Many thanks to David Weiss, Slav Petrov, Richard Socher, Chris Manning, Kevin Duh for starting point for slides
Learning in Neural Networks

- As long as the activation functions are differentiable, it is possible to compute gradient via chain rule (Backpropagation).

\[ z = F_1(W_1(F_2(.....F_n(W_n[a; b] + b_n).... + b_2) + b_1) + b_1) \]

- Can then use gradient descent to optimize the objective.
- Existing computing libraries like Tensorflow / Torch automatically compute the gradient.
- However, not guaranteed optimal solution, since problem is highly non-convex.
Learning in Neural Nets

• Typical strategy is a stochastic optimization technique:
  • Instead of computing gradient over whole data, compute only gradient over a small “minibatch”
  • Speeds up training, and the added randomness helps get out of local optima

• Other common optimizers are:
  • Adagrad [Duchi et al. 2011]
  • ADAM [Kingma and Ba, 2014]
  • AMSGrad [Reddi et al. 2018]
  and many more!
Dropout Regularization [Srivastava et al. 2014]

• Neural nets overfit very easily.

• Empirically, often better to train a massive neural network with lots of regularization than a smaller neural network

• Dropout [Srivastava et al. 2014] is a very popular regularization technique
  • Dropout random neurons in training
  • (Use all the neurons in test)
  • Intuitively some sort of model averaging
Dropout Regularization [Srivastava et al. 2014]

![Diagram of Dropout Neural Net Model](image)

(a) Standard Neural Net  (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

![Diagram of At Training Time and At Test Time](image)

(a) At training time  (b) At test time

Figure 2: **Left:** A unit at training time that is present with probability $p$ and is connected to units in the next layer with weights $w$. **Right:** At test time, the unit is always present and the weights are multiplied by $p$. The output at test time is same as the expected output at training time.
(Equivalent formulation as in Srivastava et al. 2014, just described slightly differently)

Assume simple model:

\[ y^{(m)} = \text{ReLU}(Wx^{(m)} + b) \]

\[ x^{(m)} = \begin{pmatrix} 0.35 \\ -0.2 \\ 0.45 \\ 0.7 \end{pmatrix} \]

\((m)\) indexes over examples
Dropout Regularization - Training Time
[Srivastava et al. 2014]

(Equivalent formulation as in Srivastava et al. 2014, just described slightly differently)

For each example, randomly sample a dropout mask (assume the keep probability p=0.5) and reweight surviving elements by 1/p.

\[
d^{(m)} = \begin{pmatrix} 1/p \\ 0 \\ 0 \\ 1/p \end{pmatrix} = \begin{pmatrix} 1/0.5 \\ 0 \\ 0 \\ 1/0.5 \end{pmatrix}
\]

Multiply input to hidden layer by dropout mask before multiplying by weights in training.

\[
\tilde{x}^{(m)} = x \circ d^{(m)} = \begin{pmatrix} 0.7 \\ 0 \\ 0 \\ 1.4 \end{pmatrix}
\]

\[
\tilde{y}^{(m)} = \text{ReLU}(W\tilde{x}^{(m)} + b)
\]
Dropout Regularization - Test Time
[Srivastava et al. 2014]

(Equivalent formulation as in Srivastava et al. 2014, just described slightly differently)

Use network as it is without any dropout

\[ y = \text{ReLU}(Wx + b) \]
Dropout Regularization [Srivastava et al. 2014]

Dropout

setting that do not use dropout or unsupervised pretraining achieve an error of about 1.60% (Simard et al., 2003). With dropout the error reduces to 1.35%. Replacing logistic units with rectified linear units (ReLUs) (Jarrett et al., 2009) further reduces the error to 1.25%. Adding max-norm regularization again reduces it to 1.06%. Increasing the size of the network leads to better results. A neural net with 2 layers and 8192 units per layer gets down to 0.95% error. Note that this network has more than 65 million parameters and is being trained on a data set of size 60,000. Training a network of this size to give good generalization error is very hard with standard regularization methods and early stopping. Dropout, on the other hand, prevents overfitting, even in this case. It does not even need early stopping. Goodfellow et al. (2013) showed that results can be further improved to 0.94% by replacing ReLU units with maxout units. All dropout nets use $p = 0.5$ for hidden units and $p = 0.8$ for input units. More experimental details can be found in Appendix B.1.

Dropout nets pretrained with stacks of RBMs and Deep Boltzmann Machines also give improvements as shown in Table 2. DBM—pretrained dropout nets achieve a test error of 0.79% which is the best performance ever reported for the permutation invariant setting. We note that it possible to obtain better results by using 2-D spatial information and augmenting the training set with distorted versions of images from the standard training set. We demonstrate the effectiveness of dropout in that setting on more interesting data sets.

Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.
## Parts-of-Speech (English)

- One basic kind of linguistic structure: syntactic word classes

<table>
<thead>
<tr>
<th>Open class (content) words</th>
<th>Closed class (functional) words</th>
<th>Abbreviations</th>
<th>Verbs</th>
<th>Modal</th>
<th>Punctuation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nouns</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cat, dog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proper</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM, John</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Adjectives</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Red, happy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Adverbs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quickly</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Determiners</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The, some</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Conjunctions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>And, or</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Pronouns</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>They, him</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Numbers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One, thousand, 1,983,213</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Adpositions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In, of, from</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Particles</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Off, up</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Abbreviations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Verbs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ran, ate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Modal</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can, had</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Punctuation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>., ?, !</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Parts-of-Speech (German)

- One basic kind of linguistic structure: syntactic word classes

<table>
<thead>
<tr>
<th>Open class (content) words</th>
<th>Closed class (functional) words</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nouns</strong></td>
<td><strong>Determiners</strong></td>
</tr>
<tr>
<td>Common</td>
<td><strong>Conjunctions</strong></td>
</tr>
<tr>
<td>Katze, Hund</td>
<td>die, einige</td>
</tr>
<tr>
<td>Proper</td>
<td><strong>Pronouns</strong></td>
</tr>
<tr>
<td>IBM, Jan</td>
<td>sie, ihm</td>
</tr>
<tr>
<td><strong>Adjectives</strong></td>
<td><strong>Adpositions</strong></td>
</tr>
<tr>
<td>rot, froh</td>
<td>in, aus, von</td>
</tr>
<tr>
<td><strong>Verb</strong></td>
<td><strong>Modal</strong></td>
</tr>
<tr>
<td><strong>Adverbs</strong></td>
<td>kann, hat</td>
</tr>
<tr>
<td>schnell</td>
<td><strong>Numbers</strong></td>
</tr>
<tr>
<td><strong>Abbreviations</strong></td>
<td>eins, Tausend, 1.983.213</td>
</tr>
<tr>
<td><strong>etc.</strong></td>
<td><strong>Particles</strong></td>
</tr>
<tr>
<td><strong>Punctuation</strong></td>
<td>aus, an</td>
</tr>
<tr>
<td>., ?, !</td>
<td></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.
Accuracy So Far

<table>
<thead>
<tr>
<th>Dictionary Lookup - Most frequent tag</th>
<th>Known words</th>
<th>Unknown words</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>~90%</td>
<td>~50%</td>
</tr>
</tbody>
</table>
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  VBN  VB  VBP  VBZ  NNP  NNS  NN  NNS  CD  NN

- Two basic sources of constraints:
  - Grammatical environment
  - Identity of the current word
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

\[
\begin{align*}
\text{\textless \textcircled{\text{\textdagger}}} & \quad \text{\textless t}_1 \text{\textgreater} & \quad \text{\textless t}_2 \text{\textgreater} & \quad \text{\textless t}_n \text{\textgreater} \\
 s_0 & \quad s_1 & \quad s_2 & \quad s_n \\
 \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow \\
 w_1 & \quad w_2 & \quad w_n
\end{align*}
\]
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:

1 1 - y e a r

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

- 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:

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

\[\text{recursion}\]
Why did we do suffixes instead of prefixes?

- Because suffixes tell us more about part of speech:

  **adverbs!**  
  hastily  
  quickly  
  slowly  
  .....  
  angrily

  **verbs!**  
  running  
  flowing  
  living  
  .....  
  jumping
Inference

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

\[ s^* = \arg\max_s \Pi_i P(s|w) \]

- Can ignore normalizer

\[ s^* = \arg\max_s \Pi_i P(s, w) \]
\[ s^* = \arg\max_s \Pi_i P(s_i|s_{i-1}) P(w_i|s_i) \]
Brute Force

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

<table>
<thead>
<tr>
<th>NNP</th>
<th>VBZ</th>
<th>VB</th>
<th>NNS</th>
<th>CD</th>
<th>NN</th>
<th>8.0e-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP</td>
<td>NNS</td>
<td>NN</td>
<td>NNS</td>
<td>CD</td>
<td>NN</td>
<td>3.1e-6</td>
</tr>
<tr>
<td>NNP</td>
<td>VBZ</td>
<td>NN</td>
<td>NNS</td>
<td>CD</td>
<td>NN</td>
<td>3.5e-6</td>
</tr>
</tbody>
</table>

Fed raises interest rates 0.5 percent.

- Problem: Exponentially many tag sequences
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:

\[
s^* = \arg\max_{s_1, \ldots, s_n} \prod_{i=1}^{n} P(s_i | s_{i-1}) P(w_i | s_i)
\]

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

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

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

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

\[ \vdots \]

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

START       Fed           raises       interest       rates       END
The Viterbi Algorithm

- Dynamic program for computing

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

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

\[ \delta_i(s_i) = \max_{s_{i-1}} P(s_i|s_{i-1})P(w_i|s_i)\delta_{i-1}(s_{i-1}) \]

- Also store a backtrace

\[ \psi_i(s_i) = \arg \max_{s_{i-1}} P(s_i|s_{i-1})P(w_i|s_i)\delta_{i-1}(s_{i-1}) \]
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

The average of interbank offered rates plummeted …

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

<table>
<thead>
<tr>
<th></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>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>
Supervised POS Accuracies

![Bar chart showing supervised POS accuracies for various languages](chart.png)
Parsing the Web Shared Task Results

Part-of-Speech Tagging Accuracy (POS)

Baseline \(=\) StanfordTagger v2.0 [Manning '11]

<table>
<thead>
<tr>
<th>Team</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>90.8</td>
</tr>
<tr>
<td>UPenn</td>
<td>89.9</td>
</tr>
<tr>
<td>UMass</td>
<td>90.2</td>
</tr>
<tr>
<td>NAIST</td>
<td>90.6</td>
</tr>
<tr>
<td>IMS-2</td>
<td>89.9</td>
</tr>
<tr>
<td>IMS-3</td>
<td>88.7</td>
</tr>
<tr>
<td>CPH-Trento</td>
<td>90.9</td>
</tr>
<tr>
<td>Stanford-2</td>
<td>90.5</td>
</tr>
<tr>
<td>HIT-Baseline</td>
<td>90.8</td>
</tr>
<tr>
<td>HIT-Domain</td>
<td>91.3</td>
</tr>
<tr>
<td>Stanford-1</td>
<td>91.3</td>
</tr>
<tr>
<td>DCU-Pairs13</td>
<td>90.8</td>
</tr>
<tr>
<td>DCU-Pairs13</td>
<td>92.9</td>
</tr>
</tbody>
</table>
## Common In-Domain Errors

### Common errors [from Toutanova & Manning 00]

<table>
<thead>
<tr>
<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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>2</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>
</tr>
<tr>
<td>VBP</td>
<td>5</td>
<td>34</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>49</td>
<td>6</td>
<td>3</td>
<td>0</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>
</tr>
</tbody>
</table>

- **Predicted**: The table shows the predicted part-of-speech tagging errors.
- **Gold**:
  - Leading official: VBG/NN/JJ
  - Recently sold shares: RB/VBD/VBN/NNS
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!

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

- 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

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

- 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

- Decoding maxent taggers:
  - Just like decoding HMMs
  - Viterbi, beam search, posterior decoding
- Viterbi algorithm (HMMs):
  \[ \delta_i(s_i) = \max_{s_{i-1}} P(s_i|s_{i-1})P(w_i|s_i)\delta_{i-1}(s_{i-1}) \]
- Viterbi algorithm (Maxent):
  \[ \delta_i(s_i) = \max_{s_{i-1}} P(s_i|s_{i-1}, w)\delta_{i-1}(s_{i-1}) \]
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