NLP Pipeline

Language Identification → Tokenization → Part-of-Speech (POS) Tagging → Shallow Parsing (Chunking) → Named Entity Recognition (NER)

Stemming

Normalization
NLP Pipeline

classification (Naïve Bayes) → Regular Expression
Language Identification → Tokenization → Part-of-Speech (POS) Tagging → Shallow Parsing (Chunking) → Named Entity Recognition (NER)

Stemming → Normalization

Sequential Tagging
Part-of-Speech (POS) Tagging

<table>
<thead>
<tr>
<th>Cant</th>
<th>MD</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait</td>
<td>VB</td>
</tr>
<tr>
<td>for</td>
<td>IN</td>
</tr>
<tr>
<td>the</td>
<td>DT</td>
</tr>
<tr>
<td>ravens</td>
<td>NNP</td>
</tr>
<tr>
<td>game</td>
<td>NN</td>
</tr>
<tr>
<td>tomorrow</td>
<td>NN</td>
</tr>
<tr>
<td>...</td>
<td>:</td>
</tr>
<tr>
<td>go</td>
<td>VB</td>
</tr>
<tr>
<td>ray</td>
<td>NNP</td>
</tr>
<tr>
<td>rice</td>
<td>NNP</td>
</tr>
<tr>
<td>!!!!!!!</td>
<td>.</td>
</tr>
</tbody>
</table>
Named Entity Recognition

Source: Strauss, Toma, Ritter, de Marneffe, Xu
Results of the WNUT16 Named Entity Recognition Shared Task (WNUT@COLING 2016)
Timeline of NLP on Microblogs

2010
- First Story Detection (Petrovic et al.)
- Geographic Variation (Eisenstein et al.)
- Dialog Modeling (Ritter et al.)

2011
- POS Tagging (Gimpel et al.)
- Normalization (Han and Baldwin)
- Named Entity Recog. (Ritter et al.)
- Summarization (Liu et. al)

2012
- Open-Domain Event Extraction (Ritter et al.)
- Censorship Detection (Bamman et al.)
Timeline of NLP on Microblogs

- **Summarization** (Xu et al.)
- **Normalization** (Xu et al.)
- **Paraphrase Extraction** (Xu et al.)
- **Named Entity Recognition** (Cherry and Guo)

- **Machine Translation** (Ling et al.)
  - **POS** (Owoputi et al.)
- **Parsing Weibo** (Wang et al.)
- **Dialogues Modeling** (Sordoni et al.)
- **Parsing Twitter** (Kong et al.)

2013

2014

2015
BAD LANGUAGE!
...on the INTERNET!!

Boom! Ya ur website suxx bro

michelle obama great. job. and. whit all my. respect she. look. great. congrats. to. her.

...dats why pluto is pluto it can neva be a star

What can we do about it?
Why don't they just write NORMALLY??
Can our software ever ADAPT???
How does language go bad?

Illiteracy? No. (Tagliamonte and Denis 2008; Drouin and Davis 2009)

Length limits? (probably not)

Hardware input constraints? (Gouws et al 2011)

Social variables
- Non-standard language does identity work, signaling authenticity, solidarity, etc.
- Social variation is usually inhibited in written language, but social media is less regulated than other written genres.

Source: Jacob Eisenstein
Why is Social Media Text “Bad”?

• Lack of literacy? no [Drouin and Davis, 2009]
• Length restrictions? not primarily [Eisenstein, 2013]
• Text input method? to some degree, yes [Gouws et al., 2011]
• Pragmatics (mimicking prosodic effects etc. in speech)? yeeees [Eisenstein, 2013]
• Social variables/markers of social identity? blood oath! [Eisenstein, 2013]

Source: Jacob Eisenstein & Tim Baldwin
Why is Social Media Text “Bad”?

- Pragmatics (mimicking prosodic effects etc. in speech)? yeeyees [Eisenstein, 2013]

**HELLA**

Derived from "hell of a lot". Similar to "very, really, a lot," etc.

Used mostly in Northern California though has been heard in other parts of CA and even in the media such as an infamous "hella" South Park episode. (Cartman used it outside of its meaning to annoy Kyle.)

*Before:* There's a hell of a lot of beer in that fridge.

*After:* There's hella beer in that fridge.

*As "very" or "really":*

"That's hella far away!"
Why is Social Media Text “Bad”?

- Social variables/markers of social identity? blood oath! [Eisenstein, 2013]

“I would like to believe he’s sick rather than just mean and evil.”

“You could’ve been getting down to this sick beat.”

Source: Yang Yi and Jacob Eisenstein (TACL 2017) Overcoming Language Variation in Sentiment Analysis with Social Attention
Text Normalization

- convert non-standard words to standard

Source: Tim Baldwin, Marie de Marneffe, Han Bo, Young-Bum Kim, Alan Ritter, Wei Xu
Shared Tasks of the 2015 Workshop on Noisy User-generated Text: Twitter Lexical Normalization and Named Entity Recognition
An Unsupervised Learning Method:

(1) Brown Clustering

• Input:
  - a (large) text corpus

• Output:
  1. a partition of words into word clusters
  2. or a hierarchical word clustering (generalization of 1)
Brown Clustering

- Example Clusters (from Brown et al. 1992)

Source: Miller, Guinness, Zamanian (NAACL 2004)
Name Tagging with Word Clusters and Discriminative Training
Hierarchical Word Clustering

• Each intermediate node is a cluster:

```
bit string representation

000 001
000 001
apple pear Apple IBM
```

Levels:
- Level 0: Root node
- Level 1: Intermediate nodes
- Level 2: Sub-clusters
- Level 3: Further sub-clusters
- Level 4: Leaf nodes (e.g., apple, pear, Apple, IBM)
Hierarchical Word Clustering

- Example Clusters
  (from Miller et al. 2004)

<table>
<thead>
<tr>
<th>Word</th>
<th>Binary Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>mailman</td>
<td>100000110101111</td>
</tr>
<tr>
<td>salesman</td>
<td>100000110110000</td>
</tr>
<tr>
<td>bookkeeper</td>
<td>1000001101100010</td>
</tr>
<tr>
<td>troubleshooter</td>
<td>1000001101100011</td>
</tr>
<tr>
<td>bouncer</td>
<td>1000001101100111</td>
</tr>
<tr>
<td>technician</td>
<td>1000001101100100</td>
</tr>
<tr>
<td>janitor</td>
<td>1000001101100101</td>
</tr>
<tr>
<td>saleswoman</td>
<td>1000001101100110</td>
</tr>
</tbody>
</table>

...
Hierarchical Word Clustering

- Example Clusters (from Miller et al. 2004)

<table>
<thead>
<tr>
<th>mailman</th>
<th>1000011010111</th>
</tr>
</thead>
<tbody>
<tr>
<td>salesman</td>
<td>1000011011000</td>
</tr>
<tr>
<td>bookkeeper</td>
<td>1000011010000</td>
</tr>
<tr>
<td>troubleshooter</td>
<td>1000011011000</td>
</tr>
<tr>
<td>bouncer</td>
<td>1000011010011</td>
</tr>
<tr>
<td>technician</td>
<td>1000011010010</td>
</tr>
<tr>
<td>janitor</td>
<td>1000011011011</td>
</tr>
<tr>
<td>saleswoman</td>
<td>1000011011010</td>
</tr>
<tr>
<td>Nike</td>
<td>101101100100101</td>
</tr>
<tr>
<td>Maytag</td>
<td>101101100101011</td>
</tr>
<tr>
<td>Generali</td>
<td>101101100101011</td>
</tr>
<tr>
<td>Gap</td>
<td>101101100100111</td>
</tr>
<tr>
<td>Harley-Davidson</td>
<td>101101100101011</td>
</tr>
<tr>
<td>Enfield</td>
<td>101101100101011</td>
</tr>
<tr>
<td>genus</td>
<td>101101100101011</td>
</tr>
<tr>
<td>Microsoft</td>
<td>101101100100110</td>
</tr>
<tr>
<td>Ventritex</td>
<td>101101100101000</td>
</tr>
<tr>
<td>Tractebel</td>
<td>101101100101000</td>
</tr>
<tr>
<td>Synopsys</td>
<td>101101100101000</td>
</tr>
<tr>
<td>WordPerfect</td>
<td>101101100101000</td>
</tr>
<tr>
<td>John</td>
<td>101110010000000</td>
</tr>
<tr>
<td>Consuelo</td>
<td>101110010000001</td>
</tr>
<tr>
<td>Jeffrey</td>
<td>101110001000000</td>
</tr>
<tr>
<td>Kenneth</td>
<td>101110001000000</td>
</tr>
<tr>
<td>Phillip</td>
<td>101110001000000</td>
</tr>
<tr>
<td>WILLIAM</td>
<td>101110001000000</td>
</tr>
<tr>
<td>Timothy</td>
<td>101110001000000</td>
</tr>
</tbody>
</table>

word cluster features
(bit string prefix)

Source: Miller, Guinness, Zamanian (NAACL 2004)
Name Tagging with Word Clusters and Discriminative Training
Challenges in Twitter

2m 2ma 2mar 2mara 2maro 2marrow 2mor 2mora 2moro 2morow 2morr 2morro 2morrow 2moz 2mr 2mro 2mrrw 2mrw 2mw tmmrw tmo tmoro tmorrow tmoz tmr tmro tmrow tmrrow tmrrw tmrw tmrww tmw tomaro tomarow tomararro tomararrow tomm tommarow tommarrow tommoro tommorow tommorrow tommoroww tommrow tomo tomoz tomorow tomorro tomorroww tomoz tomrw tomz
# Clusters in Twitter NER

<table>
<thead>
<tr>
<th>System</th>
<th>Fin10Dev</th>
<th>Rit11</th>
<th>Fro14</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoNLL</td>
<td>27.3</td>
<td>27.1</td>
<td>29.5</td>
<td>28.0</td>
</tr>
<tr>
<td>+ Brown</td>
<td>38.4</td>
<td>39.4</td>
<td>42.5</td>
<td>40.1</td>
</tr>
<tr>
<td>+ Vector</td>
<td>40.8</td>
<td>40.4</td>
<td>42.9</td>
<td>41.4</td>
</tr>
<tr>
<td>+ Reps</td>
<td>42.4</td>
<td>42.2</td>
<td>46.2</td>
<td>43.6</td>
</tr>
<tr>
<td>Fin10</td>
<td>36.7</td>
<td>29.0</td>
<td>30.4</td>
<td>32.0</td>
</tr>
<tr>
<td>+ Brown</td>
<td>59.9</td>
<td>53.9</td>
<td>56.3</td>
<td>56.7</td>
</tr>
<tr>
<td>+ Vector</td>
<td>61.5</td>
<td>56.4</td>
<td>58.4</td>
<td>58.8</td>
</tr>
<tr>
<td>+ Reps</td>
<td>64.0</td>
<td>58.5</td>
<td>60.2</td>
<td>60.9</td>
</tr>
<tr>
<td>CoNLL+Fin10</td>
<td>44.7</td>
<td>39.9</td>
<td>44.2</td>
<td>42.9</td>
</tr>
<tr>
<td>+ Brown</td>
<td>54.9</td>
<td>52.9</td>
<td>58.5</td>
<td>55.4</td>
</tr>
<tr>
<td>+ Vector</td>
<td>58.9</td>
<td>55.2</td>
<td>59.9</td>
<td>58.0</td>
</tr>
<tr>
<td>+ Reps</td>
<td>58.9</td>
<td>56.4</td>
<td>61.8</td>
<td>59.0</td>
</tr>
<tr>
<td>+ Weights</td>
<td>64.4</td>
<td>59.6</td>
<td>63.3</td>
<td>62.4</td>
</tr>
</tbody>
</table>

Table 5: Impact of our components on Twitter NER performance, as measured by F1, under 3 data scenarios.

Source: Colin Cherry, Hongyu Guo (NAACL 2015)
Clusters in Twitter NER

Brown clusters, for each $i$ s.t. $s \leq i < t$:

$$\{[y_j, \text{brn}(n, x_i), n]\}_{n \in \{2, 4, 8, 12\}},$$

$$\{[y_j, \text{er}_{s,t}(i), \text{brn}(n, x_i), n]\}_{n \in \{2, 4, 8, 12\}}$$

Word vectors, for each $i$ s.t. $s \leq i < t$:

$$\{[y_j, n] = \text{w2v}(n, x_i)\}_{n=1}^{300},$$

$$\{[y_j, \text{er}_{s,t}(i), n] = \text{w2v}(n, x_i)\}_{n=1}^{300}$$

Table 2: Word representation features in $\phi(s, t, y_j, x)$.

$\text{brn}(n, x_i)$ maps a word $x_i$ to the first $n$ bits of its Brown cluster bit sequence. $\text{w2v}(n, x_i)$ maps $x_i$ to the $n^{th}$ component of its word vector, and $[\text{str}] = v$ stands for a real-valued feature with name $\text{str}$ and value $v$. 

Source: Colin Cherry, Hongyu Guo (NAACL 2015)
Brown Clustering

• The Intuition:
  - similar words appear in similar contexts
  - more precisely: similar words have similar distributions of words to their immediate left and right
Brown Clustering Algorithm

• An agglomerative clustering algorithm:
  - take the top m most frequent words, put each into its own cluster, \(c_1, c_2, \ldots, c_m\)
  - repeat for \(i = (m+1) \ldots |V|\)
    - create a new cluster \(c_{m+1}\) for the \(i\)’th most frequent word
    - choose two clusters from \(c_1, c_2, \ldots, c_{m+1}\) to be merged, which give the highest \textit{Quality} based on a training corpus
Brown Clustering Algorithm

- maximize the **Quality** function that score a given partitioning $C$:

  \[
  \text{Quality}(C) = \sum_{i} \log e(w_i | C(w_i))q(C(w_i) | C(w_{i-1}))
  \]

  \[
  = \sum_{c=1}^{k} \sum_{c'=1}^{k} p(c,c') \log \frac{p(c,c')}{p(c)p(c')} + G
  \]

  - $n(c)$: count of class $c$ seen in the corpus
  - $n(c,c')$: counts of $c'$ seen following $c$

  \[
  p(c,c') = \frac{n(c,c')}{\sum_{c,c'} n(c,c')} \quad p(c,c') = \frac{n(c)}{\sum_{c} n(c)}
  \]

Learn more: Percy Liang's phd thesis - Semi-Supervised Learning for Natural Language
Brown Clustering

C++ implementation of the Brown word clustering algorithm.

Implementation of the Brown hierarchical word clustering algorithm.
Percy Liang
Release 1.3
2012.07.24

Input: a sequence of words separated by whitespace (see input.txt for an example).
Output: for each word type, its cluster (see output.txt for an example).

<cluster represented as a bit string> <word> <number of times word occurs in input>
Word Vector Representations
(a.k.a. “word embeddings”)

- 4 kinds of vector semantic models

1. Hard clustering (e.g. Brown clustering)
2. Soft clustering (e.g. SVD, LSA, LDA)
3. Neural Network inspired models (e.g. skip-grams and CBOW in word2vec)
4. Mutual-information weighted word co-occurrence metrics
Hard vs. Soft Clustering

**Hard clustering**
Each observation belongs to exactly one cluster

**Soft clustering**
An observation can belong to more than one cluster to a certain degree (e.g. likelihood of belonging to the cluster)

In Contrast To

represent word meaning by a taxonomy like WordNet

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

synonym sets (good):

[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')]

S: (adj) full, good
S: (adj) estimable, good, honorable, respectable
S: (adj) beneficial, good
S: (adj) good, just, upright
S: (adj) adept, expert, good, practiced, proficient, skillful
S: (adj) dear, good, near
S: (adj) good, right, ripe
...
S: (adv) well, good
S: (adv) thoroughly, soundly, good
S: (n) good, goodness
S: (n) commodity, trade good, good
In Contrast To represent word meaning by a taxonomy like WordNet

• problems with this discrete representation:
  
  - missing new words (impossible to keep up-to-date): 
    wicked, badass, nifty, crack, ace, wizard, genius, ninja
  
  - requires human labor to create and adapt
  
  - hard to compute accurate word similarity
  
  - and apparently not enough to handle social media data!
Distributional Intuition

• From context words, human can guess a word’s meaning:

A bottle of *tesgüino* is on the table
Everybody likes *tesgüino*
*Tesgüino* makes you drunk
We make *tesgüino* out of corn.

“You shall know a word by the company it keeps”
— J. R. Firth 1957
Distributional Intuition

• From context words, human can guess a word’s meaning:

A bottle of *tesgüino* is on the table
Everybody likes *tesgüino*
*Tesgüino* makes you drunk
We make *tesgüino* out of corn.

• similar words = similar contexts = similar vectors

• word meaning is represented by a vector of numbers
Simple Co-occurrence Vectors

- Option #1: word-document co-occurrence counts

<table>
<thead>
<tr>
<th></th>
<th>As You Like It</th>
<th>Twelfth Night</th>
<th>Julius Caesar</th>
<th>Henry V</th>
</tr>
</thead>
<tbody>
<tr>
<td>battle</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>soldier</td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>fool</td>
<td>37</td>
<td>58</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>clown</td>
<td>6</td>
<td>117</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

This will give general topics (e.g. sports terms will have similar entries), leading to Latent Semantic Analysis.
Simple Co-occurrence Vectors

- Option #2: use a sliding window over a big corpus of text and count word co-occurrences:

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>
</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>
</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>
</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>
</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>
</tr>
<tr>
<td>learning</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>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>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>
</tr>
</tbody>
</table>

This captures both syntactic (POS) and semantic information.
Simple Co-occurrence Vectors

- Problems with this representation of raw counts:
  - increase in size with vocabulary
  - high dimensionality and very sparse!
  - not a great measure of association between words: “the” and “of” are very frequent, but maybe not the most discriminative
  - unable to capture word order
    “I like NLP” and “NLP like I” will have same representation
Lower Dimensional Vectors

• **The Idea**: use dense vectors to store “most” of the important information in a fixed, small number of dimensions

• usually around 25 ~1000 dimensions
Lower Dimensional Vectors

• Word meaning is represented as a dense vector

“linguistic” =

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

How to reduce the dimensionality?
(2) Matrix Factorization

• Singular Value Decomposition (SVD)

\[ X = U S V^T \]

\[ \hat{X} = \hat{U} \hat{S} \hat{V}^T \]

\( \hat{X} \) is the best rank \( k \) approximation to \( X \), in terms of least squares.
SVD Word Vectors

example 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, 1, 1, 1, 0]])

U, s, Vh = la.svd(X, full_matrices=False)
```
SVD Word Vectors

- plot first 2 columns of U corresponding to the 2 biggest singular values:

```python
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```
Some Hacks

• Problem: function words ("the", "he", "has") are too frequent → syntax has too much impact.
  - fixes: cap the counts, or ignore them all

• ramped windows that count closer words more

• etc …

Source: Rohde et al. (2005) An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Zipf’s (Power) Law

- frequency of word is inversely proportional to its rank in the frequency table

Source: http://www.shadycharacters.co.uk/
Zipf’s (Power) Law

Source: smashing magazine
Clustering Vectors

- visualize similarity

Source: Rohde et al. (2005)

An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Interesting Syntactic Patterns

Source: Rohde et al. (2005)

An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Interesting Semantic Patterns

Source: Rohde et al. (2005)

An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
SVD Word Vectors

• Still some problems:
  - computational cost scales quadratically for $m \times n$ matrix — $O(mn^2)$ when $n < m$
  - hard to use large corpus (and vocabulary)
  - hard to incorporate new words or documents
(3) Neural Word Embeddings

• **The Idea**: directly learn low-dimensional word vectors

• **… can go back to 1980s:**
  - Learning Representations by Back-Propagating Errors (Rumelhart et al., 1986)
  - **A Neural Probabilistic Language Model** (Bengio et al., 2003)
  - NLP from Scratch (Collobert & Weston, 2008)
  - **Word2vec** (Mikolov et al. 2013)
Neural Word Embeddings

• The Basic Idea:

- We define a model that aims to predict a word given its context words (word vectors), which has a loss function, e.g. $J = 1 - P(\text{context} \mid w_i)$

- We look at many positions of $t$ in a big text corpus,

- and keep adjusting the word vectors to minimize this loss.
Word2vec

- simple and efficient

Source: Mikolov et al. (NIPS 2013)
Word2vec

• Skip-gram — predicts surrounding “outside” words given the “center” word
Word2vec

- Skip-gram — predicts surrounding “outside” words given the “center” word

Figure 16.5: The skip-gram model viewed as a network (Mikolov et al. 2013, Mikolov et al. 2013a).
Input Layer

- “one-hot” word vectors
  - a vector of dimension $|V|$ (size of vocabulary)
  - all “0”s expect a single “1” in the vector
  - different positions of that “1” represent different words
Hidden (Projection) Layer

- A simple look up — the rows of this weight matrix are actually “input” word vectors.
Hidden (Projection) Layer

• A simple look up — the rows of this weight matrix are actually “input” word vectors

\[
\begin{bmatrix}
0 & 0 & 0 & 1 & 0
\end{bmatrix}
\times
\begin{bmatrix}
17 & 24 & 1 \\
23 & 5 & 7 \\
4 & 6 & 13 \\
10 & 12 & 19 \\
11 & 18 & 25
\end{bmatrix}
= 
\begin{bmatrix}
10 \\
12 \\
19
\end{bmatrix}
\]
Output Layer

- predicts surrounding “outside” (context) words given the “center” word → A classification problem!

- Softmax Regression = Multi-class Logistic Regression
Softmax Function

- Softmax function is a generalization of logistic function

\[
\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}
\]
Softmax Function

- Softmax function is a generalization of logistic function

\[
\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}
\]

exp
normalize (to sum to one)

-2.85
0.86
0.28

0.058
2.36
1.32

0.016
0.631
0.353
Output Layer

- Intuition

Word vector for “ants”

300 features

\[ \times \]

Output weights for “car”

\[ \text{softmax} \]

\[ \frac{e^x}{\sum e^x} \]

= Probability that “car” shows up near “ants”

Source: Chris McCormick
Output Layer

- Objective function: maximize the log probability of any “outside” (context) word given the “center” word

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

• predicts surrounding “outside” (context) words given the “center” word

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

• so, every word has two vectors!
Gradient Descent

- Cost/Objective function:

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

- For a “center” word and an “outside” word:

\[ \log p(o|c) = \log \frac{\exp \left( u_o^T v_c \right)}{\sum_{w=1}^{W} \exp \left( u_w^T v_c \right)} \]
Gradient Descent

• Basics:

\[
\frac{\partial x^T a}{\partial x} = \frac{\partial a^T x}{\partial x} = a
\]

\[
\frac{\partial e^x}{\partial x} = e^x \quad \quad \quad \quad \quad \frac{\partial \log x}{\partial x} = \frac{1}{x}
\]

• Chain Rule:

\[
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} = \frac{\partial f(g)}{\partial g} \frac{\partial g(x)}{\partial x}
\]
Skipgram

\[ V \times 1 \]

\[ d \times V \]

\[ V \times d \]

\[ W \]

\[ V_c = W_w \]

\[ \text{softmax} \]

\[ \text{Truth} \]

\[ W_{e-3} \]

\[ W_{e-2} \]

\[ W_{e-1} \]

\[ e^x \]

\[ \sum_i e^x_i \]

\[ p_i = \frac{e^x_i}{\sum_i e^x_i} \]

One hot word symbol

Looks up column of word embedding matrix as representation of center word

Output word representation
Word Vector Representations
(a.k.a. “word embeddings”)

- 4 kinds of vector semantic models

  1. Hard clustering (e.g. Brown clustering)
  2. Soft clustering (e.g. SVD, LSA, LDA)
  3. Neural Network inspired models (e.g. skip-grams and CBOW in word2vec)
  4. Mutual-information weighted word co-occurrence metrics
Word2vec

• Word2vec is not a single algorithm, but a toolkit
  - which contains two distinct algorithms (Skip-gram & CBOW), two training methods (negative sampling & hierarchical softmax)

• Word2vec is not deep learning, but neural-inspired
  - only one hidden layer followed by softmax, no non-linear activation function

Learn more: Omer Levy’s answer on Quora
https://www.quora.com/How-does-word2vec-work-Can-someone-walk-through-a-specific-example
Word2vec

• Word2vec is not a single algorithm, but a toolkit
  - which contains two distinct algorithms (Skip-gram & CBOW), two training methods (negative sampling & hierarchical softmax)

• Word2vec is not deep learning, but neural-inspired
  - only one hidden layer followed by softmax, no non-linear activation function

Learn more: Omer Levy's answer on Quora
  https://www.quora.com/How-does-word2vec-work-Can-someone-walk-through-a-specific-example
Relation between Skip-gram and SVD

• Levy and Goldberg (2014) show that skip-gram is factorizing (a shifted version of) the traditional word-context PMI matrix:

$$\text{Opt}(\vec{w} \cdot \vec{c}) = \text{PMI}(w, c) - \log k$$

• So does SVD!
Visualization

Male-Female

Verb tense

Country-Capital

Source: tensorflow.org
Visualization

Figure 1: A 2-dimensional projection of the latent semantic space captured by our algorithm. Notice the semantic trajectory of the word *gay* transitioning meaning in the space.

Source: Kulkarni et al. (WWW 2015)
Statistically Significant Detection of Linguistic Change
Thank You!

Instructor: Wei Xu
www.cis.upenn.edu/~xwe/

Course Website: socialmedia-class.org