Deep Learning for Efficient Discriminative Parsing

Ronan Collobert
ronan@collobert.com

Idiap Research Institute †

†Most of this work has been achieved at NEC Laboratories America
Motivation

- We have a generic architecture (see arXiv:1103.0398) for various Natural Language Processing tasks:
  - part-of-speech
  - chunking
  - name entity recognition
  - semantic role

\[
\text{The}_{DT} \text{ cat}_{NN} \text{ sat}_{VBD} \text{ on}_{IN} \text{ the}_{DT} \text{ mat}_{NN}
\]

- Generic = “deep learning”
  trade “task-specific features” for “learning the right features”

- Leverage word representations trained on unlabeled corpus (852M words)

<table>
<thead>
<tr>
<th>france</th>
<th>jesus</th>
<th>xbox</th>
<th>reddish</th>
<th>scratched</th>
<th>megabits</th>
</tr>
</thead>
<tbody>
<tr>
<td>austria</td>
<td>god</td>
<td>amiga</td>
<td>greenish</td>
<td>nailed</td>
<td>octets</td>
</tr>
<tr>
<td>belgium</td>
<td>sati</td>
<td>playstation</td>
<td>bluish</td>
<td>smashed</td>
<td>mb/s</td>
</tr>
<tr>
<td>germany</td>
<td>christ</td>
<td>msx</td>
<td>pinkish</td>
<td>punched</td>
<td>bit/s</td>
</tr>
<tr>
<td>italy</td>
<td>satan</td>
<td>ipod</td>
<td>purplish</td>
<td>popped</td>
<td>baud</td>
</tr>
<tr>
<td>greece</td>
<td>kali</td>
<td>sega</td>
<td>brownish</td>
<td>crimped</td>
<td>carats</td>
</tr>
<tr>
<td>sweden</td>
<td>indra</td>
<td>psNUMBER</td>
<td>greyish</td>
<td>scraped</td>
<td>kbit/s</td>
</tr>
<tr>
<td>norway</td>
<td>vishnu</td>
<td>hd</td>
<td>grayish</td>
<td>screwed</td>
<td>megahertz</td>
</tr>
<tr>
<td>europe</td>
<td>ananda</td>
<td>dreamcast</td>
<td>whitish</td>
<td>sectioned</td>
<td>megapixels</td>
</tr>
<tr>
<td>hungary</td>
<td>parvati</td>
<td>geforce</td>
<td>silvery</td>
<td>slashed</td>
<td>gbit/s</td>
</tr>
<tr>
<td>switzerland</td>
<td>grace</td>
<td>capcom</td>
<td>yellowish</td>
<td>ripped</td>
<td>amperes</td>
</tr>
</tbody>
</table>
Motivation

- Can we do the same on syntactic parsing?
  (VP = Verb Phrase, NP = Noun Phrase, ...)

Not another “flat” tagging task...
- Keep the same architecture
- Make it “recurrent”
- Add tree constraints
Standard Parsing Benchmarks

But stocks kept falling

(Collins, 1999) (Charniak, 2000)

(Charniak & Johnson, 2005 & 2006)


Lexicalized Probabilistic Context-Free Grammar (PCFG), POS, head words, chart parser, deleted interpolation, ... 30 pages of details in (Bikel, 2004)

Re-ranking over the above, using lots of ad-hoc features

PCFG, dependency features

CRF or similar
**Parsing as a Tagging Task**

- **BIOES tagging:**
  
<table>
<thead>
<tr>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-S</td>
<td>I-S</td>
<td>I-S</td>
<td>E-S</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>B-VP</td>
<td>E-VP</td>
</tr>
<tr>
<td>O</td>
<td>S-NP</td>
<td>O</td>
<td>S-S#VP</td>
</tr>
</tbody>
</table>

- **Greedy Approach:**
  
  1. Predict Level 1.
  2. Predict next level according to **level history**, with the same tagger†.
  3. Update the history of levels and go to 2.

† Apply **constraints** during tagging to avoid loops and respect **tree structure** (Nodes strictly grow with tree levels)
Neural Networks

Stack several layers together (increase level of abstraction)
Parameters $W^i$ trained by gradient descent

\[
\begin{align*}
x & \quad \text{input (vector)} \\
W^1 \times \cdot & \quad \text{linear operation (embedding)} \\
\tanh(\cdot) & \quad \text{non-linearity} \\
W^2 \times \cdot & \quad \text{linear operation (embedding)} \\
& \quad \text{score per label}
\end{align*}
\]

Convolutional layer

\[
X = (X_{1}, X_{2} \cdots) \quad \text{input (matrix)}
\]

\[
W \times \begin{pmatrix} X_{1} & X_{2} \\ X_{2} & X_{3} \\ X_{3} & X_{4} \end{pmatrix} \quad \text{convolution (local embedding for each input column)}
\]

Max over time layer

\[
X = (X_{1}, X_{2} \cdots) \quad \text{input (matrix)}
\]

\[
\max_t [X]_{i,t} \quad \forall i \quad \text{remove the time “dimension”}
\]
Words into Vectors

See (Bengio et al, 2001)

a word = index in a dictionary
The cat sat on the mat = \((w_1, w_2, w_3, w_4, w_5, w_6)\)

binary code ∼ dictionary size
\[ w \leftrightarrow \begin{pmatrix} 0, \cdots, 0, & 1 & 0, \cdots, 0 \end{pmatrix}^T \]

word embedding
\[ W \sim \text{word representation size} \times \text{dictionary size} \]
\[ W \times \begin{pmatrix} 0, \cdots, 0, & 1 & 0, \cdots, 0 \end{pmatrix}^T = W \cdot w \]
lookup-table operation

Applicable to any discrete feature (words, caps, stems...)

Window Approach

How to tag “in” in the sentence “The Visigoths settled in southern Gaul”?
Window Approach (extra features)

How to tag “in” in the sentence
“The Visigoths settled in southern Gaul”?
Sentence Approach

How to tag “in” in the sentence
“The Visigoths settled in southern Gaul”?

Sentence  | Word & Dist. Representation | Local Representation | Higher-Level Representation | Tag Scorer
---|---|---|---|---
The      | $W^{1,w}$  | $W^{2}$  | max $h(W^3)$  | $W^4$
-3       | $W^{1,d}$  |               |                           | 
Visigoths| $W^{1,w}$  | $W^{2}$  |               | 
-2       | $W^{1,d}$  |               |                           | 
settled  | $W^{1,w}$  | $W^{2}$  |               | 
-1       | $W^{1,d}$  |               |                           | 
in       | $W^{1,w}$  | $W^{2}$  |               | 
0        | $W^{1,d}$  |               |                           | 
southern | $W^{1,w}$  | $W^{2}$  |               | 
1        | $W^{1,d}$  |               |                           | 
Gaul     | $W^{1,w}$  | $W^{2}$  |               | 
2        | $W^{1,d}$  |               |                           |
Sentence Scoring

- Sentence of $T$ words $[w]_1^T$
- Network parameters $\theta = \{W^1, W^2, \ldots\}$

- Tags are interdependent $\rightarrow$ structured output learning
- Network score for each word $w_t$ and tag $k$ $\rightarrow f([w]_1^T, k, t, \theta)$
- Transition score to jump from tag $k$ to tag $l$ $\rightarrow A_{kl}$

The cat sat on the mat

O
B-VP
I-VP
E-VP

Sentence score for a tag path $[i]_1^T$

$$s([w]_1^T, [i]_1^T, \theta) = \sum_{t=1}^{T} \left( A_{i_{t-1} i_t} + f([w]_1^T, [i]_t, t, \theta) \right)$$

- Inference: Viterbi algorithm
Training

- Conditional likelihood by normalizing w.r.t all possible paths:

\[
\log p([y]_1^T | [w]_1^T, \theta) = s([w]_1^T, [y]_1^T, \theta) - \log \left[ \sum_{\forall [j]_1^T} e^{s([w]_1^T, [j]_1^T, \theta)} \right]
\]

- **Log-sum** efficiently computed with recursive **Forward** algorithm
- Maximize **likelihood** over training set
- Use **stochastic gradient ascent**

\[
\theta \leftarrow \theta + \lambda \frac{\partial \log p([y]_t^T | [x]_1^T, \theta)}{\partial \theta}
\]

Fixed learning rate. **“Tricks”:**

* Divide learning rate by “fan-in”
* Initialization according to “fan-in”

- **Chain rule** ("back-propagation") through **Forward recursion and network**

- **Non-linear CRFs:** **Graph Transformer Networks** (Bottou et al, 1997)
Constrained Graph: Leverage BIOES

O
yesterday
B-NP
the
I-NP
black
E-NP
cat
O
sat
Comparison With Pure Discriminative Parsers

- Recall, Precision and F1 scores on Penn Treebank, \( \text{sentences} \leq 15 \text{ words} \)
- Standard PARSEVAL evaluation

Window approach (width=5) network

Features:
- Raw low caps words (130K) + caps feature
- History: largest previously predicted chunks
- Our own POS (if mentioned)

LM: word representations initialized with Language Model

<table>
<thead>
<tr>
<th>Model</th>
<th>( R )</th>
<th>( P )</th>
<th>( F1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collins (1999)</td>
<td>88.2</td>
<td>89.2</td>
<td>88.7</td>
</tr>
<tr>
<td>Taskar et al. (2004)</td>
<td>89.1</td>
<td>89.1</td>
<td>89.1</td>
</tr>
<tr>
<td>Turian and Melamed (2006)</td>
<td>89.3</td>
<td>89.6</td>
<td>89.4</td>
</tr>
<tr>
<td>NN</td>
<td>82.4</td>
<td>82.8</td>
<td>82.6</td>
</tr>
<tr>
<td>NN+LM</td>
<td>86.1</td>
<td>87.2</td>
<td>86.6</td>
</tr>
<tr>
<td>NN+POS</td>
<td>87.1</td>
<td>86.2</td>
<td>86.7</td>
</tr>
<tr>
<td>NN+LM+POS</td>
<td>89.2</td>
<td>89.0</td>
<td>89.1</td>
</tr>
</tbody>
</table>
Comparison With State-Of-The-Art Parsers

- Recall, Precision and F1 scores on Penn Treebank
- Standard PARSEVAL evaluation
- Window & sentence approach networks
- Comparison with:
  - Recent PCFG & CRF-based parsers

<table>
<thead>
<tr>
<th></th>
<th>(\leq 40) Words</th>
<th>(\leq 100) Words</th>
<th>Test Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(R)  (P)  (F_1)</td>
<td>(R)  (P)  (F_1)</td>
<td></td>
</tr>
<tr>
<td>Magerman (1995)</td>
<td>84.6  84.9  84.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collins (1999)</td>
<td>88.5  88.7  88.6</td>
<td>88.1  88.3  88.2</td>
<td>2640</td>
</tr>
<tr>
<td>Charniak (2000)</td>
<td>90.1  90.1  90.1</td>
<td>89.6  89.5  89.6</td>
<td>1020</td>
</tr>
<tr>
<td>McClosky et al. (2006)</td>
<td></td>
<td>92.1</td>
<td></td>
</tr>
<tr>
<td>Finkel et al. (2008)</td>
<td>88.8  89.2  89.0</td>
<td>87.8  88.2  88.0</td>
<td></td>
</tr>
<tr>
<td>Petrov et al. (2008)</td>
<td></td>
<td>90.0</td>
<td>89.4</td>
</tr>
<tr>
<td>Carreras et al. (2008)</td>
<td></td>
<td>89.9  91.1  90.5</td>
<td></td>
</tr>
<tr>
<td>NN (window)</td>
<td>81.3  81.9  81.6</td>
<td>80.3  81.0  80.6</td>
<td></td>
</tr>
<tr>
<td>NN+LM (window)</td>
<td>84.2  85.7  84.9</td>
<td>83.5  85.1  84.3</td>
<td></td>
</tr>
<tr>
<td>NN+LM+POS (window)</td>
<td>85.6  86.8  86.2</td>
<td>84.8  86.2  85.5</td>
<td></td>
</tr>
<tr>
<td>NN+LM+POS (sentence)</td>
<td>88.1  88.8  88.5</td>
<td>87.5  88.3  87.9</td>
<td>76</td>
</tr>
</tbody>
</table>
SENNA Demo

- Standalone, less than 3000 lines of C code
- Part-of-speech tagging, Chunking, Name entity recognition, Semantic Role Labeling, Parsing
- Available at http://ml.nec-labs.com/software/senna

```c
void SENNA_nn_viterbi(int *path, float *init, float *transition, float *emission, int N, int T)
{
    float *delta, *deltap;
    int *phi;
    int i, j, t;

    /* misc allocations */
    delta = SENNA_malloc(sizeof(float), N);
    deltap = SENNA_malloc(sizeof(float), N);
    phi = SENNA_malloc(sizeof(float), N*T);

    /* init */
    for(i = 0; i < N; i++)
        deltap[i] = init[i] + emission[i];

    /* recursion */
    for(t = 1; t < T; t++)
    {
        float *deltan = delta;
        for(j = 0; j < N; j++)
        {
            float maxValue = -FLT_MAX;
            int maxIndex = 0;
            for(i = 0; i < N; i++)
            {
                float z = deltap[i] + transition[i+j*N];
                if(z > maxValue)
                {
                    maxValue = z;
                    maxIndex = i;
                }
            }
        }
    }
```