Many thanks to David Weiss, Slav Petrov, Richard Socher, Chris Manning, Kevin Duh for starting point for slides
Some History

- Neural net algorithms dates from the 80’s (originally inspired by early neuroscience)
- Historically slow, complex and unwieldy
- Now: the term “neural net” is abstract enough to encompass almost any model — but useful!
- Dramatic shift in last 2-3 years away from maxent (linear, convex) to “neural net” (non-linear architecture)
The “Promise”

Most current machine learning works well because of human-designed representations and input features.

Machine learning becomes just optimizing weights to best make a final prediction.

**Representation learning** attempts to automatically learn good features or representations.

**Deep learning** algorithms attempt to learn multiple levels of representation of increasing complexity/abstraction.
From MaxEnt to Neural Nets

In NLP, a maxent classifier is normally written as:

\[
P(c \mid d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c, d)}{\sum_{c' \in C} \exp \sum_i \lambda_i f_i(c', d)}
\]

Supervised learning gives us a distribution for datum \(d\) over classes in \(C\)

Vector form:

\[
P(c \mid d, \lambda) = \frac{e^{\lambda^T f(c, d)}}{\sum_{c'} e^{\lambda^T f(c', d)}}
\]

Such a classifier is used as-is in a neural network ("a softmax layer")

- Often as the top layer: \(J = \text{softmax}(\lambda \cdot x)\)

But for now we’ll derive a two-class logistic model for one neuron
Vector form: \[ P(c \mid d, \lambda) = \frac{e^{\lambda^T f(c, d)}}{\sum_{c'} e^{\lambda^T f(c', d)}} \]

Make two class:
\[
P(c_1 \mid d, \lambda) = \frac{e^{\lambda^T f(c_1, d)}}{e^{\lambda^T f(c_1, d)} + e^{\lambda^T f(c_2, d)}} = \frac{e^{\lambda^T f(c_1, d)} \cdot e^{-\lambda^T f(c_1, d)}}{e^{\lambda^T f(c_1, d)} + e^{\lambda^T f(c_2, d)}}
\]

\[
= \frac{1}{1 + e^{\lambda^T [f(c_2, d) - f(c_1, d)]}}
\]

for \( x = f(c_1, d) - f(c_2, d) \)

\[
= \frac{1}{1 + e^{-\lambda^T x}}
\]

for \( f(z) = 1/(1 + \exp(-z)) \), the logistic function – a sigmoid non-linearity.
But that’s a neuron!

\[ h_{w,b}(x) = f(w^T x + b) \]

\[ f(z) = \frac{1}{1 + e^{-z}} \]

\( b \): We can have an “always on” feature, which gives a class prior, or separate it out, as a bias term

\( w, b \) are the parameters of this neuron i.e., this logistic regression model.
Neural Net = Several MaxEnt Models

If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs ...

But we don’t have to decide ahead of time what variables these logistic regressions are trying to predict!
Neural Net = Several MaxEnt Models

... which we can feed into another logistic regression function

It is the training criterion that will direct what the intermediate hidden variables should be, so as to do a good job at predicting the targets for the next layer, etc.
### Parts-of-Speech (English)

- **One basic kind of linguistic structure: syntactic word classes**

#### Open class (content) words

<table>
<thead>
<tr>
<th>Nouns</th>
<th>Adjectives</th>
<th>Abbreviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td><em>red, happy</em></td>
<td><em>etc.</em></td>
</tr>
<tr>
<td>Proper</td>
<td><em>IBM, John</em></td>
<td></td>
</tr>
</tbody>
</table>

#### Closed class (functional) words

<table>
<thead>
<tr>
<th>Determiners</th>
<th>Conjunctions</th>
<th>Pronouns</th>
<th>Adpositions</th>
<th>Modal</th>
<th>Punctuation</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>the, some</em></td>
<td><em>and, or</em></td>
<td><em>they, him</em></td>
<td><em>in, of, from</em></td>
<td><em>can, had</em></td>
<td>, ?, !</td>
</tr>
</tbody>
</table>

#### Other categories

- **Numbers:** *one, thousand, 1,983,213*
- **Verbs:** *ran, ate*
- **Adpositions:** *in, of, from*
- **Particles:** *off, up*
### Parts-of-Speech (German)

- One basic kind of linguistic structure: syntactic word classes

<table>
<thead>
<tr>
<th>Part of Speech</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Open class (content) words</strong></td>
<td></td>
</tr>
<tr>
<td>Nouns</td>
<td></td>
</tr>
<tr>
<td>Common</td>
<td>Katze, Hund</td>
</tr>
<tr>
<td>Proper</td>
<td>IBM, Jan</td>
</tr>
<tr>
<td>Adjectives</td>
<td>rot, froh</td>
</tr>
<tr>
<td>Adverbs</td>
<td>schnell</td>
</tr>
<tr>
<td><strong>Closed class (functional) words</strong></td>
<td></td>
</tr>
<tr>
<td>Determiners</td>
<td>die, einige</td>
</tr>
<tr>
<td>Conjunctions</td>
<td>und, oder</td>
</tr>
<tr>
<td>Pronouns</td>
<td>sie, ihm</td>
</tr>
<tr>
<td><strong>Abbreviations</strong></td>
<td>etc.</td>
</tr>
<tr>
<td><strong>Verbs</strong></td>
<td>renne, aß</td>
</tr>
<tr>
<td><strong>Numbers</strong></td>
<td>eins, Tausend, 1.983.213</td>
</tr>
<tr>
<td><strong>Adpositions</strong></td>
<td>in, aus, von</td>
</tr>
<tr>
<td><strong>Particles</strong></td>
<td>aus, an</td>
</tr>
<tr>
<td><strong>Punctuation</strong></td>
<td>., ?, !</td>
</tr>
<tr>
<td><strong>Modal</strong></td>
<td>kann, hat</td>
</tr>
</tbody>
</table>
**Common POS Categories - Nouns**

- **NN** - common noun, singular or mass
  - **Examples:** cabbage, thermostat, investment

- **NNS** - common noun, plural
  - **Examples:** undergraduates, thieves

- **NNP** - proper singular noun
  - **Examples:** Mary, Jasper

- **NNPS** - proper plural noun
  - **Examples:** Americans, Democrats
Common POS Categories - Verbs

- **VB** - verb, base form
  - **Examples:** ask, bring, fire, see, take
- **VBD** - verb, past tense
  - **Examples:** pleaded, swiped, registered, saw
- **VBG** - verb, present participle or gerund
  - **Examples:** stirring, focusing, approaching, erasing
Common POS Categories - Adjectives/Adverbs

- **JJ** - adjective or numeral, ordinal
  - **Examples:** third, ill-mannered, regrettable
- **JJR** - adjective, comparative
  - **Examples:** braver, cheaper, taller
- **RB** - adverb
  - **Examples:** occasionally, maddeningly, adventurously
- **RBR** - adverb, comparative
  - **Examples:** further, better, worse
Common POS Categories - Misc.

- **CD** - numeral, cardinal
  - **Examples**: mid-1890 nine-thirty 0.5 one

- **DT** - determiner
  - **Examples**: a, an, the

- **CC** - conjunction, coordinating
  - **Examples**: and, both, but, either, or

...
Why POS Tagging?

- Useful in and of itself (more than you’d think)
  - Text-to-speech: record, lead
  - Lemmatization: saw[v] → see, saw[n] → saw
  - Quick-and-dirty NP-chunk detection: grep {JJ | NN}* {NN | NNS}
  - Linguistically motivated word clustering

- Useful as a pre-processing step for parsing

- Useful as features to downstream systems.
Baseline Approach 1:

- Just look up the word in a dictionary and look up the part of speech tag.

- Drawback:
  - Cannot handle unknown words.
Baseline Approach 2:

- Built a classifier that maps words to part of speech tags.

**Example features:**

- **Word** the: the → DT
- **Lowercased word** Importantly: importantly → RB
- **Prefixes** unfathomable: un- → JJ
- **Suffixes** Surprisingly: -ly → RB
- **Capitalization** Meridian: CAP → NNP
- **Word shapes** 35-year: d-x → JJ
Baseline Approach 2:

- **Shortcoming:**
  - A word can map to multiple tags!
  - Example: **fire** can be both a noun and a verb
Shortcoming: Part-of-Speech Ambiguity

- Words can have multiple parts of speech

Fed  raises  interest  rates  0.5  percent

VBD   VB     VBD
VBN   VBZ    VBP
NNP   NNS    NN
NNS   CD     NN

- Two basic sources of constraints:
  - Grammatical environment
  - Identity of the current word
### Accuracy So Far

<table>
<thead>
<tr>
<th></th>
<th>Known words</th>
<th>Unknown words</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dictionary Lookup</strong></td>
<td>~90%</td>
<td>~50%</td>
</tr>
<tr>
<td><strong>Most frequent tag</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Features only based</strong></td>
<td>~93.7%</td>
<td>~82.6%</td>
</tr>
<tr>
<td>on word in question</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Classic Solution: Hidden Markov Models (HMMs)

- We want a model of sequences \( s \) and observations \( w \)

- First order (bigram) hidden markov model:

\[
P(s, w) = \prod_i P(s_i|s_{i-1})P(w_i|s_i)
\]

- Relationship to bigram language model:

\[
P(w) = \prod_i P(w_i|w_{i-1})
\]
Classic Solution: Hidden Markov Models (HMMs)

\[ P(s, w) = \prod_{i} P(s_i|s_{i-1}) P(w_i|s_i) \]

transition matrix
emission matrix

\[
\begin{array}{cccc}
\text{\textless \textbullet \textgreater} & \text{\textless t}_1\textgreater & \text{\textless t}_2\textgreater & \ldots & \text{\textless t}_n\textgreater \\

s_0 & s_1 & s_2 & \ldots & s_n \\
\downarrow & \downarrow & \downarrow & \ldots & \downarrow \\
\downarrow & \downarrow & \downarrow & \ldots & \downarrow \\
w_1 & w_2 & w_3 & \ldots & w_n \\
\end{array}
\]
Classic Solution: Hidden Markov Models (HMMs)

**Assumptions:**
- States are tag n-grams
- Usually a dedicated start and end state / word
- Tag/state sequence is generated by a Markov model
- Words are chosen independently, conditioned only on the tag/state

These are totally broken assumptions: why?

\[
P(s, w) = \prod_i P(s_i | s_{i-1}) P(w_i | s_i)
\]
Can Use Higher Order HMMs

- Second order HMM (trigram tagger):

\[ P(s, w) = \prod_i P(s_i | s_{i-1}, s_{i-2}) P(w_i | s_i) \]

- Can keep increasing the order. What are trade-offs?
Need to estimate transition and emission matrices from data.

Trivial solution (maximum likelihood):

\[
\hat{P}(w_i = j | s_i = k) = \frac{\#[w_i = j, s_i = k]}{\sum_j \#[w_i = j, s_i = k]}
\]

\[
\hat{P}(s_i = l | s_{i-1} = k) = \frac{\#[s_i = l, s_{i-1} = k]}{\sum_k \#[s_i = l, s_{i-1} = k]}
\]

What is wrong with this?
Estimating Transitions

- We need to smooth!!! (Just like in language models)

\[ P(s_i \mid s_{i-1}, s_{i-2}) = \lambda_2 \hat{P}(s_i \mid s_{i-1}, s_{i-2}) + \lambda_1 \hat{P}(s_i \mid s_{i-1}) + (1 - \lambda_1 - \lambda_2) \hat{P}(s_i) \]

- Can get a lot fancier (e.g. KN smoothing) or use higher orders, but in this case it doesn’t buy much

- One option: encode more into the state, e.g. whether the previous word was capitalized (Brants 00)

- BIG IDEA: The basic approach of state-splitting turns out to be very important in a range of tasks
Estimating Emissions

- Emissions are trickier:
  - Words we’ve never seen before, or occur with tags we’ve never seen them with
  - Can blindly use add-one smoothing (or something similar).
  - **Suboptimal**: unknown words aren’t black boxes:

  343,127.23  11-year  Minteria  reintroducibly
Suffixes for Emission Estimation [Brants 00]

- Smoothing at the character level:

\[ \text{11-year} \]

\[ c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5 \quad c_6 \quad c_7 \]

- First compute

\[ P(s|c_1, \ldots, c_7) \]

- And then use Bayes rule to obtain:

\[ P(c_1, \ldots, c_7|s) = \frac{P(s|c_1, \ldots, c_7)P(c_1, \ldots, c_7)}{P(s)} \]
Suffixes for Emission Estimation [Brants 00]

- Smoothing with (recursive) interpolation:

\[
P_{sm}(s|c_1, \ldots, c_7) = \frac{\hat{P}(s|c_1, \ldots, c_7) + \kappa P_{sm}(s|c_2, \ldots, c_7)}{1 + \kappa}
\]

recursion
Why did we do suffixes instead of prefixes?

- Because suffixes tell us more about part of speech:

<table>
<thead>
<tr>
<th>adverbs!</th>
<th>verbs!</th>
</tr>
</thead>
<tbody>
<tr>
<td>hastily</td>
<td>running</td>
</tr>
<tr>
<td>quickly</td>
<td>flowing</td>
</tr>
<tr>
<td>slowly</td>
<td>living</td>
</tr>
<tr>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>angrily</td>
<td>jumping</td>
</tr>
</tbody>
</table>
Inference

- Problem: find the most likely (Viterbi) sequence under the model

\[ s^* = \arg\max_{s} \prod_{i} P(s | w) \]

- Can ignore normalizer

\[ s^* = \arg\max_{s} \prod_{i} P(s, w) \]

\[ s^* = \arg\max_{s} \prod_{i} P(s_i | s_{i-1}) P(w_i | s_i) \]
Brute Force

- Score all the tag sequences and take highest scoring one.

Fed
raises
interest
rates
0.5
percent

- Problem: Exponentially many tag sequences
Finding the Best Trajectory

- Option 1: Beam Search (Greedy)

- A beam is a set of partial hypotheses
- At each derivation step:
  - Consider all continuations of previous hypotheses
  - Discard most, keep top k, or those within a factor of the best

- Beam search works ok in practice
But in this case we don’t need it

- Dynamic programming (Viterbi algorithm)

- Recursion:

\[
\begin{align*}
    s^* &= \arg\max_{s_1, \ldots, s_n} \prod_{i=1}^{n} P(s_i | s_{i-1}) P(w_i | s_i) \\
    &= \arg\max_{s_n, s_{n-1}} P(s_n | s_{n-1}) P(w_n | s_n) \left( \arg\max_{s_1, \ldots, s_{n-2}} \prod_{i=1}^{n-1} P(s_i | s_{i-1}) P(w_i | s_i) \right)
\end{align*}
\]

- Dynamic programming = “bottom up” computation.
Dynamic Programming

\[ \delta(s_1) = P(w_1|s_1)P(s_1) \]

\[ \delta(s_2) = \max_{s_1} P(s_2|s_1)P(w_2|s_2)\delta(s_1) \]

\[ \cdots \]

\[ \max \text{ score} = \max_{s_n} \delta(s_n) \]
The State Lattice / Trellis

START       Fed           raises       interest       rates       END

\( P(N|\Lambda) \)

\( P(Fed|N) \)
The Viterbi Algorithm

- Dynamic program for computing

\[ \delta_i(s) = \max_{s_0...s_i} P(s_0...s_{i-1}s, w_1...w_{i-1}) \]

- The score of a best path up to position \( i \) ending in state \( s \)

\[ \delta_i(s) = \max_{s'} P(s \mid s') P(w \mid s') \delta_{i-1}(s') \]

- Also store a backtrace

\[ \psi_i(s) = \arg \max_{s'} P(s \mid s') P(w \mid s') \delta_{i-1}(s') \]
So How Well Does It Work?

- **TnT (Brants, 2000):**
  - A carefully smoothed trigram tagger
  - Suffix trees for emissions
  - 96.7% on WSJ text (SOA is ~97.5%)

- **Noise in the data**
  - Many errors in the training and test corpora

```
DT     NN     IN     NN   VBD   NNS   VBD
The average of interbank offered rates plummeted …
```

- Probably about 2% guaranteed error from noise (on this data)
## Overview: Accuracies

<table>
<thead>
<tr>
<th>Method</th>
<th>Known words</th>
<th>Unknown words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary Lookup - Most frequent tag</td>
<td>~90%</td>
<td>~50%</td>
</tr>
<tr>
<td>Features only based on word in question</td>
<td>~93.7%</td>
<td>~82.6%</td>
</tr>
<tr>
<td>Trigram HMM</td>
<td>~95%</td>
<td>~55%</td>
</tr>
<tr>
<td>TNT (Trigram HMM++)</td>
<td>~ 97%</td>
<td>~86%</td>
</tr>
</tbody>
</table>
Part-of-Speech Tagging Accuracy (POS)

Baseline = StanfordTagger v2.0 [Manning '11]
Common In-Domain Errors

- **Common errors [from Toutanova & Manning 00]**

<table>
<thead>
<tr>
<th></th>
<th>JJ</th>
<th>NN</th>
<th>NNP</th>
<th>NNPS</th>
<th>RB</th>
<th>RP</th>
<th>IN</th>
<th>VB</th>
<th>VBD</th>
<th>VBN</th>
<th>VBP</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>JJ</td>
<td>0</td>
<td>177</td>
<td>56</td>
<td>0</td>
<td>61</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>108</td>
<td>0</td>
<td>488</td>
</tr>
<tr>
<td>NN</td>
<td>244</td>
<td>0</td>
<td>103</td>
<td>0</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td>29</td>
<td>5</td>
<td>6</td>
<td>19</td>
<td>525</td>
</tr>
<tr>
<td>NNP</td>
<td>107</td>
<td>106</td>
<td>0</td>
<td>132</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>427</td>
</tr>
<tr>
<td>NNPS</td>
<td>1</td>
<td>0</td>
<td>110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>142</td>
</tr>
<tr>
<td>RB</td>
<td>72</td>
<td>21</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>138</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>295</td>
</tr>
<tr>
<td>RP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>39</td>
<td>0</td>
<td>65</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>104</td>
</tr>
<tr>
<td>IN</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>169</td>
<td>103</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>323</td>
</tr>
<tr>
<td>VB</td>
<td>17</td>
<td>64</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>85</td>
<td>189</td>
</tr>
<tr>
<td>VBD</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>143</td>
<td>0</td>
<td>166</td>
</tr>
<tr>
<td>VBN</td>
<td>101</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>108</td>
<td>0</td>
<td>1</td>
<td>221</td>
</tr>
<tr>
<td>VBP</td>
<td>5</td>
<td>34</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>49</td>
<td>6</td>
<td>3</td>
<td>104</td>
</tr>
<tr>
<td>Total</td>
<td>626</td>
<td>536</td>
<td>348</td>
<td>144</td>
<td>317</td>
<td>122</td>
<td>279</td>
<td>102</td>
<td>140</td>
<td>269</td>
<td>108</td>
<td>3651</td>
</tr>
</tbody>
</table>

**predicted**

- **gold**
- **leading official**
- **recently sold shares**

43
Sequence-Free Tagging?

- What about looking at a word and its environment, but no sequence information?
  - Add in previous / next word the __
  - Previous / next word shapes X __ X
  - Occurrence pattern features [X: x X occurs]
  - Crude entity detection __ ..... (Inc.|Co.)
  - Conjunctions of these things

- All features except sequence: 96.6% / 86.8%
- Uses lots of features: > 200K
- What is the disadvantage of this approach?
Why Linear Context is Useful

- Lots of rich local information!

```
RB
PRP VBD IN RB IN PRP VBD .
They left as soon as he arrived .
```

- We could fix this with a feature that looked at the next word

```
JJ
NNP NNS VBD VBN .
Intrinsic flaws remained undetected .
```

- We could fix this by linking capitalized words to their lowercase versions

- Solution: discriminative sequence models (MEMMs, CRFs)
MEMM Taggers

- One step up: also condition on previous tags

\[ P(s|w) = \Pi_i P_{ME}(s_i|w, s_{i-1}, s_{i-2}) \]

- This is referred to as an MEMM tagger [Ratnaparkhi 96]
- Beam search effective! (Why?)
- What’s the advantage of beam size 1?

- Natural extension of MaxEnt: neural net version!
Decoding maxent taggers:
- Just like decoding HMMs
- Viterbi, beam search, posterior decoding

Viterbi algorithm (HMMs):

$$\delta_i(s) = \arg \max_{s'} P(s|s')P(w_{i-1}|s')\delta_{i-1}(s')$$

Viterbi algorithm (Maxent):

$$\delta_i(s) = \arg \max_{s'} P(s|s', w)\delta_{i-1}(s')$$
Overview

- Part-of-Speech Tagging:
  - First Step of Syntactic Analysis
  - Hidden Markov Models
  - Supervised Accuracy:
    - In-Domain: >97%
    - Out-of-Domain: <90%

- Next Class:
  - Conditional Random Fields
  - Unsupervised Techniques