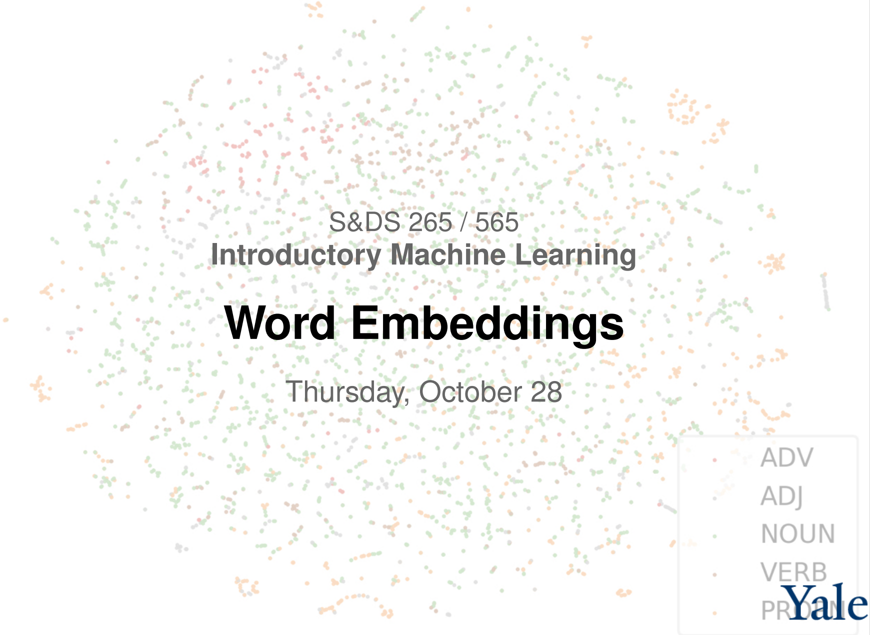
S&DS 265 / 565

Introductory Machine Learning

Word Embeddings

Thursday, October 28



Yale

Reminders

- Assignment 4 due Tuesday
- Assignment 5 out Tuesday

Language models

- A language model is a way of assigning a probability to any sequence of words (or string of text)

$$p(w_1, \ldots, w_n)$$

Language models

- A language model is a way of assigning a probability to any sequence of words (or string of text)

$$p(w_1, \ldots, w_n)$$

- By the basic rules of conditional probability we can factor this as

$$p(w_1, \ldots, w_n) = p(w_1)p(w_2 \mid w_1) \ldots p(w_n \mid w_1, \ldots, w_{n-1})$$

Language models

- A language model is a way of *generating* any sequence of words

$$
\begin{aligned}
P(\text{“the whole forest had been anesthetized”}) = \\
P(\text{“the”}) \times P(\text{“whole”} \mid \text{“the”}) \\
\times P(\text{“forest”} \mid \text{“the whole”}) \\
\times P(\text{“had”} \mid \text{“the whole forest”}) \\
\times P(\text{“been”} \mid \text{“the whole forest had”}) \\
\times P(\text{“anesthetized”} \mid \text{“the whole forest had been”})
\end{aligned}
$$

Remixing Noon

Text generated from Channel Skin by Jeff Noon

"The whole forest had been anesthetised, her temples wired, her senses stimulated, her eyes were not on Eva, not on Eva, not on Eva, not on anybody in that same realm, the land of dreams and nightmares."

Viability: 0.000000326%

https://revdancatt.com/2017/03/01/markov_noon

Text generation

- Words generated one-by-one
- A word is chosen by sampling from a probability distribution
- Then treated as if it were “real,” as in dreaming
- Result is purely synthetic text

How good is a language model? Perplexity

Perplexity is defined as

$$\text{Perplexity}(\theta) = \left(\prod_{i=1}^{n} p_\theta(w_i \mid w_{1:i-1}) \right)^{-\frac{1}{n}}$$

where $w_1, w_2, \ldots, w_n$ is a large chunk of text that wasn't used to train the language model.

How good is a language model? Perplexity

- Perplexity is the inverse of the geometric mean of the word probabilities
- If the perplexity is 100, the model predicts, on average, as if there were 100 equally likely words to follow
- This is the (geometric) average “branching factor” for the model on real text

Modern language models

Suppose a computer program assigns a “score” to possible next words:

$$s(v;\ \underbrace{w_1, \ldots, w_n}_{\text{word history}}\ )$$

https://en.wikipedia.org/wiki/Softmax_function

Modern language models

Suppose a computer program assigns a “score” to possible next words:

$$s(v;\ \underbrace{w_1, \ldots, w_n}_{\text{word history}}\ )$$

Can convert this to a language model by the “softmax” operation:

$$p(w \mid w_1, \ldots, w_n) = \frac{\exp(s(w; w_1, \ldots, w_n))}{\sum_{v \in V} \exp(s(v; w_1, \ldots, w_n))}$$

---

https://en.wikipedia.org/wiki/Softmax_function

Modern language models

Suppose a computer program assigns a “score” to possible next words:

$$s(v;\ \underbrace{w_1, \ldots, w_n}_{\text{word history}}\ )$$

Can convert this to a language model by the “softmax” operation:

$$p(w \mid w_1, \ldots, w_n) = \frac{\exp(s(w; w_1, \ldots, w_n))}{\sum_{v \in V} \exp(s(v; w_1, \ldots, w_n))}$$

In GPT-3, the function $s(v; w_{1:n})$ is learned on large amounts of text (unsupervised) using a type of deep neural network called a *transformer*.

https://en.wikipedia.org/wiki/Softmax_function

Modern language models

Suppose a computer program assigns a “score” to possible next words:

$$s(v;\ \underbrace{w_1, \ldots, w_n}_{\text{word history}}\ )$$

Modern language models

Suppose a computer program assigns a "score" to possible next words:

$$s(v;\ \underbrace{w_1, \ldots, w_n}_{\text{word history}}\ )$$

Today, we'll be working with a simple case where

$$\begin{aligned} s(v; w_1, \ldots, w_n) &= \beta_v^T \phi(w_1, \ldots, w_n) \\ &= \beta_v^T \phi(w_n) \\ &= \phi(v)^T \phi(w_n) \end{aligned}$$

Modern language models

Suppose a computer program assigns a "score" to possible next words:

$$s(v;\ \underbrace{w_1, \ldots, w_n}_{\text{word history}}\ )$$

Today, we'll be working with a simple case where

$$\begin{aligned} s(v; w_1, \ldots, w_n) &= \beta_v^T \phi(w_1, \ldots, w_n) \\ &= \beta_v^T \phi(w_n) \\ &= \phi(v)^T \phi(w_n) \end{aligned}$$

Key intuition

- Words that have similar neighbors will be similar
- Self-referential notion of similarity
- This will be an intuition behind “word embeddings”

Pointwise mutual information (PMI)

Can cluster words based on “pointwise mutual information” (PMI)

$$\log\left(\frac{p_{\mathrm{near}}(w_1, w_2)}{p(w_1)p(w_2)}\right)$$

- How likely are specific words/clusters to co-occur together within some window, compared to if they were independent?

Example clusters from PMI

we our us ourselves ours
question questions asking answer answers answering
performance performed perform performs performing
tie jacket suit
write writes writing written wrote pen
morning noon evening night nights midnight bed
attorney counsel trial court judge
problems problem solution solve analyzed solved solving
letter addressed enclosed letters correspondence
large size small larger smaller
operations operations operating operate operated
school classroom teaching grade math
street block avenue corner blocks
table tables dining chairs plate
published publication author publish writer titled
wall ceiling walls enclosure roof
sell buy selling buying sold

Core idea of embeddings

- Form a language model but replace classes by vectors, one for each word
- Use PMI-like scores to fit the vectors
- Can be applied whenever have cooccurrence data.

Constructing embeddings

Language model is

$$p(w_2 \mid w_1) = \frac{\exp(\phi(w_2)^T \phi(w_1))}{\sum_w \exp(\phi(w)^T \phi(w_1))}.$$

Carry out stochastic gradient descent over the embedding vectors $\phi \in \mathbb{R}^d$ (where $d \approx 50$–500 is chosen by hand)

This is what Mikolov et al. (2014, 2015) did at Google. With a couple of twists:

"Distributed representations of words," (2014) "Efficient representations of words in vector space" (2015)

Constructing embeddings

Heuristics used:

- Skip-gram: predict surrounding words from current word

“Distributed representations of words,” (2014) “Efficient representations of words in vector space” (2015)

Constructing embeddings

Heuristics used:

- Skip-gram: predict surrounding words from current word
- This leads to a model of nearby words $p_{\text{near}}(w_2 \mid w_1)$.

“Distributed representations of words,” (2014) “Efficient representations of words in vector space” (2015)

Constructing embeddings

Heuristics used:

- Skip-gram: predict surrounding words from current word
- This leads to a model of nearby words $p_{\text{near}}(w_2 \mid w_1)$.
- Second is computational. The bottleneck is computing the denominator in the logistic (softmax) probability.

"Distributed representations of words," (2014) "Efficient representations of words in vector space" (2015)

Constructing embeddings

Heuristics used:

- Skip-gram: predict surrounding words from current word
- This leads to a model of nearby words $p_{\text{near}}(w_2 \mid w_1)$.
- Second is computational. The bottleneck is computing the denominator in the logistic (softmax) probability.
- Use "negative sampling": Approximation

$$\sum_{w} \exp(\phi(w)^T \phi(w_1))$$

$$\approx \exp(\phi(w_2)^T \phi(w_1)) + \sum_{\text{random } w} \exp(\phi(w)^T \phi(w_1))$$

"Distributed representations of words," (2014) "Efficient representations of words in vector space" (2015)

Using PCA

A closely related approach is to use PCA of PMI:

- Form $V \times V$ matrix of pointwise mutual information values

$$\log\left(\frac{p_{\text{near}}(w_1, w_2)}{p(w_1)p(w_2)}\right)$$

- Compute top k eigenvectors $\phi_1, \ldots, \phi_k$
- For each word w, define embedding as

$$\phi(w) \equiv (\phi_{1w}, \phi_{2w}, \ldots, \phi_{kw})^T$$

Analogies

These heuristics enable training on very large text collections. Leads to vector representations of words with interesting properties.

For example, analogies:

`king` is to `man` as ? is to `woman`

Analogies

These heuristics enable training on very large text collections. Leads to vector representations of words with interesting properties.

For example, analogies:

`king` is to `man` as ? is to `woman`

`Paris` is to `France` as ? is to `Germany`

Analogies

These heuristics enable training on very large text collections. Leads to vector representations of words with interesting properties.

For example, analogies:

`king` is to `man` as ? is to `woman`

`Paris` is to `France` as ? is to `Germany`

$$\phi(\texttt{king}) - \phi(\texttt{man}) \overset{?}{\approx} \phi(\texttt{queen}) - \phi(\texttt{woman})$$

$$\widehat{w} = \arg\min_{w} \|\phi(\texttt{king}) - \phi(\texttt{man}) + \phi(\texttt{woman}) - \phi(w)\|^2$$

Does $\widehat{w} = \texttt{queen}$?

Learned Analogies

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Mikolov et al., "Distributed representations of words," (2014); "Efficient representations of words in vector space" (2015)

Evaluation Analogies

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

Mikolov et al., “Distributed representations of words,” (2014); “Efficient representations of words in vector space” (2015)

GloVe

Shortly after: Stanford group introduced a variant

$$\mathcal{O}(\phi) = \sum_{w_1, w_2} f(c_{w_1, w_2}) \left(\phi(w_1)^T \phi(w_2) - \log c_{w_1, w_2}\right)^2$$

where $c_{w,w'}$ are cooccurrence counts in a window (PMI)

- A type of regression estimator
- Main advantage is that SGD can be carried out much more efficiently

Pennington et al., "GloVe: Global vectors for word representation," (2015)

GloVe

$$\mathcal{O}(\phi) = \sum_{w_1, w_2} f(c_{w_1,w_2}) \left(\phi(w_1)^T \phi(w_2) - \log c_{w_1,w_2}\right)^2$$

where $c_{w,w'}$ are cooccurrence counts.

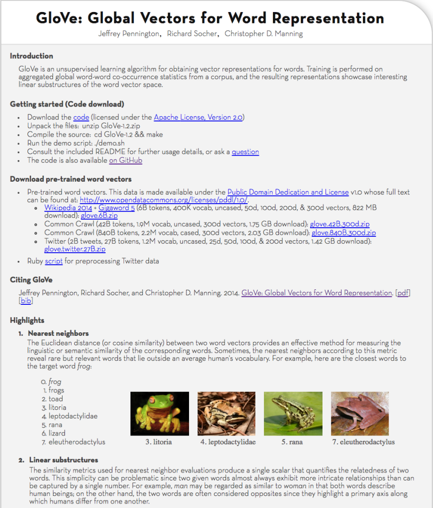
- Heuristic weighting function

  $$f(x) = \left(\frac{x}{x_{\max}}\right)^{\alpha}$$

  where $\alpha = 3/4$ set empirically.
- So $10^{-4} \mapsto 10^{-3}$. Each order of magnitude down gets "boosted" by 1/4-magnitude.

Pennington et al., "GloVe: Global vectors for word representation," (2015)

GloVe site and code



GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning

Introduction

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Getting started (Code download)

- Download the code (licensed under the Apache License, Version 2.0)
- Unpack the files: unzip GloVe-1.2.zip
- Compile the source: cd GloVe-1.2 && make
- Run the demo script: ./demo.sh
- Consult the included README for further usage details, or ask a question
- The code is also available on GitHub

Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip
- Ruby script for preprocessing Twitter data

Citing GloVe

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. [pdf] [bib]

Highlights

1. **Nearest neighbors**

   The Euclidean distance (or cosine similarity) between two word vectors provides an effective method for measuring the linguistic or semantic similarity of the corresponding words. Sometimes, the nearest neighbors according to this metric reveal rare but relevant words that lie outside an average human's vocabulary. For example, here are the closest words to the target word *frog*:

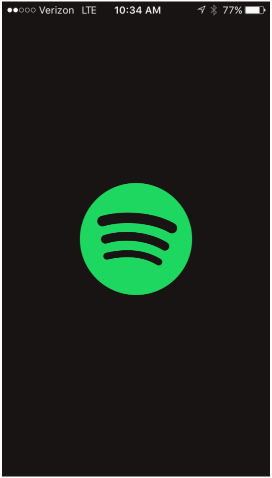
   0. *frog*
   1. frogs
   2. toad
   3. litoria
   4. leptodactylidae
   5. rana
   6. lizard
   7. eleutherodactylus

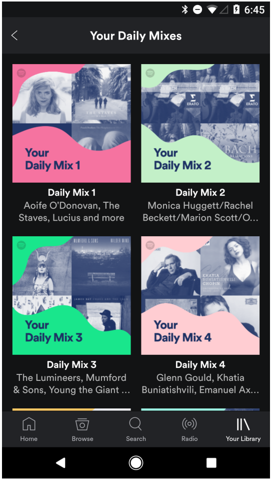
   3. litoria 4. leptodactylidae 5. rana 7. eleutherodactylus

2. **Linear substructures**

   The similarity metrics used for nearest neighbor evaluations produce a single scalar that quantifies the relatedness of two words. This simplicity can be problematic since two given words almost always exhibit more intricate relationships than can be captured by a single number. For example, *man* may be regarded as similar to *woman* in that both words describe human beings; on the other hand, the two words are often considered opposites since they highlight a primary axis along which humans differ from one another.

Recommendation via Embedding





Notebook

Let's go to the Python notebook!

Embedding embeddings: t-SNE

- How can we visualize the embeddings?

Embedding embeddings: t-SNE

- How can we visualize the embeddings?
- We’re in a very high dimensional space

Embedding embeddings: t-SNE

- How can we visualize the embeddings?
- We're in a very high dimensional space
- Could use PCA—this will tend to distort more

Embedding embeddings: t-SNE

- How can we visualize the embeddings?
- We’re in a very high dimensional space
- Could use PCA—this will tend to distort more
- Many visualization techniques exist. A currently popular one is t-SNE: “Student-t Stochastic Neighborhood Embedding”

t-SNE

Here's the idea behind t-SNE:

- Form a language model using the embeddings

Pronounced: **tee**-snee

t-SNE

Here's the idea behind t-SNE:

- Form a language model using the embeddings
- Scale and symmetrize, giving a matrix $P = [P_{ij}]$

Pronounced: **tee**-snee

t-SNE

Here's the idea behind t-SNE:

- Form a language model using the embeddings
- Scale and symmetrize, giving a matrix $P = [P_{ij}]$
- Represent word i by $y_i \in \mathbb{R}^2$. Use a heavy-tailed distribution (Student-t)

Pronounced: **tee**-snee

t-SNE

Here's the idea behind t-SNE:

- Form a language model using the embeddings
- Scale and symmetrize, giving a matrix $P = [P_{ij}]$
- Represent word i by $y_i \in \mathbb{R}^2$. Use a heavy-tailed distribution (Student-t)
- Select y_i using stochastic gradient descent

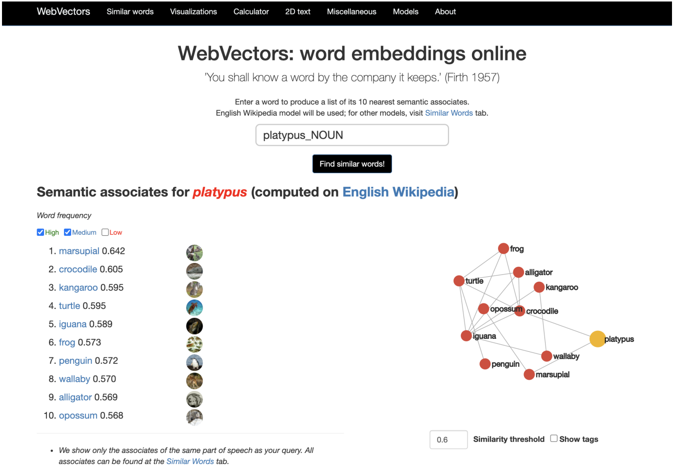
Pronounced: **tee**-snee

t-SNE: More info and examples

https://lvdmaaten.github.io/tsne/
http://cs.stanford.edu/people/karpathy/tsnejs/

Note: This is just a visualization technique, to give intuition for the high dimensional embedding

Embedding / Visualization Examples



http://vectors.nlpl.eu/explore/embeddings/en/

Summary: Word embeddings

- Word embeddings are vector representations of words, learned from cooccurrence statistics
- The models can be viewed in terms of language modeling, pointwise mutual information, and regression
- Surprising semantic relations are encoded in linear relations
- Embeddings improve with more data
- t-SNE is an algorithm for visualizing embeddings

extra slides (optional)

t-SNE: Detailed algorithm

For each word w_i compute a language model

$$P_{j\,|\,i} \propto \exp\left(-\frac{\|\phi(w_i) - \phi(w_j)\|^2}{2h_i^2}\right)$$

That is:

$$P_{j\,|\,i} = \frac{\exp\left(-\frac{\|\phi(w_i) - \phi(w_j)\|^2}{2h_i^2}\right)}{\sum_k \exp\left(-\frac{\|\phi(w_i) - \phi(w_k)\|^2}{2h_i^2}\right)}$$

Pronounced: **tee**-snee

t-SNE: Detailed algorithm

For each word w_i compute a language model

$$P_{j\,|\,i} \propto \exp\left(-\frac{\|\phi(w_i) - \phi(w_j)\|^2}{2h_i^2}\right)$$

That is:

$$P_{j\,|\,i} = \frac{\exp\left(-\frac{\|\phi(w_i) - \phi(w_j)\|^2}{2h_i^2}\right)}{\sum_k \exp\left(-\frac{\|\phi(w_i) - \phi(w_k)\|^2}{2h_i^2}\right)}$$

Choose the bandwith h_i so that the perplexity is, say, 10. This puts the probabilities all on the same scale.

Pronounced: **tee**-snee

t-SNE: Detailed algorithm

For each word w_i compute a language model

$$P_{j\,|\,i} \propto \exp\left(-\frac{\|\phi(w_i) - \phi(w_j)\|^2}{2h_i^2}\right)$$

t-SNE: Detailed algorithm

For each word w_i compute a language model

$$P_{j\,|\,i} \propto \exp\left(-\frac{\|\phi(w_i) - \phi(w_j)\|^2}{2h_i^2}\right)$$

Now form

$$P_{ij} = \frac{1}{2}\left(P_{j\,|\,i} + P_{i\,|\,j}\right)$$

as a simple way of symmetrizing.

t-SNE: Detailed algorithm

Now form Student-t distribution depending on the visualization vectors $y_i \in \mathbb{R}^2$:

$$Q_{ij} \propto \left(1 + \|y_i - y_j\|^2\right)^{-1}$$

t-SNE: Detailed algorithm

Now form Student-t distribution depending on the visualization vectors $y_i \in \mathbb{R}^2$:

$$Q_{ij} \propto \left(1 + \|y_i - y_j\|^2\right)^{-1}$$

That is:

$$Q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq \ell} \left(1 + \|y_k - y_\ell\|^2\right)^{-1}}$$

This has fatter tails than a Gaussian

t-SNE: Detailed algorithm

Finally, run stochastic gradient descent (SGD) over the vectors y_i to optimize:

$$\begin{aligned}\widehat{y} &= \arg\min \sum_{ij} P_{ij} \log P_{ij}/Q_{ij} \\ &= \arg\max \sum_{ij} P_{ij} \log Q_{ij}\end{aligned}$$

Interpretation: if $\phi(w_i)$ is very close to $\phi(w_j)$ then y_i will be close to y_j. (long distances may be stretched further...)