Natural Language Processing with Deep Learning

CS224N/Ling284

Christopher Manning

Lecture 1: Introduction and Word Vectors
Lecture Plan

Lecture 1: Introduction and Word Vectors
1. The course (10 mins)
2. Human language and word meaning (15 mins)
3. Word2vec introduction (15 mins)
4. Word2vec objective function gradients (25 mins)
5. Optimization basics (5 mins)
6. Looking at word vectors (10 mins or less)

Key learning today: The (really surprising!) result that word meaning can be represented rather well by a (high-dimensional) vector of real numbers
Course logistics in brief

• Instructor: Christopher Manning
• Head TA: Anna Goldie
• Coordinator: Amelie Byun
• TAs: Many wonderful people! See website
• Time: Tu/Th 3:15–4:45 Pacific time, Zoom U. (→ video)

• We’ve put a lot of other important information on the class webpage. Please read it!
  • http://cs224n.stanford.edu/
    a.k.a., http://www.stanford.edu/class/cs224n/
  • TAs, syllabus, help sessions/office hours, Ed (for all course questions/discussion)
    • Office hours start Thursday evening!
    • Python/numpy and then PyTorch tutorials: First two Fridays 10:00–11:20 Pacific time on Zoom U.
  • Slide PDFs uploaded before each lecture
What do we hope to teach? (A.k.a. “learning goals”)

1. The foundations of the effective modern methods for deep learning applied to NLP
   • Basics first, then key methods used in NLP: Word vectors, feed-forward networks, recurrent networks, attention, encoder-decoder models, transformers, etc.

2. A big picture understanding of human languages and the difficulties in understanding and producing them via computers

3. An understanding of and ability to build systems (in PyTorch) for some of the major problems in NLP:
   • Word meaning, dependency parsing, machine translation, question answering
Course work and grading policy

• 5 x 1-week Assignments: 6% + 4 x 12%: 54%
  • HW1 is released today! Due next Tuesday! At 3:15 p.m.
  • Submitted to Gradescope in Canvas (i.e., using @stanford.edu email for your Gradescope account)
• Final Default or Custom Course Project (1–3 people): 43%
  • Project proposal: 5%, milestone: 5%, poster or web summary: 3%, report: 30%
• Participation: 3%
  • Guest lecture reactions, Ed, course evals, karma – see website!
• Late day policy
  • 6 free late days; afterwards, 1% off course grade per day late
  • Assignments not accepted more than 3 days late per assignment unless given permission in advance
• Collaboration policy: Please read the website and the Honor Code!
  Understand allowed collaboration and how to document it: Don’t take code off the web; acknowledge working with other students; write your own assignment solutions
High-Level Plan for Assignments (to be completed individually!)

• Ass1 is hopefully an easy on ramp – a Jupyter/IPython Notebook
• Ass2 is pure Python (numpy) but expects you to do (multivariate) calculus so you really understand the basics
• Ass3 introduces PyTorch, building a feed-forward network for dependency parsing
• Ass4 and Ass5 use PyTorch on a GPU (Microsoft Azure)
  • Libraries like PyTorch and Tensorflow are now the standard tools of DL
• For Final Project, more details presented later, but you either:
  • Do the default project, which is a question answering system
    • Open-ended but an easier start; a good choice for many
  • Propose a custom final project, which we approve
    • You will receive feedback from a mentor (TA/prof/postdoc/PhD)
  • Can work in teams of 1–3; can use any language/packages
Lecture Plan

1. The course (10 mins)
2. Human language and word meaning (15 mins)
3. Word2vec introduction (15 mins)
4. Word2vec objective function gradients (25 mins)
5. Optimization basics (5 mins)
6. Looking at word vectors (10 mins or less)
...Anyway, I could care less.

I think you mean you couldn’t care less. Saying you could care less implies you care at least some amount.

I dunno.

We’re these unbelievably complicated brains drifting through a void, trying in vain to connect with one another by blindly flinging words out into the darkness.

Every choice of phrasing and spelling and tone and timing carries countless signals and contexts and subtexts and more, and every listener interprets those signals in their own way.

Language isn’t a formal system. Language is glorious chaos.

You can never know for sure what any words will mean to anyone.

All you can do is try to get better at guessing how your words affect people, so you can have a chance of finding the ones that will make them feel something like what you want them to feel.

Everything else is pointless.

I assume you’re giving me tips on how you interpret words because you want me to feel less alone.

If so, then thank you, that means a lot.

But if you’re just running my sentences past some mental checklist so you can show off how well you know it,

Then I could care less.
Trained on text data, neural machine translation is quite good!

Malawi yawayapoteza mawaziri 2 kutokana na maafa ya COVID-19
TUKO.co.ke imefahamishwa kuwa waziri wa serikali ya mitaa Lingson Belekanyama na mwenzake wa uchukuzi Sidik Mia walifariki dunia ndani ya saa mbili tofauti.

Malawi loses 2 ministers due to COVID-19 disaster
TUKO.co.ke has been informed that local government minister Lingson Belekanyama and his transport counterpart Sidik Mia died within two separate hours.

https://kiswahili.tuko.co.ke/
The SEC said, “Musk, your tweets are a blight. They really could cost you your job, if you don't stop all this tweeting at night.”

Then Musk cried, “Why? The tweets I wrote are not mean, I don't use all-caps and I'm sure that my tweets are clean.”

“But your tweets can move markets and that's why we're sore. You may be a genius and a billionaire, but it doesn't give you the right to be a bore!”

How many users have signed up since the start of 2020?
SELECT count(id) FROM users WHERE created_at > '2020-01-01'

What is the average number of influencers each user is subscribed to?
SELECT avg(count) FROM ( SELECT user_id, count(*) FROM subscribers GROUP BY user_id ) AS avg_subscriptions_per_user
How do we represent the meaning of a word?

Definition: **meaning** (Webster dictionary)
- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.
- the idea that is expressed in a work of writing, art, etc.

Commonest linguistic way of thinking of meaning:

- signifier (symbol) ⇔ signified (idea or thing)
- = denotational semantics

- tree ⇔ {🌳, 🌲, 🌴, ...}
How do we have usable meaning in a computer?

Previously commonest NLP solution: Use, e.g., WordNet, a thesaurus containing lists of synonym sets and hypernyms (“is a” relationships)

e.g., synonym sets containing “good”:

```python
from nltk.corpus import wordnet as wn
poses = { 'n': 'noun', 'v': 'verb', 's': 'adj (s)', 'a': 'adj', 'r': 'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()], ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
...
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g., hypernyms of “panda”:

```python
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
 Synset('carnivore.n.01'),
 Synset('placental.n.01'),
 Synset('mammal.n.01'),
 Synset('vertebrate.n.01'),
 Synset('chordate.n.01'),
 Synset('animal.n.01'),
 Synset('organism.n.01'),
 Synset('living_thing.n.01'),
 Synset('whole.n.02'),
 Synset('object.n.01'),
 Synset('physical_entity.n.01'),
 Synset('entity.n.01')]
```
Problems with resources like WordNet

- A useful resource but missing nuance:
  - e.g., "proficient" is listed as a synonym for "good"
    This is only correct in some contexts
  - Also, WordNet list offensive synonyms in some synonym sets without any coverage of the connotations or appropriateness of words
- Missing new meanings of words:
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Can’t be used to accurately compute word similarity (see following slides)
Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols: hotel, conference, motel – a localist representation

Such symbols for words can be represented by one-hot vectors:

motel = \[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0\]
hotel = \[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0\]

Vector dimension = number of words in vocabulary (e.g., 500,000+)
Problem with words as discrete symbols

Example: in web search, if a user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel”

But:

\[
\begin{align*}
motel &= [0 0 0 0 0 0 0 0 0 1 0 0 0 0] \\
hotel &= [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
\end{align*}
\]

These two vectors are orthogonal

There is no natural notion of similarity for one-hot vectors!

Solution:

• Could try to rely on WordNet’s list of synonyms to get similarity?
  • But it is well-known to fail badly: incompleteness, etc.

• Instead: learn to encode similarity in the vectors themselves
Representing words by their context

- **Distributional semantics**: A word’s meaning is given by the words that frequently appear close-by
  - “You shall know a word by the company it keeps” (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word \( w \) appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of \( w \) to build up a representation of \( w \)

\[
\ldots \text{government debt problems turning into } \text{banking} \text{ crises as happened in 2009}\ldots \\
\ldots \text{saying that Europe needs unified } \text{banking} \text{ regulation to replace the hodgepodge}\ldots \\
\ldots \text{India has just given its } \text{banking} \text{ system a shot in the arm}\ldots \\
\]

These context words will represent **banking**
Word vectors

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector dot (scalar) product.

Note: word vectors are also called (word) embeddings or (neural) word representations.
They are a distributed representation.

\[
\text{banking} = \begin{pmatrix}
0.286 \\
0.792 \\
-0.177 \\
-0.107 \\
0.109 \\
-0.542 \\
0.349 \\
0.271
\end{pmatrix}
\]

\[
\text{monetary} = \begin{pmatrix}
0.413 \\
0.582 \\
-0.007 \\
0.247 \\
0.216 \\
-0.718 \\
0.147 \\
0.051
\end{pmatrix}
\]
Word meaning as a neural word vector – visualization

\[
\text{expect} = \begin{pmatrix}
-0.177 \\
-0.107 \\
0.109 \\
-0.542 \\
0.349 \\
0.271 \\
0.487 \\
\end{pmatrix}
\]
3. Word2vec: Overview

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors.

Idea:

- We have a large corpus (“body”) of text: a long list of words.
- Every word in a fixed vocabulary is represented by a vector.
- Go through each position $t$ in the text, which has a center word $c$ and context (“outside”) words $o$.
- Use the similarity of the word vectors for $c$ and $o$ to calculate the probability of $o$ given $c$ (or vice versa).
- Keep adjusting the word vectors to maximize this probability.
Example windows and process for computing $P(w_{t+j} | w_t)$
Word2Vec Overview

Example windows and process for computing $P(w_{t+j} | w_t)$
Word2vec: objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size $m$, given center word $w_j$. Data likelihood:

$$L(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} \mid w_t; \theta)$$

$\theta$ is all variables to be optimized

Sometimes called a cost or loss function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\iff$ Maximizing predictive accuracy
Word2vec: objective function

• We want to minimize the objective function:

\[ J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} | w_t; \theta) \]

• **Question:** How to calculate \( P(w_{t+j} | w_t; \theta) \)?

• **Answer:** We will use two vectors per word \( w \):
  
  • \( v_w \) when \( w \) is a center word
  
  • \( u_w \) when \( w \) is a context word

• Then for a center word \( c \) and a context word \( o \):

\[
P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}
\]
Word2Vec with Vectors

- Example windows and process for computing $P(w_{t+j} \mid w_t)$
- $P(u_{\text{problems}} \mid v_{\text{into}})$ short for $P(\text{problems} \mid \text{into} ; u_{\text{problems}}, v_{\text{into}}, \theta)$

All words vectors $\theta$ appear in denominator
Word2vec: prediction function

2. Exponentiation makes anything positive

\[ P(o \mid c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \]

1. Dot product compares similarity of \( o \) and \( c \).

\[ u^T v = u \cdot v = \sum_{i=1}^{n} u_i v_i \]

Larger dot product = larger probability

3. Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** \( \mathbb{R}^n \rightarrow (0,1)^n \)

\[ \text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i \]

- The softmax function maps arbitrary values \( x_i \) to a probability distribution \( p_i \)
  - “max” because amplifies probability of largest \( x_i \)
  - “soft” because still assigns some probability to smaller \( x_i \)
  - Frequently used in Deep Learning

27
To train the model: Optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

- Recall: $\theta$ represents all the model parameters, in one long vector
- In our case, with $d$-dimensional vectors and $V$-many words, we have →
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

- We optimize these parameters by walking down the gradient (see right figure)
- We compute all vector gradients!
4. Word2vec derivations of gradient

• Zoom U. Whiteboard – see video or revised slides

• The basic Lego piece: The chain rule

• Useful basic fact: $$\frac{\partial x^T a}{\partial x} = \frac{\partial a^T x}{\partial x} = a$$

• If in doubt: write it out with indices
Chain Rule

- Chain rule! If $y = f(u)$ and $u = g(x)$, i.e., $y = f(g(x))$, then:

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx} = \frac{df(u)}{du} \frac{dg(x)}{dx}$$

- Simple example:

$$\frac{dy}{dx} = \frac{d}{dx} 5(x^3 + 7)^4$$

$$y = f(u) = 5u^4$$

$$u = g(x) = x^3 + 7$$

$$\frac{dy}{du} = 20u^3$$

$$\frac{du}{dx} = 3x^2$$

$$\frac{dy}{dx} = 20(x^3 + 7)^3 \cdot 3x^2$$
Let’s derive gradient for center word together
For one example window and one example outside word:
\[
J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j} | w_t)
\]

\[
\log p(o|c) = \log \frac{\exp(u_o^T v_c)}{\sum_{V}^{v} \exp(u_w^T v_c)}
\]

You then also need the gradient for context words (it’s similar; left for homework). That’s all of the parameters \( \theta \) here.
Calculating all gradients!

- We went through the gradient for each center vector $v$ in a window.
- We also need gradients for outside vectors $u$.
  - Derive at home!
- Generally, in each window we will compute updates for all parameters that are being used in that window. For example:
Word2vec: More details

Why two vectors? ➔ Easier optimization. Average both at the end.

Two model variants:

1. Skip-grams (SG)
   Predict context (“outside”) words (position independent) given center word

2. Continuous Bag of Words (CBOW)
   Predict center word from (bag of) context words

This lecture so far: Skip-gram model

Additional efficiency in training:

1. Negative sampling

So far: Focus on naïve softmax (simpler but more expensive training method)
5. Optimization: Gradient Descent

- We have a cost function $J(\theta)$ we want to minimize
- **Gradient Descent** is an algorithm to minimize $J(\theta)$
- **Idea:** for current value of $\theta$, calculate gradient of $J(\theta)$, then take small step in direction of negative gradient. Repeat.

Note: Our objectives may not be convex like this 😞

But life turns out to be okay 😊
Gradient Descent

• Update equation (in matrix notation):

\[ \theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta) \]

\[ \alpha = \text{step size or learning rate} \]

• Update equation (for single parameter):

\[ \theta_{j}^{new} = \theta_{j}^{old} - \alpha \frac{\partial}{\partial \theta_{j}^{old}} J(\theta) \]

• Algorithm:

```python
while True:
    theta_grad = evaluate_gradient(J, corpus, theta)
    theta = theta - alpha * theta_grad
```
Stochastic Gradient Descent

- **Problem**: $J(\theta)$ is a function of all windows in the corpus (potentially billions!)
  - So $\nabla_{\theta} J(\theta)$ is very expensive to compute
  - You would wait a very long time before making a single update!

- **Very** bad idea for pretty much all neural nets!
- **Solution**: Stochastic gradient descent (SGD)
  - Repeatedly sample windows, and update after each one
- Algorithm:

```python
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J, window, theta)
    theta = theta - alpha * theta_grad
```
Lecture Plan

1. The course (10 mins)
2. Human language and word meaning (15 mins)
3. Word2vec introduction (15 mins)
4. Word2vec objective function gradients (25 mins)
5. Optimization basics (5 mins)
6. Looking at word vectors (10 mins or less)