Distributed representation of text

He He

NEW YORK UNIVERSITY

September 11, 2024

Logistics

- HW1 released. Due by next Friday. Fixing issues on Gradescope.
- Waitlist: class cap increased to 150.
- Plan for today:
  - Wrap up logistic regression from last week
  - Word embeddings
  - Neural networks basics

Table of Contents

Review

Introduction

Count-based word embeddings

Prediction-based word embeddings

Neural networks

Last week

Generative vs discriminative models for text classification

- (Multinomial) naive Bayes

What's the key assumption?

Last week

Generative vs discriminative models for text classification

- (Multinomial) naive Bayes
  - Assumes conditional independence
  - Very efficient in practice (closed-form solution)

What's the key assumption?

Last week

Generative vs discriminative models for text classification

- (Multinomial) naive Bayes — What's the key assumption?
  - Assumes conditional independence
  - Very efficient in practice (closed-form solution)
- Logistic regression — What's the main advantage?

Last week

Generative vs discriminative models for text classification

- (Multinomial) naive Bayes — What's the key assumption?
  - Assumes conditional independence
  - Very efficient in practice (closed-form solution)
- Logistic regression — What's the main advantage?
  - Works with all kinds of features
  - Wins with more data [Ng and Jordan, 2001]

Last week

Generative vs discriminative models for text classification

- (Multinomial) naive Bayes — What's the key assumption?
  - Assumes conditional independence
  - Very efficient in practice (closed-form solution)
- Logistic regression — What's the main advantage?
  - Works with all kinds of features
  - Wins with more data [Ng and Jordan, 2001]

Feature vector of text input

- BoW representation
- N-gram features (usually $n \leq 3$)

Table of Contents

Review

Introduction

Count-based word embeddings

Prediction-based word embeddings

Neural networks

Objective

Goal: come up with a good representation of text

- What is a representation?

Objective

Goal: come up with a good representation of text

- What is a representation?
  - Feature map: $\phi\colon \text{text} \to \mathbb{R}^d$, e.g., BoW, handcrafted features

Objective

Goal: come up with a good representation of text

- What is a representation?
  - Feature map: $\phi\colon \text{text} \rightarrow \mathbb{R}^d$, e.g., BoW, handcrafted features
  - "Representation" often refers to learned features of the input

Objective

Goal: come up with a good representation of text

- What is a representation?
  - Feature map: $\phi\colon \text{text} \rightarrow \mathbb{R}^d$, e.g., BoW, handcrafted features
  - "Representation" often refers to learned features of the input
- What is a good representation?

Objective

Goal: come up with a good representation of text

- What is a representation?
  - Feature map: $\phi\colon \text{text} \rightarrow \mathbb{R}^d$, e.g., BoW, handcrafted features
  - “Representation” often refers to learned features of the input
- What is a good representation?
  - Leads to good task performance (with less training data)

Objective

Goal: come up with a good representation of text

- What is a representation?
  - Feature map: $\phi\colon \text{text} \to \mathbb{R}^d$, e.g., BoW, handcrafted features
  - "Representation" often refers to learned features of the input
- What is a good representation?
  - Leads to good task performance (with less training data)
  - Enables a notion of distance over text: $d(\phi(a), \phi(b))$ is small for semantically similar texts a and b

Distance functions

Euclidean distance

For $a, b \in \mathbb{R}^d$,

$$d(a, b) = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2}\ .$$

Distance functions

Euclidean distance

For $a, b \in \mathbb{R}^d$,

$$d(a, b) = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2} \, .$$

Assume a and b are BoW vectors. What if b repeats each sentence in a twice?

Distance functions

Euclidean distance

For $a, b \in \mathbb{R}^d$,

$$d(a,b) = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2} \; .$$

Assume a and b are BoW vectors. What if b repeats each sentence in a twice? ($b_i = 2a_i$)

Distance functions

Euclidean distance

For $a, b \in \mathbb{R}^d$,

$$d(a, b) = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2} \ .$$

Assume a and b are BoW vectors. What if b repeats each sentence in a twice? ($b_i = 2a_i$)

Cosine similarity

For $a, b \in \mathbb{R}^d$,

$$\text{sim}(a, b) = \frac{a \cdot b}{\|a\| \|b\|}$$

Defines angle between two vectors ($= \cos \alpha$ in 2D)

Example: information retrieval

Given a set of documents and a query, use the BoW representation and cosine similarity to find the most relevant document.

What are potential problems?

Example:

Q: Who has watched Barbie?

D1: She has watched Oppenheimer.

D2: Barbie was shown here last week.

Example: information retrieval

Given a set of documents and a query, use the BoW representation and cosine similarity to find the most relevant document.

What are potential problems?

Example:

Q: Who has watched Barbie?

D1: She has watched Oppenheimer.

D2: Barbie was shown here last week.

- Similarity may be dominated by common words
- Only considers the surface form (e.g., does not account for synonyms)

TFIDF

Key idea: upweight words that carry more information about the document

TFIDF

Key idea: upweight words that carry more information about the document

Construct a feature map $\phi\colon$ document $\to \mathbb{R}^{|\mathcal{V}|}$

TFIDF weight for token w:

$$\phi_w(d) = \underbrace{\text{count}(w, d)}_{\text{tf}(w,d)} \times$$

- **Term frequency (TF)**: count of the word in the document (same as BoW)

TFIDF

Key idea: upweight words that carry more information about the document

Construct a feature map ϕ: document $\rightarrow \mathbb{R}^{|\mathcal{V}|}$

TFIDF weight for token w:

$$\phi_w(d) = \underbrace{\text{count}(w, d)}_{\mathrm{tf}(w,d)} \times \underbrace{\log \frac{\text{\# documents}}{\text{\# documents containing } w}}_{\mathrm{idf}(w)} .$$

- **Term frequency (TF)**: count of the word in the document (same as BoW)
- Reweight by **inverse document frequency (IDF)**: how specific is the word to any particular document

TFIDF

Key idea: upweight words that carry more information about the document

Construct a feature map ϕ: document $\rightarrow \mathbb{R}^{|\mathcal{V}|}$

TFIDF weight for token w:

$$\phi_w(d) = \underbrace{\text{count}(w, d)}_{\text{tf}(w,d)} \times \underbrace{\log \frac{\text{\# documents}}{\text{\# documents containing } w}}_{\text{idf}(w)} .$$

- **Term frequency (TF)**: count of the word in the document (same as BoW)
- Reweight by **inverse document frequency (IDF)**: how specific is the word to any particular document
- Higher weight on frequent words that only occur in a few documents

TFIDF example

	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

Figure 6.9 A tf-idf weighted term-document matrix for four words in four Shakespeare plays, using the counts in Fig. 6.2. For example the 0.049 value for *wit* in *As You Like It* is the product of $\text{tf} = \log_{10}(20+1) = 1.322$ and $\text{idf} = .037$. Note that the idf weighting has eliminated the importance of the ubiquitous word *good* and vastly reduced the impact of the almost-ubiquitous word *fool*.

Figure: From Jurafsky and Martin.

Why do some words have zero weights?

Table of Contents

Review

Introduction

Count-based word embeddings

Prediction-based word embeddings

Neural networks

The distributional hypothesis

"You shall know a word by the company it keeps." (Firth, 1957)

The distributional hypothesis

"You shall know a word by the company it keeps." (Firth, 1957)

Word guessing! (example from Eisenstein's book)

Everybody likes tezgüino.

The distributional hypothesis

"You shall know a word by the company it keeps." (Firth, 1957)

Word guessing! (example from Eisenstein's book)

> Everybody likes tezgüino.
>
> We make tezgüino out of corn.

The distributional hypothesis

"You shall know a word by the company it keeps." (Firth, 1957)

Word guessing! (example from Eisenstein's book)

Everybody likes tezgüino.

We make tezgüino out of corn.

A bottle of tezgüino is on the table.

The distributional hypothesis

"You shall know a word by the company it keeps." (Firth, 1957)

Word guessing! (example from Eisenstein's book)

Everybody likes tezgüino.

We make tezgüino out of corn.

A bottle of tezgüino is on the table.

Don't have tezgüino before you drive.

The distributional hypothesis

"You shall know a word by the company it keeps." (Firth, 1957)

Word guessing! (example from Eisenstein's book)

Everybody likes tezgüino.

We make tezgüino out of corn.

A bottle of tezgüino is on the table.

Don't have tezgüino before you drive.

Idea: Represent a word by its neighbors.

Step 1: Choose the context

What are the neighbors? (What type of co-occurence are we interested in?)

Example:

- word × document

	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

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Figure: Jurafsky and Martin.

Step 1: Choose the context

What are the neighbors? (What type of co-occurence are we interested in?)

Example:

- word $\times$ document
- word $\times$ word

	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

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Figure: Jurafsky and Martin.

Step 1: Choose the context

What are the neighbors? (What type of co-occurence are we interested in?)

Example:

- word $\times$ document
- word $\times$ word
- person $\times$ movie

	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

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Figure: Jurafsky and Martin.

Step 1: Choose the context

What are the neighbors? (What type of co-occurence are we interested in?)

Example:

- word $\times$ document
- word $\times$ word
- person $\times$ movie
- note $\times$ song

	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

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Figure: Jurafsky and Martin.

Step 1: Choose the context

What are the neighbors? (What type of co-occurence are we interested in?)

Example:

- word $\times$ document
- word $\times$ word
- person $\times$ movie
- note $\times$ song

	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

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Figure: Jurafsky and Martin.

Construct a matrix where

- Row and columns represent two sets of objects
- Each entry is the (adjusted) co-occurence counts of the two objects

Step 2: Reweight counts

Upweight informative words

	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

Figure 6.9 A tf-idf weighted term-document matrix for four words in four Shakespeare

Figure: Jurafsky and Martin.

Each row/column gives us a word/document representation.

Step 2: Reweight counts

Upweight informative words

	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

Figure 6.9 A tf-idf weighted term-document matrix for four words in four Shakespeare

Figure: Jurafsky and Martin.

Each row/column gives us a word/document representation.

Using cosine similarity, we can cluster documents, find synonyms, discover word meanings, etc.

An alternative way to reweighting using pointwise mutual information

$$\text{PMI}(x; y) \stackrel{\text{def}}{=} \log \frac{p(x, y)}{p(x)p(y)} = \log \frac{p(x \mid y)}{p(x)} = \log \frac{p(y \mid x)}{p(y)}$$

An alternative way to reweighting using pointwise mutual information

$$\text{PMI}(x; y) \stackrel{\text{def}}{=} \log \frac{p(x, y)}{p(x)p(y)} = \log \frac{p(x \mid y)}{p(x)} = \log \frac{p(y \mid x)}{p(y)}$$

- Symmetric: $\text{PMI}(x; y) = \text{PMI}(y; x)$
- Estimates:

$$\hat{p}(x \mid y) = \frac{\text{count}(x, y)}{\sum_{x' \in \mathcal{X}} \text{count}(x', y)}$$ how often does word x occur in the neighborhood of y

$$\hat{p}(x) = \frac{\text{count}(x)}{\sum_{x' \in \mathcal{X}} \text{count}(x')}$$ how often does word x occur in the corpus

Positive PMI / PPMI

- Range: $(-\infty, \min(-\log p(x), -\log p(y)))$
- What does negative PMI mean?

Positive PMI / PPMI

- Range: $(-\infty, \min(-\log p(x), -\log p(y)))$
- What does negative PMI mean?
  - Two words co-occur less frequently than chance
  - Need large data to estimate small probabilities
  - Difficult to judge unrelatedness by humans
- **Positive PMI**: $\text{PPMI}(x; y) \stackrel{\text{def}}{=} \max(0, \text{PMI}(x; y))$
- Application in NLP: measure association between words

Step 3: Dimensionality reduction

What's the size of this matrix?

	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

Figure 6.9 A tf-idf weighted term-document matrix for four words in four Shakespeare

Figure: Jurafsky and Martin.

Step 3: Dimensionality reduction

What's the size of this matrix?

	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

Figure 6.9 A tf-idf weighted term-document matrix for four words in four Shakespeare

Figure: Jurafsky and Martin.

Motivation: want a lower-dimensional, dense representation for efficiency

Step 3: Dimensionality reduction

Recall **SVD**: a $m \times n$ matrix $A_{m \times n}$ (e.g., a word-document matrix), can be decomposed to

$$U_{m \times m} \Sigma_{m \times n} V^T_{n \times n} ,$$

where U and V are orthogonal matrices, and Σ is a diagonal matrix.

Step 3: Dimensionality reduction

Recall **SVD**: a $m \times n$ matrix $A_{m\times n}$ (e.g., a word-document matrix), can be decomposed to

$$U_{m\times m}\Sigma_{m\times n}V^T_{n\times n} \ ,$$

where U and V are orthogonal matrices, and Σ is a diagonal matrix.

Interpretation:

$$AA^T = (U\Sigma V^T)(V\Sigma U^T) = U\Sigma^2 U^T \ .$$

- σ_i^2 are eigenvalues of AA^T
- Connection to PCA: If columns of A have zero mean (i.e. AA^T is the covariance matrix), then columns of U are principle components of the column space of A.

SVD for the word-document matrix

$$A = \begin{array}{c} \\ \\ \\ \\ \end{array}\begin{array}{c} \begin{matrix} d_1 & d_2 & \cdots & d_n \end{matrix} \\ \begin{pmatrix} 12 & 5 & \cdots & 8 \\ 7 & 10 & \cdots & 3 \\ 4 & 8 & \cdots & 6 \\ 9 & 3 & \cdots & 7 \end{pmatrix} \end{array} \begin{array}{l} \\ \text{US} \\ \text{gov} \\ \text{gene} \\ \text{lab} \end{array}$$

SVD for the word-document matrix

$$A = \begin{array}{c} \begin{matrix} d_1 & d_2 & \cdots & d_n \end{matrix} \\ \begin{pmatrix} 12 & 5 & \cdots & 8 \\ 7 & 10 & \cdots & 3 \\ 4 & 8 & \cdots & 6 \\ 9 & 3 & \cdots & 7 \end{pmatrix} \end{array} \begin{matrix} \text{US} \\ \text{gov} \\ \text{gene} \\ \text{lab} \end{matrix}$$

$$= \begin{pmatrix} 0.50 & 0.02 & \cdots \\ 0.60 & 0.03 & \cdots \\ 0.01 & 0.72 & \cdots \\ 0.02 & 0.84 & \cdots \end{pmatrix}_{4\times 4} \begin{pmatrix} 15 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 10 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 5 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 2 & \cdots & 0 \end{pmatrix}_{4\times n} \begin{pmatrix} 0.50 & 0.60 & \cdots \\ 0.64 & 0.48 & \cdots \\ 0.12 & 0.24 & \cdots \\ 0.36 & 0.12 & \cdots \end{pmatrix}^T_{n\times 4}$$

SVD for the word-document matrix

$$A = \begin{array}{c} \\ \\ \end{array}\begin{matrix} & d_1 & d_2 & \cdots & d_n & \\ & \end{matrix}$$

$$A = \begin{pmatrix} 12 & 5 & \cdots & 8 \\ 7 & 10 & \cdots & 3 \\ 4 & 8 & \cdots & 6 \\ 9 & 3 & \cdots & 7 \end{pmatrix} \begin{matrix} \text{US} \\ \text{gov} \\ \text{gene} \\ \text{lab} \end{matrix} \quad (\text{columns } d_1, d_2, \cdots, d_n)$$

$$= \begin{pmatrix} 0.50 & 0.02 & \cdots \\ 0.60 & 0.03 & \cdots \\ 0.01 & 0.72 & \cdots \\ 0.02 & 0.84 & \cdots \end{pmatrix}_{4\times 4} \begin{pmatrix} 15 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 10 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 5 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 2 & \cdots & 0 \end{pmatrix}_{4\times n} \begin{pmatrix} 0.50 & 0.60 & \cdots \\ 0.64 & 0.48 & \cdots \\ 0.12 & 0.24 & \cdots \\ 0.36 & 0.12 & \cdots \end{pmatrix}^T_{n\times 4}$$

- u_i are document clusters and v_i are word clusters
- Take top-k components to obtain word vectors: $W = U_k\Sigma_k$ (or just U_k)

SVD for the word-document matrix

Computing the dense word vectors:

- Run **truncated SVD** of the word-document matrix $A_{m \times n}$
- Each row of $U_{m \times k} \Sigma_k$ corresponds to a word vector of dimension k
- Each coordinate of the word vector corresponds to a cluster of documents (i.e., topics such as politics, music, etc.)

Summary

Count-based word embeddings

1. Design the matrix, e.g. word $\times$ document, people $\times$ movie.
2. Reweight the raw counts, e.g. TFIDF, PPMI.
3. Reduce dimensionality by truncated SVD.
4. Use word/person/etc. vectors in downstream tasks.

Key idea:

- Intuition: Represent an object by its connection to other objects.
- Lexical semantics: the word meaning can be represented by the context it occurs in.
- Linear algebra: Infer clusters (e.g., concepts, topics) using co-occurence statistics

Table of Contents

Review

Introduction

Count-based word embeddings

Prediction-based word embeddings

Neural networks

Learning word embeddings

Goal: map each word to a vector in $\mathbb{R}^d$ such that *similar* words also have *similar* word vectors.

Learning word embeddings

Goal: map each word to a vector in $\mathbb{R}^d$ such that *similar* words also have *similar* word vectors.

Can we formalize this as a prediction problem?

- Needs to be self-supervised since our data is unlabeled.

Learning word embeddings

Goal: map each word to a vector in $\mathbb{R}^d$ such that *similar* words also have *similar* word vectors.

Can we formalize this as a prediction problem?

- Needs to be self-supervised since our data is unlabeled.

Distributional hypothesis: Similar words occur in similar contexts

- Predict the context given a word $f : \text{word} \rightarrow \text{context}$
- Words that tend to occur in same contexts will have similar representation

The skip-gram model

Task: given a word, predict its neighboring words within a window

The quick brown fox jumps over the lazy dog

The skip-gram model

Task: given a word, predict its neighboring words within a window

The quick brown fox jumps over the lazy dog

Assume **conditional independence** of the context words:

$$p(w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k} \mid w_i) = \prod_{j=i-k, j \neq i}^{i+k} p(w_j \mid w_i)$$

The skip-gram model

Task: given a word, predict its neighboring words within a window

The quick brown fox jumps over the lazy dog

Assume **conditional independence** of the context words:

$$p(w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k} \mid w_i) = \prod_{j=i-k, j\neq i}^{i+k} p(w_j \mid w_i)$$

How to model $p(w_j \mid w_i)$?

The skip-gram model

Task: given a word, predict its neighboring words within a window

The quick brown fox jumps over the lazy dog

Assume **conditional independence** of the context words:

$$p(w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k} \mid w_i) = \prod_{j=i-k, j \neq i}^{i+k} p(w_j \mid w_i)$$

How to model $p(w_j \mid w_i)$? Multiclass classification

The skip-gram model

Use the softmax function to predict context words from the center word

$$p(w_j \mid w_i) = \frac{\exp\left[\phi_{\text{ctx}}(w_j) \cdot \phi_{\text{wrd}}(w_i)\right]}{\sum_{w \in \mathcal{V}} \exp\left[\phi_{\text{ctx}}(w_j) \cdot \phi_{\text{wrd}}(w_i)\right]}$$

The skip-gram model

Use the softmax function to predict context words from the center word

$$p(w_j \mid w_i) = \frac{\exp\left[\phi_{\text{ctx}}(w_j) \cdot \phi_{\text{wrd}}(w_i)\right]}{\sum_{w \in \mathcal{V}} \exp\left[\phi_{\text{ctx}}(w_j) \cdot \phi_{\text{wrd}}(w_i)\right]}$$

🤔 What's the difference from multinomial logistic regression?

The skip-gram model

Use the softmax function to predict context words from the center word

$$p(w_j \mid w_i) = \frac{\exp\left[\phi_{\text{ctx}}(w_j) \cdot \phi_{\text{wrd}}(w_i)\right]}{\sum_{w \in \mathcal{V}} \exp\left[\phi_{\text{ctx}}(w_j) \cdot \phi_{\text{wrd}}(w_i)\right]}$$

What's the difference from multinomial logistic regression?

Implementation:

- $\phi \colon \mathcal{V} \to \mathbb{R}^d$. ϕ can be implemented as a dictionary from words (ids) to vectors.

The skip-gram model

Use the softmax function to predict context words from the center word

$$p(w_j \mid w_i) = \frac{\exp\left[\phi_{\text{ctx}}(w_j) \cdot \phi_{\text{wrd}}(w_i)\right]}{\sum_{w \in \mathcal{V}} \exp\left[\phi_{\text{ctx}}(w_j) \cdot \phi_{\text{wrd}}(w_i)\right]}$$

What's the difference from multinomial logistic regression?

Implementation:

- $\phi \colon \mathcal{V} \to \mathbb{R}^d$. ϕ can be implemented as a dictionary from words (ids) to vectors.
- For each word w, learn two vectors.

The skip-gram model

Use the softmax function to predict context words from the center word

$$p(w_j \mid w_i) = \frac{\exp\left[\phi_{\text{ctx}}(w_j) \cdot \phi_{\text{wrd}}(w_i)\right]}{\sum_{w \in \mathcal{V}} \exp\left[\phi_{\text{ctx}}(w_j) \cdot \phi_{\text{wrd}}(w_i)\right]}$$

🤔 What's the difference from multinomial logistic regression?

Implementation:

- $\phi \colon \mathcal{V} \to \mathbb{R}^d$. ϕ can be implemented as a dictionary from words (ids) to vectors.
- For each word w, learn two vectors.
- Learn parameters by MLE and SGD (Is the objective convex?)

The skip-gram model

Use the softmax function to predict context words from the center word

$$p(w_j \mid w_i) = \frac{\exp\left[\phi_{\text{ctx}}(w_j) \cdot \phi_{\text{wrd}}(w_i)\right]}{\sum_{w \in \mathcal{V}} \exp\left[\phi_{\text{ctx}}(w_j) \cdot \phi_{\text{wrd}}(w_i)\right]}$$

🤔 What's the difference from multinomial logistic regression?

Implementation:

- $\phi \colon \mathcal{V} \to \mathbb{R}^d$. ϕ can be implemented as a dictionary from words (ids) to vectors.
- For each word w, learn two vectors.
- Learn parameters by MLE and SGD (Is the objective convex?)
- ϕ_{wrd} is taken as the word embedding

Negative sampling

Challenge in MLE: computing the normalizer is expensive (try calculate the gradient)!

Negative sampling

Challenge in MLE: computing the normalizer is expensive (try calculate the gradient)!

Key idea: solve a binary classification problem instead

Is the (word, context) pair real or fake?

positive examples +

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

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 sampling

Challenge in MLE: computing the normalizer is expensive (try calculate the gradient)!

Key idea: solve a binary classification problem instead

Is the (word, context) pair real or fake?

positive examples +

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

negative examples -

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

$$p_\theta(\text{real} \mid w, c) = \frac{1}{1 + e^{-\phi_{\text{ctx}}(c)\cdot\phi_{\text{wrd}}(w)}}$$

Large dot product between w and c if they co-occur.

The continuous bag-of-words model

Task: given the context, predict the word in the middle

The quick brown fox jumps over the lazy dog

Similary, we can use multiclass classification for the prediction

$$p(w_i \mid w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k})$$

How to represent the context (input)?

The continuous bag-of-words model

The context is a sequence of words.

$$c = w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k}$$

$$p(w_i \mid c) = \frac{\exp\left[\phi_{\mathrm{wrd}}(w_i) \cdot \phi_{\mathrm{BoW}}(c)\right]}{\sum_{w \in \mathcal{V}} \exp\left[\phi_{\mathrm{wrd}}(w) \cdot \phi_{\mathrm{BoW}}(c)\right]}$$

The continuous bag-of-words model

The context is a sequence of words.

$$c = w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k}$$

$$\begin{aligned} p(w_i \mid c) &= \frac{\exp\left[\phi_{\text{wrd}}(w_i) \cdot \phi_{\text{BoW}}(c)\right]}{\sum_{w \in \mathcal{V}} \exp\left[\phi_{\text{wrd}}(w) \cdot \phi_{\text{BoW}}(c)\right]} \\ &= \frac{\exp\left[\phi_{\text{wrd}}(w_i) \cdot \sum_{w' \in c} \phi_{\text{ctx}}(w')\right]}{\sum_{w \in \mathcal{V}} \exp\left[\phi_{\text{wrd}}(w) \cdot \sum_{w' \in c} \phi_{\text{ctx}}(w')\right]} \end{aligned}$$

- $\phi_{\text{BoW}}(c)$ sums over representations of each word in c
- Implementation is similar to the skip-gram model.

Semantic properties of word embeddings

Find similar words: top-k nearest neighbors using cosine similarity

- Size of window influences the type of similarity
- Shorter window produces syntactically similar words, e.g., Hogwarts and Sunnydale (fictional schools)
- Longer window produces topically related words, e.g., Hogwarts and Dumbledore (Harry Porter entities)

Semantic properties of word embeddings

Solve word analogy problems: a is to b as a' is to what?

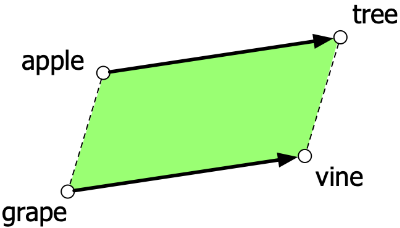


Figure: Parallelogram model (from J&H).

- man : woman :: king : queen

  $\phi_{\text{wrd}}(\text{man}) - \phi_{\text{wrd}}(\text{king}) \approx \phi_{\text{wrd}}(\text{woman}) - \phi_{\text{wrd}}(\text{queen})$
- Caveat: must exclude the three input words
- Does not work for general relations

Comparison

Count-based	Prediction-based
matrix factorization	prediction problem
fast to compute	slow (with large corpus)
interpretable components	hard to interprete but has intriguing properties

- Both uses the **distributional hypothesis**.
- Both generalize beyond text: using co-occurence between any types of objects
  - Learn product embeddings from customer orders
  - Learn region embeddings from images

Evaluate word vectors

Intrinsic evaluation

- Evaluate on the proxy task (related to the learning objective)
- Word similarity/analogy datasets (e.g., WordSim-353, SimLex-999)

Extrinsic evaluation

- Evaluate on the real/downstream task we care about
- Use word vectors as features in applications, e.g., text classification.

Summary

Key idea: formalize word representation learning as a self-supervised prediction problem

Prediction problems:

- Skip-gram: Predict context from words
- CBOW: Predict word from context
- Other possibilities:
  - Predict $\log \hat{p}(\text{word} \mid \text{context})$, e.g. GloVe
  - Contextual word embeddings (later)

Table of Contents

Review

Introduction

Count-based word embeddings

Prediction-based word embeddings

Neural networks

Feature learning

Linear predictor with handcrafted features: $h(x) = w \cdot \phi(x)$.

Can we learn features from data?

Feature learning

Linear predictor with handcrafted features: $h(x) = w \cdot \phi(x)$.

Can we learn features from data?

Example:

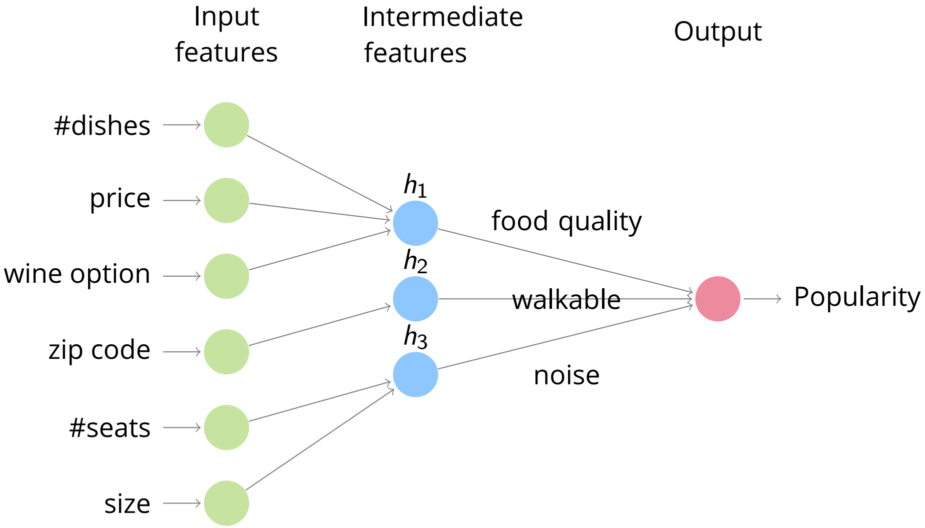
- Predict popularity of restaurants.
- Raw input: #dishes, price, wine option, zip code, #seats, size
- Decompose into subproblems:

$$h_1([\text{\#dishes, price, wine option}]) = \text{food quality}$$

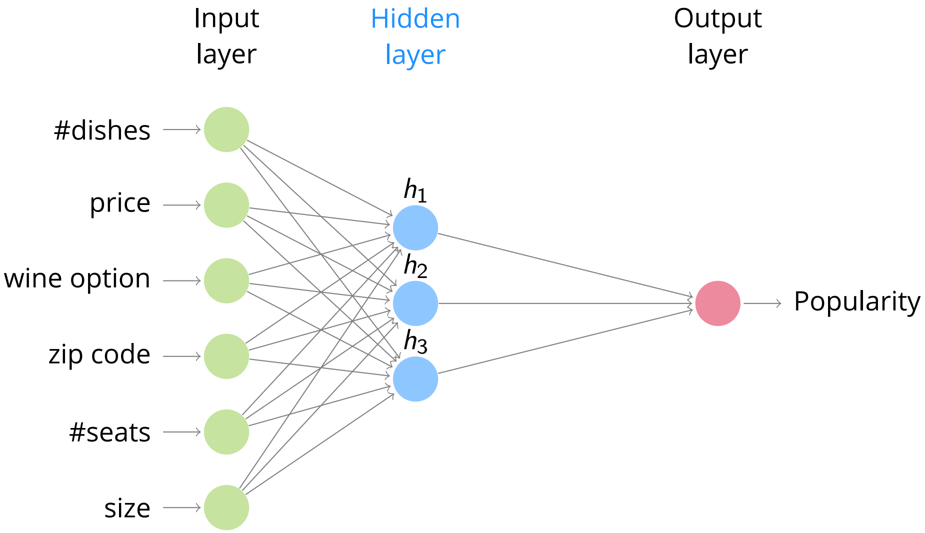
$$h_2([\text{zip code}]) = \text{walkable}$$

$$h_3([\text{\#seats, size}]) = \text{nosie}$$

Predefined subproblems



Learning intermediate features



Neural networks

Key idea: automatically learn the intermediate features.

Feature engineering: Manually specify $\phi(x)$ based on domain knowledge and learn the weights:

$$f(x) = w^T \phi(x).$$

Feature learning: Automatically learn both the features (K hidden units) and the weights:

$$h(x) = [h_1(x), \dots, h_K(x)], \quad f(x) = w^T h(x)$$

Activation function

- How should we parametrize h_i's?

Activation function

- How should we parametrize h_i's?

$$h_i(x) = \sigma(v_i^T x). \tag{1}$$

- σ is the **activation function**.

Activation function

- How should we parametrize h_i's?

$$h_i(x) = \sigma(v_i^T x). \tag{1}$$

- σ is the **activation function**.
- What might be some activation functions we want to use?

Activation function

- How should we parametrize h_i's?

$$h_i(x) = \sigma(v_i^T x). \tag{1}$$

- σ is the **activation function**.
- What might be some activation functions we want to use?
  - sign function? Non-differentiable.

Activation function

- How should we parametrize h_i's?

$$h_i(x) = \sigma(v_i^T x). \tag{1}$$

- σ is the **activation function**.
- What might be some activation functions we want to use?
  - sign function? Non-differentiable.
  - *Differentiable* approximations: sigmoid functions.
    - E.g., logistic function, hyperbolic tangent function, ReLU

Activation function

- How should we parametrize h_i's?

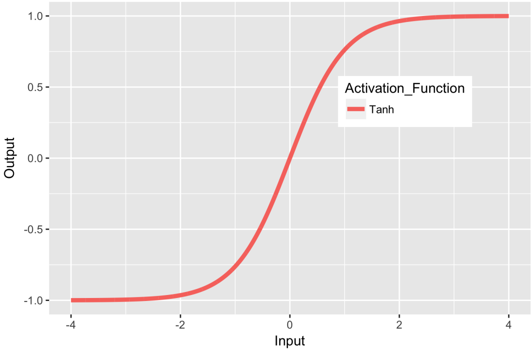
$$h_i(x) = \sigma(v_i^T x). \tag{1}$$

- σ is the **activation function**.
- What might be some activation functions we want to use?
  - sign function? Non-differentiable.
  - *Differentiable* approximations: sigmoid functions.
    - E.g., logistic function, hyperbolic tangent function, ReLU
  - Non-linearity

Activation Functions

- The **hyperbolic tangent** is a common activation function:

$$\sigma(x) = \tanh(x) .$$

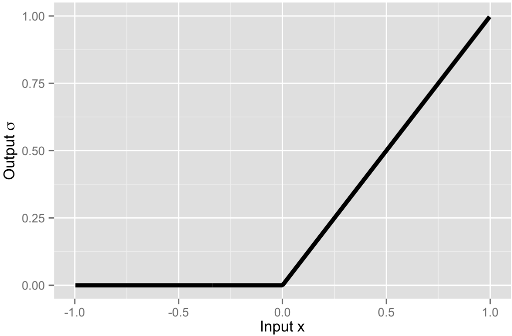


Activation Functions

- The **rectified linear** (**ReLU**) function:

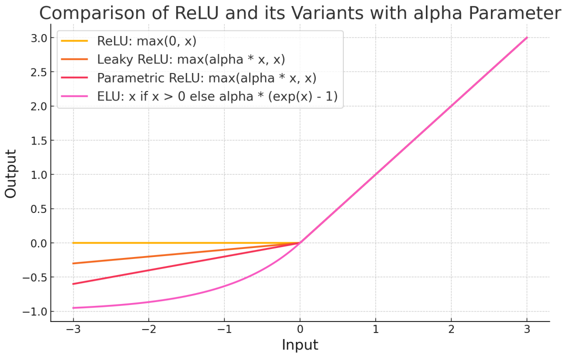
$$\sigma(x) = \max(0, x).$$

- Efficient backpropogation, sparsity, avoiding vanishing gradient
- Work well empirically.



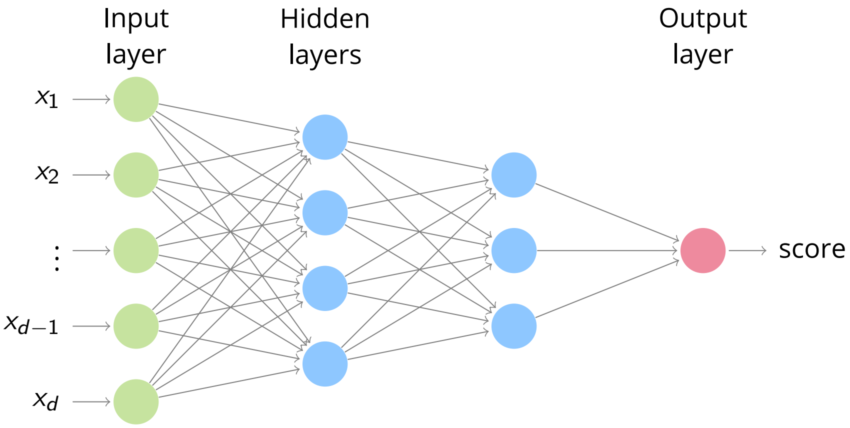
Activation Functions

- The dying ReLU problem: neurons become inactive
- Solution: allow small gradients when the neuron is not active



Multilayer perceptron / Feed-forward neural networks

- Wider: more hidden units.
- Deeper: more hidden layers.



Multilayer Perceptron: Standard Recipe

- Each subsequent hidden layer takes the output $o \in \mathbb{R}^m$ of previous layer and produces

$$h^{(j)}(o^{(j-1)}) = \sigma\left(W^{(j)} o^{(j-1)} + b^{(j)}\right), \text{ for } j = 2, \ldots, L$$

where $W^{(j)} \in \mathbb{R}^{m \times m}$, $b^{(j)} \in \mathbb{R}^m$.

Multilayer Perceptron: Standard Recipe

- Each subsequent hidden layer takes the output $o \in \mathbb{R}^m$ of previous layer and produces

$$h^{(j)}(o^{(j-1)}) = \sigma\left(W^{(j)} o^{(j-1)} + b^{(j)}\right), \text{ for } j = 2, \ldots, L$$

  where $W^{(j)} \in \mathbb{R}^{m \times m}$, $b^{(j)} \in \mathbb{R}^m$.
- Last layer is an *affine* mapping (no activation function):

$$a(o^{(L)}) = W^{(L+1)} o^{(L)} + b^{(L+1)},$$

  where $W^{(L+1)} \in \mathbb{R}^{k \times m}$ and $b^{(L+1)} \in \mathbb{R}^k$.

Multilayer Perceptron: Standard Recipe

- Each subsequent hidden layer takes the output $o \in \mathbb{R}^m$ of previous layer and produces
$$h^{(j)}(o^{(j-1)}) = \sigma\left(W^{(j)}o^{(j-1)} + b^{(j)}\right), \text{ for } j = 2, \dots, L$$
where $W^{(j)} \in \mathbb{R}^{m\times m}$, $b^{(j)} \in \mathbb{R}^m$.
- Last layer is an *affine* mapping (no activation function):
$$a(o^{(L)}) = W^{(L+1)}o^{(L)} + b^{(L+1)},$$
where $W^{(L+1)} \in \mathbb{R}^{k\times m}$ and $b^{(L+1)} \in \mathbb{R}^k$.
- The full neural network function is given by the *composition* of layers:
$$f(x) = \left(a \circ h^{(L)} \circ \cdots \circ h^{(1)}\right)(x) \tag{2}$$