Natural Language Processing with Deep Learning

Introduction and Word Vectors
What do we hope to teach?

1. An understanding of the effective modern methods for deep learning
   • Basics first, then key methods used in NLP: Recurrent networks, attention, etc.

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

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
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:

\[
\begin{align*}
\text{signifier (symbol)} & \iff \text{signified (idea or thing)} \\
= \text{denotational semantics}
\end{align*}
\]

cf. connotational: implied feeling, context, etc
How do we have usable meaning in a computer?

Common 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

• Great as a resource but missing nuance
  • e.g. “proficient” is listed as a synonym for “good”. This is only correct in some contexts.

• 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 compute accurate word similarity
Representing words as discrete symbols

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

Means one 1, the rest 0s

Words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
hotel = [0 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 user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel”.

But:

\[
\text{motel} = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]
\]
\[
\text{hotel} = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0]
\]

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).

• Use the many contexts of $w$ to build up a representation of $w$

  …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**
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.

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

Note: word vectors are sometimes called word embeddings or word representations. They are a distributed representation.
Word meaning as a neural word vector – visualization

expect =

\[
\begin{pmatrix}
0.286 \\
0.792 \\
-0.177 \\
-0.107 \\
0.109 \\
-0.542 \\
0.349 \\
0.271 \\
0.487
\end{pmatrix}
\]
Word2vec: Overview

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

Idea:
• We have a large corpus of text
• 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
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 $w_j$.

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

$\theta$ is all variables to be optimized

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

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

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

• We want to minimize the objective function:

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

• **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 Overview with Vectors

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

![Diagram](image)

- $\ldots$ problems turning into banking crises as $\ldots$
- Outside context words in window of size 2
- Center word at position $t$
- Outside context words in window of size 2
**Word2vec: prediction function**

Exponentiation makes anything positive

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

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

Normalize over entire vocabulary to give probability distribution

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

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

- 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
Training a model by optimizing parameters

To train a model, we adjust parameters to minimize a loss. E.g., below, for a simple convex function over two parameters.

Contour lines show levels of objective function.
To train the model: Compute all vector gradients!

- Recall: $\theta$ represents all model parameters, in one long vector
- In our case with $d$-dimensional vectors and $V$-many words:

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

- Remember: every word has two vectors
- We optimize these parameters by walking down the gradient
Word2vec derivations of gradient

• The basic Lego piece

• Useful basics: 
  \[
  \frac{\partial x^T a}{\partial x} = \frac{\partial a^T x}{\partial x} = a
  \]

• If in doubt: write out with indices

• Chain rule! If \( y = f(u) \) and \( u = g(x) \), i.e. \( y = f(g(x)) \), then:
  \[
  \frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}
  \]
Interactive Whiteboard Session!

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

Let's derive gradient for center word together
For one example window and one example outside word:

\[ \log p(o|c) = \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \]

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 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**
   1 positive and 2–5 negative word pairs

So far: Focus on **naïve softmax** (simpler training method)
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 :(
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
  ```
  cf. mini-batch update
Review: Main idea of word2vec

- Iterate through each word of the whole corpus
- Predict surrounding words using word vectors

\[ P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \]
Word2vec parameters and computations

\[
\begin{bmatrix}
\vdots & \vdots & \vdots & \cdots \\
\vdots & \vdots & \vdots & \cdots \\
\vdots & \vdots & \vdots & \cdots \\
\end{bmatrix} \quad \begin{bmatrix}
\vdots & \vdots & \vdots & \cdots \\
\vdots & \vdots & \vdots & \cdots \\
\vdots & \vdots & \vdots & \cdots \\
\end{bmatrix} \quad \begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix} \quad \begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\]

\(U\) outside \quad \(V\) center \quad \(U \cdot v_4^T\) dot product \quad \text{softmax}(U \cdot v_4^T) probabilities

Same predictions at each position

We want a model that gives a reasonably high probability estimate to all words that occur in the context (fairly often)
Word2vec maximizes objective function by putting similar words nearby in space.
Stochastic gradients with word vectors!

- Iteratively take gradients at each such window for SGD
- But in each window, we only have at most $2m + 1$ words, so $\nabla_\theta J_t(\theta)$ is very sparse!

\[
\nabla_\theta J_t(\theta) = \begin{bmatrix}
0 \\
\vdots \\
\nabla_{v_{like}} \\
\vdots \\
0 \\
\nabla_{u_I} \\
\vdots \\
\nabla_{u_{learning}} \\
\vdots \\
\end{bmatrix} \in \mathbb{R}^{2dV}
\]
Stochastic gradients with word vectors!

- We might only update the word vectors that actually appear!

- Solution: either you need sparse matrix update operations to only update certain rows of full embedding matrices $U$ and $V$, or you need to keep around a hash for word vectors.

- If you have millions of word vectors and do distributed computing, it is important to not have to send gigantic updates around!
The skip-gram model with negative sampling

- The normalization factor is too computationally expensive.

- \[ P(o|c) = \frac{\exp(u_o^Tv_c)}{\sum_{w \in V} \exp(u_w^Tv_c)} \]

- Hence, in standard word2vec you implement the skip-gram model with negative sampling

- Main idea: train binary logistic regressions for a true pair (center word and word in its context window) versus several noise pairs (the center word paired with a random word)
The skip-gram model with negative sampling

- From paper: “Distributed Representations of Words and Phrases and their Compositionality” (Mikolov et al. 2013)
- Overall objective function (they maximize): \( J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta) \)

\[
J_t(\theta) = \log \sigma \left( u_o^T v_c \right) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)} \left[ \log \sigma \left( -u_j^T v_c \right) \right]
\]

- The sigmoid function: \( \sigma(x) = \frac{1}{1+e^{-x}} \)
  (we’ll become good friends soon)
- So we maximize the probability of two words co-occurring in first log
The skip-gram model with negative sampling

- Notation more similar to class:

\[
J_{\text{neg-sample}}(o, v_c, U) = -\log(\sigma(u_o^T v_c)) - \sum_{k=1}^{K} \log(\sigma(-u_k^T v_c))
\]

- We take \( k \) negative samples (using word probabilities)
- Maximize probability that real outside word appears, minimize prob. that random words appear around center word

- \( P(w) = U(w)^{3/4}/Z \), (sampling prob for random words)
  the unigram distribution \( U(w) \) raised to the 3/4 power.
- The power makes less frequent words be sampled more often
But why not capture co-occurrence counts directly?

With a co-occurrence matrix $X$

- 2 options: windows vs. full document
- Window: Similar to word2vec, use window around each word → captures both syntactic (POS) and semantic information
- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to “Latent Semantic Analysis”
Example: Window based co-occurrence matrix

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.
Window based co-occurrence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

<table>
<thead>
<tr>
<th>counts</th>
<th>I</th>
<th>like</th>
<th>enjoy</th>
<th>deep</th>
<th>learning</th>
<th>NLP</th>
<th>flying</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>like</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>enjoy</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>deep</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>learning</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NLP</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>flying</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

*window size = 1
Problems with simple co-occurrence vectors

Increase in size with vocabulary

Very high dimensional: requires a lot of storage

Subsequent classification models have sparsity issues

→ Models are less robust
Solution: Low dimensional vectors

- Idea: store “most” of the important information in a fixed, small number of dimensions: a dense vector

- Usually 25–1000 dimensions, similar to word2vec

- How to reduce the dimensionality?
Dimensionality Reduction on $X$

Singular Value Decomposition of co-occurrence matrix $X$

Factorizes $X$ into $U\Sigma V^T$, where $U$ and $V$ are orthonormal

Retain only $k$ singular values, in order to generalize. $\hat{X}$ is the best rank $k$ approximation to $X$, in terms of least squares. Classic linear algebra result. Expensive to compute for large matrices.
Simple SVD word vectors in Python

Corpus:
I like deep learning. I like NLP. I enjoy flying.

```python
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy", "deep", "learning", "NLP", "flying", "."]
X = np.array([[0, 2, 1, 0, 0, 0, 0, 0],
              [2, 0, 0, 1, 0, 1, 0, 0],
              [1, 0, 0, 0, 0, 0, 1, 0],
              [0, 1, 0, 0, 1, 0, 0, 0],
              [0, 0, 0, 1, 0, 0, 0, 1],
              [0, 1, 0, 0, 0, 0, 0, 1],
              [0, 0, 1, 0, 0, 0, 0, 1],
              [0, 0, 0, 0, 0, 1, 1, 1]])

U, s, Vh = la.svd(X, full_matrices=False)
```
Simple SVD word vectors in Python

Corpus: I like deep learning. I like NLP. I enjoy flying.
Printing first two columns of U corresponding to the 2 biggest singular values

*2dim word vectors

```python
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```
Scaling the counts in the cells can help a lot

- Problem: function words (the, he, has) are too frequent → syntax has too much impact. Some fixes:
  - $\min(X,t)$, with $t \approx 100$
  - Ignore them all
  - Ramped windows that count closer words more
  - Use Pearson correlations instead of counts, then set negative values to 0
  - Etc.
Interesting syntactic patterns emerge in the vectors

COALS model from
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. ms., 2005
Interesting semantic patterns emerge in the vectors

COALS model from
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. ms., 2005
## Count based vs. direct prediction

<table>
<thead>
<tr>
<th>Count based Models</th>
<th>Direct Prediction Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSA, HAL (Lund &amp; Burgess), COALS, Hellinger-PCA (Rohde et al, Lebret &amp; Collobert)</td>
<td>Skip-gram/CBOW (Mikolov et al) NNLM, HLBL, RNN (Bengio et al; Collobert &amp; Weston; Huang et al; Mnih &amp; Hinton)</td>
</tr>
</tbody>
</table>

- **Fast training**
- **Efficient usage of statistics**
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

- **Scales with corpus size**
- **Inefficient usage of statistics**
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity
**Crucial insight:** Ratios of co-occurrence probabilities can encode meaning components

<table>
<thead>
<tr>
<th></th>
<th>$x = \text{solid}$</th>
<th>$x = \text{gas}$</th>
<th>$x = \text{water}$</th>
<th>$x = \text{random}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(x</td>
<td>\text{ice})$</td>
<td>large</td>
<td>small</td>
<td>large</td>
</tr>
<tr>
<td>$P(x</td>
<td>\text{steam})$</td>
<td>small</td>
<td>large</td>
<td>large</td>
</tr>
<tr>
<td>(- \frac{P(x</td>
<td>\text{ice})}{P(x</td>
<td>\text{steam})})</td>
<td>large</td>
<td>small</td>
</tr>
</tbody>
</table>
**Crucial insight:** Ratios of co-occurrence probabilities can encode meaning components

<table>
<thead>
<tr>
<th></th>
<th>$x = \text{solid}$</th>
<th>$x = \text{gas}$</th>
<th>$x = \text{water}$</th>
<th>$x = \text{fashion}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(x</td>
<td>\text{ice})$</td>
<td>$1.9 \times 10^{-4}$</td>
<td>$6.6 \times 10^{-5}$</td>
<td>$3.0 \times 10^{-3}$</td>
</tr>
<tr>
<td>$P(x</td>
<td>\text{steam})$</td>
<td>$2.2 \times 10^{-5}$</td>
<td>$7.8 \times 10^{-4}$</td>
<td>$2.2 \times 10^{-3}$</td>
</tr>
<tr>
<td>$\frac{P(x</td>
<td>\text{ice})}{P(x</td>
<td>\text{steam})}$</td>
<td>8.9</td>
<td>$8.5 \times 10^{-2}$</td>
</tr>
</tbody>
</table>
Encoding meaning in vector differences

Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

A: Log-bilinear model:

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

with vector differences

\[ w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)} \]
Combining the best of both worlds

GloVe  [Pennington et al., EMNLP 2014]

Global vectors

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

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

left side = right side => minimize J

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors

\[ f \sim \]
GloVe results

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus
How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs extrinsic
- Intrinsic:
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
  - Evaluation on a real task
  - Can take a long time to compute accuracy
  - Unclear if the subsystem is the problem or its interaction or other subsystems
  - If replacing exactly one subsystem with another improves accuracy → Winning!
Intrinsic word vector evaluation

- Word Vector Analogies
  
  \[ a : b : c : ? \]

  man : woman :: king : ?

- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions

- Discarding the input words from the search!

- Problem: What if the information is there but not linear?

\[
 d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}
\]
Glove Visualizations
Glove Visualizations: Company - CEO
Glove Visualizations: Superlatives
Details of intrinsic word vector evaluation

- Word Vector Analogies: Syntactic and **Semantic** examples from 
  http://code.google.com/p/word2vec/source/browse/trunk/questions-words.txt

: city-in-state
Chicago Illinois Houston Texas
Chicago Illinois Philadelphia Pennsylvania
Chicago Illinois Phoenix Arizona
Chicago Illinois Dallas Texas
Chicago Illinois Jacksonville Florida
Chicago Illinois Indianapolis Indiana
Chicago Illinois Austin Texas
Chicago Illinois Detroit Michigan
Chicago Illinois Memphis Tennessee
Chicago Illinois Boston Massachusetts

problem: different cities may have same name
Details of intrinsic word vector evaluation

- Word Vector Analogies: **Syntactic** and Semantic examples from:

  : gram4-superlative
  bad worst big biggest
  bad worst bright brightest
  bad worst cold coldest
  bad worst cool coolest
  bad worst dark darkest
  bad worst easy easiest
  bad worst fast fastest
  bad worst good best
  bad worst great greatest
Analogy evaluation and hyperparameters

- Glove word vectors evaluation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ivLBL</td>
<td>100</td>
<td>1.5B</td>
<td>55.9</td>
<td>50.1</td>
<td>53.2</td>
</tr>
<tr>
<td>HPCA</td>
<td>100</td>
<td>1.6B</td>
<td>4.2</td>
<td>16.4</td>
<td>10.8</td>
</tr>
<tr>
<td>GloVe</td>
<td>100</td>
<td>1.6B</td>
<td>67.5</td>
<td>54.3</td>
<td>60.3</td>
</tr>
<tr>
<td>SG</td>
<td>300</td>
<td>1B</td>
<td>61</td>
<td>61</td>
<td>61</td>
</tr>
<tr>
<td>CBOw</td>
<td>300</td>
<td>1.6B</td>
<td>16.1</td>
<td>52.6</td>
<td>36.1</td>
</tr>
<tr>
<td>vLBL</td>
<td>300</td>
<td>1.5B</td>
<td>54.2</td>
<td>64.8</td>
<td>60.0</td>
</tr>
<tr>
<td>ivLBL</td>
<td>300</td>
<td>1.5B</td>
<td>65.2</td>
<td>63.0</td>
<td>64.0</td>
</tr>
<tr>
<td>GloVe</td>
<td>300</td>
<td>1.6B</td>
<td>80.8</td>
<td>61.5</td>
<td>70.3</td>
</tr>
<tr>
<td>SVD</td>
<td>300</td>
<td>6B</td>
<td>6.3</td>
<td>8.1</td>
<td>7.3</td>
</tr>
<tr>
<td>SVD-S</td>
<td>300</td>
<td>6B</td>
<td>36.7</td>
<td>46.6</td>
<td>42.1</td>
</tr>
<tr>
<td>SVD-L</td>
<td>300</td>
<td>6B</td>
<td>56.6</td>
<td>63.0</td>
<td>60.1</td>
</tr>
<tr>
<td>CBOw†</td>
<td>300</td>
<td>6B</td>
<td>63.6</td>
<td>67.4</td>
<td>65.7</td>
</tr>
<tr>
<td>SG†</td>
<td>300</td>
<td>6B</td>
<td>73.0</td>
<td>66.0</td>
<td>69.1</td>
</tr>
<tr>
<td>GloVe</td>
<td>300</td>
<td>6B</td>
<td>77.4</td>
<td>67.0</td>
<td>71.7</td>
</tr>
<tr>
<td>CBOw</td>
<td>1000</td>
<td>6B</td>
<td>57.3</td>
<td>68.9</td>
<td>63.7</td>
</tr>
<tr>
<td>SG</td>
<td>1000</td>
<td>6B</td>
<td>66.1</td>
<td>65.1</td>
<td>65.6</td>
</tr>
<tr>
<td>SVD-L</td>
<td>300</td>
<td>42B</td>
<td>38.4</td>
<td>58.2</td>
<td>49.2</td>
</tr>
<tr>
<td>GloVe</td>
<td>300</td>
<td>42B</td>
<td>81.9</td>
<td>69.3</td>
<td>75.0</td>
</tr>
</tbody>
</table>
Analogy evaluation and hyperparameters

- Good dimension is ~300
- Asymmetric context (only words to the left) are not as good *for semantics, but good for syntactic
- But this might be different for downstream tasks!
- Window size of 8 around each center word is good for Glove vectors

**Figure 2:** Accuracy on the analogy task as function of vector size and window size/type. All models are trained on the 6 billion token corpus. In (a), the window size is 10. In (b) and (c), the vector size is 100.

<table>
<thead>
<tr>
<th>Dimensionality</th>
<th>Window size</th>
<th>Window size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall</td>
<td>Syntactic</td>
<td>Semantic</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
<td>80</td>
</tr>
</tbody>
</table>

Table: Accuracy [%]

<table>
<thead>
<tr>
<th>Overall</th>
<th>Syntactic</th>
<th>Semantic</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>Accuracy</td>
<td>Accuracy</td>
<td>Accuracy</td>
</tr>
<tr>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
</tr>
<tr>
<td>80</td>
<td>90</td>
<td>100</td>
<td>110</td>
</tr>
</tbody>
</table>

- For all our experiments, we set \( \epsilon = 0.05 \). We run 50 iterations for vectors smaller than 300 dimensions, and 100 iterations otherwise (see Section 4.6 for more details about the convergence rate). Unless otherwise noted, we use a context of seven words to the left and seven words to the right. We explore the effect of these choices below.

- When \( \mathbf{W} \) is symmetric, \( \mathbf{d} = \mathbf{d}^{\top} \). We must choose how large the context window should be and whether to distinguish left context from right context. We choose window sizes of 2, 3, 4, and 5 for \( \mathbf{d} \) in order to evaluate the impact of context size on accuracy.

- In constructing \( \mathbf{W} \), we must account for the fact that very distant word pairs contribute 1 to the occurrence counts. This number is evaluated on a subset of the Gigaword5 + Wikipedia2014, combination Gigaword5 + Wikipedia2014, which has 6 billion tokens; Gigaword 5 which has 4.3 billion tokens; the 2010 Wikipedia dump with 1 billion tokens; and a 2014 Wikipedia dump with 1.6 billion tokens. We trained our model on five corpora of varying sizes: a 2010 Wikipedia dump with 1 billion tokens; a 2014 Wikipedia dump with 1.6 billion tokens; Gigaword 5 which has 4.3 billion tokens; and the combined Gigaword5 + Wikipedia2014, which has 6 billion tokens. In (a), the window size is 10. In (b) and (c), the vector size is 100.

- With this in mind, we choose to use a 2010 Wikipedia dump with 1 billion tokens for the CoNLL-2003 training dataset. In addition, 50-dimensional discrete features were generated for the CoNLL-2003 model (Finkel et al., 2005). A total of 437,905 annotated documents from Reuters newswire articles, including WordSim-353 (Finkelstein et al., 2001), were used for the analogy task. This number is evaluated on a subset of the combination Gigaword5 + Wikipedia2014, which has 6 billion tokens. In (a), the window size is 10. In (b) and (c), the vector size is 100.

- The model generates two sets of word vectors, \( \mathbf{W} \) and \( \mathbf{W}^{\top} \), as well as all the preprocessing steps described in (Wang and Manning, 2013). We use a standard distribution of the Stanford NER model (Finkel et al., 2005). A total of 437,905 annotated documents from Reuters newswire articles, including WordSim-353 (Finkelstein et al., 2001), were used for the analogy task. This number is evaluated on a subset of the combination Gigaword5 + Wikipedia2014, which has 6 billion tokens. In (a), the window size is 10. In (b) and (c), the vector size is 100.

- Formally, we define \( \mathbf{d} \) as a vector containing the counts of words within a window centered at the current word. We also define \( \mathbf{D} \) as a matrix of occurrence counts. We explore the effect of these choices below.

- We use the Stanford NER model (Finkel et al., 2005) and a comprehensive set of discrete features that comes with it. We use a standard distribution of the Stanford NER model (Finkel et al., 2005). A total of 437,905 annotated documents from Reuters newswire articles, including WordSim-353 (Finkelstein et al., 2001), were used for the analogy task. This number is evaluated on a subset of the combination Gigaword5 + Wikipedia2014, which has 6 billion tokens. In (a), the window size is 10. In (b) and (c), the vector size is 100.

- The CoNLL-2003 dataset is a comprehensive and widely used English benchmark dataset for Named entity recognition. The model generates two sets of word vectors, \( \mathbf{W} \) and \( \mathbf{W}^{\top} \), as well as all the preprocessing steps described in (Wang and Manning, 2013). We use a standard distribution of the Stanford NER model (Finkel et al., 2005). A total of 437,905 annotated documents from Reuters newswire articles, including WordSim-353 (Finkelstein et al., 2001), were used for the analogy task. This number is evaluated on a subset of the combination Gigaword5 + Wikipedia2014, which has 6 billion tokens. In (a), the window size is 10. In (b) and (c), the vector size is 100.

- The CoNLL-2003 dataset is a comprehensive and widely used English benchmark dataset for Named entity recognition. The model generates two sets of word vectors, \( \mathbf{W} \) and \( \mathbf{W}^{\top} \), as well as all the preprocessing steps described in (Wang and Manning, 2013). We use a standard distribution of the Stanford NER model (Finkel et al., 2005). A total of 437,905 annotated documents from Reuters newswire articles, including WordSim-353 (Finkelstein et al., 2001), were used for the analogy task. This number is evaluated on a subset of the combination Gigaword5 + Wikipedia2014, which has 6 billion tokens. In (a), the window size is 10. In (b) and (c), the vector size is 100.
Analogies evaluation and hyperparameters

- More training time helps
Analogy evaluation and hyperparameters

- More data helps, Wikipedia is better than news text!

### Table 4: F1 score on NER task with 50d vectors.

<table>
<thead>
<tr>
<th>Model</th>
<th>Dev</th>
<th>Test</th>
<th>ACE</th>
<th>MUC7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete</td>
<td>91.0</td>
<td>85.4</td>
<td>77.4</td>
<td>73.4</td>
</tr>
<tr>
<td>SVD</td>
<td>90.8</td>
<td>85.7</td>
<td>77.3</td>
<td>73.7</td>
</tr>
<tr>
<td>SVD-S</td>
<td>91.0</td>
<td>85.5</td>
<td>77.6</td>
<td>74.3</td>
</tr>
<tr>
<td>SVD-L</td>
<td>90.5</td>
<td>84.8</td>
<td>73.6</td>
<td>71.5</td>
</tr>
<tr>
<td>HPCA</td>
<td>92.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HSMN</td>
<td>90.5</td>
<td>85.7</td>
<td>78.7</td>
<td>74.7</td>
</tr>
<tr>
<td>CW</td>
<td>92.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBOW</td>
<td>93.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GloVe</td>
<td>93.2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

shown for neural vectors in (Turian et al., 2010).

### 4.4 Model Analysis: Vector Length and Context Size

In Fig. 2, we show the results of experiments that vary vector length and context window. A context window that extends to the left and right of a target word will be called symmetric, and one which extends only to the left will be called asymmetric. In (a), we observe diminishing returns for vectors larger than about 200 dimensions. In (b) and (c), we examine the effect of varying the window size for symmetric and asymmetric context windows. Performance is better on the syntactic subtask for small and asymmetric context windows, which aligns with the intuition that syntactic information is mostly drawn from the immediate context and can depend strongly on word order. Semantic information, on the other hand, is more frequently non-local, and more of it is captured with larger window sizes.

### 4.5 Model Analysis: Corpus Size

In Fig. 3, we show performance on the word analogy task for 300-dimensional vectors trained on different corpora. On the syntactic subtask, there is a monotonic increase in performance as the corpus size increases. This is to be expected since larger corpora typically produce better statistics. Interestingly, the same trend is not true for the semantic subtask, where the models trained on the smaller Wikipedia corpora do better than those trained on the larger Gigaword corpus. This is likely due to the large number of city- and country-based analogies in the analogy dataset and the fact that Wikipedia has fairly comprehensive articles for most such locations. Moreover, Wikipedia's entries are updated to assimilate new knowledge, whereas Gigaword is a fixed news repository with outdated and possibly incorrect information.

### 4.6 Model Analysis: Run-time

The total run-time is split between populating X and training the model. The former depends on many factors, including window size, vocabulary size, and corpus size. Though we did not do so, this step could easily be parallelized across multiple machines (see, e.g., Lebret and Collobert (2014) for some benchmarks). Using a single thread of a dual 2.1GHz Intel Xeon E5-2658 machine, populating X with a 10 word symmetric context window, a 400,000 word vocabulary, and a 6 billion token corpus takes about 85 minutes. Given X, the time it takes to train the model depends on the vector size and the number of iterations. For 300-dimensional vectors with the above settings (and using all 32 cores of the above machine), a single iteration takes 14 minutes. See Fig. 4 for a plot of the learning curve.

### 4.7 Model Analysis: Comparison with word2vec

A rigorous quantitative comparison of GloVe with word2vec is complicated by the existence of many parameters that have a strong effect on performance. We control for the main sources of variation that we identified in Sections 4.4 and 4.5 by setting the vector length, context window size, corpus, and vocabulary size to the configuration mentioned in the previous subsection. The most important remaining variable to control for is training time. For GloVe, the relevant parameter is the number of training iterations. For word2vec, the obvious choice would be the number of training epochs. Unfortunately, the code is currently designed for only a single epoch.
Another intrinsic word vector evaluation

- Word vector distances and their correlation with human judgments
- Example dataset: WordSim353

<table>
<thead>
<tr>
<th>Word 1</th>
<th>Word 2</th>
<th>Human (mean)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tiger</td>
<td>cat</td>
<td>7.35</td>
</tr>
<tr>
<td>tiger</td>
<td>tiger</td>
<td>10</td>
</tr>
<tr>
<td>book</td>
<td>paper</td>
<td>7.46</td>
</tr>
<tr>
<td>computer</td>
<td>internet</td>
<td>7.58</td>
</tr>
<tr>
<td>plane</td>
<td>car</td>
<td>5.77</td>
</tr>
<tr>
<td>professor</td>
<td>doctor</td>
<td>6.62</td>
</tr>
<tr>
<td>stock</td>
<td>phone</td>
<td>1.62</td>
</tr>
<tr>
<td>stock</td>
<td>CD</td>
<td>1.31</td>
</tr>
<tr>
<td>stock</td>
<td>jaguar</td>
<td>0.92</td>
</tr>
</tbody>
</table>
Correlation evaluation

- Word vector distances and their correlation with human judgments

<table>
<thead>
<tr>
<th>Model</th>
<th>Size</th>
<th>WS353</th>
<th>MC</th>
<th>RG</th>
<th>SCWS</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVD</td>
<td>6B</td>
<td>35.3</td>
<td>35.1</td>
<td>42.5</td>
<td>38.3</td>
<td>25.6</td>
</tr>
<tr>
<td>SVD-S</td>
<td>6B</td>
<td>56.5</td>
<td>71.5</td>
<td>71.0</td>
<td>53.6</td>
<td>34.7</td>
</tr>
<tr>
<td>SVD-L</td>
<td>6B</td>
<td>65.7</td>
<td>72.7</td>
<td>75.1</td>
<td>56.5</td>
<td>37.0</td>
</tr>
<tr>
<td>CBOW†</td>
<td>6B</td>
<td>57.2</td>
<td>65.6</td>
<td>68.2</td>
<td>57.0</td>
<td>32.5</td>
</tr>
<tr>
<td>SG†</td>
<td>6B</td>
<td>62.8</td>
<td>65.2</td>
<td>69.7</td>
<td>58.1</td>
<td>37.2</td>
</tr>
<tr>
<td>GloVe</td>
<td>6B</td>
<td>65.8</td>
<td>72.7</td>
<td>77.8</td>
<td>53.9</td>
<td>38.1</td>
</tr>
<tr>
<td>SVD-L</td>
<td>42B</td>
<td>74.0</td>
<td>76.4</td>
<td>74.1</td>
<td>58.3</td>
<td>39.9</td>
</tr>
<tr>
<td>GloVe</td>
<td>42B</td>
<td>75.9</td>
<td>83.6</td>
<td>82.9</td>
<td>59.6</td>
<td>47.8</td>
</tr>
<tr>
<td>CBOW*</td>
<td>100B</td>
<td>68.4</td>
<td>79.6</td>
<td>75.4</td>
<td>59.4</td>
<td>45.5</td>
</tr>
</tbody>
</table>

- Some ideas from Glove paper have been shown to improve skip-gram (SG) model also (e.g. sum both vectors)
Word senses and word sense ambiguity

• Most words have lots of meanings!
  • Especially common words
  • Especially words that have existed for a long time

• Example: pike

• Does one vector capture all these meanings or do we have a mess?
pike

• A sharp point or staff
• A type of elongated fish
• A railroad line or system
• A type of road
• The future (coming down the pike)
• A type of body position (as in diving)
• To kill or pierce with a pike
• To make one’s way (pike along)
• In Australian English, pike means to pull out from doing something: I reckon he could have climbed that cliff, but he piked!
Improving Word Representations Via Global Context And Multiple Word Prototypes (Huang et al. 2012)

- Idea: Cluster word windows around words, retrain with each word assigned to multiple different clusters bank₁, bank₂, etc.
Linear Algebraic Structure of Word Senses, with Applications to Polysemy

- Different senses of a word reside in a linear superposition (weighted sum) in standard word embeddings like word2vec
  \[ \vec{v}_{\text{pike}} = \alpha_1 \vec{v}_{\text{pike}_1} + \alpha_2 \vec{v}_{\text{pike}_2} + \alpha_3 \vec{v}_{\text{pike}_3} \]
- Where \( \alpha_1 = \frac{f_1}{f_1 + f_2 + f_3} \), etc., for frequency \( f \)
- Surprising result:
  - Because of ideas from sparse coding you can actually separate out the senses (providing they are relatively common)

<table>
<thead>
<tr>
<th>tie</th>
<th>trousers</th>
<th>season</th>
<th>scoreline</th>
<th>wires</th>
<th>operatic</th>
</tr>
</thead>
<tbody>
<tr>
<td>blouse</td>
<td></td>
<td>teams</td>
<td>goalless</td>
<td>cables</td>
<td>soprano</td>
</tr>
<tr>
<td>waistcoat</td>
<td></td>
<td>winning</td>
<td>equaliser</td>
<td>wiring</td>
<td>mezzo</td>
</tr>
<tr>
<td>skirt</td>
<td></td>
<td>league</td>
<td>clinching</td>
<td>electrical</td>
<td>contralto</td>
</tr>
<tr>
<td>sleeved</td>
<td></td>
<td>finished</td>
<td>scoreless</td>
<td>wire</td>
<td>baritone</td>
</tr>
<tr>
<td>pants</td>
<td></td>
<td>championship</td>
<td>replay</td>
<td>cable</td>
<td>coloratura</td>
</tr>
</tbody>
</table>
Extrinsic word vector evaluation

- Extrinsic evaluation of word vectors: All subsequent tasks in this class

- One example where good word vectors should help directly: named entity recognition: finding a person, organization or location

<table>
<thead>
<tr>
<th>Model</th>
<th>Dev</th>
<th>Test</th>
<th>ACE</th>
<th>MUC7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete</td>
<td>91.0</td>
<td>85.4</td>
<td>77.4</td>
<td>73.4</td>
</tr>
<tr>
<td>SVD</td>
<td>90.8</td>
<td>85.7</td>
<td>77.3</td>
<td>73.7</td>
</tr>
<tr>
<td>SVD-S</td>
<td>91.0</td>
<td>85.5</td>
<td>77.6</td>
<td>74.3</td>
</tr>
<tr>
<td>SVD-L</td>
<td>90.5</td>
<td>84.8</td>
<td>73.6</td>
<td>71.5</td>
</tr>
<tr>
<td>HPCA</td>
<td>92.6</td>
<td><strong>88.7</strong></td>
<td>81.7</td>
<td>80.7</td>
</tr>
<tr>
<td>HSMN</td>
<td>90.5</td>
<td>85.7</td>
<td>78.7</td>
<td>74.7</td>
</tr>
<tr>
<td>CW</td>
<td>92.2</td>
<td>87.4</td>
<td>81.7</td>
<td>80.2</td>
</tr>
<tr>
<td>CBOW</td>
<td>93.1</td>
<td>88.2</td>
<td>82.2</td>
<td>81.1</td>
</tr>
<tr>
<td>GloVe</td>
<td><strong>93.2</strong></td>
<td>88.3</td>
<td><strong>82.9</strong></td>
<td><strong>82.2</strong></td>
</tr>
</tbody>
</table>

- Next: How to use word vectors in neural net models!