Vector Semantics

CSC485

Lecture 10

Announcements

- Guest lecture from Prof. Zhijing Jin on Friday!
  - Incoming Prof at UofT CL Group (2025-)
  - THE researcher on Causal LLMs for Social Good.

- A1 Extension?
  - Vote!
  - If I give you an extension, you give me an extension.
  - Slower TA response after the original deadline, potential marking delay.

Representing Data

- Earlier success in computer vision.
  - Navlab 5 (Jochem et al., 1995)

- Much more intuitive to convert images into vector representations.

Representing Data

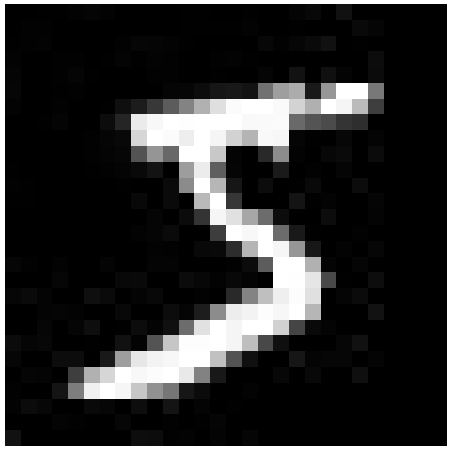
- Numeric Data:
  - E.g. credit score:

Monthly Income	Number Of Open Credit Lines And Loans	Number Of Times 90 Days Late	Number Real Estate Loans Or Lines	Number Of Time 60-89 Days Past Due Not Worse	Number Of Dependents
9120	13	0	6	0	2
2600	4	0	0	0	1
3042	2	1	0	0	0
3300	5	0	0	0	0
63588	7	0	1	0	0
3500	3	0	1	0	1
NA	8	0	3	0	0
3500	8	0	0	0	0

Give Me Some Credit (gmsc): https://www.kaggle.com/c/GiveMeSomeCredit

Representing Data

- Numeric Data:
  - E.g. credit score:
- Images:
  - Gray scale or RGB



MNIST dataset
Handwritten numbers

Representing `Textual` Data

- The vast majority of rule--based and statistical NLP work regarded words as atomic symbols.
  - Recall Lecture 3:

```
Det → the | a | an
Adj → old | red | happy | …
N   → dog | park | ice-cream | contumely | run | …
V   → saw | ate | run | disdained | …
P   → in | to | on | under | with | … }
```

- Vector space: this is a vector with one 1 and a lot of zeroes:

`[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]`

  - The "one-hot" representation
  - i-th word in the dictionary:

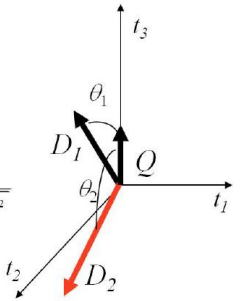
$$v_i = 1, \forall j \neq i, v_j = 0$$

Representing `Textual` Data: Problems

- There are a lot of words!
  - Oxford English Dictionary: 500,000+ entries
  - Longman Dictionary of Contemporary English: 230,000 words
  - Brysbaert et al. (2016): 42,000 lemmas
- As a result, a lot of BIG vectors!
- For reference, (L)LM dimensions:
  - BERT, GPT-2: 768
  - Llama-3-8B: 4096
  - Llama-3.1-405B: 16384
- No useful similarity information:
  - Motel: `[0 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 0]`
  - Linguist: `[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]`
- cos_sim(motel, hotel)=0, cos_sim(motel, linguist)=0, cos_sim(hotel, linguist)=0

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.

$$\text{CosSim}(\vec{d}_j, \vec{q}) = \frac{\vec{d}_j \cdot \vec{q}}{|\vec{d}_j| \cdot |\vec{q}|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$

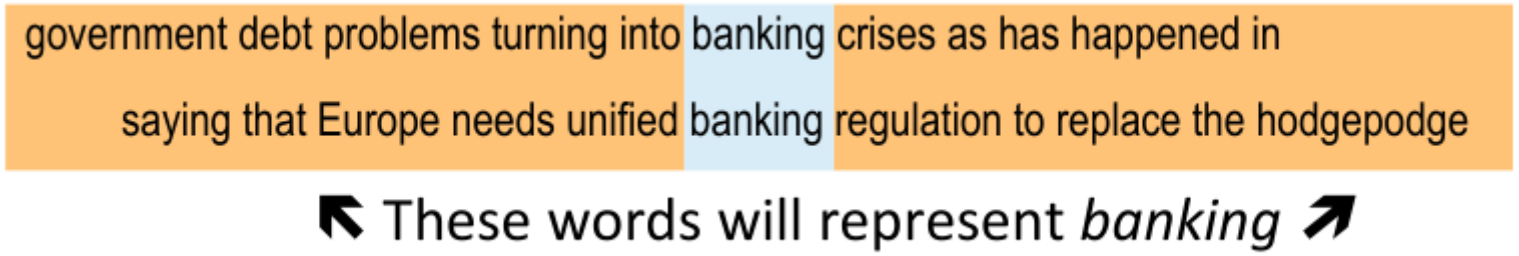


$D_1 = 2T_1 + 3T_2 + 5T_3$ $\quad \text{CosSim}(D_1, Q) = 10 / \sqrt{(4+9+25)(0+0+4)} = 0.81$
$D_2 = 3T_1 + 7T_2 + 1T_3$ $\quad \text{CosSim}(D_2, Q) = 2 / \sqrt{(9+49+1)(0+0+4)} = 0.13$
$Q = 0T_1 + 0T_2 + 2T_3$

D_1 is 6 times better than D_2 using cosine similarity but only 5 times better using inner product.

Distributional similarity based representations

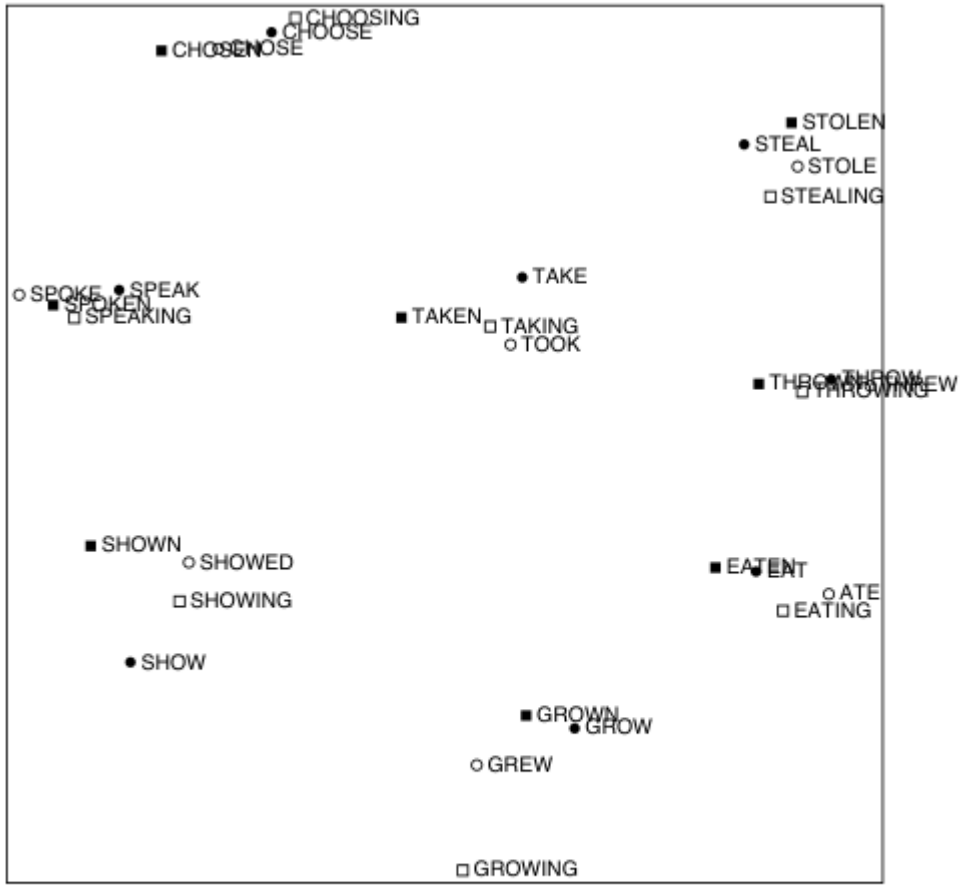
- You can get a lot of value by representing a word by means of its neighbors:
- “Noscitur a sociis”
  - The meaning of an unclear or ambiguous word should be determined by considering the words with which it is associated in the context.
  - 19th-century rule of interpretation in English civil courts.
- One of the most successful ideas of modern NLP



With distributed, distributional representations, syntactic and semantic information can be captured

$$shown = \begin{bmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{bmatrix}$$

Synonymy? Hyponymy?
Morphology?



[Rohde et al. 2005. An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence]

Outline

1. Vector space representations of language
2. Predict! vs. Count!: The GloVe model of word vectors
3. Wanted: meaning composition functions
4. Tree-structured Recursive Neural Networks for Semantics
5. Revisit Transformers

Two Kinds of Vectors

- Count:
  - tf-idf, PMI, LSA
  - **Sparse!**
  - Information Retrieval workhorse!
  - Words are represented by (a simple function of) the counts of nearby words
- Predict:
  - word2vec, GloVe, BERT, GPT-2, …
  - **Dense!**
  - Representation is created by training a classifier to predict whether a word is likely to appear nearby
  - *Contextual embeddings*.

TF-IDF

- Corpus: collection of N documents
  - “cherry” & “strawberry vs. “digital” & “information”
- Term frequency:

$$tf(t, d) = count(t, d)$$

- Inverse document frequency:

$$idf(t) = \log(N / df)$$

Size of corpus (N = |D|)

number of documents where the term t appers

- tf-idf(t, d) = tf(t, d) * idf(t)

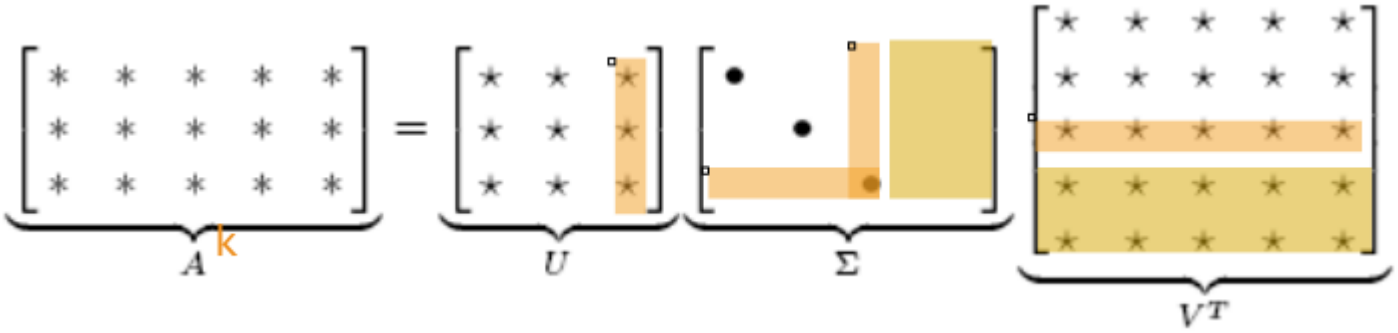
Example


```
from sklearn.feature_extraction.text import TfidfVectorizer
import torch
from torch.nn.functional import cosine_similarity as sim

if __name__ == '__main__':
    corpus = ['I did not hit her',
              'I did not',
              'Oh hi Mark']

    tfidf = TfidfVectorizer(stop_words='english')
    x = torch.tensor(tfidf.fit_transform(corpus).todense())
    fs = tfidf.get_feature_names_out()
    print(fs)
    print(x)

    print(corpus[0], corpus[1], sim(x[0], x[1], dim=0), sep='\t')
    print(corpus[0], corpus[2], sim(x[0], x[2], dim=0), sep='\t')
    print(corpus[1], corpus[2], sim(x[1], x[2], dim=0), sep='\t')

    print(fs[0], fs[2], sim(x[:,0], x[:,2], dim=0), sep='\t')
    print(fs[1], fs[2], sim(x[:,1], x[:,2], dim=0), sep='\t')
```

Latent semantic analysis

LSA: Count!

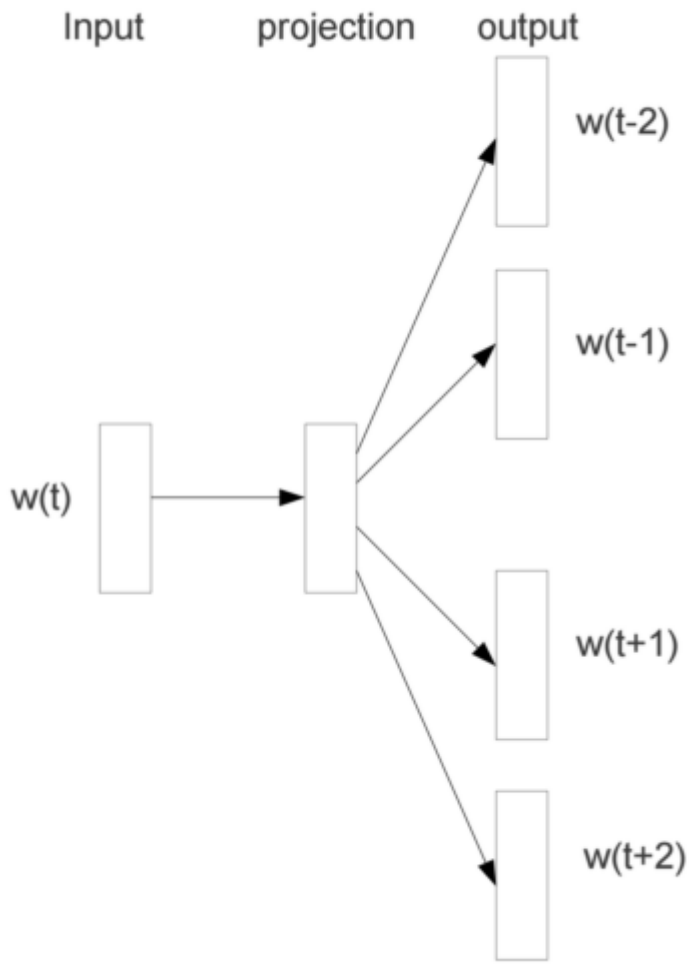
- Factorize a (maybe weighted, maybe log scaled) term-document or word-context matrix (Schütze 1992) into $U\Sigma V^T$
  - Singular value decomposition (SVD)
- Retain only *k* singular values, in order to generalize



Word2vec

Word2vec CBOW/SkipGram: Predict!

- Train word vectors to try to either
  - Predict a word given its bag-of-words context (CBOW); or
  - Predict a context word (position-independent) from the center word
- Update word vectors until they can do this prediction well



Skip-Gram Training Data

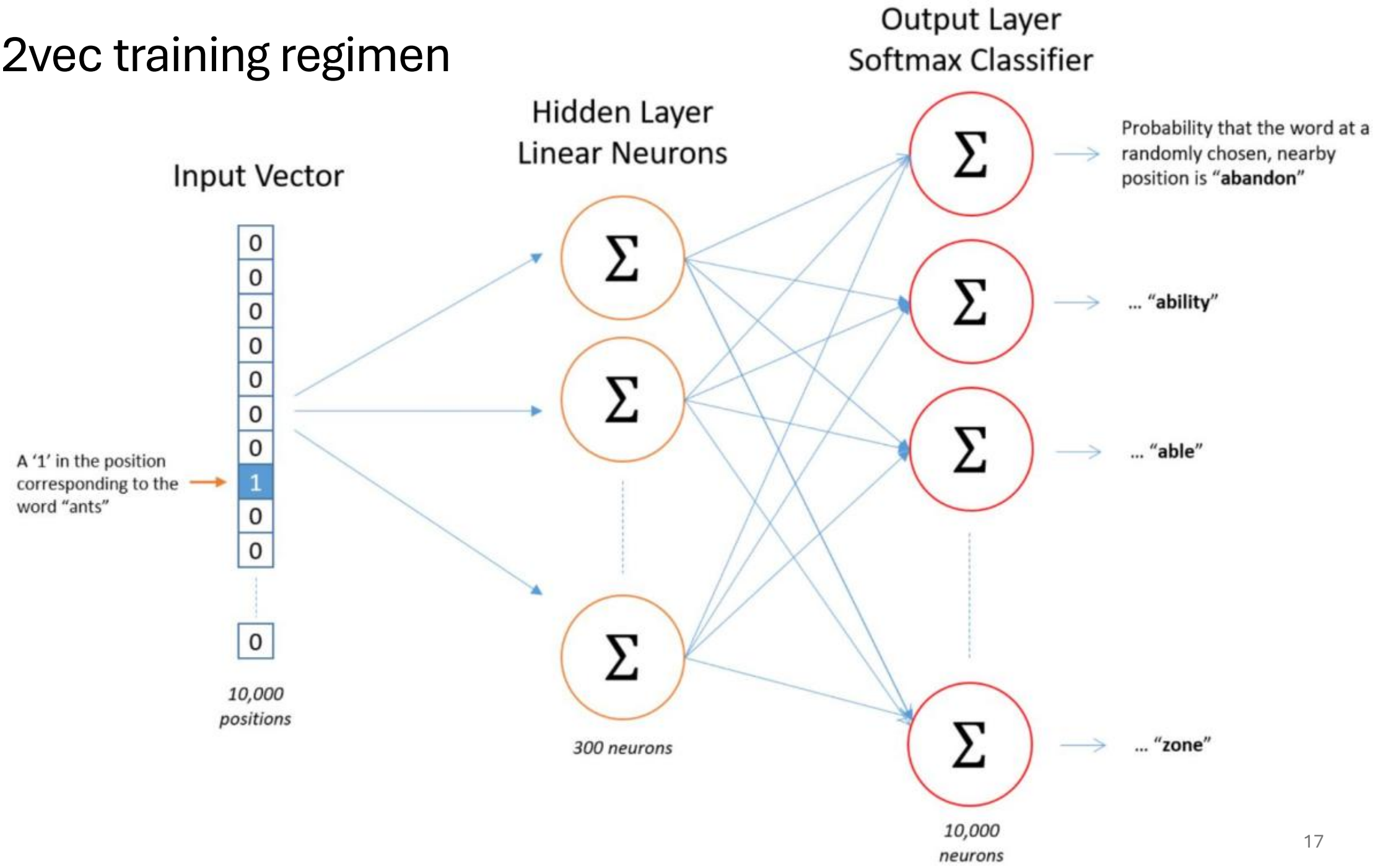
- Assume a +/- 2 word window, given training sentence:

…lemon, a [tablespoon of apricot jam, a] pinch

c1 c2 [target] c3 c4

- Goal: train a classifier that is given a candidate (word, context) pair
- And assigns each pair a probability:
  - P(+|w, c)
  - P(−|w, c) = 1 − P(+|w, c)

Word2vec training regimen



```python
class Word2Vec(nn.Module):

    def __init__(self, vocab_size, embedding_size):
        super().__init__()
        self.embed = nn.Embedding(vocab_size, embedding_size)
        self.expand = nn.Linear(embedding_size, vocab_size, bias=False)

    def forward(self, input):
        # Encode input to lower-dimensional representation
        hidden = self.embed(input)
        # Expand hidden layer to predictions
        logits = self.expand(hidden)
        return logits
```

Approach: predict if candidate word *c* is a "neighbor"

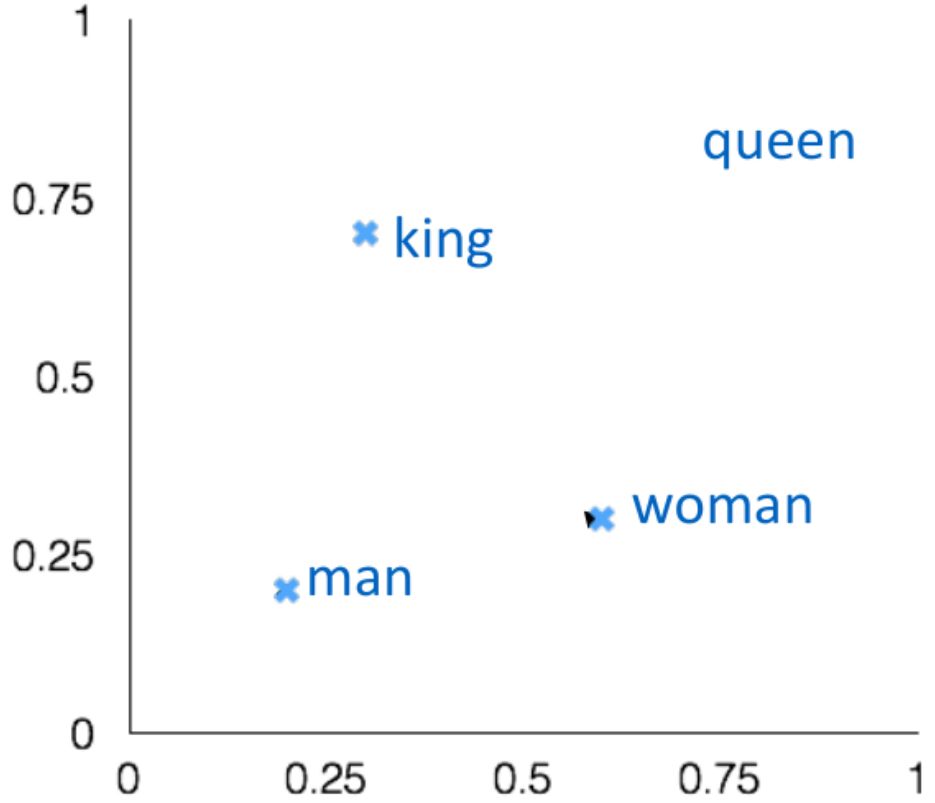
1. Treat the target word t and a neighboring context word c as **positive examples**.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

Word Analogies: word2vec captures dimensions of similarity as linear relations

Test for linear relationships, examined by Mikolov et al. (2013)

man:woman :: king:?

+	king	[0.30 0.70]
–	man	[0.20 0.20]
+	woman	[0.60 0.30]
.		
	queen	[0.70 0.80]



Word Analogies
[Mikolov et al., 2012, 2013]

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

Count based vs. direct prediction

- Fast training
- Efficient usage of statistics

- Long & Sparse!
  - Length = |V|
  - most elements are zero
- Primarily used to capture word similarity
- Disproportionate importance given to small counts

- Scales with corpus size
- Inefficient usage of statistics

- Short and Dense
  - Length = any hidden size (50-10000)
  - Nearly nothing is zero
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

Encoding meaning in vector differences

- Key idea:
  - Ratios of co-occurrence probabilities can encode meaning components

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

Pennington et al. (2014)

Encoding meaning in vector differences

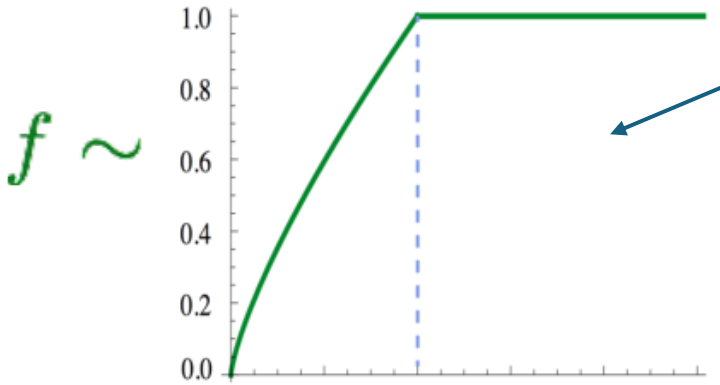
- How can we capture ratios of co-occurrence probabilities as meaning components in a word vector space?
- Solution:
  - Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$
  - with vector differences: $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

Pennington et al. (2014)

GloVe: A new model for learning word representations

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

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



weighting function

biases

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

dot product (similarity)

Coocurrence count

```python
# https://github.com/noaRricky/pytorch-glove
class GloVeModel(nn.Module):

    def __init__(self, embedding_size, context_size, vocab_siz)
        self.focal_embeddings = nn.Embedding(
            vocab_size, embedding_size)
        self.context_embeddings = nn.Embedding(
            vocab_size, embedding_size)

        self.focal_biases = nn.Embedding(vocab_size, 1)
        self.context_biases = nn.Embedding(vocab_size, 1)

    def loss(self, focal_input, context_input, coocurrence_count):

        focal_embed = self.focal_embeddings(focal_input)
        context_embed = self.context_embeddings(context_input)
        focal_bias = self.focal_biases(focal_input)
        context_bias = self.context_biases(context_input)

        # count weight factor
        weight_factor = torch.pow(coocurrence_count / x_max, alpha)
        weight_factor[weight_factor > 1] = 1

        embedding_products = torch.sum(focal_embed * context_embed, dim=1)
        log_cooccurrences = torch.log(coocurrence_count)

        distance_expr = (embedding_products + focal_bias +
                         context_bias + log_cooccurrences) ** 2

        single_losses = weight_factor * distance_expr
        mean_loss = torch.mean(single_losses)
        return mean_loss
```

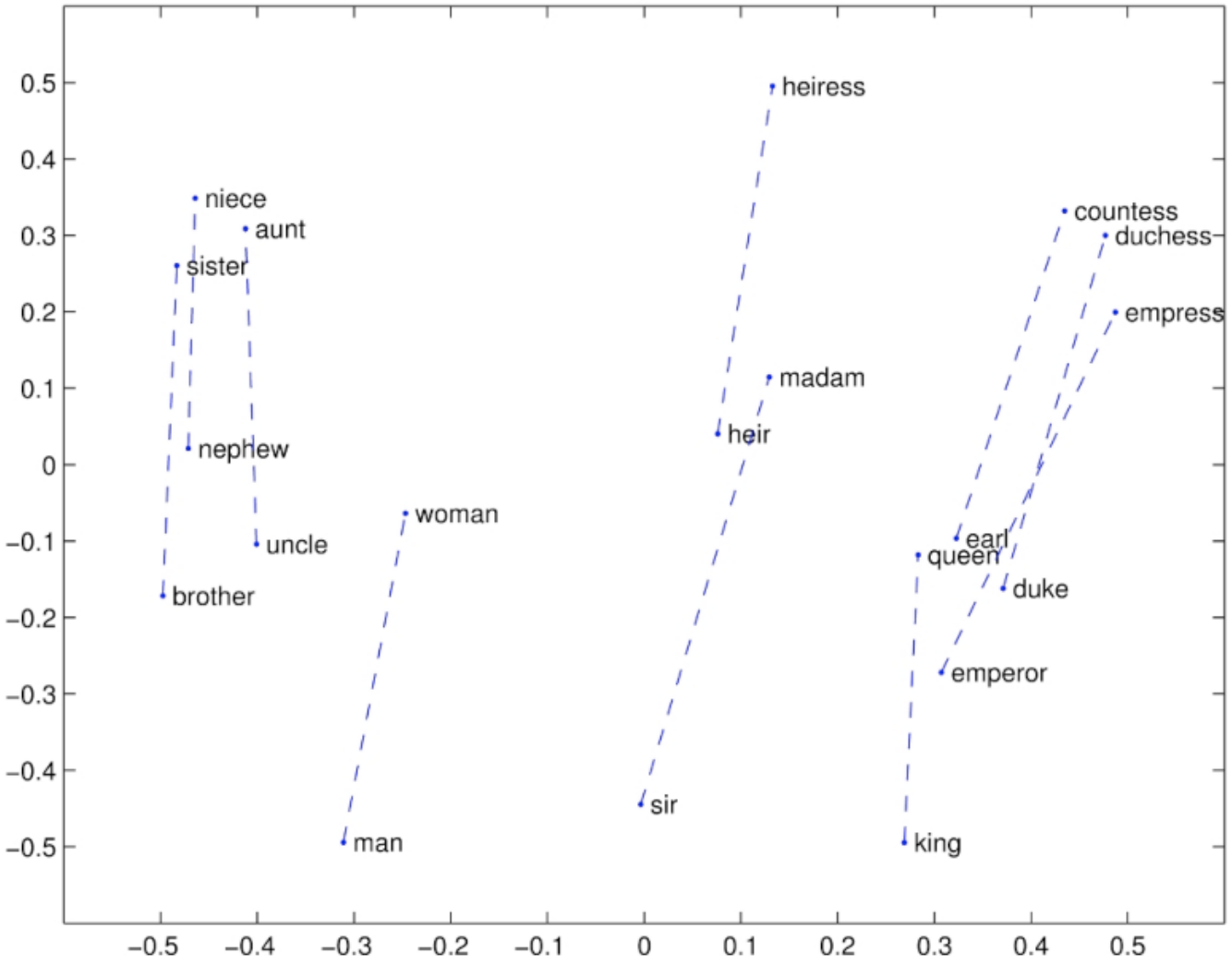
$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

Word Similarities

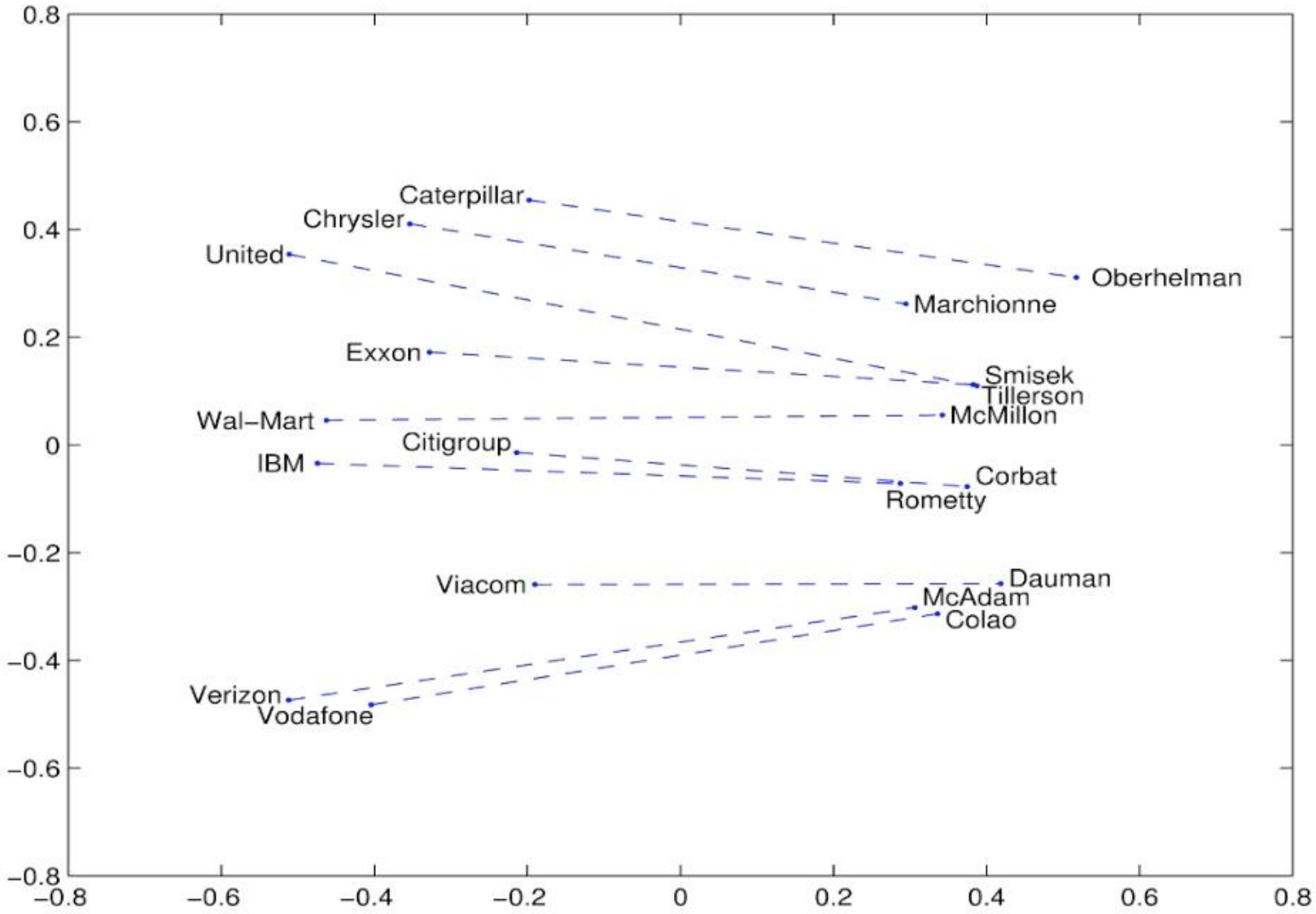
Nearest words to frog:

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

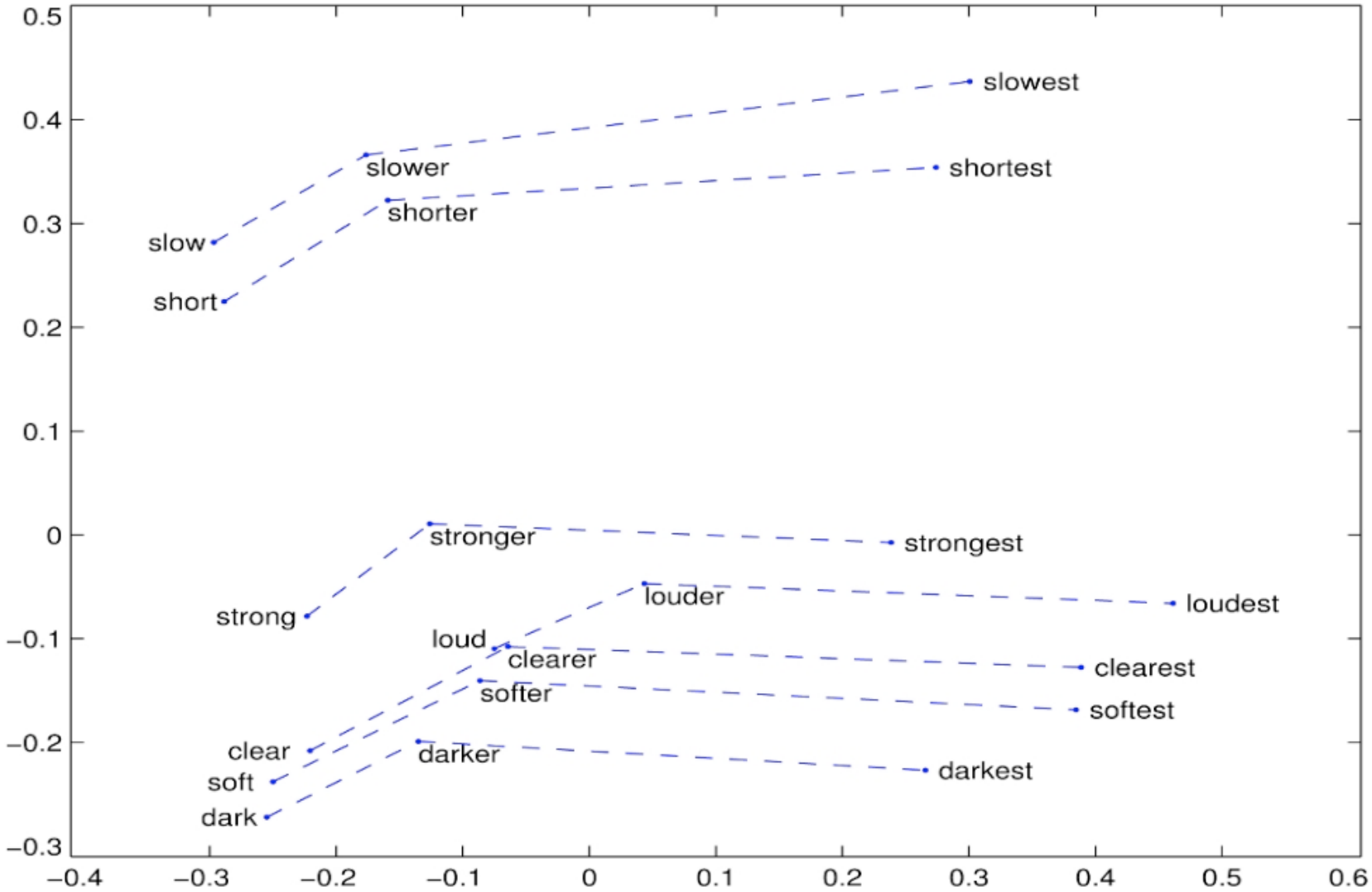
Linear Structures: Visualizations



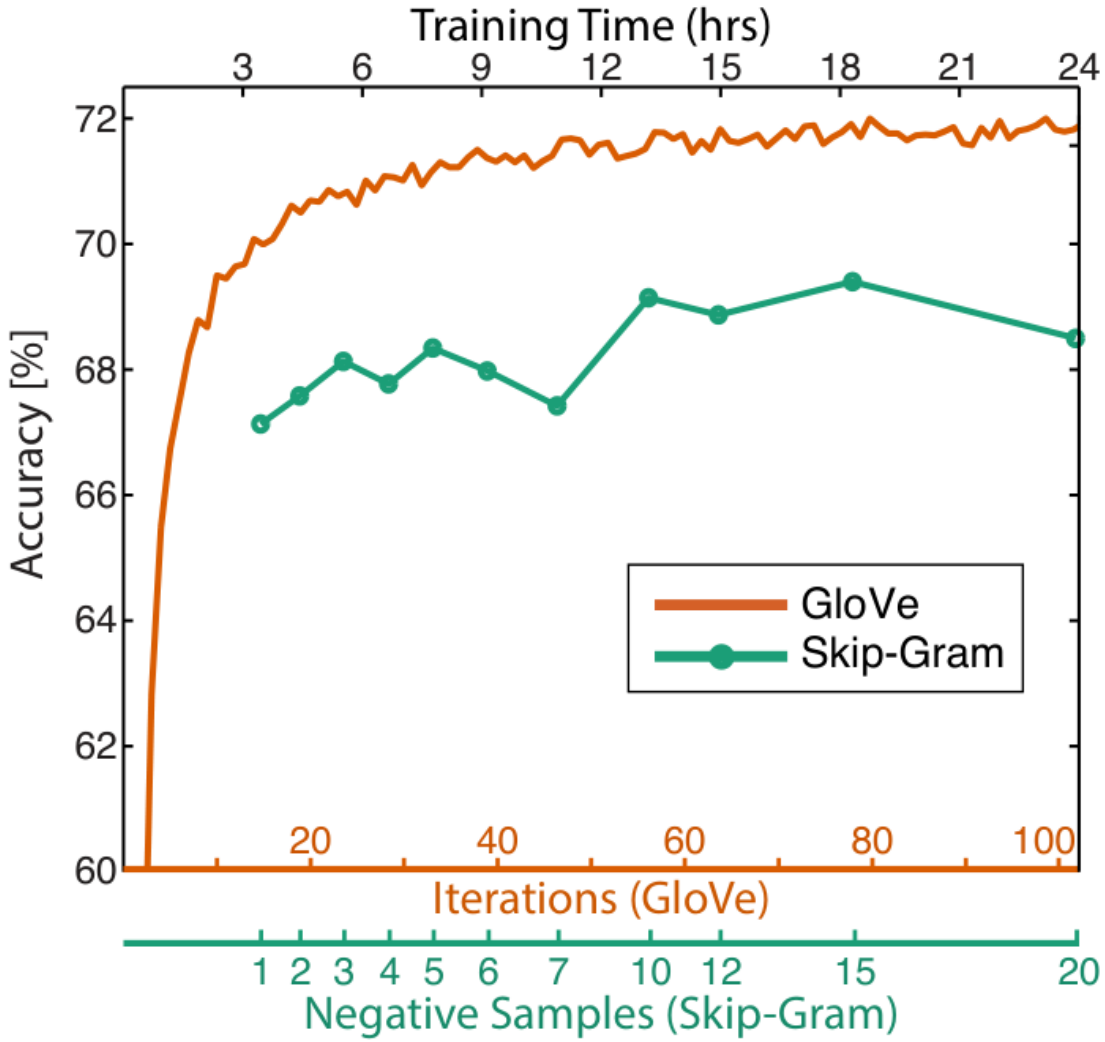
Linear Structures: Visualizations



Linear Structures: Visualizations



Analogy evaluation and hyperparameters



Word Embedding Conclusion

- Developed a model that can translate meaningful relationships between word-word co-occurrence probabilities into linear relations in the word vector space.
- GloVe shows the connection between Count! work and Predict! work – appropriate scaling of counts gives the properties and performance of Predict! models

Quiz

Select all systems and algorithms that involve some vectorized representation of words:

A. GloVe
B. BERT
C. Lesk’s algorithm
D. word2vec