Parsing Natural Scenes and Natural Language with Recursive Neural Networks

Socher et. al
Parsing Natural Scenes and Natural Language with Recursive Neural Networks

Socher et al.
A general purpose model

• Model works with two different types of input: natural language, and images

• Exploits common recursive nature of parsing

• Works by recursively “merging” components

• Uses **Recursive** Neural Network (RNN) (not “Recurrent Neural Network”)

Overview

- Generate general purpose features based on input. This is the only “non-general” step
- Train model on annotated tree data
- Test the predictor on new data
  - Generates parse trees
- Use model output to classify data
Input representations

Images

• Images split into segments
• Features generated for segments based on texture, colour, and shape features (and lots more)
• Use auxiliary neuron for each segment:

\[ a_i = f(W^{sem}F_i + b^{sem}) \]

Sentences

• Sentence split into individual words
• Features generated for words based on co-occurrence statistics
• This is not covered by the paper :(
Structure Prediction

• Learn a function $f : X \rightarrow Y$ where $Y$ is the set of possible binary trees representing input $X$. $X$ is split into two parts:

  • (i) A set of activation vectors (outputs from earlier)

  • (ii) A symmetric adjacency matrix representing neighbourhood
Structure Prediction

<table>
<thead>
<tr>
<th>Input Instance</th>
<th>Image</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image1.png" alt="Image" /></td>
<td>The house has a window</td>
</tr>
</tbody>
</table>

| Adjacency Matrix | ![Matrix](matrix1.png) | ![Matrix](matrix2.png) |

| Set of Correct Tree Structures | ![Trees](trees1.png) | ![Trees](trees2.png) |
Max Margin Estimation

- A learning framework in which we maximise the margin between the best and the rest. More specifically, larger than some loss $\Delta$. 
Max Margin Estimation

• Loss is measured as the number of merges which result in a subtree which doesn’t appear in the training data

• Merge possible neighbours according to loss function:

\[
\Delta(x, l, \hat{y}) = \kappa \sum_{d \in N(\hat{y})} 1\{\text{subTree}(d) \notin Y(x, l)\}
\]

where $N(\hat{y})$ is the set of non-terminal nodes and $\kappa$ is a scaling parameter.
Risk Function

\[ f_\theta(x) = \arg \max_{\hat{y}\in\mathcal{T}(x)} s(\text{RNN}(\theta, x, \hat{y})) \]

- s is a scoring function (high if tree is correct with confidence). \( \theta \) are the parameters needed to calculate s

- We want a risk function which minimises expected loss on an unseen input
Risk Function

- As said, we want highest scoring correct tree to be better than the rest by a margin defined by the loss $\Delta$:

$$s(RNN(\theta, x_i, y_i)) \geq s(RNN(\theta, x_i, \hat{y})) + \Delta(x_i, l_i, \hat{y})$$

- This gives us the regularised risk function:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} r_i(\theta) + \frac{\lambda}{2} \|\theta\|^2,$$

where

$$r_i(\theta) = \max_{\hat{y} \in \mathcal{T}(x_i)} (s(RNN(\theta, x_i, \hat{y})) + \Delta(x_i, l_i, \hat{y}))$$

$$- \max_{y_i \in \mathcal{Y}(x_i, l_i)} (s(RNN(\theta, x_i, y_i)))$$
Greedy Structure Prediction

• Now we can define RNN to predict the tree structures. This takes the two inputs as described earlier; the adjacency matrix and the activation vectors. This vector is called C.

---

**Input vector C**

**Output parse tree**
Greedy Structure Prediction

• Training aims to increase scores of pairs with the same label (unless no more of such pairs are left)

• Generate scores for pairs and select pair with best score.

• Update C by removing c1;c2 and adding a new segment, with all its neighbours (merge)

• This process is repeated using the same layer until there is only one segment remaining

\[
s = W^{score}p \quad (9) \\
p = f(W[c_1; c_2] + b)
\]
Greedy Structure Prediction

• Finally have a definition for the scoring function:

\[ s(RNN(\theta, x_i, \hat{y})) = \sum_{d \in N(\hat{y})} s_d. \]
Image Classification

• Simply add softmax neuron layer to predict classes:

\[ \text{label}_p = \text{softmax}(W^{\text{label}_p}) \]
Learning

- Using subgradient descent, via backpropagation
- Use L-BFGS to minimize objective function:

\[
\frac{\partial J}{\partial \theta} = \frac{1}{n} \sum_{i} \frac{\partial s(\hat{y}_i)}{\partial \theta} - \frac{\partial s(y_i)}{\partial \theta} + \lambda \theta
\]
Results (images)

<table>
<thead>
<tr>
<th>Method and Semantic Pixel Accuracy in</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel CRF, Gould et al. (2009)</td>
<td>74.3</td>
</tr>
<tr>
<td>Log. Regr. on Superpixel Features</td>
<td>75.9</td>
</tr>
<tr>
<td>Region-based energy, Gould et al. (2009)</td>
<td>76.4</td>
</tr>
<tr>
<td>Local Labeling, TL (2010)</td>
<td>76.9</td>
</tr>
<tr>
<td>Superpixel MRF, TL (2010)</td>
<td>77.5</td>
</tr>
<tr>
<td>Simultaneous MRF, TL (2010)</td>
<td>77.5</td>
</tr>
<tr>
<td><strong>RNN (our method)</strong></td>
<td>78.1</td>
</tr>
</tbody>
</table>

- 16 seconds to parse 143 test images on 2.6GHz laptop (in matlab though)
Results (images)
Results (Text)

• F-score of language parser is 90.29% compared with Berkeley parser: 91.36%

• Could potentially be improved with larger feature vectors

• 2.6GHz laptop took 72 seconds to parse 421 sentences of length < 15

• (again in matlab though)
Conclusion

• RNNs can do cool stuff!

• Images and sentences can be treated as similar things (and so can any recursively divisible inputs!)

• Neural Network models can be repurposed fairly easily
Questions?