Parsing Natural Scenes and Natural Language with Recursive Neural Networks

Socher et. al

Parsing Natural Scenes and Natural Language with Recursive Neural Networks

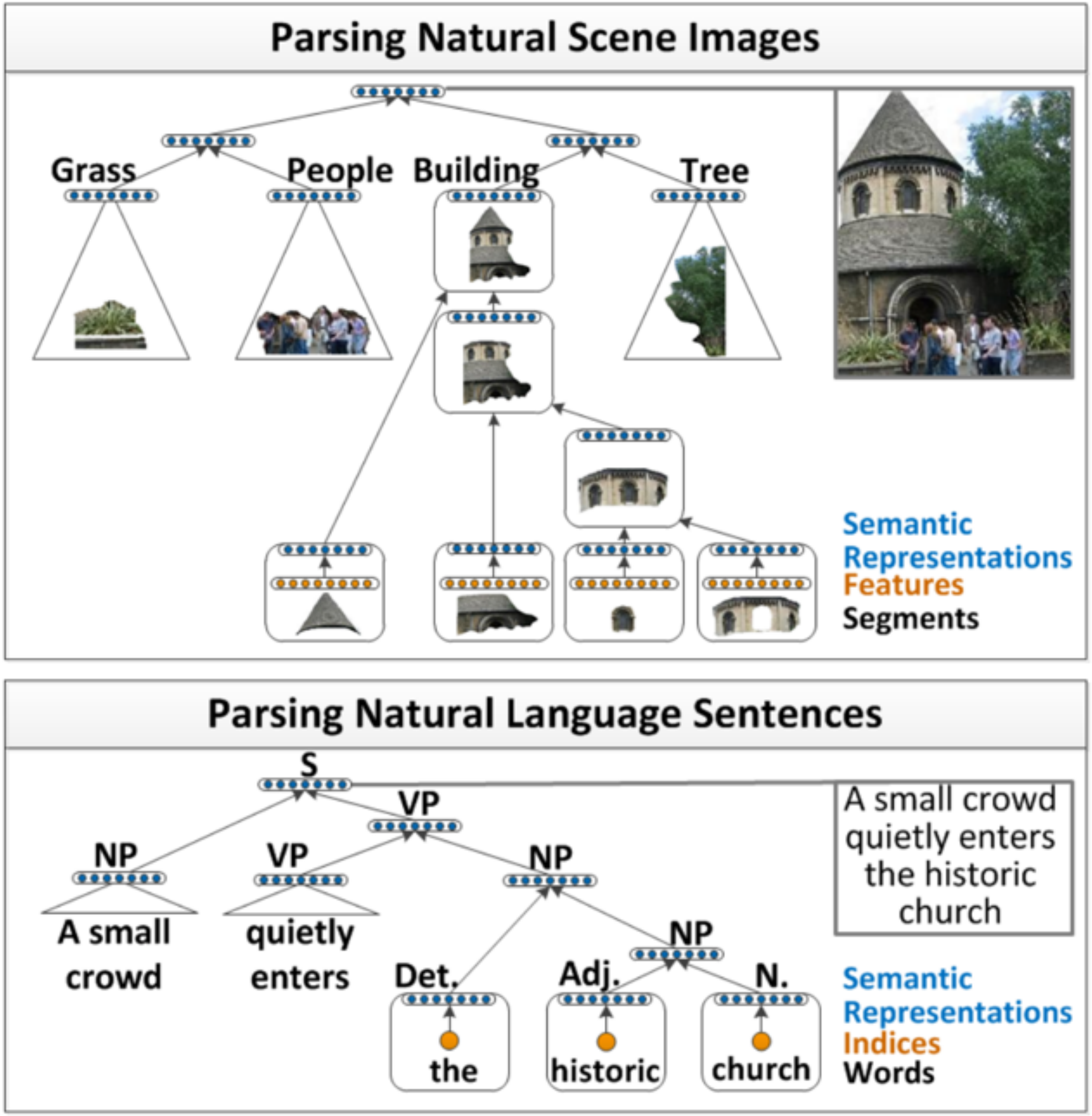
Socher et. al

✗

✓

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")


Parsing Natural Scene Images
Grass
People
Building
Tree
Semantic
Representations
Features
Segments
Parsing Natural Language Sentences
S
VP
NP
VP
NP
A small
crowd
quietly
enters
Det.
Adj.
NP
N.
the
historic
church
A small crowd
quietly enters
the historic
church
Semantic
Representations
Indices
Words

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:

$$a_i = f(W^{sem} F_i + b^{sem})$$

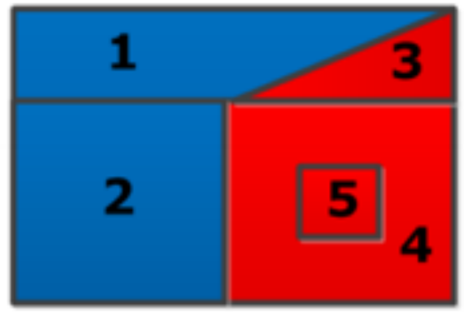
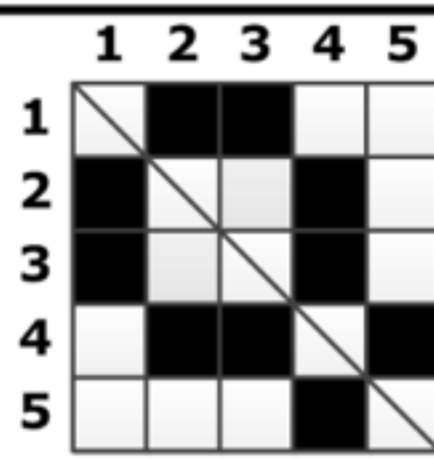
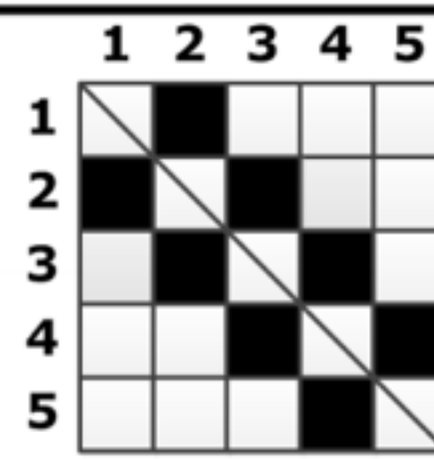
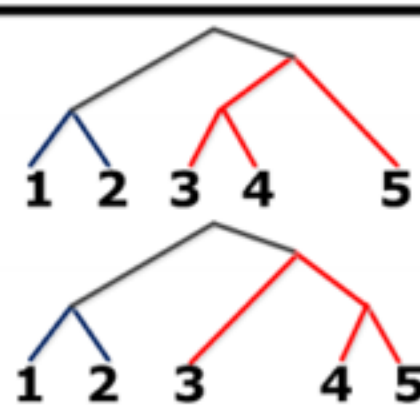
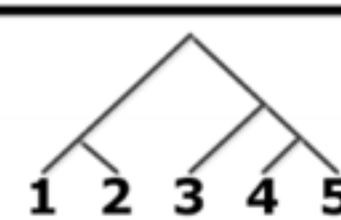
Sentences

- 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 -> 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

	Image	Text
Input Instance		The house has 1 2 3 a window 4 5
Adjacency Matrix		
Set of Correct Tree Structures		

Max Margin Estimation

- A learning framework in which we maximise the margin between the best and the rest. More specifically, larger than some loss Δ.

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:

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

where N(ˆy) is the set of non-terminal nodes and κ is a scaling parameter.

Risk Function

$$f_\theta(x) = \arg\max_{\hat{y} \in \mathcal{T}(x)} \mathrm{s}(\mathrm{RNN}(\theta, x, \hat{y}))$$

- s is a scoring function (high if tree is correct with confidence). θ 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 Δ:

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

- This gives us the regularised risk function:

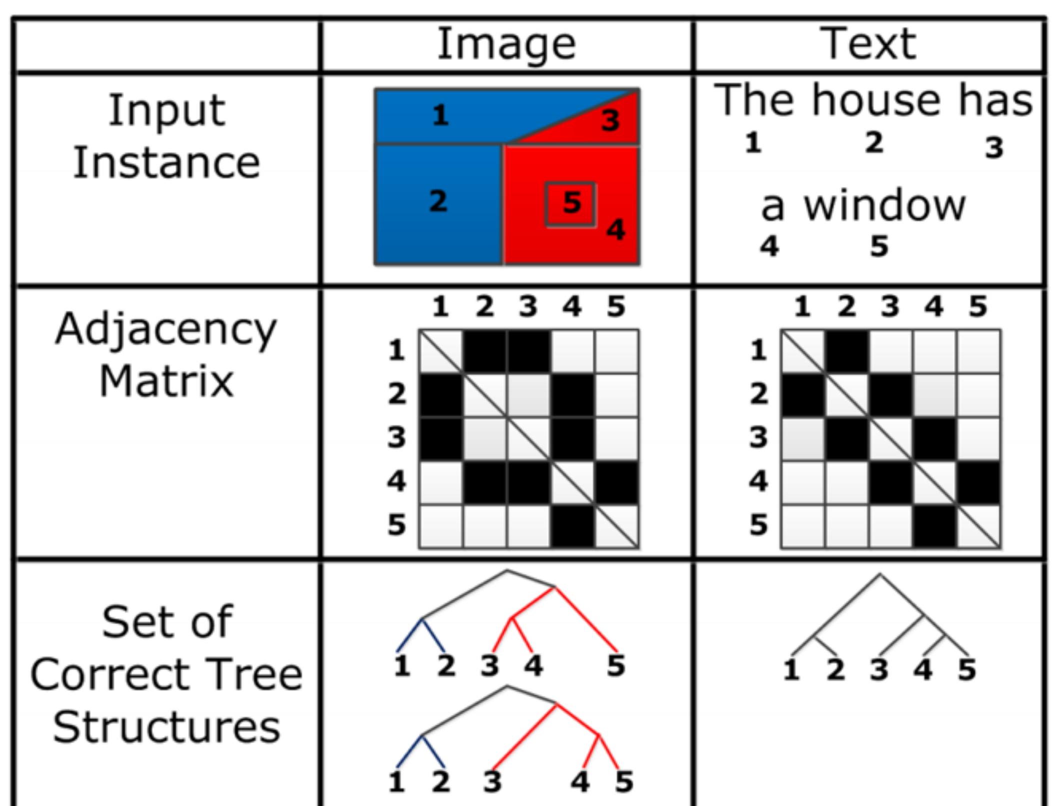
$$\begin{aligned} J(\theta) &= \frac{1}{N} \sum_{i=1}^{N} r_i(\theta) + \frac{\lambda}{2} ||\theta||^2, \quad \text{where} \qquad (5) \\ r_i(\theta) &= \max_{\hat{y} \in \mathcal{T}(x_i)} \big(s(\mathrm{RNN}(\theta, x_i, \hat{y})) + \Delta(x_i, l_i, \hat{y}) \big) \\ &- \max_{y_i \in Y(x_i, l_i)} \big(s(\mathrm{RNN}(\theta, x_i, y_i)) \big) \end{aligned}$$

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

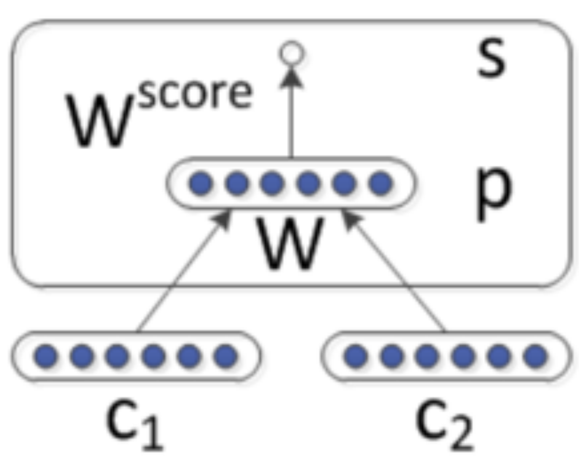
Input vector C

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



$$
\begin{aligned}
s &= W^{score} p \qquad (9) \\
p &= f(W[c_1; c_2] + b)
\end{aligned}
$$

Greedy Structure Prediction

- Finally have a definition for the scoring function:

$$s(\mathrm{RNN}(\theta, x_i, \hat{y})) = \sum\nolimits_{d \in N(\hat{y})} s_d.$$

Image Classification

- Simply add softmax neuron layer to predict classes:

$$label_p = softmax(W^{label} p)$$

WOW!

Learning

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

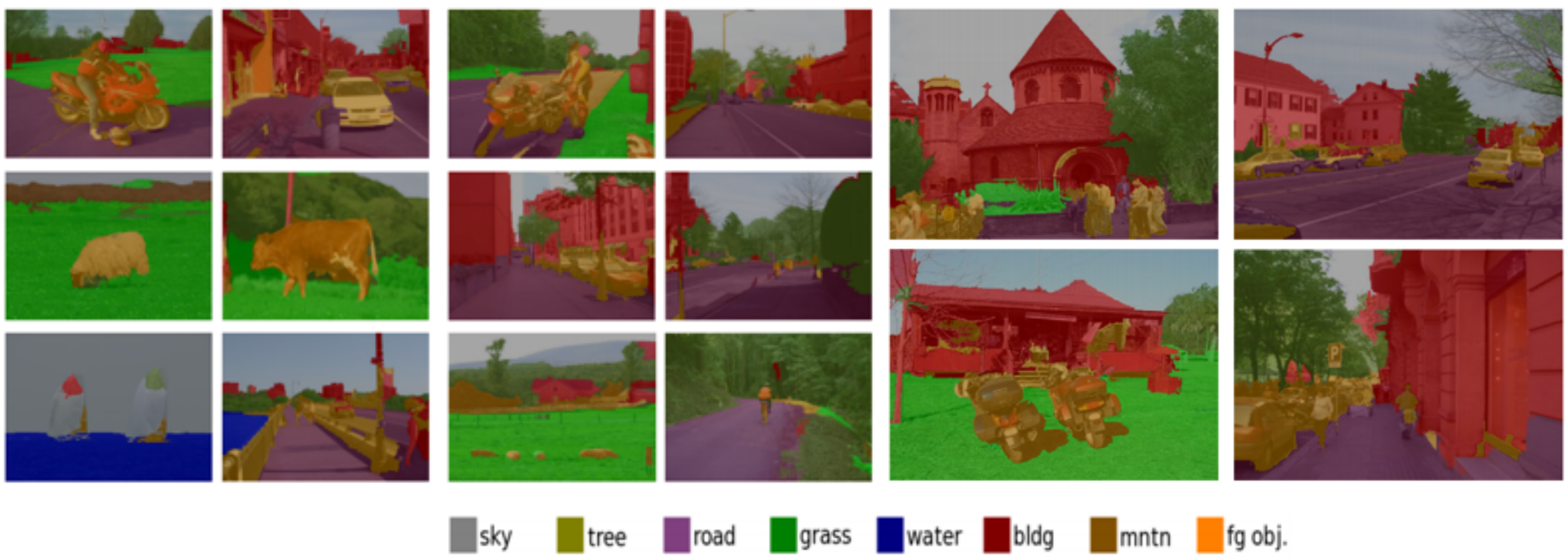
$$\frac{\partial J}{\partial \theta} = \frac{1}{n} \sum_i \frac{\partial s(\hat{y}_i)}{\partial \theta} - \frac{\partial s(y_i)}{\partial \theta} + \lambda \theta$$

Results (images)

Method and Semantic Pixel Accuracy in	%
Pixel CRF, Gould et al.(2009)	74.3
Log. Regr. on Superpixel Features	75.9
Region-based energy, Gould et al.(2009)	76.4
Local Labeling,TL(2010)	76.9
Superpixel MRF,TL(2010)	77.5
Simultaneous MRF,TL(2010)	77.5
RNN (our method)	**78.1**

- 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?