Search Engine Architecture

9. Distributed Word Representations
Agenda

• Distributed representations / distributional hypothesis
• Dimensionality reduction
• Artificial neural networks
• Representation learning
• Word2vec
  • Skip-gram
  • CBOW
• Doc2vec
• SVD reduction
Word2vec Linear Relationships

Source: Mikolov et al. (2013) “Distributed Representations of Words and Phrases and their Compositionality.” *NIPS.*
Distributional Hypothesis

“You shall know a word by the company it keeps”

(J. R. Firth 1957: 11)
How to represent meaning?

• One answer: use a taxonomy like WordNet

Hypernyms (is-a relationships):

```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')]```

Synonym sets (good):

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

Problems with discrete representation

• Missing nuances
  • adept, expert, good, practiced, proficient, skillful
• Impossible to keep up to date
  • wicked, badass, nifty, legend, ninja
• Subjective
• Requires human labor
• *Hard to compute word similarity*

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016
How can we represent term relations?

• With the standard symbolic encoding of terms, each term is a dimension

• Different terms have no inherent similarity

• \[
\begin{align*}
\text{motel} & \left[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \right]^T \\
\text{hotel} & \left[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 3 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \right] = 0
\end{align*}
\]

• If query on hotel and document has motel, then our query and document vectors are orthogonal

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016
Distributional similarity-based representations

• You can get a lot of value by representing a word by means of its neighbors

• “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

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016
Representing words with neighbors

• Use co-occurrence matrix
• Two options for context – document or window
• Word-document co-occurrence matrix leads to general topics (sports terms will have similar entries)
  • “Latent Semantic Analysis”
• Instead – window around each word captures syntactic and semantic information

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016
Window-based co-occurrence matrix

- 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>

Problems with co-occurrence vectors

- Increase in size with vocabulary
- High dimensional
- Extreme sparsity leads to less robust downstream models
- *How to reduce dimensionality?*

Traditional Way: Latent Semantic Indexing/Analysis

- Use Singular Value Decomposition (SVD) – kind of like Principal Components Analysis (PCA) for an arbitrary rectangular matrix
- Theory is that similarity is preserved as much as possible
- Weakly, you can actually gain in IR by doing LSA as “noise” of term variation gets replaced by semantic “concepts”
Dimensionality Reduction

Word meaning in terms of vectors

- In all subsequent models, including deep learning models, a word is represented as a dense vector

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

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016
What’s the point?

- Word models have many applications
- E.g. find synonyms with Spotify’s Annoy:

Detour:
Artificial Neural Networks
Neuron

Sigmoid (logistic) activation function: \[ g(z) = \frac{1}{1 + e^{-z}} \]

\[ h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \]

\[ x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \]

"bias unit"\[ x_0 = 1 \]
Neural Network

Layer 1 (Input Layer)  Layer 2 (Hidden Layer)  Layer 3 (Output Layer)

Based on slide by Andrew Ng
Feed-Forward Process

• Input layer units are set by some external function (think of these as **sensors**), which causes their output links to be **activated** at the given level

• Working forward through the network, the **input function** of each unit is applied to compute the input value
  
  • Usually this is just the weighted sum of the activation on the links feeding into the node

• The **activation function** transforms this input into a final value
  
  • Typically this is a **nonlinear** function, often a **sigmoid** function corresponding to the “threshold” of the node
Feed-Forward Process

\[ a_i^{(j)} = \text{"activation" of unit } i \text{ in layer } j \]

\[ \Theta^{(j)} = \text{weight matrix controlling function mapping from layer } j \text{ to layer } j + 1 \]

\[
\begin{align*}
    a_1^{(2)} &= g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\
    a_2^{(2)} &= g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\
    a_3^{(2)} &= g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \\
    h_{\Theta}(x) &= a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})
\end{align*}
\]

If network has \( s_j \) units in layer \( j \) and \( s_{j+1} \) units in layer \( j+1 \), then \( \Theta^{(j)} \) has dimension \( s_{j+1} \times (s_j + 1) \).

\[
\begin{align*}
    \Theta^{(1)} &\in \mathbb{R}^{3\times 4} \\
    \Theta^{(2)} &\in \mathbb{R}^{1\times 4}
\end{align*}
\]
Vectorization

\[ a_1^{(2)} = g \left( \Theta_{10} x_0 + \Theta_{11} x_1 + \Theta_{12} x_2 + \Theta_{13} x_3 \right) = g \left( z_1^{(2)} \right) \]

\[ a_2^{(2)} = g \left( \Theta_{20} x_0 + \Theta_{21} x_1 + \Theta_{22} x_2 + \Theta_{23} x_3 \right) = g \left( z_2^{(2)} \right) \]

\[ a_3^{(2)} = g \left( \Theta_{30} x_0 + \Theta_{31} x_1 + \Theta_{32} x_2 + \Theta_{33} x_3 \right) = g \left( z_3^{(2)} \right) \]

\[ h_\Theta(x) = g \left( \Theta_{10} a_0^{(2)} + \Theta_{11} a_1^{(2)} + \Theta_{12} a_2^{(2)} + \Theta_{13} a_3^{(2)} \right) = g \left( z_1^{(3)} \right) \]

\[
\begin{align*}
\mathbf{z}^{(2)} &= \Theta^{(1)} \mathbf{x} \\
\mathbf{a}^{(2)} &= g(\mathbf{z}^{(2)}) \\
\text{Add } a_0^{(2)} &= 1 \\
\mathbf{z}^{(3)} &= \Theta^{(2)} \mathbf{a}^{(2)} \\
h_\Theta(\mathbf{x}) &= \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})
\end{align*}
\]

Based on slide by Andrew Ng
Multiple Output Units: One Vs. Rest

We want:

\[
\begin{align*}
\mathbf{h}_\Theta(\mathbf{x}) &\approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} & \text{when pedestrian} \\
\mathbf{h}_\Theta(\mathbf{x}) &\approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} & \text{when car} \\
\mathbf{h}_\Theta(\mathbf{x}) &\approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} & \text{when motorcycle} \\
\mathbf{h}_\Theta(\mathbf{x}) &\approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} & \text{when truck}
\end{align*}
\]
Learning Via Backpropagation

• Cycle through examples
  • If the output of the network is correct, no changes are made
  • If there is an error, weights are adjusted to reduce the error
• The trick is to assess the blame for the error and divide it among the contributing weights
• Iteratively update weights by gradient descent
Cost Function

Logistic Regression:

\[ J(\theta) = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log h_{\theta}(x_i) + (1 - y_i) \log (1 - h_{\theta}(x_i))] + \frac{\lambda}{2n} \sum_{j=1}^{d} \theta_j^2 \]

Neural Network:

\[ h_{\theta} \in \mathbb{R}^K \]  
\[ (h_{\theta}(x))_i = i^{th} \text{ output} \]

\[ J(\Theta) = -\frac{1}{n} \left[ \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log (h_{\theta}(x_i))_k + (1 - y_{ik}) \log \left(1 - (h_{\theta}(x_i))_k\right) \right] \]

\[ + \frac{\lambda}{2n} \sum_{l=1}^{L-1} \sum_{s_{l-1}=1}^{s_l} \sum_{j=1}^{s_l} \left( \Theta_{ji}^{(l)} \right)^2 \]

\[ k^{th} \text{ class: true, predicted} \]
\[ \text{not } k^{th} \text{ class: true, predicted} \]

Based on slide by Andrew Ng
Backpropagation Intuition

\[ \delta_j^{(l)} = \text{"error" of node } j \text{ in layer } l \]

Formally, \[ \delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(x_i) \]

where \[ \text{cost}(x_i) = y_i \log h_\Theta(x_i) + (1 - y_i) \log(1 - h_\Theta(x_i)) \]
Backpropagation Intuition

\[ \delta_j^{(l)} = \text{“error” of node } j \text{ in layer } l \]

Formally, \[ \delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(x_i) \]

where \[ \text{cost}(x_i) = y_i \log h_\Theta(x_i) + (1 - y_i) \log(1 - h_\Theta(x_i)) \]

Based on slide by Andrew Ng
Backpropagation Intuition

\[ \delta_2^{(2)} = \Theta_{12}^{(2)} \times \delta_1^{(3)} + \Theta_{22}^{(2)} \times \delta_2^{(3)} \]

\( \delta_j^{(l)} \) = “error” of node \( j \) in layer \( l \)

Formally, \( \delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(x_i) \)

where \( \text{cost}(x_i) = y_i \log h_\Theta(x_i) + (1 - y_i) \log (1 - h_\Theta(x_i)) \)

Based on slide by Andrew Ng
Gradient Descent with Backprop

Given: training set \((x_1, y_1), \ldots, (x_n, y_n)\)

Initialize all \(\Theta^{(l)}\) randomly (NOT to 0!)

Loop // each iteration is called an epoch

Set \(\Delta^{(l)}_{ij} = 0\) \(\forall l, i, j\)

For each training instance \((x_i, y_i)\):

Set \(a^{(1)} = x_i\)

Compute \(\{a^{(2)}, \ldots, a^{(L)}\}\) via forward propagation

Compute \(\delta^{(L)} = a^{(L)} - y_i\)

Compute errors \(\{\delta^{(L-1)}, \ldots, \delta^{(2)}\}\)

Compute gradients \(\Delta^{(l)}_{ij} = \Delta^{(l)}_{ij} + a^{(l)}_j \delta^{(l+1)}_i\)

Compute avg regularized gradient \(D^{(l)}_{ij} = \begin{cases} \frac{1}{n} \Delta^{(l)}_{ij} + \lambda \Theta^{(l)}_{ij} & \text{if } j \neq 0 \\ \frac{1}{n} \Delta^{(l)}_{ij} & \text{otherwise} \end{cases}\)

Update weights via gradient step \(\Theta^{(l)}_{ij} = \Theta^{(l)}_{ij} - \alpha D^{(l)}_{ij}\)

Until weights converge or max #epochs is reached

Based on slide by Andrew Ng
Neural Embeddings
• Deep model trained on a GPU on 6M random pics downloaded from Yelp

What is word2vec?

- word2vec is **not** a single algorithm
- It is a **software package** for representing words as vectors, containing:
  - Two distinct models
    - Skip-Gram
    - CBOW
  - Various training methods
    - Negative Sampling
    - Hierarchical Softmax
  - A rich preprocessing pipeline
    - Dynamic Context Windows
    - Subsampling
    - Deleting Rare Words
- C.f. GloVe

Source: Levy et al. (2015) “Improving Distributional Similarity with Lessons Learned from Word Embeddings.”
Skip-gram

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016
Continuous Bag of Words (CBOW)

**Input layer**
1-hot input vectors for each context word

\[ \begin{align*}
W & \in |V| \times d \\
\sum & \text{sum of embeddings for context words}
\end{align*} \]

**Projection layer**

**Output layer**
probability of \( w_t \)

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016
CBOW with 1-Word Context

A CBOW model attempts to predict the one-hot vector for the context words based on the word vector for the target word. The model uses a context of one word for each input word. The input layer contains one-hot vectors for the context words. The hidden layer includes the word vectors and hidden unit activations. The output layer generates the one-hot vector for the target word. These matrices have word vectors!

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016
Word2vec Training

- Start with small, random vectors for words
- Iteratively go through millions of words in contexts
  - Work out prediction, work out error
  - Backpropagate error to update word vectors
  - Repeat
- Result is dense vectors for all words

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

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016
Word2vec Linear Relationships

- These representations are very good at encoding similarity and dimensions of similarity!
- Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

Syntactically
- \( x_{\text{apple}} - x_{\text{apples}} \approx x_{\text{car}} - x_{\text{cars}} \approx x_{\text{family}} - x_{\text{families}} \)
- Similarly for verb and adjective morphological forms

Semantically (Semeval 2012 task 2)
- \( x_{\text{shirt}} - x_{\text{clothing}} \approx x_{\text{chair}} - x_{\text{furniture}} \)
- \( x_{\text{king}} - x_{\text{man}} \approx x_{\text{queen}} - x_{\text{woman}} \)

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016
Word2vec Linear Relationships

Source: Mikolov et al. (2013) “Distributed Representations of Words and Phrases and their Compositionality.” *NIPS.*
Doc2vec

Distributed Memory

Classifier

Average/Concatenate

Paragraph Matrix

Paragraph id

D

W

W

W

Source: Le, Mikolov (2014) “Distributed Representations of Sentences and Documents.” ICML.
Doc2vec

Distributed Bag of Words

Classifier

Paragraph Matrix

Paragraph id

Source: Le, Mikolov (2014) “Distributed Representations of Sentences and Documents.” ICML.
Doc2vec > Bag of Words

- Semantic similarity
- Takes into account word order within context
  - (Distributed Memory only)
  - Similar to bag-of-n-grams, but dense

Source: Le, Mikolov (2014) “Distributed Representations of Sentences and Documents.” ICML.
SVD Reduction

• Levy and Goldberg demonstrated skip-gram with negative sampling reduces to SVD on “SPPMI” matrix
  • SPMI is PMI shifted by constant log(k)
    • $k := \text{number of negative samples}$
  • SPPMI is positive SPMI
    • $\max(0, \text{SPMI})$
  • Demonstrated near equivalence among many embedding methods
    • (Given the right hyperparameters)
  • Also refuted a bunch of the GloVe claims...

Source: Levy et al. (2015) “Improving Distributional Similarity with Lessons Learned from Word Embeddings.”
Questions?