CS224d: Deep NLP

Lecture 13:
Convolutional Neural Networks (for NLP)

Richard Socher
richard@metamind.io
Midterm

- Generally good performance

\[
\frac{\partial \| \mathbf{x} \|_2^2}{\partial \mathbf{x}} = \frac{\partial \| \mathbf{x} \|_2^2}{\partial \mathbf{x}} = \frac{\partial \| \mathbf{x}^T \mathbf{x} \|_2}{\partial \mathbf{x}} = 2\mathbf{x}
\]
Overview of today

• From RNNs to CNNs

• CNN Variant 1: Simple single layer
• Application: Sentence classification
• More details and tricks
• Evaluation
• Comparison between sentence models: BoV, RNNs, CNNs

• CNN Variant 2: Complex multi layer
From RNNs to CNNs

The country of my birth

0.4 0.3
2.1 3.3
7 7
4 4.5
2.3 3.6

0.4 0.3
2.1 3.3
7 7
4 4.5
2.3 3.6
From RNNs to CNNs

- Recursive neural nets require a parser to get tree structure

- Recurrent neural nets cannot capture phrases without prefix context and often capture too much of last words in final vector
From RNNs to CNNs

• RNN: Get compositional vectors for grammatical phrases only

• CNN: What if we compute vectors for every possible phrase?
• Example: “the country of my birth” computes vectors for:
  • the country, country of, of my, my birth, the country of, country of my, of my birth, the country of my, country of my birth

• Regardless of whether it is grammatical
• Wouldn’t need parser
• Not very linguistically or cognitively plausible
What is convolution anyway?

- 1d discrete convolution generally:
  \[(f * g)[n] = \sum_{m=-M}^{M} f[n - m]g[m].\]

- Convolution is great to extract features from images

- 2d example →
- Yellow shows filter weights
- Green shows input

![Convolution Example](convolution_example.png)

Image

Convolved Feature

Stanford UFLDL wiki

Lecture 1, Slide 7

Richard Socher

5/12/15
From RNNs to CNNs

• First layer: compute all bigram vectors

• Same computation as in RNN but for every pair

\[ p = \tanh \left( W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b \right) \]

• This can be interpreted as a convolution over the word vectors
From RNNs to CNNs

- Now multiple options to compute higher layers.
- First option (simple to understand but not necessarily best)
- Just repeat with different weights:

\[ p = \tanh \left( W^{(2)} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b \right) \]
From RNNs to CNNs

- First option (simple to understand but not necessarily best)
From RNNs to CNNs

- First option (simple to understand but not necessarily best)
Single Layer CNN

- A simple variant using one convolutional layer and **pooling**
- Based on Collobert and Weston (2011) and Kim (2014) “Convolutional Neural Networks for Sentence Classification”
- Word vectors: \( x_i \in \mathbb{R}^k \)
- Sentence: \( x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n \) (vectors concatenated)
- Concatenation of words in range: \( x_{i:i+j} \)
- Convolutional filter: \( w \in \mathbb{R}^{hk} \) (goes over window of h words)
- Could be 2 (as before) higher, e.g. 3:

```
     1.1
    /   \
 0.4    2.1    7
 0.3     3.3    7
```

the     country of my birth
Single layer CNN

- Convolutional filter: \( \mathbf{w} \in \mathbb{R}^{hk} \) (goes over window of h words)
- Note, filter is vector!
- Window size h could be 2 (as before) or higher, e.g. 3:
- To compute feature for CNN layer:

\[
c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)
\]

\[
\begin{pmatrix}
1.1 \\
0.4 \\
0.3 \\
2.1 \\
3.3 \\
7 \\
4 \\
4.5 \\
2.3 \\
3.6
\end{pmatrix}
\]

the   country   of   my   birth
Single layer CNN

- Filter $w$ is applied to all possible windows (concatenated vectors)

- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$

- All possible windows of length $h$: $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$

- Result is a feature map: $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$

---

The diagram illustrates a sentence with the words "the", "country", "of", "my", and "birth", each represented as a feature vector with values corresponding to each word. The diagram shows how the filter is applied to all possible windows of length $h$ along the sentence.
Single layer CNN

- Filter $w$ is applied to all possible windows (concatenated vectors)

- Sentence: $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$

- All possible windows of length $h$: $\{x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}\}$

- Result is a feature map: $c = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$

![Diagram of feature map and word vectors]
Single layer CNN: Pooling layer

- New building block: Pooling
- In particular: max-over-time pooling layer
- Idea: capture most important activation (maximum over time)

- From feature map \( c = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1} \)

- Pooled single number: \( \hat{c} = \max\{c\} \)

- But we want more features!
Solution: Multiple filters

- Use multiple filter weights $w$
- Useful to have different window sizes $h$
- Because of max pooling $\hat{c} = \max\{c\}$, length of $c$ irrelevant
  \[ c = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1} \]
- So we can have some filters that look at unigrams, bigrams, trigrams, 4-grams, etc.
Multi-channel idea

- Initialize with pre-trained word vectors (word2vec or Glove)
- Start with two copies
- Backprop into only one set, keep other “static”
- Both channels are added to $c_i$ before max-pooling
Classification after one CNN layer

- First one convolution, followed by one max-pooling

- To obtain final feature vector: \( z = [\hat{c}_1, \ldots, \hat{c}_m] \) (assuming \( m \) filters \( w \))

- Simple final softmax layer \( y = \text{softmax}(W^{(S)} z + b) \)
In one of the model variants, we experiment with having two 'channels' of word vectors—one for the static channels and one for the non-static channels. Whenever weight vectors by rescaling unmasked units. At test time, the learned weight vectors are scaled by \( \|w\|_2 \). We employ language from computer vision where a color image has red, green, and blue channels. We additionally constrain the unmasked units. At test time, the learned weight vectors are scaled by \( \|w\|_2 \).

Gradients are backpropagated only through the masked units. At test time, the learned weight vectors are scaled by \( \|w\|_2 \). We then apply a max-over-time pooling operation (Collobert et al., 2011) to obtain multiple features. These features form the penultimate layer and are passed to a fully connected layer with dropout and softmax output.

In the multichannel architecture, illustrated in figure 1, we use multiple filters (with varying window sizes) to deal with variable sentence lengths. Here, \( c_i \) is the important feature—one with the highest value—for each possible window of words in the sentence.

The idea is to capture the most important feature and then pool across all filters. The feature map is the concatenation operator. In general, let \( h_i = \{ x_1, \ldots, x_{i+1} \} \) be the \( i \)th filter. The model uses multiple filters (with varying window sizes) to capture the most important feature—one with the highest value—for each possible window of words in the sentence. For example, a feature \( x \) is generated from a window of words to produce a new feature. For \( x_i \):\( x_i+1 \), \( x_i+2 \),..., \( x_k \) representation of sentence with static and non-static channels, we apply a convolutional layer with multiple filter widths and feature maps. Max-over-time pooling is applied to each possible window of words in the sentence. We then apply a fully connected layer with dropout and softmax output.

The model variant that we experiment with has two 'channels' of word vectors—one for the static channels and one for the non-static channels. Whenever weight vectors by rescaling unmasked units. At test time, the learned weight vectors are scaled by \( \|w\|_2 \). We employ language from computer vision where a color image has red, green, and blue channels. We additionally constrain the unmasked units. At test time, the learned weight vectors are scaled by \( \|w\|_2 \). We then apply a max-over-time pooling operation (Collobert et al., 2011) to obtain multiple features. These features form the penultimate layer and are passed to a fully connected layer with dropout and softmax output.

In the multichannel architecture, illustrated in figure 1, we use multiple filters (with varying window sizes) to deal with variable sentence lengths. Here, \( c_i \) is the important feature—one with the highest value—for each possible window of words in the sentence. The idea is to capture the most important feature and then pool across all filters. The feature map is the concatenation operator. In general, let \( h_i = \{ x_1, \ldots, x_{i+1} \} \) be the \( i \)th filter. The model uses multiple filters (with varying window sizes) to capture the most important feature—one with the highest value—for each possible window of words in the sentence. For example, a feature \( x \) is generated from a window of words to produce a new feature. For \( x_i \):\( x_i+1 \), \( x_i+2 \),..., \( x_k \) representation of sentence with static and non-static channels, we apply a convolutional layer with multiple filter widths and feature maps. Max-over-time pooling is applied to each possible window of words in the sentence. We then apply a fully connected layer with dropout and softmax output.
Tricks to make it work better: Dropout

• Idea: randomly mask/dropout/set to 0 some of the feature weights $z$

• Create masking vector $r$ of Bernoulli random variables with probability $p$ (a hyperparameter) of being 1

• Delete features during training:

$$y = \text{softmax} \left( W^{(S)} (r \circ z) + b \right)$$

• Reasoning: Prevents co-adaptation (overfitting to seeing specific feature constellations)
Tricks to make it work better: Dropout

\[ y = \text{softmax} \left( W^{(S)} (r \circ z) + b \right) \]

- At training time, gradients are backpropagated only through those elements of z vector for which \( r_i = 1 \)
- At test time, there is no dropout, so feature vectors z are larger.
- Hence, we scale final vector by Bernoulli probability \( p \)
  \[ \hat{W}^{(S)} = pW^{(S)} \]
- Kim (2014) reports 2 – 4% improved accuracy and ability to use very large networks without overfitting
Another regularization trick

- Somewhat less common

- Constrain $l_2$ norms of weight vectors of each class (row in softmax weight $W^{(S)}$) to fixed number $s$ (also a hyperparameter)

- If $\|W_{c.}^{(S)}\| > s$, then rescale it so that: $\|W_{c.}^{(S)}\| = s$
All hyperparameters in Kim (2014)

- Find hyperparameters based on dev set
- Nonlinearity: reLu
- Window filter sizes $h = 3, 4, 5$
- Each filter size has 100 feature maps
- Dropout $p = 0.5$
- L2 constraint $s$ for rows of softmax $s = 3$
- Mini batch size for SGD training: 50
- Word vectors: pre-trained with word2vec, $k = 300$

- During training, keep checking performance on dev set and pick highest accuracy weights for final evaluation
### Experiments

<table>
<thead>
<tr>
<th>Model</th>
<th>MR</th>
<th>SST-1</th>
<th>SST-2</th>
<th>Subj</th>
<th>TREC</th>
<th>CR</th>
<th>MPQA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-rand</td>
<td>76.1</td>
<td>45.0</td>
<td>82.7</td>
<td>89.6</td>
<td>91.2</td>
<td>79.8</td>
<td>83.4</td>
</tr>
<tr>
<td>CNN-static</td>
<td>81.0</td>
<td>45.5</td>
<td>86.8</td>
<td>93.0</td>
<td>92.8</td>
<td>84.7</td>
<td><strong>89.6</strong></td>
</tr>
<tr>
<td>CNN-non-static</td>
<td><strong>81.5</strong></td>
<td>48.0</td>
<td>87.2</td>
<td>93.4</td>
<td>93.6</td>
<td>84.3</td>
<td>89.5</td>
</tr>
<tr>
<td>CNN-multichannel</td>
<td>81.1</td>
<td>47.4</td>
<td><strong>88.1</strong></td>
<td>93.2</td>
<td>92.2</td>
<td><strong>85.0</strong></td>
<td>89.4</td>
</tr>
<tr>
<td>RAE (Socher et al., 2011)</td>
<td>77.7</td>
<td>43.2</td>
<td>82.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MV-RNN (Socher et al., 2012)</td>
<td>79.0</td>
<td>44.4</td>
<td>82.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNTN (Socher et al., 2013)</td>
<td></td>
<td>45.7</td>
<td>85.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCNN (Kalchbrenner et al., 2014)</td>
<td></td>
<td>48.5</td>
<td>86.8</td>
<td>93.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paragraph-Vec (Le and Mikolov, 2014)</td>
<td></td>
<td><strong>48.7</strong></td>
<td>87.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCAE (Hermann and Blunsom, 2013)</td>
<td>77.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>87.2</td>
</tr>
<tr>
<td>Sent-Parser (Dong et al., 2014)</td>
<td>79.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86.3</td>
</tr>
<tr>
<td>NBSVM (Wang and Manning, 2012)</td>
<td>79.4</td>
<td></td>
<td></td>
<td>93.2</td>
<td></td>
<td>81.8</td>
<td>86.3</td>
</tr>
<tr>
<td>MNB (Wang and Manning, 2012)</td>
<td>79.0</td>
<td></td>
<td></td>
<td></td>
<td>93.6</td>
<td>80.0</td>
<td>86.3</td>
</tr>
<tr>
<td>G-Dropout (Wang and Manning, 2013)</td>
<td>79.0</td>
<td></td>
<td></td>
<td>93.4</td>
<td></td>
<td>82.1</td>
<td>86.1</td>
</tr>
<tr>
<td>F-Dropout (Wang and Manning, 2013)</td>
<td>79.0</td>
<td></td>
<td></td>
<td></td>
<td>93.6</td>
<td>81.9</td>
<td>86.3</td>
</tr>
<tr>
<td>Tree-CRF (Nakagawa et al., 2010)</td>
<td>77.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>81.4</td>
<td>86.1</td>
</tr>
<tr>
<td>CRF-PR (Yang and Cardie, 2014)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>82.7</td>
<td></td>
</tr>
<tr>
<td>SVM$_S$ (Silva et al., 2011)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>95.0</strong></td>
<td></td>
</tr>
</tbody>
</table>
Problem with comparison?

- Dropout gives 2 – 4 % accuracy improvement
- Several baselines didn’t use dropout

- Still remarkable results and simple architecture!

- Difference to window and RNN architectures we described in previous lectures: many filters and dropout
- Both ideas can be used in RNN^2s too
- Only explored in computer vision: Socher et al (2012): "Convolutional-Recursive Deep Learning for 3D Object Classification"
- Tree-LSTMs obtain better performance on sentence datasets
Relationship between RNNs and CNNs

- CNN
- RNN
Relationship between RNNs and CNNs

- **Stride size** difference
- Tying (sharing) weights of filters inside vs across different layers
- CNN: multiple filters, additional layer type: pooling
- Balanced input independent structure vs input specific tree
CNN alternatives

- Narrow vs wide convolution

- Complex pooling schemes (over sequences) and deeper convolutional layers

- Kalchbrenner et al. (2014)
CNN application: Translation

- One of the first successful neural machine translation efforts
- Uses CNN for encoding and RNN for decoding
- Kalchbrenner and Blunsom (2013) “Recurrent Continuous Translation Models”
Model comparison

- **Bag of Vectors**: Surprisingly good baseline for simple classification problems. Especially if followed by a few layers!

- **Window Model**: Good for single word classification for problems that do not need wide context

- **CNNs**: good for classification, unclear how to incorporate phrase level annotation (can only take a single label), need zero padding for shorter phrases, hard to interpret, easy to parallelize on GPUs
Model comparison

- **Recursive Neural Networks**: most linguistically plausible, interpretable, provide most important phrases, need parse trees
- **Recurrent Neural Networks**: Most cognitively plausible (reading from left to right), not usually the highest classification performance but lots of improvements right now with gates (GRUs, LSTMs, etc).
Next week:

• Guest Lecture: Speech recognition

• Guest Lecture: Efficient implementations and GPUs