
CMPT 413/713: Natural Language Processing

# Word Embeddings

Spring 2026
2026-01–19

Adapted from slides from Dan Jurafsky, Chris Manning, Danqi Chen and Karthik Narasimhan

# How to represent words?

In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel — a localist representation

one 1, the rest 0's

Words can be represented by one-hot vectors:

Each word is one dimension!

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

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

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

There is no way to encode similarity of words in these vectors!

# Representing words

- One-hot vectors

  cat = [0 0 0 1 0 0 0]
  dog = [0 0 0 0 0 1 0]

- Continuous embeddings

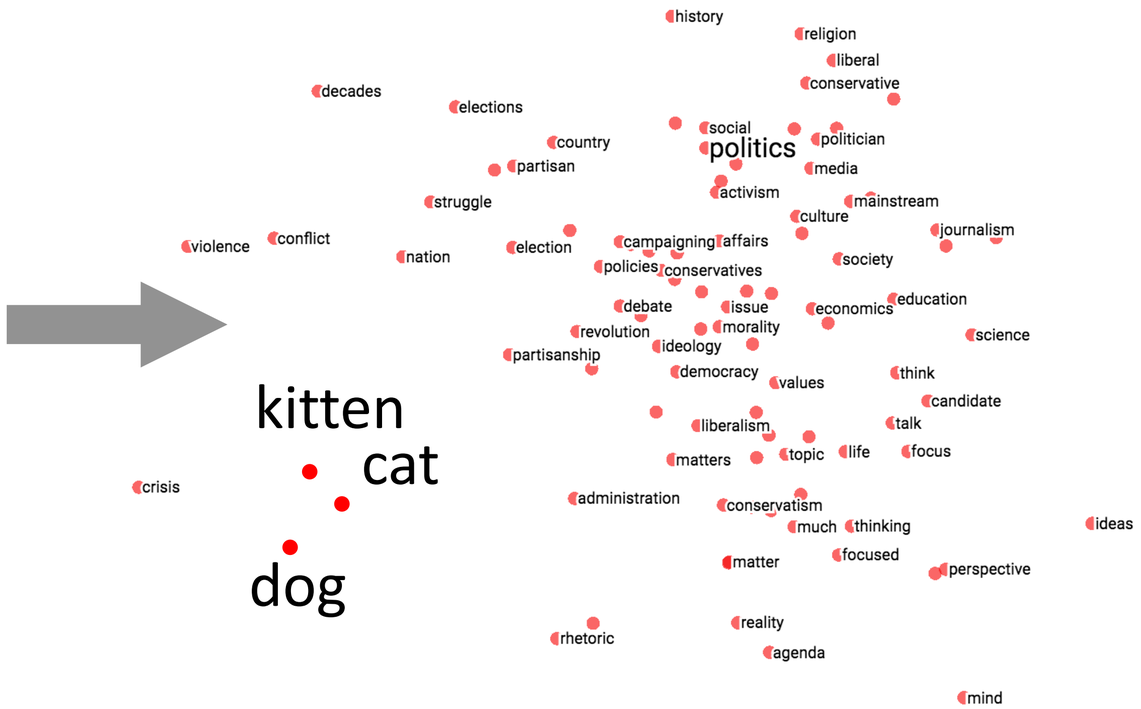
  cat = [0.04 1.79 -1.79 1.07 0.48]
  dog = [0.61 1.84 -1.12 0.52 0.53]

  Distributed representation

  (Meaning not isolated to one dimension)

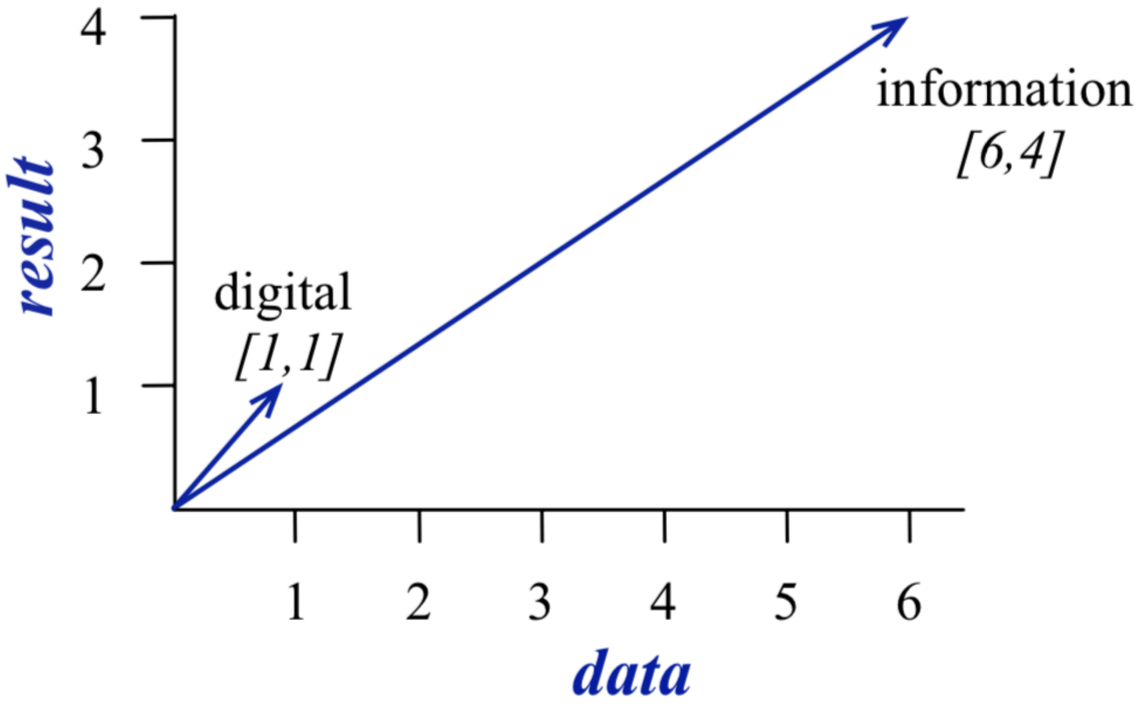
Representing meaning as vectors

- common representation space
- enables information sharing
- can be learned from data




Continuous space for words:
Similar words closer to each other

# Can measure similarity of words




$$cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$$

$$cos(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{V} u_i v_i}{\sqrt{\sum_{i=1}^{V} u_i^2}\sqrt{\sum_{i=1}^{V} v_i^2}}$$

# Representing words

- One-hot vectors

  cat = [0 0 0 1 0 0 0]
  dog = [0 0 0 0 0 1 0]

- Static embeddings

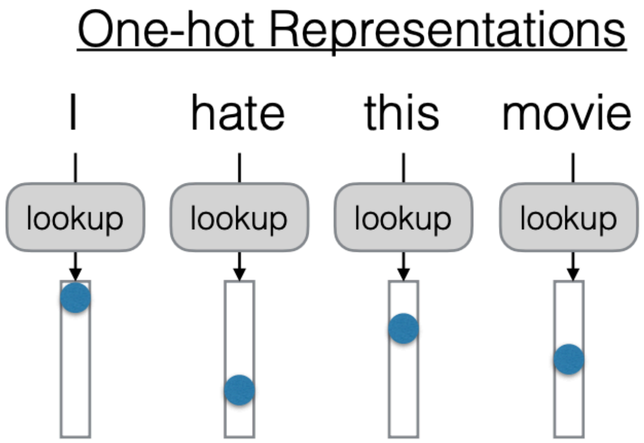
  cat = [0.04 1.79 -1.79 1.07 0.48]
  dog = [0.61 1.84 -1.12 0.52 0.53]

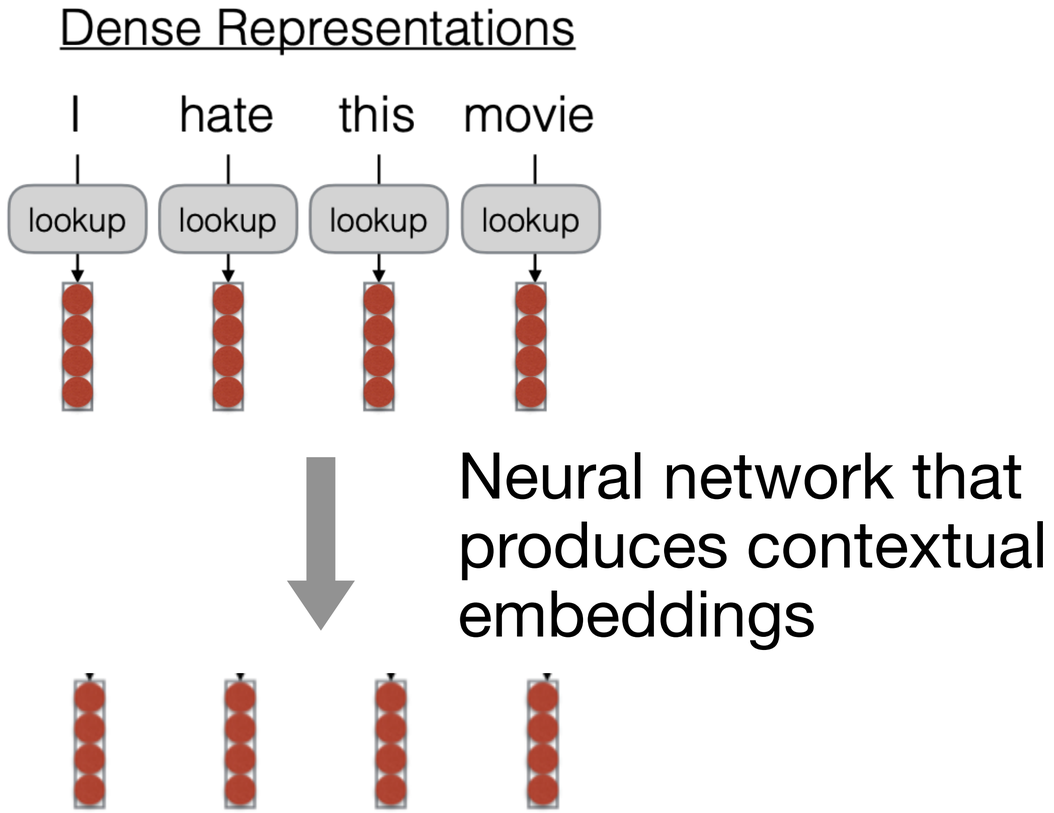
- Contextual embeddings
  - Token embedding depend on other tokens

$$\phi : V \rightarrow \mathbb{R}^d$$

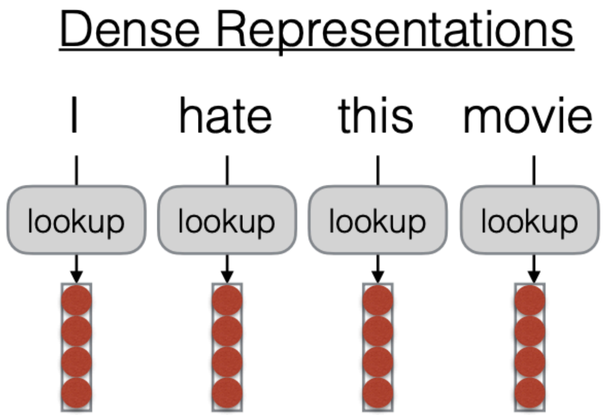
$$\phi : V^L \rightarrow \mathbb{R}^{d \times L}$$

# Words in sentences

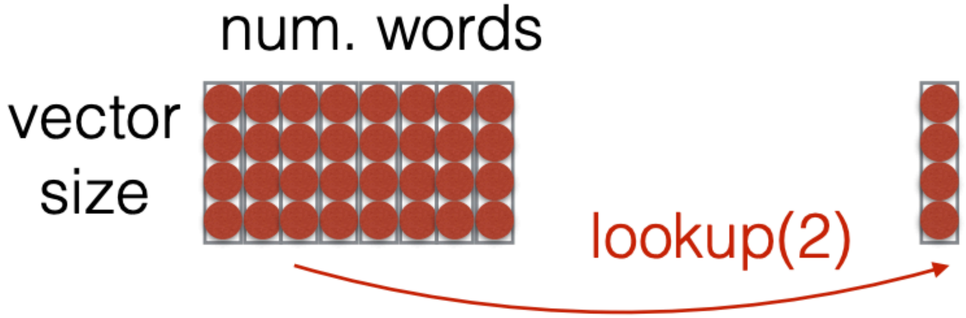


# Looking up embeddings

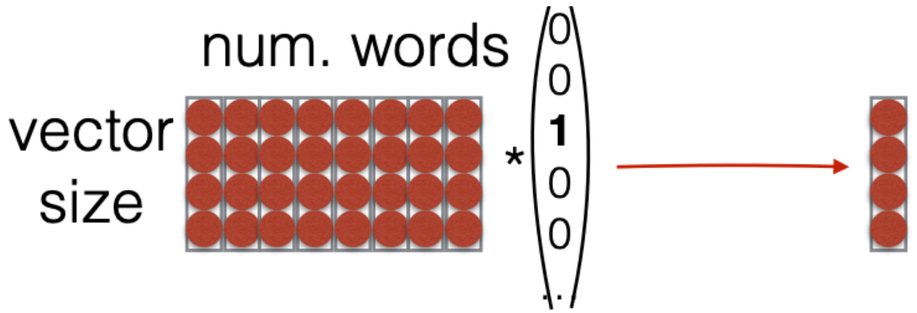



- Lookup: grabbing a vector from a big matrix of word embeddings




Typically faster

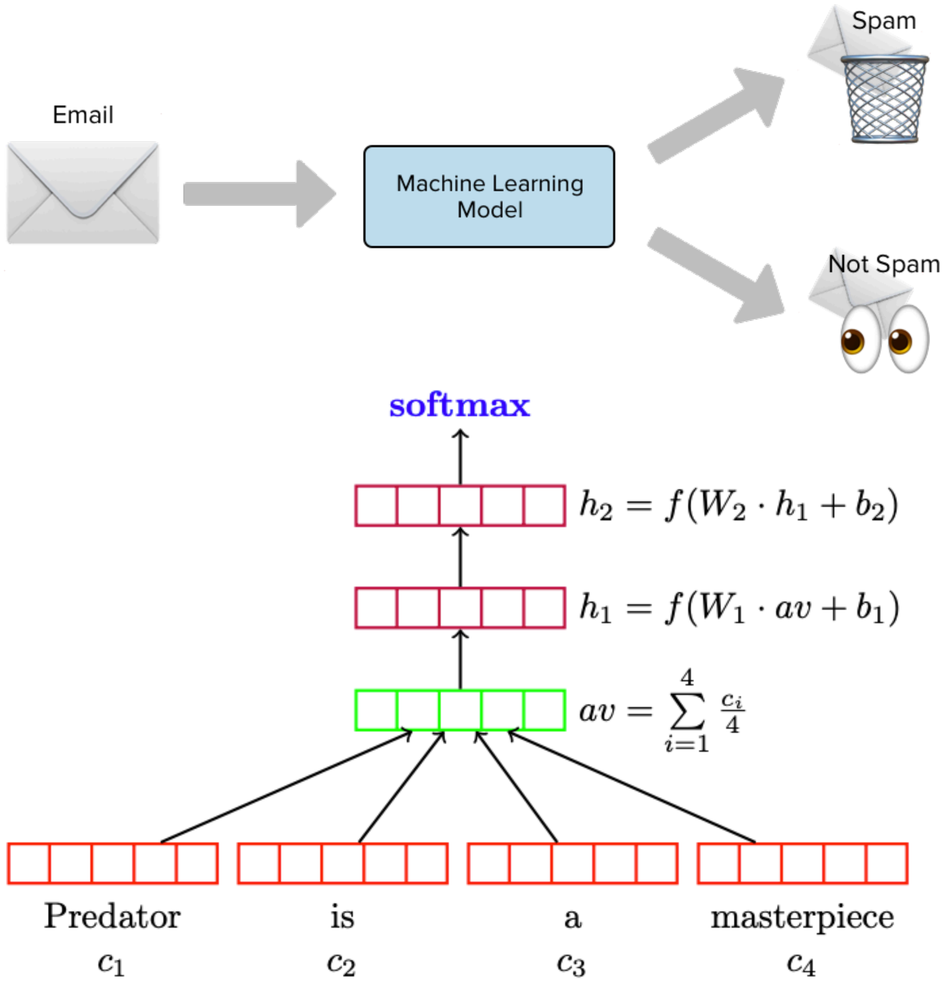
- Can also view lookup as multiplying by a “one-hot” vector



# Where do these word embeddings come from?

# Task specific training

- Assume you have annotated data specific to a task
- Initialize with random vectors
- Lookup table from word to vector
- Train your classifier
  - Classifier parameters are updated during training
  - These parameters include the word vectors!
  - After training, you get word vectors that are good for your task!



# Pretraining and task-specific fine-tuning

## Pretraining

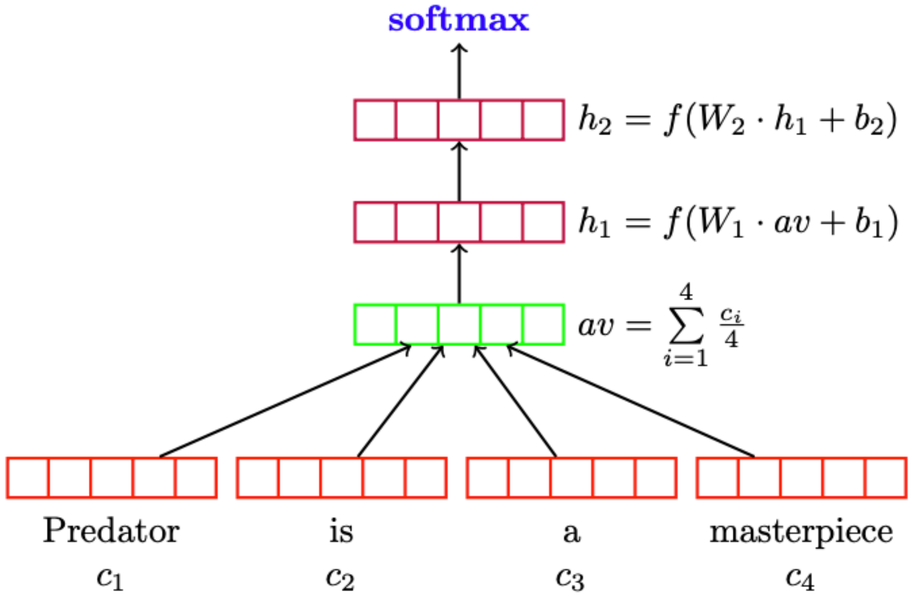
- Big pile of unlabeled text data!
- Lots of resources to train!

## Task-specific fine-tuning

- Annotated data specific to a task (usually small)
- Initialize with pre-trained model

# Summary of three options

- **Random + train** - Initialize with random embeddings and learn them when you train your classifier.
- **Pretrain + fixed** - Initialize with pretrained embeddings + keep them fixed
- **Pretrain + fine-tune** - Initialize with pretrained embeddings and then allow embedding weights to change as classifier is trained



# Neural Networks with Word Embeddings

# Estimating the probability of the next word

$$P(s) = P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t | w_{<t})$$

Statistical LM (n-grams) - used from 1990s to early 2010s

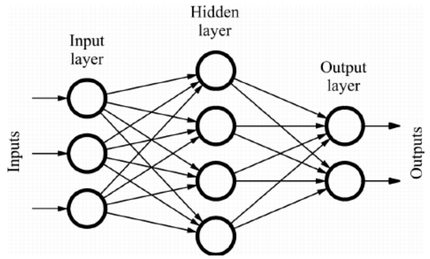
$$P(w_t | w_{<t}) \approx P(w_t | w_{t-n+1,t-1})$$

Neural LM: introduced in 2003, popularized in 2010s

$$P(w_t \mid w_{<t}) \approx P(w_t \mid \phi(w_1, \ldots, w_{t-1}))$$

Large language models (2020s):

- train with large amount of data (2020s)




Use neural network to encode the history and then estimate the probability of the next word

Word representations
- continuous vector representations
- learned from lots of data

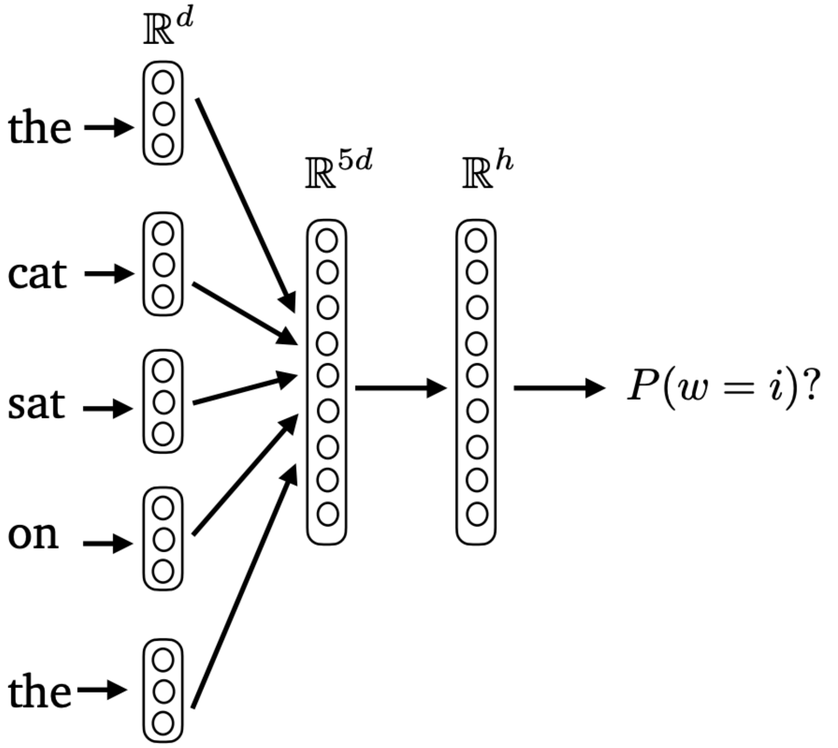
# Language modelling with neural networks

- Words are represented as continuous word embeddings
- Parameters are weights of the neural network
- Train to predict the next word given the previous words
  - For each word: classification problem over the entire vocabulary
  - Recall: use softmax to model probability of output class and cross-entropy loss for classification

$$P(y = c \mid x) = \frac{\exp(w_c \cdot x + b_c)}{\sum_{j=1}^{k} \exp(w_j \cdot x + b_j)}$$ softmax

$$L_{CE}(\hat{y}, y) = -\sum_{c=1}^{k} 1\{y = k\} \log P(y = k \mid x)$$

P(mat | the cat sat on the) = ?

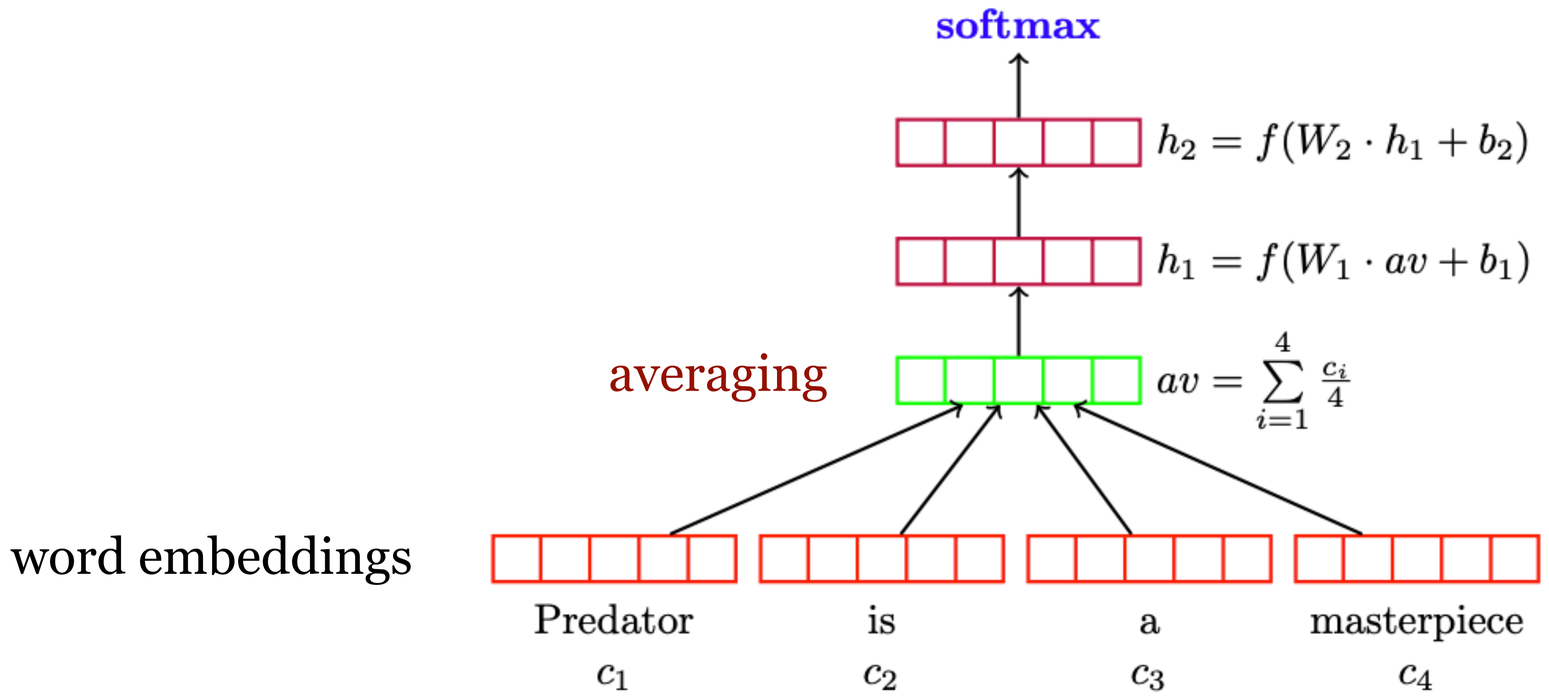



$$p(w_t \mid w_{<t}) \approx p(w_t \mid \phi(w_1, \ldots, w_{t-1}))$$

A Neural Probabilistic Language Model [Bengio et al. 2003]

# Neural Bag-of-Words (NBOW)

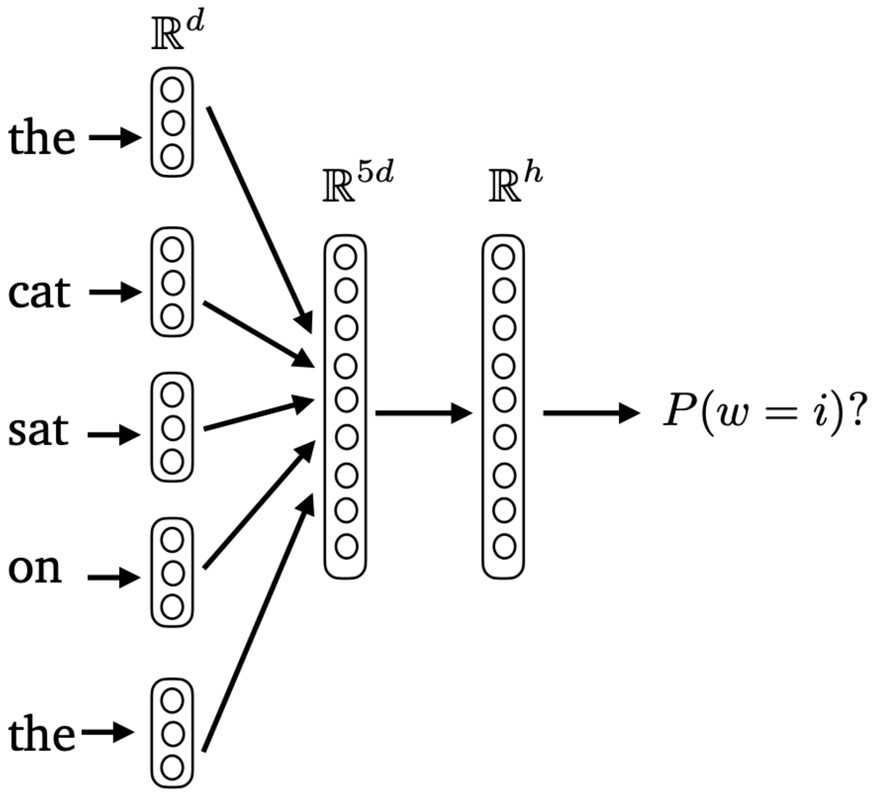
- Deep Averaging Networks (DAN) for Text Classification




(Iyyer et 2015): Deep Unordered Composition Rivals Syntactic Methods for Text Classification

# Feedforward Neural LMs

- P(mat | the cat sat on the) = ?




- Input layer (context size n = 5):

$$\mathbf{x} = [\mathbf{e}_{\text{the}}; \mathbf{e}_{\text{cat}}; \mathbf{e}_{\text{sat}}; \mathbf{e}_{\text{on}}; \mathbf{e}_{\text{the}}] \in \mathbb{R}^{dn}$$

concatenate word embeddings

- Hidden layer

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^h$$

- Output layer (softmax)

$$\mathbf{z} = \mathbf{U}\mathbf{h} \in \mathbb{R}^{|V|}$$

$$P(w = i \mid \text{context}) = softmax_i(\mathbf{z})$$

(Bengio et 2003): A Neural Probabilistic Language Model

# Language modelling with neural networks

- Words are represented as continuous word embeddings
- Parameters are weights of the neural network
- Train to predict the next word given the previous words
  - For each word: classification problem over the entire vocabulary
  - Recall: use softmax to model probability of output class and cross-entropy loss for classification
- Different neural network architectures
  - FFNs

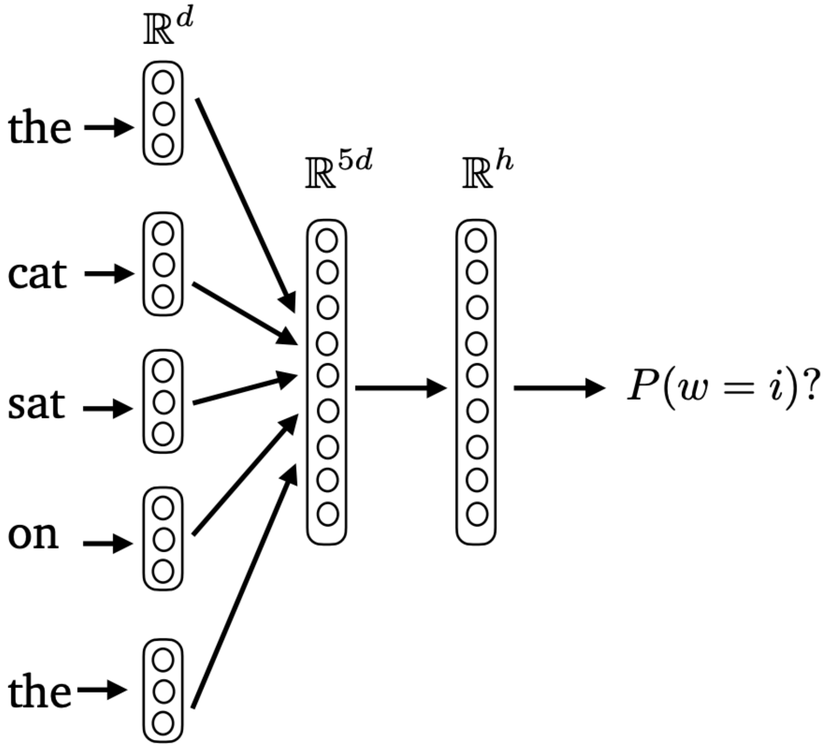
    $$P(w_t|w_{<t}) \approx P(w_t|\phi(w_{t-n+1,t-1}))$$

  - RNNs

    $$P(w_t|w_{<t}) \approx P(w_t|\mathbf{h}_{t-1}), \mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

  - Transformers

P(mat | the cat sat on the) = ?




$$p(w_t \mid w_{<t}) \approx p(w_t \mid \phi(w_1, \ldots, w_{t-1}))$$

A Neural Probabilistic Language Model [Bengio et al. 2003]

# Feedforward Neural Language Model (Neural BOW LM)

Issues: Fixed window, no parameter sharing

Larger window → more parameters

**output distribution**

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

**hidden layer**

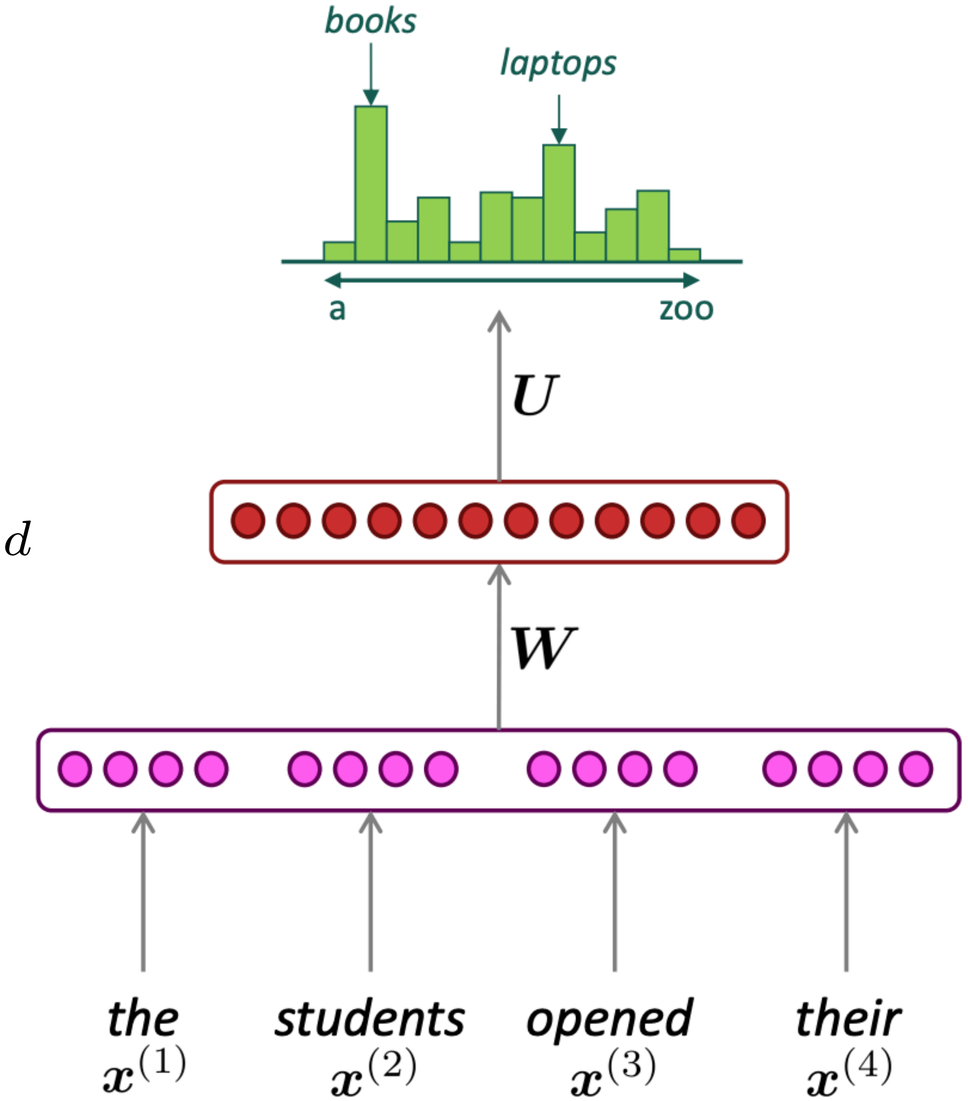
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1) \quad \mathbf{W} \in \mathbb{R}^{h \times nd}$$

**concatenated word embeddings**

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

**words / one-hot vectors**

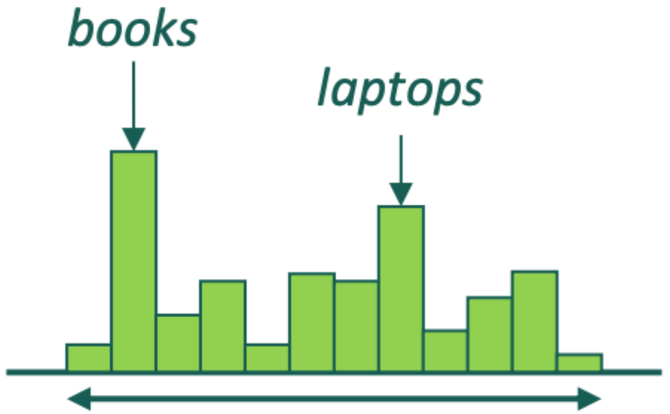
$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$




*Slide credit: Chris Manning*

# RNNs for Language Modeling

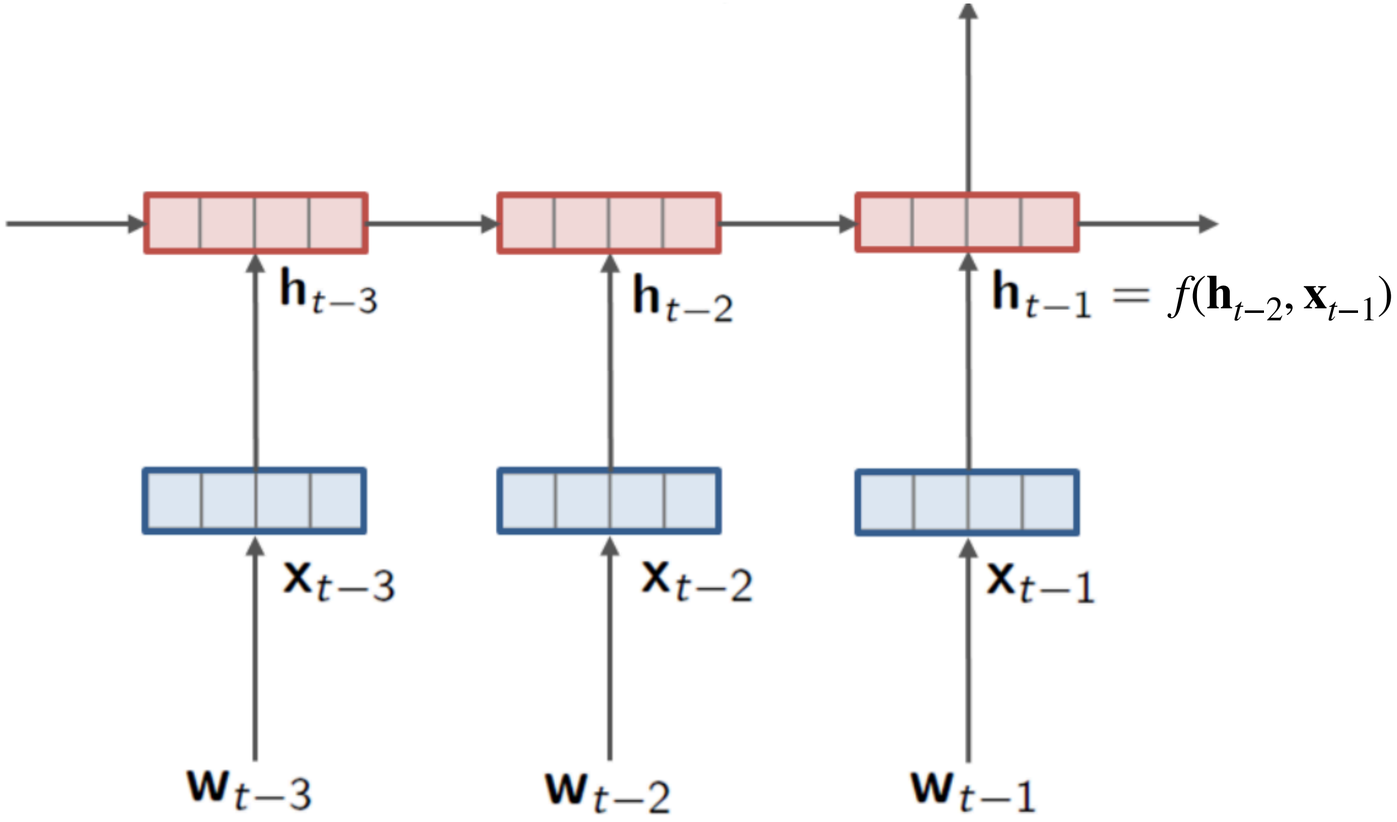
$$P(w_t|w_{<t}) \approx P(w_t|\mathbf{h}_{t-1}), \mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$



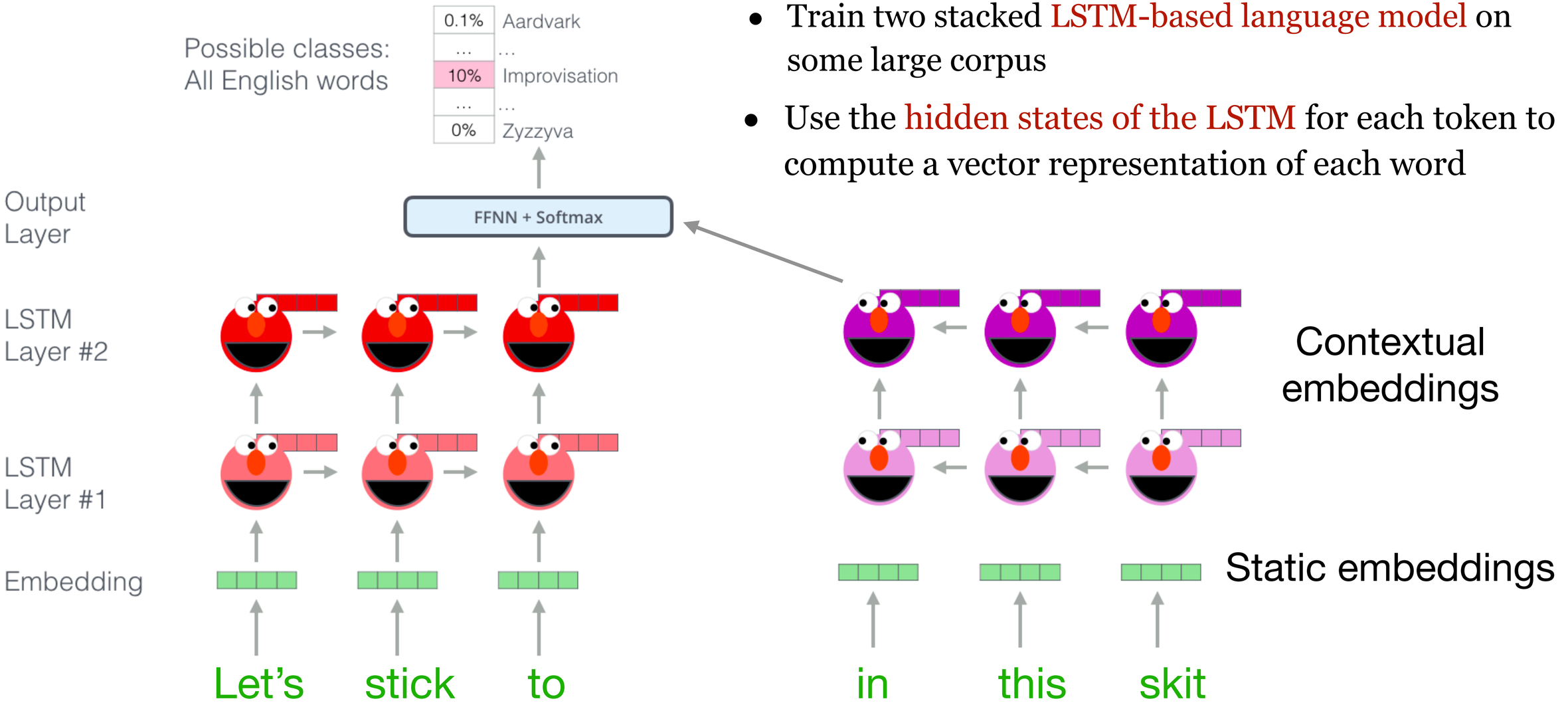

Use a RNN to

- capture the history of the previous words as a hidden state
- use hidden state to estimate the probability of the next word

$$P(w_t \,|\, w_1, \ldots, w_{t-1}) = P(w_t \,|\, h_{t-1})$$



# Learning contextualized word embeddings with RNNs: ELMo




- Train two stacked LSTM-based language model on some large corpus
- Use the hidden states of the LSTM for each token to compute a vector representation of each word

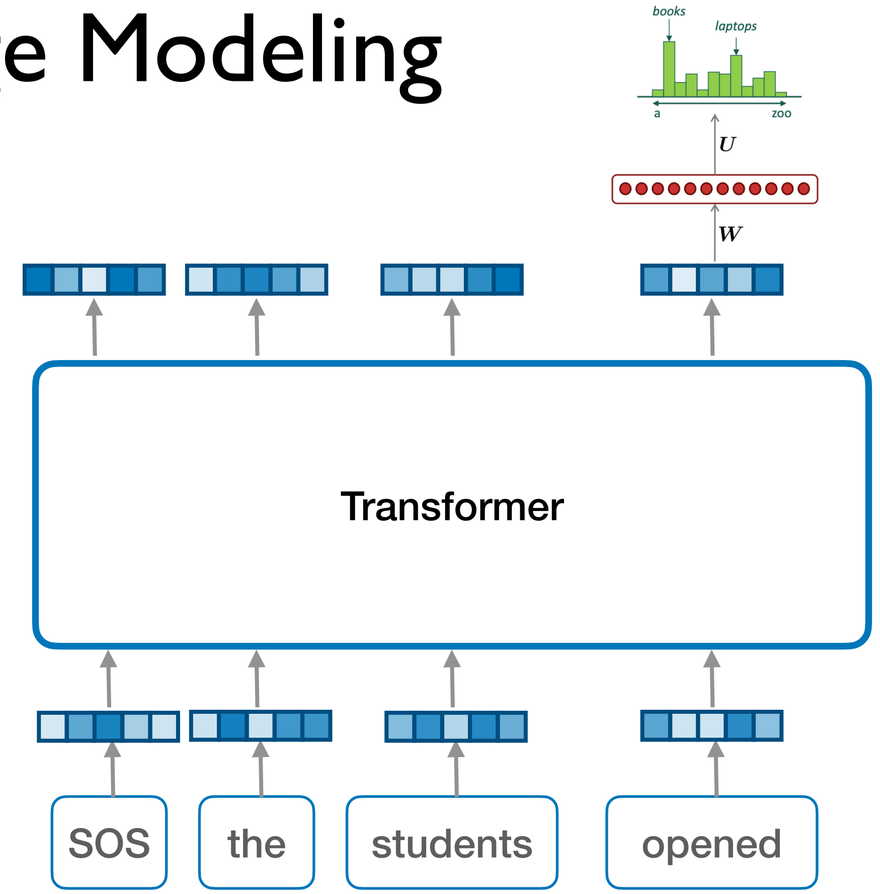
*(figure credit: Jay Alammar http://jalammar.github.io/illustrated-bert/)*

Deep contextualized word representations [Peter et al. 2018]

# Transformers for Language Modeling

Contextual embeddings

Static embeddings




$$P(w_t|w_{<t}) \approx P(w_t|\phi(w_{t-n+1,t-1}))$$

# How does this pretraining work?

Big pile of unlabeled text data!  Learn using language modelling!

# Representing words by their context

**Distributional hypothesis**: words that occur in similar contexts tend to have similar meanings

J.R.Firth 1957

- "You shall know a word by the company it keeps"
- One of the most successful ideas of modern statistical NLP!

*…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***.

# Distributional hypothesis

“tejuino”

C1: A bottle of ____ is on the table.

C2: Everybody likes ____.

C3: Don’t have ____ before you drive.

C4: We make ____ out of corn.

# Distributional hypothesis

"words that occur in similar contexts tend to have similar meanings"

C1: A bottle of ____ is on the table.

C2: Everybody likes ____.

C3: Don't have ____ before you drive.

C4: We make ____ out of corn.

| | C1 | C2 | C3 | C4 |
|---|---|---|---|---|
| tejuino | 1 | 1 | 1 | 1 |
| loud | 0 | 0 | 0 | 0 |
| motor-oil | 1 | 0 | 0 | 0 |
| tortillas | 0 | 1 | 0 | 1 |
| choices | 0 | 1 | 0 | 0 |
| wine | 1 | 1 | 1 | 0 |

Use as context: other words that appear in a span around the target word

# How are these embeddings learned?

Get embeddings by counting or by predicting (i.e. training a classifier)!

C1: A bottle of ____ is on the table.

C2: Everybody likes ____.

C3: Don't have ____ before you drive.

C4: We make ____ out of corn.

| | C1 | C2 | C3 | C4 |
|---|---|---|---|---|
| tejuino | 1 | 1 | 1 | 1 |
| loud | 0 | 0 | 0 | 0 |
| motor-oil | 1 | 0 | 0 | 0 |
| tortillas | 0 | 1 | 0 | 1 |
| choices | 0 | 1 | 0 | 0 |
| wine | 1 | 1 | 1 | 0 |

Use as context: other words that appear in a span around the target word

"words that occur in similar contexts tend to have similar meanings"

# How are these embeddings learned?

Get embeddings by counting or by predicting (i.e. training a classifier)!

C1: A bottle of ____ is on the table.

C2: Everybody likes ____.

C3: Don't have ____ before you drive.

C4: We make ____ out of corn.

| | C1 | C2 | C3 | C4 |
|---|---|---|---|---|
| tejuino | 1 | 1 | 1 | 1 |
| loud | 0 | 0 | 0 | 0 |
| motor-oil | 1 | 0 | 0 | 0 |
| tortillas | 0 | 1 | 0 | 1 |
| choices | 0 | 1 | 0 | 0 |
| wine | 1 | 1 | 1 | 0 |

Use as context: other words that appear in a span around the target word

"words that occur in similar contexts tend to have similar meanings"

# Representing words as vectors

- What we are aiming for:
  - Each word is a vector
  - Similar words are “nearby in space”
- Our first solution: use context vectors to represent the meaning of words

word-word (term-context) co-occurrence matrix

sugar, a sliced lemon, a tablespoonful of **apricot** jam, a pinch each of,
their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the purpose of gathering data and **information** necessary for the study authorized in the

| | aardvark | computer | data | pinch | result | sugar | ... |
|---|---|---|---|---|---|---|---|
| apricot | 0 | 0 | 0 | 1 | 0 | 1 | |
| pineapple | 0 | 0 | 0 | 1 | 0 | 1 | |
| digital | 0 | 2 | 1 | 0 | 1 | 0 | |
| information | 0 | 1 | 6 | 0 | 4 | 0 | |

Example from: Dan Jurafsky

# Problem with raw frequencies

**Problem**: using raw frequency counts is not always very good..

- if sugar appears a lot near apricot, that's useful information.
- But overly frequent words like *the*, *it*, or *they* are not very informative about the context

**Solution**: let's weight the counts!

- TF-IDF = Traditional method used in document retrieval
- PPMI = Positive Pointwise Mutual Information

# PPMI

Do two events co-occur more than if they were independent?

PPMI = Positive Pointwise Mutual Information

Joint probability

$$\text{PPMI}(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0)$$

Marginals

PMI ranges from
$-\infty$ to $+\infty$

Negative if co-occurs
less than if independent

Negative values are problematic so cap bottom to 0

# Sparse vs dense vectors

Vectors we get from word-word (term-context) co-occurrence matrix are

- **long** (length |V|= 20,000 to 50,000)
- **sparse** (most elements are zero)

True for both one-hot and PPMI vectors

Alternative: we want to represent words as

- **short** (50-300 dimensional)
- **dense** (real-valued) vectors

$$\text{employees} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 10.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

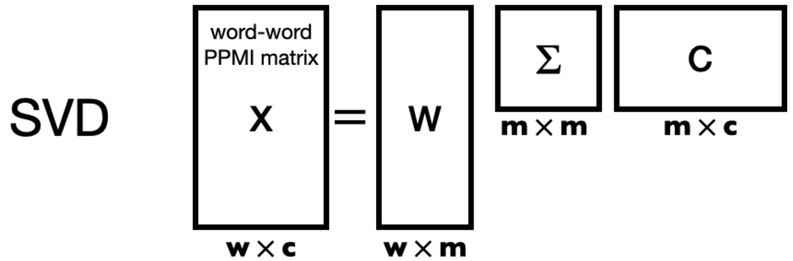
Distributed representation


More memory efficient and easier to work with
Capture similarity between words better

# Why dense vectors?

- Short vectors are easier to use as features in ML systems
- Dense vectors may generalize better than storing explicit counts
- They do better at capturing synonymy
  - $w_1$ co-occurs with “car”, $w_2$ co-occurs with “automobile”

- Different methods for getting dense vectors:
  - Singular value decomposition (SVD)
  - word2vec and friends: “learn” the vectors!

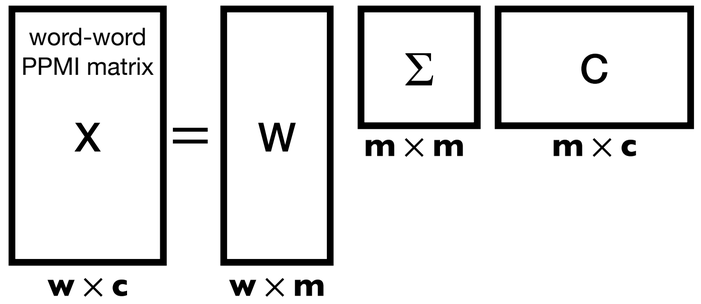



Count based method
(known since the 1990s)

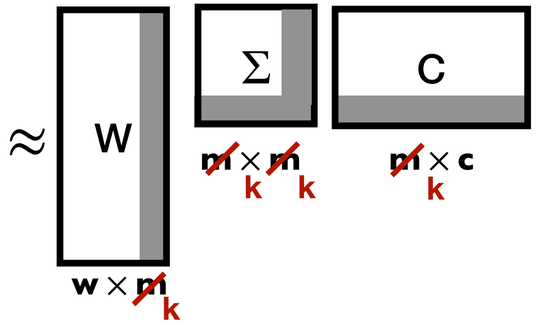
# Using SVD for obtaining dense vectors

1) SVD

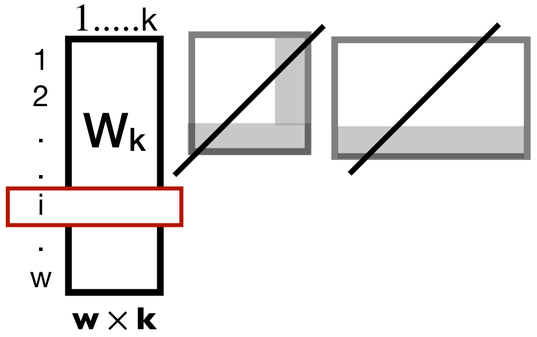
$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$$




2) Truncation




3) Embeddings




embedding for word i

SVD = Singular value decomposition

$\boldsymbol{\Sigma}$ is a diagonal matrix with singular values:

$\sigma_1, \ldots, \sigma_i, \ldots, \sigma_m$ where $m$ is the rank of the matrix $\mathbf{X}$

$\mathbf{U} = \mathbf{W}$ has orthonormal columns, $\mathbf{V}^\top = \mathbf{C}$ has orthonormal rows

In our case, $\mathbf{X}$ is a $|V| \times |V|$ matrix. Assume $m = |V|$

Truncation:

- Select the first $k$ columns of $\mathbf{W}$ to get $k$-dimensional row vectors
- Note that the singular values are ordered from largest to smallest, which each singular value representing the variance captured by that dimension
- So the first $k$ dimensions are the dimensions with the most variance

Finally, we take $i$th row of $\mathbf{W}_k$ as the embedding of word $i$

Note there are variants where the word embedding matrix is taken to be $\mathbf{W}_k\boldsymbol{\Sigma}^\lambda$ (with common values being $\lambda = 1$, 0.5, or 0)

traditional

# What is wrong with using SVD?

- Computational complexity is high: $O(|V|^3)$
- Cannot be trained as part of a larger model
  - Not a component that can be part of a larger neural network
- Cannot be trained discriminatively for a particular task

# Why dense vectors?

- Short vectors are easier to use as features in ML systems
- Dense vectors may generalize better than storing explicit counts
- They do better at capturing synonymy
  - $w_1$ co-occurs with “car”, $w_2$ co-occurs with “automobile”

- Different methods for getting dense vectors:
  - Singular value decomposition (SVD)
  - word2vec and friends: “learn” the vectors!

# How are these embeddings learned?

Learn predictor to fill in the blank!

C1: A bottle of ____ is on the table.

- Represent each word as a vector
- Train classifier to predict word using context words.
- During training, the word vector is updated so that it is possible to predict the center word using the context words

| | bottle | likes | before | make | corn |
|---|---|---|---|---|---|
| tejuino | 1 | 1 | 1 | 1 | 1 |
| loud | 0 | 0 | 0 | 0 | 0 |
| motor-oil | 1 | 0 | 0 | 0 | 0 |
| tortillas | 0 | 1 | 0 | 1 | 1 |
| choices | 0 | 1 | 0 | 0 | 0 |
| wine | 1 | 1 | 1 | 0 | 0 |

Use as context: other words that appear in a span around the target word

"words that occur in similar contexts tend to have similar meanings"

# Many different ways to learn the representations

- From context words, predict target word (Masked LM)

  A bottle of ____ is on the table.

- From target word, predict other context words.

  What words go with “tejuino”?

- From previous words, predict next (target) word (Traditional LM)

  A bottle of ____

  A bottle of tejuino is on the ____

# Many different ways to learn the representations

- Different architectures and models to learn the representations
- Get sentence embeddings by combining word embeddings
- Two types of word embeddings
  - Static - mapping of word to embedding
  - Contextual - embedding of word changes based on the context

# Summary

- Representing words as vectors
  - One-hot vectors vs vectors built using context
  - Cosine similarity for measuring similarity between words
- Using context to represent words
  - Distributional hypothesis:

    words that occur in similar contexts tend to have similar meanings
- Co-occurrence matrix
  - Raw counts
  - TF-IDF (term-frequency inverse-document-frequency)
  - PPMI (positive pointwise mutual information)
- Dense vectors via SVD or learned by predicting the missing word

# Notes on count based word vectors

# Representing words as vectors

- What we are aiming for:
  - Each word is a vector
  - Similar words are “nearby in space”
- Our first solution: use context vectors to represent the meaning of words

word-word (term-context) co-occurrence matrix

sugar, a sliced lemon, a tablespoonful of **apricot** jam, a pinch each of,
their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the purpose of gathering data and **information** necessary for the study authorized in the

| | aardvark | computer | data | pinch | result | sugar | ... |
|---|---|---|---|---|---|---|---|
| apricot | 0 | 0 | 0 | 1 | 0 | 1 | |
| pineapple | 0 | 0 | 0 | 1 | 0 | 1 | |
| digital | 0 | 2 | 1 | 0 | 1 | 0 | |
| information | 0 | 1 | 6 | 0 | 4 | 0 | |

Example from: Dan Jurafsky

# Problem with raw frequencies

**Problem**: using raw frequency counts is not always very good..

- if sugar appears a lot near apricot, that's useful information.
- But overly frequent words like *the*, *it*, or *they* are not very informative about the context

**Solution**: let's weight the counts!

- TF-IDF = Traditional method used in document retrieval
- PPMI = Positive Pointwise Mutual Information

# Problem with raw frequencies

**Problem**: using raw frequency counts is not always very good..

- if sugar appears a lot near apricot, that's useful information.
- But overly frequent words like *the*, *it*, or *they* are not very informative about the context

**Solution**: let's weight the counts!

- TF-IDF = Traditional method used in document retrieval
- PPMI = Positive Pointwise Mutual Information

# Tf-idf

**Term-document matrix**

count$(t, d)$ = count of times term $t$ occurred in document $d$

| | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|---|---|---|---|---|
| **battle** | 1 | 0 | 7 | 13 |
| **good** | 114 | 80 | 62 | 89 |
| **fool** | 36 | 58 | 1 | 4 |
| **wit** | 20 | 15 | 2 | 3 |

- Documents are represented by the column vectors

# Tf-idf

**Term-document matrix**

count$(t, d)$ = count of times term $t$ occurred in document $d$

| | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|---|---|---|---|---|
| **battle** | 1 | 0 | 7 | 13 |
| **good** | 114 | 80 | 62 | 89 |
| **fool** | 36 | 58 | 1 | 4 |
| **wit** | 20 | 15 | 2 | 3 |

- Documents are represented by the column vectors
- Words are represented by row vectors

Tf-idf: re-weighing scheme for information retrieval

- Down-weighs frequent words such as *this*, *that*, *the*

# Tf-idf combine two factors

**tf: term frequency**. frequency count (usually log-transformed):

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

**idf: inverse document frequency**:

$$\mathrm{idf}_i = \log\left(\frac{N}{\mathrm{df}_i}\right)$$

Total # of docs in collection

# of docs that have word i

+1 (avoid 0 in denominator)

There are variations on the exact formulation of tf and idf

Words like "the" or "it" will have very low idf

tf-idf value for word t in document d

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

# Raw counts vs tf-idf

| Word | df | idf |
|---|---|---|
| Romeo | 1 | 1.57 |
| salad | 2 | 1.27 |
| Falstaff | 4 | 0.967 |
| forest | 12 | 0.489 |
| battle | 21 | 0.246 |
| wit | 34 | 0.037 |
| fool | 36 | 0.012 |
| good | 37 | 0 |
| sweet | 37 | 0 |

37 plays total

## Raw Counts

| | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|---|---|---|---|---|
| **battle** | 1 | 0 | 7 | 13 |
| **good** | 114 | 80 | 62 | 89 |
| **fool** | 36 | 58 | 1 | 4 |
| **wit** | 20 | 15 | 2 | 3 |

## Tf-idf

| | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|---|---|---|---|---|
| **battle** | 0.074 | 0 | 0.22 | 0.28 |
| **good** | 0 | 0 | 0 | 0 |
| **fool** | 0.019 | 0.021 | 0.0036 | 0.0083 |
| **wit** | 0.049 | 0.044 | 0.018 | 0.022 |

Common words are down weighted

# Tf-idf summary

- Tf-idf: term-frequency inverse-document frequency
- Re-weighing scheme originally designed for information retrieval
  - Down-weighs frequent words such as *this*, *that*, *the*
- Can be used to
  - measure the similarity between words
  - measure the similarity between documents
  - measure the similarity between a query (mini-document) and a document
  - as features for classifiers
- Useful to know about (good baseline method)
- But not typically used for word embeddings

# Problem with raw frequencies

**Problem**: using raw frequency counts is not always very good..

- if sugar appears a lot near apricot, that's useful information.
- But overly frequent words like *the*, *it*, or *they* are not very informative about the context

**Solution**: let's weight the counts!

- TF-IDF = Traditional method used in document retrieval
- PPMI = Positive Pointwise Mutual Information

# PPMI

Do two events co-occur more than if they were independent?

PPMI = Positive Pointwise Mutual Information

Joint probability

$$\text{PPMI}(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0)$$

Marginals

PMI ranges from
$-\infty$ to $+\infty$

Negative if co-occurs
less than if independent

Negative values are problematic so cap bottom to 0

# Computing PPMI on a term-context matrix

- Matrix $F$ with $W$ rows (words) and $C$ columns (contexts)
- $f_{ij}$ is # of times $w_i$ occurs in context $c_j$

| | aardvark | computer | data | pinch | result | sugar |
|---|---|---|---|---|---|---|
| apricot | 0 | 0 | 0 | 1 | 0 | 1 |
| pineapple | 0 | 0 | 0 | 1 | 0 | 1 |
| digital | 0 | 2 | 1 | 0 | 1 | 0 |
| information | 0 | 1 | 6 | 0 | 4 | 0 |

$$ppmi_{ij} = \begin{cases} pmi_{ij} & \text{if } pmi_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Joint probability

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

Marginals

$$p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

# Computing PPMI on a term-context matrix

count($w_i$, $context_j$)

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

Total count N

**Count(w,context)**

| | computer | data | pinch | result | sugar |
|---|---|---|---|---|---|
| apricot | 0 | 0 | 1 | 0 | 1 |
| pineapple | 0 | 0 | 1 | 0 | 1 |
| digital | 2 | 1 | 0 | 1 | 0 |
| information | 1 | 6 | 0 | 4 | 0 |

$p(w = \text{information}, c = \text{data}) = 6/19 = 0.32$

**p(w,context)** **p(w)**

| | computer | data | pinch | result | sugar | |
|---|---|---|---|---|---|---|
| apricot | 0.00 | 0.00 | 0.05 | 0.00 | 0.05 | 0.11 |
| pineapple | 0.00 | 0.00 | 0.05 | 0.00 | 0.05 | 0.11 |
| digital | 0.11 | 0.05 | 0.00 | 0.05 | 0.00 | 0.21 |
| information | 0.05 | 0.32 | 0.00 | 0.21 | 0.00 | 0.58 |
| **p(context)** | 0.16 | 0.37 | 0.11 | 0.26 | 0.11 | |

$$p(c_j) = \frac{\sum_{i=1}^{W} f_{ij}}{N}$$

$$p(w_i) = \frac{\sum_{j=1}^{C} f_{ij}}{N}$$

$p(c = \text{data}) = 7/19 = 0.37$

$p(w = \text{information}) = 11/19 = 0.58$

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

$$pmi(w = \text{information}, c = \text{data}) = \log_2 \frac{0.32}{0.37 \times 0.58} = 0.58$$

**p(w,context)**

| | computer | data | pinch | result | sugar | **p(w)** |
|---|---|---|---|---|---|---|
| apricot | 0.00 | 0.00 | 0.05 | 0.00 | 0.05 | 0.11 |
| pineapple | 0.00 | 0.00 | 0.05 | 0.00 | 0.05 | 0.11 |
| digital | 0.11 | 0.05 | 0.00 | 0.05 | 0.00 | 0.21 |
| information | 0.05 | 0.32 | 0.00 | 0.21 | 0.00 | 0.58 |
| **p(context)** | 0.16 | 0.37 | 0.11 | 0.26 | 0.11 | |

**PPMI(w,context)**

| | computer | data | pinch | result | sugar |
|---|---|---|---|---|---|
| apricot | - | - | 2.25 | - | 2.25 |
| pineapple | - | - | 2.25 | - | 2.25 |
| digital | 1.66 | 0.00 | - | 0.00 | - |
| information | 0.00 | 0.57 | - | 0.47 | - |

Actually 0.57 using full precision

$$ppmi_{ij} = \begin{cases} pmi_{ij} & \text{if } pmi_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

# Issues with PMI

PMI is biased toward infrequent events

- Very rare words have very high PMI values

Two solutions:

- Weighted PMI: Give rare words slightly higher probabilities
- Use add-one smoothing (which has a similar effect)

# Weighting PMI

Give rare context words slightly higher probability

Raise the context probabilities to $\alpha = 0.75$:

$$\text{PPMI}_\alpha(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0)$$

$$P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$$

$P(c)$ is low, so PPMI is high

Higher $P_\alpha(c) \rightarrow$ lower $\text{PPMI}_\alpha$

This helps because $P_\alpha(c) > P(c)$ for rare $c$

Consider two events, $P(a) = .99$ and $P(b) = .01$

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

# Smoothed PPMI (add-2)

- Use Laplace smoothing
- Alternative to using weighted PPMI

**Add-2 Smoothed Count(w,context)**

| | computer | data | pinch | result | sugar |
|---|---|---|---|---|---|
| apricot | 2 | 2 | 3 | 2 | 3 |
| pineapple | 2 | 2 | 3 | 2 | 3 |
| digital | 4 | 3 | 2 | 3 | 2 |
| information | 3 | 8 | 2 | 6 | 2 |

**p(w,context) [add-2]**

| | computer | data | pinch | result | sugar | **p(w)** |
|---|---|---|---|---|---|---|
| apricot | 0.03 | 0.03 | 0.05 | 0.03 | 0.05 | 0.20 |
| pineapple | 0.03 | 0.03 | 0.05 | 0.03 | 0.05 | 0.20 |
| digital | 0.07 | 0.05 | 0.03 | 0.05 | 0.03 | 0.24 |
| information | 0.05 | 0.14 | 0.03 | 0.10 | 0.03 | 0.36 |
| **p(context)** | 0.19 | 0.25 | 0.17 | 0.22 | 0.17 | |

# PPMI versus add-2 smoothed PPMI

**PPMI(w,context)**

| | computer | data | pinch | result | sugar |
|---|---|---|---|---|---|
| apricot | - | - | 2.25 | - | 2.25 |
| pineapple | - | - | 2.25 | - | 2.25 |
| digital | 1.66 | 0.00 | - | 0.00 | - |
| information | 0.00 | 0.57 | - | 0.47 | - |

**PPMI(w,context) [add-2]**

| | computer | data | pinch | result | sugar |
|---|---|---|---|---|---|
| apricot | 0.00 | 0.00 | 0.56 | 0.00 | 0.56 |
| pineapple | 0.00 | 0.00 | 0.56 | 0.00 | 0.56 |
| digital | 0.62 | 0.00 | 0.00 | 0.00 | 0.00 |
| information | 0.00 | 0.58 | 0.00 | 0.37 | 0.00 |

# Building word vectors by counting

- Build word-word (term-context) co-occurrence matrix

| | computer | data | result | pie | sugar |
|---|---|---|---|---|---|
| **cherry** | 2 | 8 | 9 | 442 | 25 |
| **strawberry** | 0 | 0 | 1 | 60 | 19 |
| **digital** | 1670 | 1683 | 85 | 5 | 4 |
| **information** | 3325 | 3982 | 378 | 5 | 13 |

Raw counts

| | computer | data | result | pie | sugar |
|---|---|---|---|---|---|
| **cherry** | 0 | 0 | 0 | 4.38 | 3.30 |
| **strawberry** | 0 | 0 | 0 | 4.10 | 5.51 |
| **digital** | 0.18 | 0.01 | 0 | 0 | 0 |
| **information** | 0.02 | 0.09 | 0.28 | 0 | 0 |

Positive pointwise mutual information

Practically, use smoothed/weighted PPMI to help with rare words having very high PMI values

$$\text{PPMI}(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0)$$

Vectors are very long, sparse. How to get short dense vectors?

# Sparse vs dense vectors

Vectors we get from word-word (term-context) co-occurrence matrix are

- **long** (length |V|= 20,000 to 50,000)
- **sparse** (most elements are zero)

True for both one-hot and PPMI vectors

Alternative: we want to represent words as

- **short** (50-300 dimensional)
- **dense** (real-valued) vectors

$$\text{employees} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 10.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

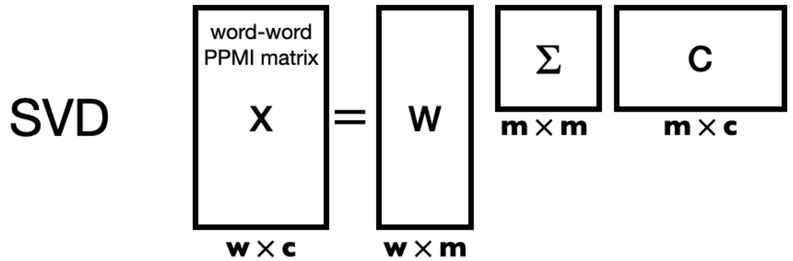
Distributed representation


More memory efficient and easier to work with
Capture similarity between words better

# Why dense vectors?

- Short vectors are easier to use as features in ML systems
- Dense vectors may generalize better than storing explicit counts
- They do better at capturing synonymy
  - $w_1$ co-occurs with “car”, $w_2$ co-occurs with “automobile”

- Different methods for getting dense vectors:
  - Singular value decomposition (SVD)
  - word2vec and friends: “learn” the vectors!

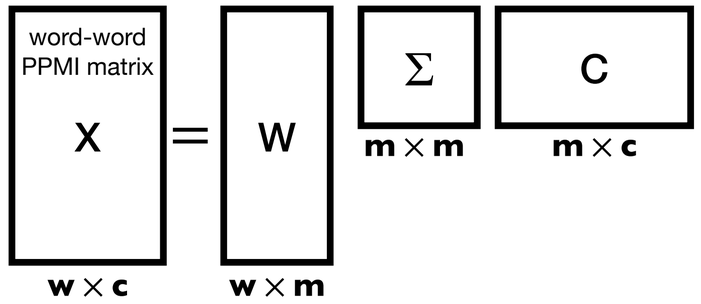



Count based method
(known since the 1990s)

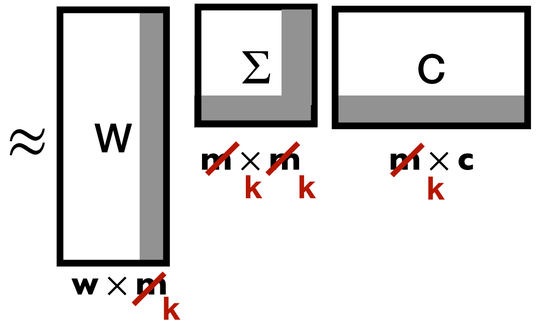
# Using SVD for obtaining dense vectors

1) SVD

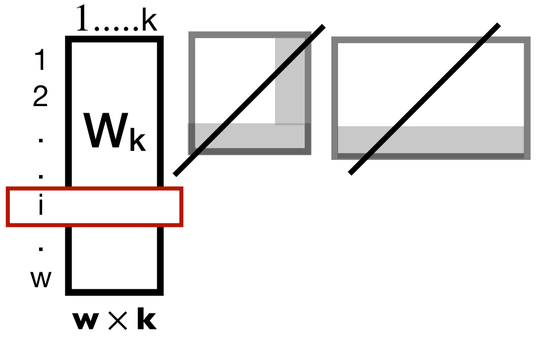
$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$$




2) Truncation




3) Embeddings




embedding for word i

SVD = Singular value decomposition

$\boldsymbol{\Sigma}$ is a diagonal matrix with singular values:

$\sigma_1, \ldots, \sigma_i, \ldots, \sigma_m$ where $m$ is the rank of the matrix $\mathbf{X}$

$\mathbf{U} = \mathbf{W}$ has orthonormal columns, $\mathbf{V}^\top = \mathbf{C}$ has orthonormal rows

In our case, $\mathbf{X}$ is a $|V| \times |V|$ matrix. Assume $m = |V|$

Truncation:

- Select the first $k$ columns of $\mathbf{W}$ to get $k$-dimensional row vectors
- Note that the singular values are ordered from largest to smallest, which each singular value representing the variance captured by that dimension
- So the first $k$ dimensions are the dimensions with the most variance

Finally, we take $i$th row of $\mathbf{W}_k$ as the embedding of word $i$

Note there are variants where the word embedding matrix is taken to be $\mathbf{W}_k\boldsymbol{\Sigma}^\lambda$ (with common values being $\lambda = 1$, 0.5, or 0)

traditional

# What is wrong with using SVD?

- Computational complexity is high: $O(|V|^3)$
- Cannot be trained as part of a larger model
  - Not a component that can be part of a larger neural network
- Cannot be trained discriminatively for a particular task

# Why dense vectors?

- Short vectors are easier to use as features in ML systems
- Dense vectors may generalize better than storing explicit counts
- They do better at capturing synonymy
  - $w_1$ co-occurs with “car”, $w_2$ co-occurs with “automobile”

- Different methods for getting dense vectors:
  - Singular value decomposition (SVD)
  - word2vec and friends: “learn” the vectors!

# How are these embeddings learned?

Learn predictor to fill in the blank!

C1: A bottle of ____ is on the table.

- Represent each word as a vector
- Train classifier to predict word using context words.
- During training, the word vector is updated so that it is possible to predict the center word using the context words

| | bottle | likes | before | make | corn |
|---|---|---|---|---|---|
| tejuino | 1 | 1 | 1 | 1 | 1 |
| loud | 0 | 0 | 0 | 0 | 0 |
| motor-oil | 1 | 0 | 0 | 0 | 0 |
| tortillas | 0 | 1 | 0 | 1 | 1 |
| choices | 0 | 1 | 0 | 0 | 0 |
| wine | 1 | 1 | 1 | 0 | 0 |

Use as context: other words that appear in a span around the target word

"words that occur in similar contexts tend to have similar meanings"

# Summary

- Representing words as vectors
  - One-hot vectors vs vectors built using context
  - Cosine similarity for measuring similarity between words
- Using context to represent words
  - Distributional hypothesis:

    words that occur in similar contexts tend to have similar meanings
- Co-occurrence matrix
  - Raw counts
  - TF-IDF (term-frequency inverse-document-frequency)
  - PPMI (positive pointwise mutual information)
- Dense vectors via SVD or learned by predicting the missing word