Recall: CRFs

\[ P(y|\mathbf{x}) = \frac{1}{Z} \exp \left( \sum_{k=1}^{n} w^T f_k(\mathbf{x}, y) \right) \]

- Conditional model: x’s are observed
- Naive Bayes: logistic regression :: HMMs : CRFs
  - local vs. global normalization <-> generative vs. discriminative
    - (locally normalized discriminative models do exist (MEMMs))
- HMMs: in the standard setup, emissions consider one word at a time
- CRFs: features over many words simultaneously, non-independent features (e.g., suffixes and prefixes), doesn’t have to be a generative model

Recall: Sequential CRFs

- Model: \( P(y|\mathbf{x}) \propto \exp w^T \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, \mathbf{x}) \right] \)
  - Emission features capture word-level info, transitions enforce tag consistency
- Inference: argmax \( P(y|\mathbf{x}) \) from Viterbi
  - Learning: run forward-backward to compute posterior probabilities; then
  \[
  \frac{\partial}{\partial w} \mathcal{L}(\mathbf{y}^*, \mathbf{x}) = \sum_{i=1}^{n} f_e(y_i^*, i, \mathbf{x}) - \sum_{i=1}^{n} \sum_{s} P(y_i = s|\mathbf{x}) f_e(s, i, \mathbf{x})
  \]
This Lecture

- Finish discussion of NER
- Beam search: in a few lectures
- Neural network history
- Neural network basics
- Feedforward neural networks + backpropagation
- Applications
- Implementing neural networks (if time)

NER

- CRF with lexical features can get around 85 F1 on this problem
- Other pieces of information that many systems capture
- World knowledge:
  The delegation met the president at the airport, Tanjug said.

Tanjug
From Wikipedia, the free encyclopedia
Tanjug (Serbian Cyrillic: Tanjug) is a Serbian state news agency based in Belgrade.

NER

- More complex factor graph structures can let you capture this, or just decode sentences in order and use features on previous sentences

Finkel and Manning (2008), Ratino and Roth (2009)
Semi-Markov Models

- Barack Obama will travel to Hangzhou today for the G20 meeting.
  - Chunk-level prediction rather than token-level BIO
  - \( y \) is a set of spans covering the sentence
  - Pros: features can look at whole span at once
  - Cons: there’s an extra factor of \( n \) in the dynamic programs

How well do NER systems do?

<table>
<thead>
<tr>
<th>System</th>
<th>Resources Used</th>
<th>( F_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBJ-NER</td>
<td>Wikipedia, Nonlocal Features, Word-class Model</td>
<td>90.80</td>
</tr>
<tr>
<td>(Suzuki and Isozaki, 2008)</td>
<td>Semi-supervised on 1G-word unlabeled data</td>
<td>89.92</td>
</tr>
<tr>
<td>(Ando and Zhang, 2005)</td>
<td>Semi-supervised on 27M-word unlabeled data</td>
<td>89.31</td>
</tr>
<tr>
<td>(Kazama and Torisawa, 2007a)</td>
<td>Wikipedia</td>
<td>88.02</td>
</tr>
<tr>
<td>(Krishnan and Manning, 2006)</td>
<td>Non-local Features</td>
<td>87.24</td>
</tr>
<tr>
<td>(Kazama and Torisawa, 2007b)</td>
<td>Non-local Features</td>
<td>87.17</td>
</tr>
<tr>
<td>(Finkel et al., 2005)</td>
<td>Non-local Features</td>
<td>86.86</td>
</tr>
<tr>
<td>Lample et al. (2016)</td>
<td>LSTM-CRF (no char)</td>
<td>90.20</td>
</tr>
<tr>
<td></td>
<td>LSTM-CRF (no char)</td>
<td>90.44</td>
</tr>
<tr>
<td></td>
<td>S-LSTM-CRF</td>
<td>87.96</td>
</tr>
<tr>
<td></td>
<td>S-LSTM</td>
<td>89.03</td>
</tr>
<tr>
<td>BiLSTM-CRF + ELMo Peters et al. (2018)</td>
<td>92.2</td>
<td></td>
</tr>
<tr>
<td>BERT Devlin et al. (2019)</td>
<td>92.8</td>
<td></td>
</tr>
</tbody>
</table>

Modern Entity Typing

- More and more classes (17 -> 112 -> 10,000+)

Choi et al. (2018)
Neural Net History

2008-2013: A glimmer of light...

- Collobert and Weston 2011: “NLP (almost) from scratch”
  - Feedforward neural nets induce features for sequential CRFs (“neural CRF”)
  - 2008 version was marred by bad experiments, claimed SOTA but wasn’t, 2011 version tied SOTA
- Socher 2011-2014: tree-structured RNNs working okay
- Krizhevsky et al. (2012): AlexNet for vision

2014: Stuff starts working

- Sutskever et al. + Bahdanau et al.: seq2seq for neural MT (LSTMs)
- Chen and Manning transition-based dependency parser (based on feedforward networks)
- 2015: explosion of neural nets for everything under the sun

What made these work? Data (not as important as you might think), optimization (initialization, adaptive optimizers), representation (good word embeddings)
Neural Net Basics

Neural Networks

- Linear classification: \( \arg\max_y w^T f(x, y) \)
- Want to learn intermediate conjunctive features of the input:
  
  \[
  \text{the movie was not all that good}
  \]
  
  \(I[\text{contains not} \& \text{contains good}]\)

- How do we learn this if our feature vector is just the unigram indicators?
  
  \(I[\text{contains not}], I[\text{contains good}]\)

Neural Networks: XOR

- Let’s see how we can use neural nets to learn a simple nonlinear function
- Inputs \( x_1, x_2 \)
  
  (generally \( x = (x_1, \ldots, x_m) \))
- Output \( y \)
  
  (generally \( y = (y_1, \ldots, y_n) \))

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( y = x_1 \text{ XOR } x_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( 1 \text{ XOR } 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Neural Networks: XOR

\[
 y = a_1 x_1 + a_2 x_2
\]

\[
 y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2)
\]

“or”

(looks like action potential in neuron)
Neural Networks:

\[
\begin{align*}
y &= a_1 x_1 + a_2 x_2 \\
y &= a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2) \\
y &= -x_1 - x_2 + 2 \tanh(x_1 + x_2)
\end{align*}
\]

"or"

\[
\begin{align*}
y &= g(w \cdot x + b) \\
y &= g(Wx + b)
\end{align*}
\]

Nonlinear transformation
Warps space

Check: what happens if no nonlinearity?
More powerful than basic linear models?

\[
z = V(g(Wx + b) + c)
\]

output of first layer

Neural Networks: XOR

<table>
<thead>
<tr>
<th>x_1</th>
<th>x_2</th>
<th>x_1 XOR x_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Neural Networks

Linear classifier
Neural network ...
possible because we transformed the space!

Deep Neural Networks

\[
\begin{align*}
y &= g(Wx + b) \\
z &= g(Vy + c) \\
z &= g(Vg(Wx + b) + c)
\end{align*}
\]

output of first layer

Check: what happens if no nonlinearity?
More powerful than basic linear models?

\[
z = V(Wx + b) + c
\]

Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
Feedforward Networks, Backpropagation

Logistic Regression with NNs

\[ P(y|x) = \frac{\exp(w^T f(x, y))}{\sum_{y'} \exp(w^T f(x, y'))} \]
- Single scalar probability

\[ P(y|x) = \text{softmax} \left( \left[ w^T f(x, y) \right]_{y \in Y} \right) \]
- Compute scores for all possible labels at once (returns vector)

\[ \text{softmax}(p)_i = \frac{\exp(p_i)}{\sum_{i'} \exp(p_{i'})} \]
- \( \text{softmax} \): \( \exp \)s and normalizes a given vector

\[ P(y|x) = \text{softmax}(W f(x)) \]
- Weight vector per class; \( W \) is \( [\text{num classes} \times \text{num feats}] \)

\[ P(y|x) = \text{softmax}(W g(V f(x))) \]
- Now one hidden layer

Neural Networks for Classification

\[ P(y|x) = \text{softmax}(W g(V f(x))) \]

Training Neural Networks

\[ P(y|x) = \text{softmax}(W z) \quad z = g(V f(x)) \]
- Maximize log likelihood of training data

\[ \mathcal{L}(x, i^*) = \log P(y = i^* | x) = \log (\text{softmax}(W z) \cdot e_{i^*}) \]
- \( i^* \): index of the gold label

\[ \mathcal{L}(x, i^*) = W z \cdot e_{i^*} - \log \sum_j \exp(W z) \cdot e_j \]
- \( e_i \): 1 in the \( i \)th row, zero elsewhere. Dot by this = select \( i \)th index
Computing Gradients

\[ \mathcal{L}(\mathbf{x}, i^*) = W \mathbf{z} \cdot e_{i^*} - \log \sum_j \exp(W \mathbf{z}) \cdot e_j \]

- Gradient with respect to \( W \)
  \[ \frac{\partial}{\partial W_{ij}} \mathcal{L}(\mathbf{x}, i^*) = \begin{cases} 
  z_j - P(y = i|\mathbf{x})z_j & \text{if } i = i^* \\
  -P(y = i|\mathbf{x})z_j & \text{otherwise}
  \end{cases} \]

- Looks like logistic regression with \( z \) as the features!

Neural Networks for Classification

\[ P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x}))) \]

Backpropagation: Picture

\[ P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x}))) \]

- Can forget everything after \( \mathbf{z} \), treat it as the output and keep backproping
Backpropagation: Takeaways

- Gradients of output weights $W$ are easy to compute — looks like logistic regression with hidden layer $z$ as feature vector
- Can compute derivative of loss with respect to $z$ to form an “error signal” for backpropagation
- Easy to update parameters based on “error signal” from next layer, keep pushing error signal back as backpropagation
- Need to remember the values from the forward computation

Applications

NLP with Feedforward Networks

- Part-of-speech tagging with FFNNs
- Word embeddings for each word form input
- ~1000 features here — smaller feature vector than in sparse models, but every feature fires on every example
- Weight matrix learns position-dependent processing of the words

Botha et al. (2017)

NLP with Feedforward Networks

- Hidden layer mixes these different signals and learns feature conjunctions

Botha et al. (2017)
NLP with Feedforward Networks

- Multilingual tagging results:

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc.</th>
<th>Wts.</th>
<th>MB</th>
<th>Ops.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gillick et al. (2016)</td>
<td>95.06</td>
<td>900k</td>
<td>-</td>
<td>6.63m</td>
</tr>
<tr>
<td>Small FF</td>
<td>94.76</td>
<td>241k</td>
<td>0.6</td>
<td>0.27m</td>
</tr>
<tr>
<td>+Clusters</td>
<td>95.56</td>
<td>261k</td>
<td>1.0</td>
<td>0.31m</td>
</tr>
<tr>
<td>1/2 Dim.</td>
<td>95.39</td>
<td>143k</td>
<td>0.7</td>
<td>0.18m</td>
</tr>
</tbody>
</table>

- Gillick used LSTMs; this is smaller, faster, and better

Botha et al. (2017)

Sentiment Analysis

- Deep Averaging Networks: feedforward neural network on average of word embeddings from input

Iyyer et al. (2015)

<table>
<thead>
<tr>
<th>Model</th>
<th>RT</th>
<th>SST fine</th>
<th>SST bin</th>
<th>IMDB</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAN-ROOT</td>
<td>-</td>
<td>46.9</td>
<td>85.7</td>
<td>—</td>
<td>31</td>
</tr>
<tr>
<td>DAN-RAND</td>
<td>77.3</td>
<td>45.4</td>
<td>83.2</td>
<td>88.8</td>
<td>136</td>
</tr>
<tr>
<td>DAN</td>
<td>80.3</td>
<td>47.7</td>
<td>86.3</td>
<td>89.4</td>
<td>136</td>
</tr>
<tr>
<td>NBOW-RAND</td>
<td>76.2</td>
<td>42.3</td>
<td>81.4</td>
<td>88.9</td>
<td>91</td>
</tr>
<tr>
<td>NBOW</td>
<td>79.0</td>
<td>43.6</td>
<td>83.6</td>
<td>89.0</td>
<td>91</td>
</tr>
<tr>
<td>BiNB</td>
<td>41.9</td>
<td>83.1</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>NBSVM-bi</td>
<td>79.4</td>
<td>—</td>
<td>91.2</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Bag-of-words

Iyyer et al. (2015)

<table>
<thead>
<tr>
<th>Model</th>
<th>RT</th>
<th>SST fine</th>
<th>SST bin</th>
<th>IMDB</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecNN</td>
<td>77.7</td>
<td>43.2</td>
<td>82.4</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>RecNTN</td>
<td>45.7</td>
<td>85.4</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>DRecNN</td>
<td>49.8</td>
<td>86.6</td>
<td>—</td>
<td>—</td>
<td>431</td>
</tr>
<tr>
<td>TreeLSTM</td>
<td>50.6</td>
<td>86.9</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>DCNN</td>
<td>48.5</td>
<td>86.9</td>
<td>89.4</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>PVEC</td>
<td>48.7</td>
<td>87.8</td>
<td>92.6</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>CNN-MC</td>
<td>81.1</td>
<td>47.4</td>
<td>88.1</td>
<td>—</td>
<td>2,452</td>
</tr>
<tr>
<td>WRRBM</td>
<td>—</td>
<td>—</td>
<td>89.2</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Tree RNNs / CNNS / LSTMS

Wang and Manning (2012)

<table>
<thead>
<tr>
<th>Model</th>
<th>RT</th>
<th>SST fine</th>
<th>SST bin</th>
<th>IMDB</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-MC</td>
<td>81.1</td>
<td>47.4</td>
<td>88.1</td>
<td>—</td>
<td>2,452</td>
</tr>
<tr>
<td>WRRBM</td>
<td>—</td>
<td>—</td>
<td>89.2</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Kim (2014)

Implementation Details
Computing gradients is hard! Computation graph abstraction allows us to define a computation symbolically and will do this for us.

Automatic differentiation: keep track of derivatives / be able to backpropagate through each function:

\[ y = x \times x \quad \rightarrow \quad (y, dy) = (x \times x, 2 \times x \times dx) \]

codegen

Use a library like Pytorch or Tensorflow. This class: Pytorch

Define forward pass for \( P(y|x) = \text{softmax}(Wg(Vf(x))) \)

```python
class FFNN(nn.Module):
    def __init__(self, inp, hid, out):
        super(FFNN, self).__init__()
        self.V = nn.Linear(inp, hid)
        self.g = nn.Tanh()
        self.W = nn.Linear(hid, out)
        self.softmax = nn.Softmax(dim=0)

    def forward(self, x):
        return self.softmax(self.W(self.g(self.V(x))))
```

Training a Model

Define a computation graph

For each epoch:

For each batch of data:

- Compute loss on batch
- Autograd to compute gradients
- Take step with optimizer

Decode test set
Next Time

- Training neural networks
- Word representations / word vectors
- word2vec, GloVe