CNNs and Neural CRFs

Wei Xu

(many slides from Greg Durrett, Stanford 231n)
Recall: RNNs

- Cell that takes some input $x$, has some hidden state $h$, and updates that hidden state and produces output $y$ (all vector-valued)
Recall: LSTM & GRU

Standard RNN
(with a single layer)

LSTM
(Long Short-term Memory)

GRU
(Gated Recurrent Unit)
Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence

Encoding of each word — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)

RNN can be viewed as a transformation of a sequence of vectors into a sequence of context-dependent vectors
This Lecture

- CNNs
- CNNs for Sentiment, Entity Linking
- Neural CRFs
A Primer on Neural Network Models for Natural Language Processing

Yoav Goldberg
Draft as of October 5, 2015.

The most up-to-date version of this manuscript is available at http://www.cs.biu.ac.il/~yogev/nlp.pdf. Major updates will be published on arxiv periodically. I welcome any comments you may have regarding the content and presentation. If you spot a missing reference or have relevant work you’d like to see mentioned, do let me know. first.last@gmail

Abstract

Over the past few years, neural networks have re-emerged as powerful machine-learning models, yielding state-of-the-art results in fields such as image recognition and speech processing. More recently, neural network models started to be applied also to textural natural language signals, again with very promising results. This tutorial surveys neural network models from the perspective of natural language processing research, in an attempt to bring natural-language researchers up to speed with the neural techniques. The tutorial covers input encoding for natural language tasks, feed-forward networks, convolutional networks, recurrent networks and recursive networks, as well as the computation graph abstraction for automatic gradient computation.
CNNs
Convolutional Neural Networks

LeCun et al. (1998)
A Bit of History

- The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.
- Perceptron (Frank Rosenblatt, 1957)
- Artificial Neuron (McCulloch & Pitts, 1943)

The IBM Automatic Sequence Controlled Calculator, called Mark I by Harvard University’s staff. It was designed for image recognition: it had an array of 400 photocells, randomly connected to the “neurons”. Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.

[Link to video](https://www.youtube.com/watch?v=cNxadbrN.ai&feature=emb_logo)
A Bit of History

- Adaline/Madeline - single and multi-layer “artificial neurons” (Widrow and Hoff, 1960)
A Bit of History

First time back-propagation became popular (Rumelhart et al, 1986)

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California,
  San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University,
  Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for
networks of neuron-like units. The procedure repeatedly adjusts
the weights of the connections in the network so as to minimize a
measure of the difference between the actual output vector of the
net and the desired output vector. As a result of the weight
adjustments, internal 'hidden' units which are not part of the input
or output come to represent important features of the task domain,
and the regularities in the task are captured by the interactions
of these units. The ability to create new features distinguishes
back-propagation from earlier, simpler methods such as the
perceptron-convergence procedure.

There have been many attempts to design self-organizing
neural networks. The aim is to find a powerful synaptic
modification rule that will allow an arbitrarily connected neural
network to develop an internal structure that is appropriate for
a particular task domain. The task is specified by giving the
desired state vector of the output units for each state vector of
the input units. If the input units are directly connected to the
output units it is relatively easy to find learning rules that
iteratively adjust the relative strengths of the connections so as
to progressively reduce the difference between the actual and
desired output vectors. Learning becomes more interesting but
more difficult when we introduce hidden units whose actual or
desired states are not specified by the task. (In perceptrons,
there are 'feature analysers' between the input and output that
are not true hidden units because their input connections are
fixed by hand, so their states are completely determined by the
input vector: they do not learn representations.) The learning
procedure must decide under what circumstances the hidden
units should be active in order to help achieve the desired
input-output behaviour. This amounts to deciding what these
units should represent. We demonstrate that a general purpose
and relatively simple procedure is powerful enough to construct
appropriate internal representations.

The simplest form of the learning procedure is for layered
networks which have a layer of input units at the bottom; any
number of intermediate layers; and a layer of output units at
the top. Connections within a layer or from higher to lower
layers are forbidden, but connections can skip intermediate
layers. An input vector is presented to the network by setting
the states of the input units. Then the states of the units in each
layer are determined by applying equations (1) and (2) to the
connections coming from lower layers. All units within a layer
have their states set in parallel, but different layers have their
states set sequentially, starting at the bottom and working
upwards until the states of the output units are determined.

The total input, \(x_i\), to unit \(j\) is a linear function of the outputs,
\(y_i\), of the units that are connected to \(j\) and of the weights,
\(w_{ij}\), on these connections:

\[
x_j = \sum_i w_{ij} y_i
\]

(1)

Units can be given biases by introducing an extra input to each
unit which always has a value of 1. The weight on this extra
input is called the bias and is equivalent to a threshold of the
opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, \(y_j\), which is a non-linear
function of its total input

\[
y_j = \frac{1}{1 + e^{-x_j}}
\]

(2)
A Bit of History

First time back-propagation became popular (Rumelhart et al, 1986)

Fig. 5 A synchronous iterative net that is run for three iterations and the equivalent layered net. Each time-step in the recurrent net corresponds to a layer in the layered net. The learning procedure for layered nets can be mapped into a learning procedure for iterative nets. Two complications arise in performing this mapping: first, in a layered net the output levels of the units in the intermediate layers during the forward pass are required for performing the backward pass (see equations (5) and (6)). So in an iterative net it is necessary to store the history of output states of each unit. Second, for a layered net to be equivalent to an iterative net, corresponding weights between different layers must have the same value. To preserve this property, we average $\Delta E/\Delta w$ for all the weights in each set of corresponding weights and then change each weight in the set by an amount proportional to this average gradient. With these two provisos, the learning procedure can be applied directly to iterative nets. These nets can then either learn to perform iterative searches or learn sequential structures.  

To break symmetry we start with small random weights. Variants on the learning procedure have been discovered independently by David Parker (personal communication) and by Yann Le Cun.

One simple task that cannot be done by just connecting the input units to the output units is the detection of symmetry. To detect whether the binary activity levels of a one-dimensional array of input units are symmetrical about the centre point, it is essential to use an intermediate layer because the activity in an individual input unit, considered alone, provides no evidence about the symmetry or non-symmetry of the whole input vector, so simply adding up the evidence from the individual input units is insufficient. (A more formal proof that intermediate units are required is given in ref. 2.) The learning procedure discovered an elegant solution using just two intermediate units, as shown in Fig. 1.

Another interesting task is to store the information in the two family trees (Fig. 2). Figure 3 shows the network we used, and Fig. 4 shows the 'receptive fields' of some of the hidden units after the network was trained on 100 of the 104 possible triples.

So far, we have only dealt with layered, feed-forward networks. The equivalence between layered networks and recurrent networks that are run iteratively is shown in Fig. 5. The most obvious drawback of the learning procedure is that the error-surface may contain local minima so that gradient descent is not guaranteed to find a global minimum. However, experience with many tasks shows that the network very rarely gets stuck in poor local minima that are significantly worse than the global minimum. We have only encountered this undesirable behaviour in networks that have just enough connections to perform the task. Adding a few more connections creates extra dimensions in weight-space and these dimensions provide paths around the barriers that create poor local minima in the lower dimensional subspaces.

©1986 Nature Publishing Group
A Bit of History

- Long Short-term Memory (Hochreiter & Schmidhuber, 1997)

![Diagram of Long Short-term Memory](image)

Figure 1: Architecture of memory cell $c_j$ (the box) and its gate units $in_j$, $out_j$. The self-recurrent connection (with weight 1.0) indicates feedback with a delay of 1 time step. It builds the basis of the “constant error carousel” CEC. The gate units open and close access to CEC. See text and appendix A.1 for details.
A Bit of History

LeCun et al. (1998)
A Bit of History

Figure 1: Neural architecture: $f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1}))$ where $g$ is the neural network and $C(i)$ is the $i$-th word feature vector.

Bengio et al. (2003)
A Bit of History

- Reinvigorated research in deep learning (Hinton & Salakhutdinov, 2006)
Convolutional Neural Networks

- AlexNet - one of the first strong results
- more filters per layer as well as stacked convolutional layers
- use of ReLU for the non-linear part instead of sigmoid or Tanh

Krizhevsky et al. (2012)
ImageNet - Object Recognition

The Image Classification Challenge:
1,000 object classes
1,431,167 images

Output:
- Scale
- T-shirt
- Steel drum
- Drumstick
- Mud turtle

Output:
- Scale
- T-shirt
- Giant panda
- Drumstick
- Mud turtle

Russakovsky et al. (2012)
ImageNet - Object Recognition

- 28% in 2010
- 26% in 2011
- 16% in 2012
- 12% in 2013
- 7.3% in 2014
- 6.7% in 2014
- 3.6% in 2015
- 3.0% in 2016
- 2.25% in 2017

- AlexNet, 8 layers
- ZF, 8 layers
- VGG, 19 layers
- GoogLeNet, 22 layers
- ResNet, 152 layers
- (Ensemble)
- SENet

100% accuracy and reliability not realistic

Human error
### Convolutional Layer

- Applies a **filter** over patches of the input and returns that filter’s activations.
- **Convolution**: take dot product of filter with a patch of the input.

**Images**: RGB values (3 dim)

**Image**: $n \times n \times k$

**Filter**: $m \times m \times k$

Each of these cells is a vector with multiple values.

\[
activation_{ij} = \sum_{i_o=0}^{m-1} \sum_{j_o=0}^{m-1} \text{image}(i + i_o, j + j_o) \cdot \text{filter}(i_o, j_o)
\]

Offsets

Sum over dot products
Convolutional Layer

- An animated example: \( k = 1 \), and a filter of size 3x3.

![Image](image.png)

**Image**

**Convolved Feature**
Convoluational Layer

- Applies a *filter* over patches of the input and returns that filter’s activations
- Convolution: take dot product of filter with a patch of the input

image: \( n \times n \times k \)  
filter: \( m \times m \times k \)  
activations: \( (n - m + 1) \times (n - m + 1) \times 1 \)
Convolutions for NLP

- Input and filter are 2-dimensional instead of 3-dimensional

sentence: $n$ words x $k$ vec dim

the movie was good

filter: $m$ x $k$

activations: $(n - m + 1) \times 1$

Combines evidence locally in a sentence and produces a new (but still variable-length) representation
Compare: CNNs vs. LSTMs

- Both LSTMs and convolutional layers transform the input using context.
- LSTM: “globally” looks at the entire sentence (but local for many problems).
- CNN: local depending on filter width + number of layers.

The movie was good
CNNs for Sentiment
CNNs for Sentiment Analysis

\[ P(y|x) \]
projection + softmax

c-dimensional vector

max pooling over the sentence

n x c
c filters, 
m x k each

the movie was good

Max pooling: return the max activation of a given filter over the entire sentence; like a logical OR (sum pooling is like logical AND)
the  |  movie  | was  | good
---|---|---|---
0.03 | 0.02 | 0.1 | 1.1

Filter “looks like” the things that will cause it to have high activation

“good” filter output
max = 1.1
Understanding CNNs for Sentiment

The movie was good.

\[
\begin{align*}
\text{the} & \quad 0.03 \\
\text{movie} & \quad 0.02 \\
\text{was} & \quad 0.1 \\
\text{good} & \quad 1.1 \\
. & \quad 0.0
\end{align*}
\]

\[
\begin{align*}
\text{“bad”} & \quad 0.1 \\
\text{“okay”} & \quad 0.3 \\
\text{“terrible”} & \quad 0.1
\end{align*}
\]

max = 1.1
Understanding CNNs for Sentiment

- Takes variable-length input and turns it into fixed-length output
- Filters are initialized randomly and then learned
Understanding CNNs for Sentiment

the  0.03
movie  0.02
was  0.1

max = 1.8

great  1.8
.

Word vectors for similar words are similar, so convolutional filters will have similar outputs
Understanding CNNs for Sentiment

- Analogous to bigram features in bag-of-words models
- Indicator feature of text containing bigram <-> max pooling of a filter that matches that bigram

```
the  0.03
movie 0.14
was  0.1
not  1.5
good 0.0
```

"not good"

max = 1.5
What can CNNs learn?

- CNNs let us take advantage of word similarity
  
  *really not very good* vs. *really not very enjoyable*

- CNNs are translation-invariant like bag-of-words
  
  *The movie was bad, but blah blah blah ...* vs. *... blah blah blah, but the movie was bad.*

- CNNs can capture local interactions with filters of width $> 1$
  
  *It was not good, it was actually quite bad* vs. *it was not bad, it was actually quite good*
Deep Convolutional Networks

- Low-level filters: extract low-level features from the data

Layer 2

Zeiler and Fergus (2014)
Deep Convolutional Networks

- High-level filters: match larger and more “semantic patterns”

Zeiler and Fergus (2014)
CNNs: Implementation

- Input is batch_size x n x k matrix, filters are c x m x k matrix (c filters)
- Typically use filters with m ranging from 1 to 5 or so (multiple filter widths in a single convnet)
- All computation graph libraries support efficient convolution operations

```
CLASS torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0,
                        dilation=1, groups=1, bias=True, padding_mode='zeros')
```

Applies a 1D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size \((N, C_{\text{in}}, L)\) and output \((N, C_{\text{out}}, L_{\text{out}})\) can be precisely described as:

\[
\text{out}(N_i, C_{out,j}) = \text{bias}(C_{out,j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out,j}, k) \times \text{input}(N_i, k)
\]

where \(\times\) is the valid cross-correlation operator, \(N\) is a batch size, \(C\) denotes a number of channels, \(L\) is a length of signal sequence.

- `stride` controls the stride for the cross-correlation, a single number or a one-element tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for padding number of points.
CNNs for Sentence Classification

- Question classification, sentiment, etc.
- Conv+pool, then use feedforward layers to classify
- Can use multiple types of input vectors (fixed initializer and learned)

Kim (2014)
CNNs for Sentence Classification

Figure 1: Model architecture with two channels for an example sentence.

- **wait for the video and do n't rent it**
- **n x k** representation of sentence with static and non-static channels
- Convolutional layer with multiple filter widths and feature maps
- Max-over-time pooling
- Fully connected layer with dropout and softmax output

Kim (2014)
Sentence Classification

- movie review
- sentiment
- subjectivity/objectivity detection
- product reviews
- question type classification

Also effective at document-level text classification

Kim (2014)
Entity Linking

- CNNs can produce good representations of both sentences and documents like typical bag-of-words features.
- Can distill topic representations for use in entity linking.

- They had disqualified Armstrong from his seven consecutive Armstrong County

- Lance Armstrong

- Armstrong County

- Cycling domain

- Geopolitical domain
Although he originally won the event, the United States Anti-Doping Agency announced in August 2012 that they had disqualified Armstrong from his seven consecutive Tour de France wins from 1999–2005.

Lance Edward Armstrong is an American former professional road cyclist.

Armstrong County is a county in Pennsylvania...

\[
P(y|x) = \text{softmax}(s)
\]

\[
s_{\text{Lance}} = d \cdot a_{\text{Lance}}
\]

\[
s_{\text{County}} = d \cdot a_{\text{County}}
\]

Francis-Landau et al. (2016)
Neural CRF Basics
\begin{itemize}
  \item Features in CRFs: \texttt{I[tag=B-LOC & curr\_word=Hangzhou]}, \texttt{I[tag=B-LOC & prev\_word=to]}, \texttt{I[tag=B-LOC & curr\_prefix=Han]}
  \item Linear model over features
  \item Downsides:
    \begin{itemize}
      \item Lexical features mean that words need to be seen in the training data
      \item Linear model can’t capture feature conjunctions as effectively (doesn’t work well to look at more than 2 words with a single feature)
    \end{itemize}
\end{itemize}

\textit{Barack Obama will travel to Hangzhou today for the G20 meeting.}
Barack Obama will travel to Hangzhou today for the G20 meeting.

Transducer (LM-like model)

What are the strengths and weaknesses of this model compared to CRFs?
LSTMs for NER

Barack Obama will travel to Hangzhou today for the G20 meeting.

- Bidirectional transducer model
- What are the strengths and weaknesses of this model compared to CRFs?
Recall: Sequential CRFs

- Model: \( P(y|x) = \frac{1}{Z} \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, x)) \)

- Normalizing constant \( Z = \sum_y \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, x)) \)
Recall: Sequential CRFs

- Inference: use Viterbi algorithm
  \[ P(y_{\text{max}}|x) = \max_{y_1,\ldots,y_n} P(y|x) \]

- Learning: run forward-backward to compute marginals
  \[ P(y_i = s|x) = \sum_{y_1,\ldots,y_{i-1},y_{i+1},\ldots,y_n} P(y|x) \]
  \[ P(y_i = s_1, y_{i+1} = s_2|x) \], then update gradient
Neural CRFs

Barack Obama will travel to Hangzhou today for the G20 meeting.

> Neural CRFs: bidirectional LSTMs (or some NN) compute emission potentials, capture structural constraints in transition potentials.
Neural CRFs

\[
P(y|x) = \frac{1}{Z} \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, x))
\]

- Conventional: \( \phi_e(y_i, i, x) = w^\top f_e(y_i, i, x) \)
- Neural: \( \phi_e(y_i, i, x) = W_{y_i}^\top f(i, x) \) \( W \) is a num_tags x len(f) matrix
- \( f(i, x) \) could be the output of a feedforward neural network looking at the words around position \( i \), or the \( i \)th output of an LSTM, ...
- Neural network computes unnormalized potentials that are consumed and “normalized” by a structured model
- Inference: compute \( f \), use Viterbi
Computing Gradients

\[ P(y|x) = \frac{1}{Z} \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, x)) \]

- Conventional: \( \phi_e(y_i, i, x) = w^\top f_e(y_i, i, x) \)
- Neural: \( \phi_e(y_i, i, x) = W_{y_i}^\top f(i, x) \)

\[ \frac{\partial L}{\partial \phi_{e,i}} = -P(y_i = s|x) + I[s \text{ is gold}] \]

“error signal”, compute with F-B chain rule say to multiply together, gives our update

- For linear model: \( \frac{\partial \phi_{e,i}}{w_i} = f_{e,i}(y_i, i, x) \)
- For neural model: compute gradient of phi w.r.t. parameters of neural net
Neural CRFs

Barack Obama will travel to Hangzhou today for the G20 meeting.

1) Compute $f(x)$
2) Run forward-backward
3) Compute error signal
4) Backprop (no knowledge of sequential structure required)
Barack Obama will travel to Hangzhou today for the G20 meeting.

\[
\phi_e = W g(V f(x, i))
\]

\[
f(x, i) = [\text{emb}(x_{i-1}), \text{emb}(x_i), \text{emb}(x_{i+1})]
\]
Barack Obama will travel to Hangzhou today for the G20 meeting.

How does this compare to neural CRF?
### “NLP (Almost) From Scratch”

<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>

- **WLL**: independent classification; **SLL**: neural CRF
- **LM2**: word vectors learned from a precursor to word2vec/GloVe, trained for 2 weeks (!) on Wikipedia

CNN Neural CRFs

- Append to each word vector an *embedding of the relative position* of that word
- Convolution over the sentence produces a position-dependent representation
### CNN NCRFs vs. FFNN NCRFs

<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><strong>Window Approach</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN+SLL+LM2</td>
<td>97.20</td>
<td>93.63</td>
<td>88.67</td>
<td>–</td>
</tr>
<tr>
<td><strong>Sentence Approach</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN+SLL+LM2</td>
<td>97.12</td>
<td>93.37</td>
<td>88.78</td>
<td>74.15</td>
</tr>
</tbody>
</table>

- Sentence approach (CNNs) is comparable to window approach (FFNNs) except for SRL where they claim it works much better.

Collobert and Weston 2008, 2011
Neural CRFs with LSTMs

- Neural CRF using character LSTMs to compute word representations

Chiu and Nichols (2015), Lample et al. (2016)
Neural CRFs with LSTMs

- Chiu+Nichols: character CNNs instead of LSTMs
- Lin/Passos/Luo: use external resources like Wikipedia
- LSTM-CRF captures the important aspects of NER: word context (LSTM), sub-word features (character LSTMs), outside knowledge (word embeddings)

<table>
<thead>
<tr>
<th>Model</th>
<th>F₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collobert et al. (2011)*</td>
<td>89.59</td>
</tr>
<tr>
<td>Lin and Wu (2009)</td>
<td>83.78</td>
</tr>
<tr>
<td>Lin and Wu (2009)*</td>
<td>90.90</td>
</tr>
<tr>
<td>Huang et al. (2015)*</td>
<td>90.10</td>
</tr>
<tr>
<td>Passos et al. (2014)</td>
<td>90.05</td>
</tr>
<tr>
<td>Passos et al. (2014)*</td>
<td>90.90</td>
</tr>
<tr>
<td>Luo et al. (2015)* + gaz</td>
<td>89.9</td>
</tr>
<tr>
<td>Luo et al. (2015)* + gaz + linking</td>
<td>91.2</td>
</tr>
<tr>
<td>Chiu and Nichols (2015)</td>
<td>90.69</td>
</tr>
<tr>
<td>Chiu and Nichols (2015)*</td>
<td>90.77</td>
</tr>
<tr>
<td>LSTM-CRF (no char)</td>
<td>90.20</td>
</tr>
<tr>
<td>LSTM-CRF</td>
<td>90.94</td>
</tr>
</tbody>
</table>

Chiu and Nichols (2015), Lample et al. (2016)
Takeaways

- CNNs are a flexible way of extracting features analogous to bag of n-grams, can also encode positional information.

- All kinds of NNs can be integrated into CRFs for structured inference. Can be applied to NER, other tagging, parsing, ...