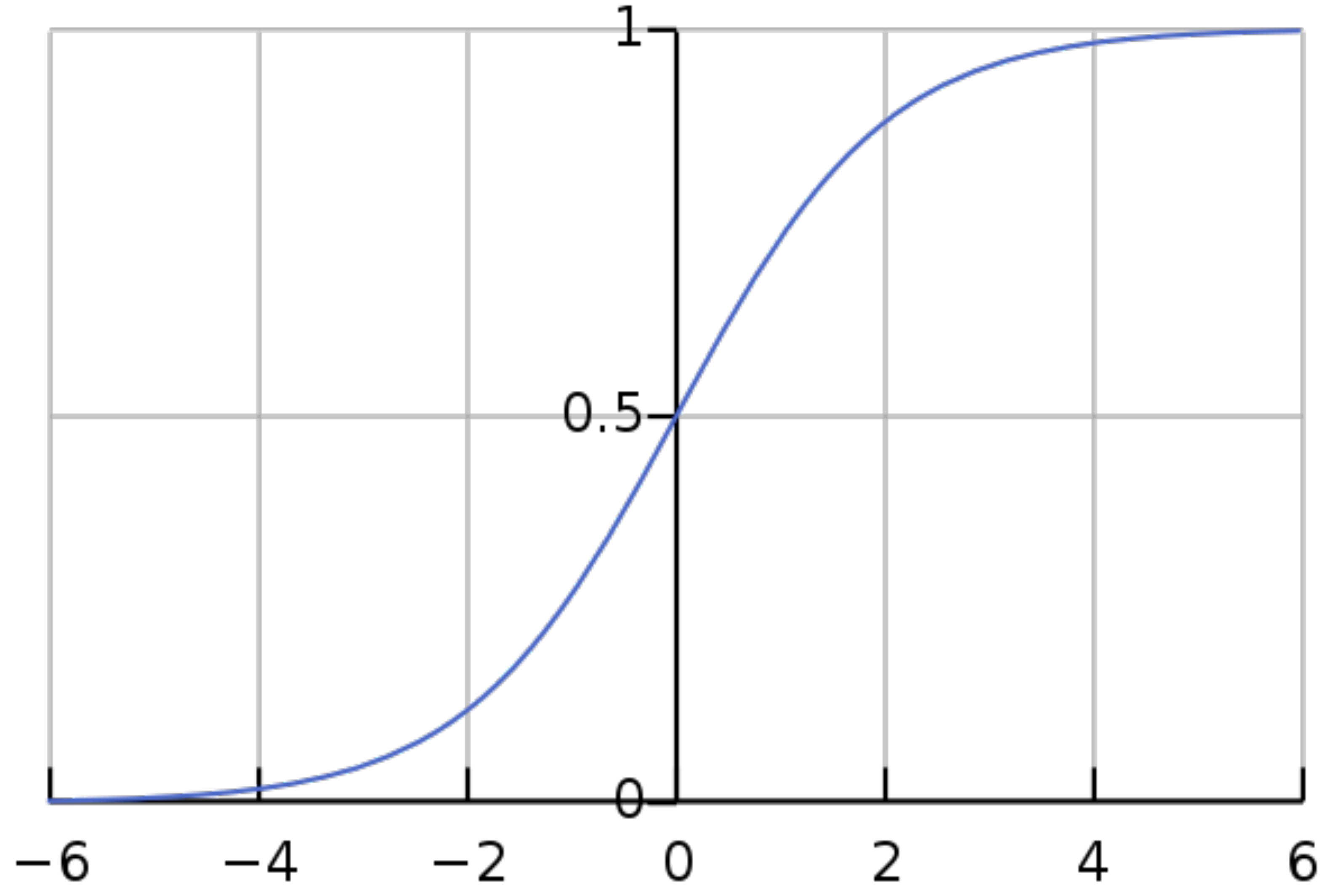
Similarity (dot-product)

The Model

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

sigmoid

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



Target word embedding

Similarity (dot-product)

Context word embedding

The Model

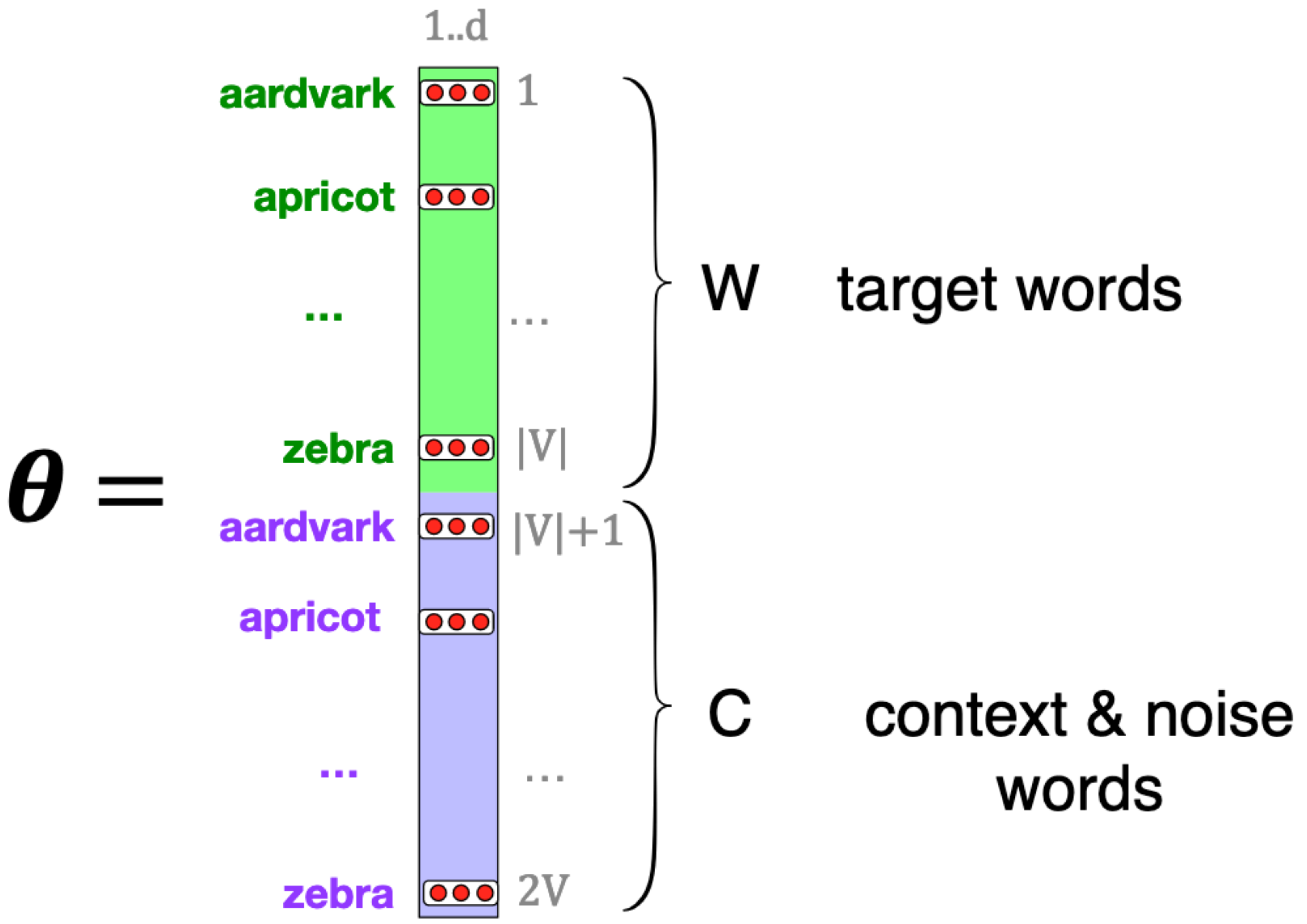
- Target and context words that are **more similar** to each other (have more similar embeddings) have a **higher probability** of being a positive example

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

Learning

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

Learning: Parameters



Learning: Loss

Learning: Loss

- We want our model to:

Learning: Loss

- We want our model to:
  - Assign high $P(1 \mid w, c_+)$ (c+ is a positive context word)

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 negative context word)

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 negative context word)
    - Equivalently: assign high $P(0 \mid w, c_-)$

Loss: Binary Cross-Entropy

Loss: Binary Cross-Entropy

- Recall: y is our “gold standard” label. $\hat{y}$ is the model’s prediction

Loss: Binary Cross-Entropy

- Recall: y is our “gold standard” label. $\hat{y}$ is the model’s prediction
- When y = 1
  - minimize: $-\log(\hat{y}) = -\log P(1 \mid w, c)$

Loss: Binary Cross-Entropy

- Recall: y is our “gold standard” label. $\hat{y}$ is the model’s prediction
- When y = 1
  - minimize: $-\log(\hat{y}) = -\log P(1 \mid w, c)$
- When y = 0
  - minimize: $-\log(1 - \hat{y}) = -\log(1 - P(1 \mid w, c)) = -\log P(0 \mid w, c)$

Loss: Binary Cross-Entropy

- Recall: y is our “gold standard” label. $\hat{y}$ is the model’s prediction
- When y = 1
  - minimize: $-\log(\hat{y}) = -\log P(1 \mid w, c)$
- When y = 0
  - minimize: $-\log(1-\hat{y}) = -\log(1 - P(1 \mid w, c)) = -\log P(0 \mid w, c)$
- I.e. the negative log probability that the model assigns to the true label

Loss: Binary Cross-Entropy

- Recall: y is our “gold standard” label. $\hat{y}$ is the model’s prediction
- When y = 1
  - minimize: $-\log(\hat{y}) = -\log P(1 \mid w, c)$
- When y = 0
  - minimize: $-\log(1 - \hat{y}) = -\log(1 - P(1 \mid w, c)) = -\log P(0 \mid w, c)$
- I.e. the negative log probability that the model assigns to the true label
- BCE loss incorporates both into the closed form:

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

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
```

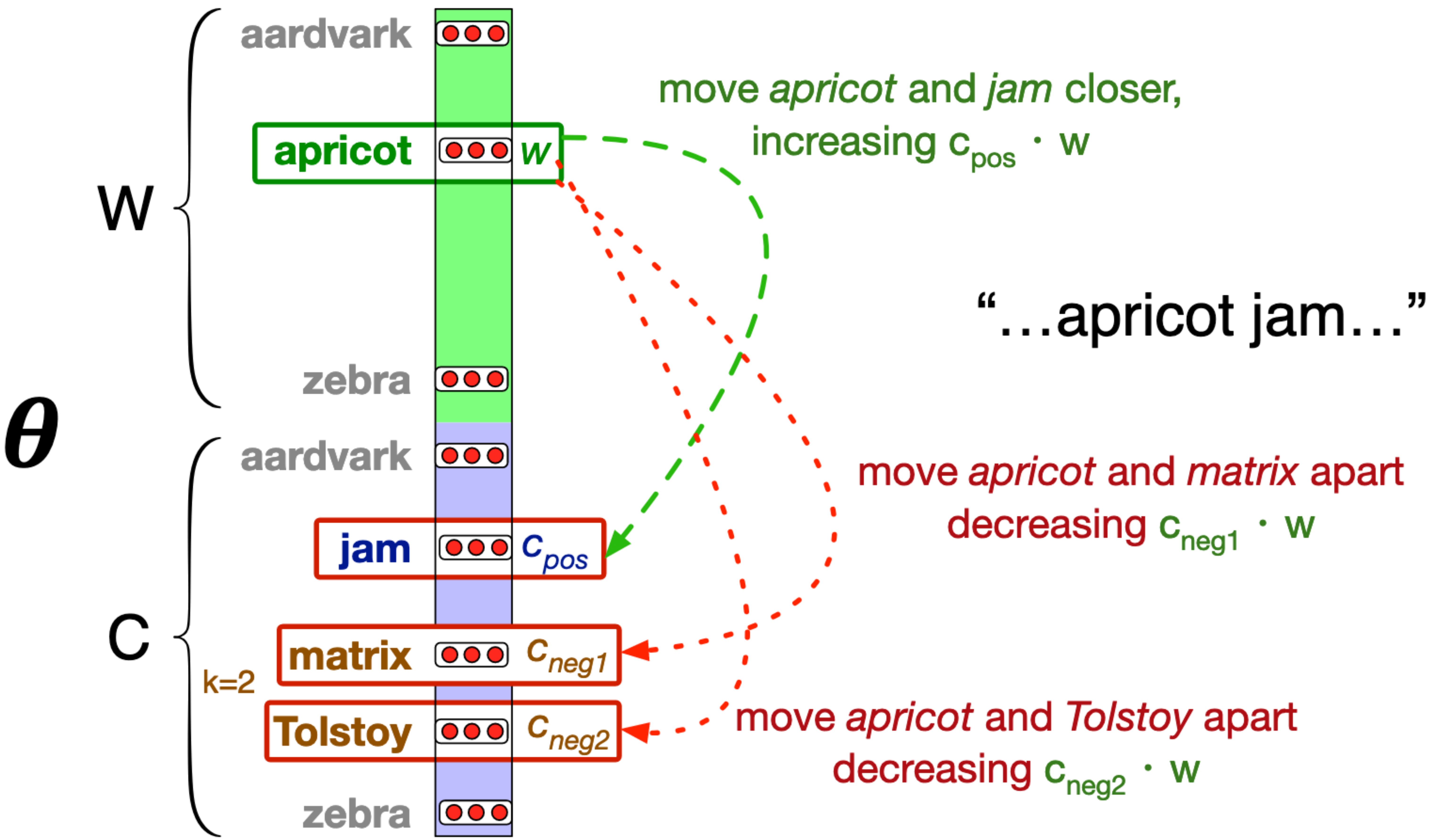
Combo Loss

$$L_{CE} = -\log P(1,0,0,\ldots,0 \,|\, w, c_+, c_{-1}, c_{-2}, \ldots, c_{-k})$$

$$= -\log(P(1 \,|\, w, c_+)\prod_{i=1}^{k} P(0 \,|\, w, c_{-i}))$$

$$= -\log P(1 \,|\, w, c_+) - \sum_{i=1}^{k} \log P(0 \,|\, w, c_{-i})$$

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

Examples

- Spam detection:
    - Input: e-mail
    - Output: spam vs. not spam

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

Sentiment Analysis

- Input: text

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

Sentiment Analysis

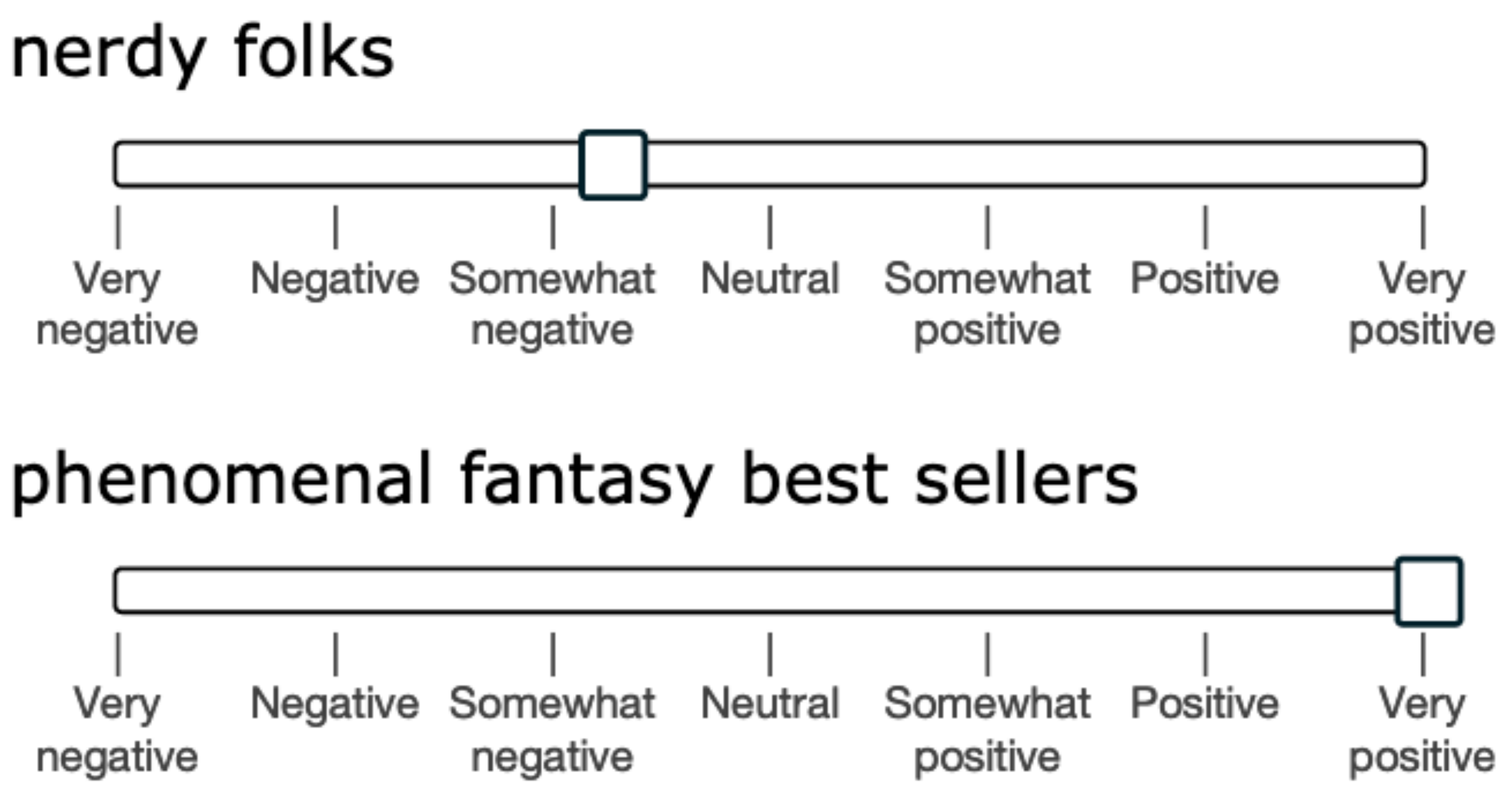
- Input: text
- Output: sentiment labels
    - e.g. negative, positive
    - e.g. very negative, somewhat negative, neutral, somewhat positive, very positive
    - e.g. # of stars
- Example inputs:
    - Product reviews
    - Movie reviews
    - Social media posts

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 starting repository for homework

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