Part-of-Speech Tagging
+
Neural Networks 3: Word Embeddings

CS 287
One-layer multi-layer perceptron architecture,

\[ NN_{MLP_1}(x) = g(xW^1 + b^1)W^2 + b^2 \]

- \( xW + b; \) perceptron
- \( x \) is the dense representation in \( \mathbb{R}^{1 \times d_{in}} \)
- \( W^1 \in \mathbb{R}^{d_{in} \times d_{hid}}, b^1 \in \mathbb{R}^{1 \times d_{hid}}; \) first affine transformation
- \( W^2 \in \mathbb{R}^{d_{hid} \times d_{out}}, b^2 \in \mathbb{R}^{1 \times d_{out}}; \) second affine transformation
- \( g: \mathbb{R}^{d_{hid} \times d_{hid}} \) is an activation non-linearity (often pointwise)
- \( g(xW^1 + b^1) \) is the hidden layer
Review: Non-Linearities Tanh

Hyperbolic Tangent:

$$
tanh(t) = \frac{\exp(t) - \exp(-t)}{\exp(t) + \exp(-t)}
$$

- Intuition: Similar to sigmoid, but range between 0 and -1.
Review: Backpropagation

\[ f_i(\ldots f_1(x^0)) \]

\[ f_{i+1}(f_i(\ldots f_1(x^0))) \]

\[ \frac{\partial L}{\partial \theta_{i+1}} \]

\[ \frac{\partial L}{\partial f_i(\ldots f_1(x^0))} \]

\[ \frac{\partial L}{\partial f_{i+1}(\ldots f_1(x^0))} \]
Quiz

One common class of operations in neural network models is known as *pooling*. Informally a pooling layer consists of an aggregation unit, typically unparameterized, that reduces the input to a smaller size.

Consider three pooling functions of the form $f : \mathbb{R}^n \mapsto \mathbb{R}$,

1. $f(x) = \max_i x_i$
2. $f(x) = \min_i x_i$
3. $f(x) = \sum_i x_i / n$

What action do each of these functions have? What are their gradients? How would you implement backpropagation for these units?
Quiz

- **Max pooling**: $f(x) = \max_i x_i$
  - Keeps only the most activated input
  - Fprop is simple; however must store arg max ("switch")
  - Bprop gradient is zero except for switch, which gets gradoutput

- **Min pooling**: $f(x) = \min_i x_i$
  - Keeps only the least activated input
  - Fprop is simple; however must store arg min ("switch")
  - Bprop gradient is zero except for switch, which gets gradoutput

- **Avg pooling**: $f(x) = \frac{\sum_i x_i}{n}$
  - Keeps the average activation input
  - Fprop is simply mean.
  - Gradoutput is averaged and passed to all inputs.
Quiz

- **Max pooling:** $f(x) = \max_i x_i$
  - Keeps only the most activated input
  - Fprop is simple; however must store arg max ("switch")
  - Bprop gradient is zero except for switch, which gets gradoutput

- **Min pooling:** $f(x) = \min_i x_i$
  - Keeps only the least activated input
  - Fprop is simple; however must store arg min ("switch")
  - Bprop gradient is zero except for switch, which gets gradoutput

- **Avg pooling:** $f(x) = \frac{\sum_i x_i}{n}$
  - Keeps the average activation input
  - Fprop is simply mean.
  - Gradoutput is averaged and passed to all inputs.
Quiz

- **Max pooling**: $f(x) = \max_i x_i$
  - Keeps only the most activated input
  - Fprop is simple; however must store arg max ("switch")
  - Bprop gradient is zero except for switch, which gets gradoutput

- **Min pooling**: $f(x) = \min_i x_i$
  - Keeps only the least activated input
  - Fprop is simple; however must store arg min ("switch")
  - Bprop gradient is zero except for switch, which gets gradoutput

- **Avg pooling**: $f(x) = \sum_i x_i / n$
  - Keeps the average activation input
  - Fprop is simply mean.
  - Gradoutput is averaged and passed to all inputs.
Contents

Embedding Motivation

C&W Embeddings

word2vec

Evaluating Embeddings
1. Use dense representations instead of sparse
2. Use windowed area instead of sequence models
3. Use neural networks to model windowed interactions
What about rare words?
Word Embeddings

Embedding layer,

\[ x^0 W^0 \]

- \( x^0 \in \mathbb{R}^{1 \times d_0} \) one-hot word.
- \( W^0 \in \mathbb{R}^{d_0 \times d_{\text{in}}} \), \( d_0 = |\mathcal{V}| \)

Notes:
- \( d_0 >> d_{\text{in}} \), e.g. \( d_0 = 10000, d_{\text{in}} = 50 \)
Pretraining Representations

- We would strongly shared representations of words.

- However, PTB only 1M labeled words, relatively small.


- (Close connection to Bengio et al (2003), next topic)
Semi-Supervised Training

Idea: Train representations separately on more data

1. Pretrain word embeddings $W^0$ first.
2. Substitute them in as first NN layer
3. Fine-tune embeddings for final task
   - Modify the first layer based on supervised gradients
   - Optional, some work skips this step
Large Corpora

To learn rare word embeddings, need many more tokens,

- **C&W**
  - English Wikipedia (631 million words tokens)
  - Reuters Corpus (221 million word tokens)
  - Total vocabulary size: 130,000 word types

- **word2vec**
  - Google News (6 billion word tokens)
  - Total vocabulary size: \(\approx 1M\) word types

But this data has no labels...
Contents

Embedding Motivation

C&W Embeddings

word2vec

Evaluating Embeddings
C&W Embeddings

- Assumption: Text in Wikipedia is *coherent* (in some sense).
- Most randomly corrupted text is *incoherent*.
- Embeddings should distinguish coherence.
- Common idea in unsupervised learning (distributional hypothesis).
C&W Setup

Let $\mathcal{V}$ be the vocabulary of English and let $s$ score any window of size $d_{\text{win}} = 5$, if we see the phrase

\[
[ \text{the dog walks to the} ]
\]

It should score higher by $s$ than

\[
[ \text{the dog house to the} ]
\]
\[
[ \text{the dog cats to the} ]
\]
\[
[ \text{the dog skips to the} ]
\]
\[
\ldots
\]
C&W Setup

Can estimate score $s$ as a windowed neural network.

$$s(w_1, \ldots, w_{d_{\text{win}}}) = \text{hardtanh}(xW^1 + b^1)W^2 + b$$

with

$$x = [v(w_1) \ v(w_2) \ \ldots \ v(w_{d_{\text{win}}})]$$

- $d_{\text{in}} = d_{\text{win}} \times 50$, $d_{\text{hid}} = 100$, $d_{\text{win}} = 11$, $d_{\text{out}} = 1$!

Example: Function $s$

$$x = [v(w_3) \ v(w_4) \ v(w_5) \ v(w_6) \ v(w_7)]$$
Training?

- Different setup than previous experiments.
- No direct supervision $y$
- Train to rank good examples better.
Ranking Loss

Given only example \{x_1, \ldots, x_n\} and for each example have set \(D(x)\) of alternatives.

\[
\mathcal{L}(\theta) = \sum_i \sum_{x' \in D(x)} L_{\text{ranking}}(s(x_i; \theta), s(x'; \theta))
\]

\[
L_{\text{ranking}}(y, \hat{y}) = \max\{0, 1 - (y - \hat{y})\}
\]

**Example:** C&W ranking

\[
x = \text{[the dog walks to the]}
\]

\[
D(x) = \{ \text{[the dog skips to the]}, \text{[the dog in to the]}, \ldots \}
\]

- (Torch `nn.RankingCriterion`)
- Note: slightly different setup.
C&W Embeddings in Practice

- Vocabulary size $|\mathcal{D}(x)| > 100,000$
- Training time for 4 weeks
- (Collobert is main an author of Torch)
Sampling (Sketch of WSABIE (Weston, 2011))

**Observation:** in many contexts

\[ L_{\text{ranking}}(y, \hat{y}) = \max\{0, 1 - (y - \hat{y})\} = 0 \]

Particularly true later in training.

For difficult contexts, may be easy to find

\[ L_{\text{ranking}}(y, \hat{y}) = \max\{0, 1 - (y - \hat{y})\} \neq 0 \]

We can therefore sample from \( D(x) \) to find an update.
C&W Results

<table>
<thead>
<tr>
<th>Approach</th>
<th>POS (PWA)</th>
<th>CHUNK (F1)</th>
<th>NER (F1)</th>
<th>SRL (F1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark Systems</td>
<td>97.24</td>
<td>94.29</td>
<td>89.31</td>
<td>77.92</td>
</tr>
<tr>
<td>NN+WLL</td>
<td>96.31</td>
<td>89.13</td>
<td>79.53</td>
<td>55.40</td>
</tr>
<tr>
<td>NN+SLL</td>
<td>96.37</td>
<td>90.33</td>
<td>81.47</td>
<td>70.99</td>
</tr>
<tr>
<td>NN+WLL+LM1</td>
<td>97.05</td>
<td>91.91</td>
<td>85.68</td>
<td>58.18</td>
</tr>
<tr>
<td>NN+SLL+LM1</td>
<td>97.10</td>
<td>93.65</td>
<td>87.58</td>
<td>73.84</td>
</tr>
<tr>
<td>NN+WLL+LM2</td>
<td>97.14</td>
<td>92.04</td>
<td>86.96</td>
<td>58.34</td>
</tr>
<tr>
<td>NN+SLL+LM2</td>
<td>97.20</td>
<td>93.63</td>
<td>88.67</td>
<td>74.15</td>
</tr>
</tbody>
</table>

1. Use dense representations instead of sparse
2. Use windowed area instead of sequence models
3. Use neural networks to model windowed interactions
4. Use semi-supervised learning to pretrain representations.
Contents

Embedding Motivation

C&W Embeddings

word2vec

Evaluating Embeddings
word2vec

- Contributions:
  - Scale embedding process to massive sizes
  - Experiments with several architectures
  - Empirical evaluations of embeddings
  - Influential release of software/data.

- Differences with C&W
  - Instead of MLP uses (bi)linear model (linear in paper)
  - Instead of ranking model, directly predict word (cross-entropy)
  - Various other extensions.

- Two different models
  1. Continuous Bag-of-Words (CBOW)
  2. Continuous Skip-gram
word2vec

- Contributions:
  - Scale embedding process to massive sizes
  - Experiments with several architectures
  - Empirical evaluations of embeddings
  - Influential release of software/data.

- Differences with C&W
  - Instead of MLP uses (bi)linear model (linear in paper)
  - Instead of ranking model, directly predict word (cross-entropy)
  - Various other extensions.

- Two different models
  1. Continuous Bag-of-Words (CBOW)
  2. Continuous Skip-gram
word2vec (Bilinear Model)

Back to pure bilinear model, but with much bigger output space

\[
\hat{y} = \text{softmax}\left(\frac{\sum_i x_i^0 W^0}{d_{\text{win}} - 1} W^1\right)
\]

- \(x_i^0 \in \mathbb{R}^{1 \times d_0}\) input words one-hot vectors.
- \(W^0 \in \mathbb{R}^{d_0 \times d_{\text{in}}}; d_0 = |\mathcal{V}|, \) word embeddings
- \(W^1 \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}; d_{\text{out}} = |\mathcal{V}| \) output embeddings

Notes:
- Bilinear parameter interaction.
- \(d_0 \gg d_{\text{in}}, \) e.g. \(50 \leq d_{\text{in}} \leq 1000, \ 10000 \leq |\mathcal{V}| \leq 1M \) or more
word2vec (Mikolov, 2013)
Continuous Bag-of-Words (CBOW)

\[ \hat{y} = \text{softmax}\left( \frac{\sum_i x_i^0 W^0}{d_{\text{win}} - 1} \right) W^1 \]

- Attempt to predict the middle word

[ the dog walks to the ]

Example: CBOW

\[ \mathbf{x} = \frac{\nu(w_3) + \nu(w_4) + \nu(w_6) + \nu(w_7)}{d_{\text{win}} - 1} \]

\[ \mathbf{y} = \delta(w_5) \]

\( W^1 \) is no longer partitioned by row (order is lost)
Continuous Skip-gram

\[ \hat{y} = \text{softmax}(x^0W^0)W^1 \]

- Also a bilinear model
- Attempt to predict each context-word from middle

\[
\begin{bmatrix}
\text{the} & \_ & \_ & \_ & \text{dog} & \_ & \_ & \_ & \_ \\
\end{bmatrix}
\]

Example: Skip-gram

\[ x = \nu(w_5) \]
\[ y = \delta(w_3) \]

Done for each word in window.
Additional aspects

- The window $d_{\text{win}}$ is sampled for each SGD step.
- SGD is done less for frequent words.
- We have slightly simplified the training objective.
Softmax Issues

Use a softmax to force a distribution,

$$\text{softmax}(z) = \frac{\exp(z)}{\sum_{c \in C} \exp(z_c)}$$

$$\log \text{softmax}(z) = z - \log \sum_{c \in C} \exp(z_c)$$

- **Issue:** class $C$ is huge.
- For C&W, 100,000, for word2vec 1,000,000 types
- Note largest dataset is 6 billion words
Two-Layer Softmax

First, clustering words into hard classes (for instance Brown clusters)

Groups words into classes based on word-context.
Two-Layer Softmax

Assume that we first generate a class $C$ and then a word,

$$p(Y|X) \approx P(Y|C, X; \theta)P(C|X; \theta)$$

Estimate distributions with a shared embedding layer,

$P(C|X; \theta)$

$$\hat{y}_1 = \text{softmax}((x^0 W^0) W^1 + b)$$

$P(Y|C = \text{class}, X; \theta)$

$$\hat{y}_2 = \text{softmax}((x^0 W^0) W^{\text{class}} + b))$$
Softmax as Tree

\[
\hat{y}^{(1)} = \text{softmax}((x^0 W^0) W^1 + b)
\]

\[
\hat{y}^{(2)} = \text{softmax}((x^0 W^0) W^{\text{class}} + b))
\]

\[
L_{2SM}(y^{(1)}, y^{(2)}, \hat{y}^{(1)}, \hat{y}^{(2)}) = -\log p(y|x, \text{class}(y)) - \log p(\text{class}(y)|x)
\]

\[
= -\log \hat{y}_{c_1}^{(1)} - \log \hat{y}_{c_2}^{(2)}
\]
Speed

- Computing loss only requires walking path.
- Two-layer a balanced tree.
- Computing loss requires $O(\sqrt{|\mathcal{V}|})$
- (Note: computing full distribution requires $O(|\mathcal{V}|)$)
Hierarchical Softmax (HSM)

- Build multiple layer tree

\[ L_{HSM}(y^{(1)}, \ldots, y^{(C)}, \hat{y}^{(1)}, \ldots, \hat{y}^{(C)}) = - \sum_i \log \hat{y}_{ci}^{(i)} \]

- Balanced tree only requires \( O(\log_2 |\mathcal{V}|) \)

- Experiments on website (Mnih and Hinton, 2008)
HSM with Huffman Encoding

- Requires $O(\log_2 \text{perp}(\text{unigram}))$
- Reduces time to only 1 day for 1.6 million tokens
Contents

Embedding Motivation

C&W Embeddings

word2vec

Evaluating Embeddings
How good are embeddings?

- Qualitative Analysis/Visualization
- Analogy task
- Extrinsic Metrics
Metrics

Dot-product

\[ \mathbf{x}_{\text{cat}} \mathbf{x}_{\text{dog}}^{\top} \]

Cosine Similarity

\[ \frac{\mathbf{x}_{\text{cat}} \mathbf{x}_{\text{dog}}^{\top}}{\| \mathbf{x}_{\text{cat}} \| \| \mathbf{x}_{\text{dog}} \|} \]
<table>
<thead>
<tr>
<th>Word</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>0.921800527377</td>
</tr>
<tr>
<td>dogs</td>
<td>0.851315870426</td>
</tr>
<tr>
<td>horse</td>
<td>0.790758298322</td>
</tr>
<tr>
<td>puppy</td>
<td>0.775492121034</td>
</tr>
<tr>
<td>pet</td>
<td>0.772470734611</td>
</tr>
<tr>
<td>rabbit</td>
<td>0.772081457265</td>
</tr>
<tr>
<td>pig</td>
<td>0.749006160038</td>
</tr>
<tr>
<td>snake</td>
<td>0.73991884888</td>
</tr>
</tbody>
</table>

- **Intuition:** trained to match words that act the same.
Empirical Measures: Analogy task

Analogy questions:

A : B :: C : __

- 5 types of semantic questions, 9 types of syntactic
## Embedding Tasks

<table>
<thead>
<tr>
<th>Type of relationship</th>
<th>Word Pair 1</th>
<th>Word Pair 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common capital city</td>
<td>Athens</td>
<td>Oslo</td>
</tr>
<tr>
<td>All capital cities</td>
<td>Astana</td>
<td>Harare</td>
</tr>
<tr>
<td>Currency</td>
<td>Angola</td>
<td>Iran</td>
</tr>
<tr>
<td>City-in-state</td>
<td>Chicago</td>
<td>Stockton</td>
</tr>
<tr>
<td>Man-Woman</td>
<td>brother</td>
<td>grandson</td>
</tr>
<tr>
<td>Adjective to adverb</td>
<td>apparently</td>
<td>rapidly</td>
</tr>
<tr>
<td>Opposite</td>
<td>possibly</td>
<td>unethically</td>
</tr>
<tr>
<td>Comparative</td>
<td>great</td>
<td>ethical</td>
</tr>
<tr>
<td>Superlative</td>
<td>easy</td>
<td>tough</td>
</tr>
<tr>
<td>Present Participle</td>
<td>think</td>
<td>lucky</td>
</tr>
<tr>
<td>Nationality adjective</td>
<td>Switzerland</td>
<td>read</td>
</tr>
<tr>
<td>Past tense</td>
<td>walking</td>
<td>reading</td>
</tr>
<tr>
<td>Plural nouns</td>
<td>mouse</td>
<td>Cambodia</td>
</tr>
<tr>
<td>Plural verbs</td>
<td>work</td>
<td>swam</td>
</tr>
</tbody>
</table>

|                     |                   |                 |
|                     |                   |                 |
|                     |                   |                 |
Analogy Prediction

A:B::C: __

\[ x' = x_B - x_A + x_C \]

Project to the closest word,

\[ \arg \max_{D \in \mathcal{V}} \frac{\mathbf{x}_D \mathbf{x}'^\top}{\|\mathbf{x}_D\| \|\mathbf{x}'\|} \]

▶ Code example
Extrinsic Tasks

- Text classification
- Part-of-speech tagging
- Many, many others over last couple years
Conclusion

- Word Embeddings
- Scaling issues and tricks
- Next Class: Language Modeling