Recall: Multiclass Classification

- Logistic regression: \[ P(y|x) = \frac{\exp(w^T f(x, y))}{\sum_{y' \in \mathcal{Y}} \exp(w^T f(x, y'))} \]
  
  Gradient (unregularized): \[ \frac{\partial}{\partial w_i} \mathcal{L}(x_j, y_j^*) = f_i(x_j, y_j^*) - \mathbb{E}_y[f_i(x_j, y)] \]

- SVM: defined by quadratic program (minimization, so gradients are flipped)
  Loss-augmented decode
  \[ \xi_j = \max_{y \in \mathcal{Y}} w^T f(x_j, y) + \ell(y, y_j^*) - w^T f(x_j, y_j^*) \]

  Subgradient (unregularized) on \( j \)th example \( = f_i(x_j, y_{\max}) - f_i(x_j, y_j^*) \)

Structured Prediction

- Four elements of a structured machine learning method:
  - Model: probabilistic, max-margin, deep neural network

  Objective

  Inference: just maxes and simple expectations so far, but will get harder

  Training: gradient descent
Optimization

- Stochastic gradient ascent
  - Very simple to code up
  - “First-order” technique: only relies on having gradient
  - Difficult to tune step size
- Newton’s method
  - Second-order technique
  - Optimizes quadratic instantly
  - Inverse Hessian: \( n \times n \) mat, expensive!
- Quasi-Newton methods: L-BFGS, etc.
  - Approximate inverse Hessian with gradients over time

\[
\frac{\partial^2 L}{\partial w^2} \quad g \quad w \leftarrow w + \alpha g, \quad g = \frac{\partial}{\partial w} L
\]

AdaGrad

- Optimized for problems with sparse features
- Sparse features are often heterogeneous: some fire on every example, some fire on one example in the corpus (but are still valuable!)
  \[
  w_i \leftarrow w_i + \frac{1}{\sqrt{\epsilon + \sum_{\tau=1}^{t} g_{\tau,i}^2}} \cdot \frac{g_{t,i}}{g_{t,i}} \quad \text{per-parameter learning rate based on sum of previous gradients}
  \]
- Avoids common features getting large values compared to rare features
- Usually works out-of-the-box with little tuning
- Other techniques for optimizing deep models — more later!

Duchi et al. (2011)

Implementation Details

- SGD/AdaGrad have a batch size parameter
  - Large batches (>50 examples): can parallelize within batch
    - ...but bigger batches often means more epochs required because you make fewer parameter updates
  - Shuffling: online methods are sensitive to dataset order
    - Fixed shuffle: breaks correlations between neighboring sentences
    - Per-epoch shuffle: lower final model variance
  - Regularization: makes SGD slower to implement with sparse features
    - Either don’t regularize (might work better than you think!), or do it lazily (see adagrad_trainer.py in Project 1)

This Lecture

- Sequence modeling
- HMMs for POS tagging
- HMM parameter estimation
- Viterbi algorithm
Linguistic Structures

- Language is tree-structured
  - I ate the spaghetti with chopsticks
  - I ate the spaghetti with meatballs

- Understanding syntax fundamentally requires trees — the sentences have the same shallow analysis

```
PRP VBZ DT NN IN NNS  PRP VBZ DT NN IN NNS
I ate the spaghetti with chopsticks  I ate the spaghetti with meatballs
```

Linguistic Structures

- Language is sequentially structured: interpreted in an online way

```
Tanenhaus et al. (1995)
```

POS Tagging

<table>
<thead>
<tr>
<th>Open class (lexical) words</th>
<th>Closed class (functional)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nouns</td>
<td>Determiners the some</td>
</tr>
<tr>
<td>Proper IBM Italy</td>
<td>Conjunctions and or</td>
</tr>
<tr>
<td>Common cat / cats snow</td>
<td>Pronouns he its</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>Structures</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRP VBZ DT NN IN NNS</td>
<td>PRP VBZ DT NN IN NNS</td>
</tr>
<tr>
<td>I ate the spaghetti with chopsticks</td>
<td>I ate the spaghetti with meatballs</td>
</tr>
</tbody>
</table>
```

Fed raises interest rates 0.5 percent

- Other paths are also plausible but even more semantically weird...
- What governs the correct choice? Word + context
  - Word identity: most words have <=2 tags, many have one (percent, the)
  - Context: nouns start sentences, nouns follow verbs, etc.
### What is this good for?
- Text-to-speech: record, lead
- Preprocessing step for syntactic parsers
- Domain-independent disambiguation for other tasks
- (Very) shallow information extraction

### Sequence Models
- **Input** $x = (x_1, ..., x_n)$  
  **Output** $y = (y_1, ..., y_n)$
- POS tagging: $x$ is a sequence of words, $y$ is a sequence of tags (most of the time...)
- Today: generative models $P(x, y)$; discriminative models next time

### Hidden Markov Models
- **Input** $x = (x_1, ..., x_n)$  
  **Output** $y = (y_1, ..., y_n)$
- Model the sequence of $y$ as a Markov process (dynamics model)
- Markov property: future is conditionally independent of the past given the present

\[
P(y_3|y_1, y_2) = P(y_3|y_2)
\]
- Lots of mathematical theory about how Markov chains behave
- If $y$ are tags, this roughly corresponds to assuming that the next tag only depends on the current tag, not anything before

### Hidden Markov Models
- **Input** $x = (x_1, ..., x_n)$  
  **Output** $y = (y_1, ..., y_n)$
- Observation ($x$) depends only on current state ($y$)
- Multinomials: tag $x$ tag transitions, tag $x$ word emissions
- $P(x|y)$ is a distribution over all words in the vocabulary — not a distribution over features
Transitions in POS Tagging

- Dynamics model:
  \[ P(y_1) \prod_{i=2}^{n} P(y_i | y_{i-1}) \]
  Fed raises interest rates 0.5 percent
  - Should \( y \) be a single tag?
  - Trigram model: \( y_1 = (<S>, \text{NNP}), y_2 = (\text{NNP}, \text{VBZ}), ... \)
  - \( P(\text{VBZ}, \text{NN}) | (\text{NNP}, \text{VBZ}) \) — more context! Noun-verb-noun S-V-O
  - Tradeoff between model capacity and data size

- \( P(y_1 = \text{NNP}) \) likely because start of sentence
- \( P(y_2 = \text{VBZ} | y_1 = \text{NNP}) \) likely because verb often follows noun
- \( P(y_3 = \text{NN} | y_2 = \text{VBZ}) \) direct object follows verb, other verb rarely follows past tense verb (main verbs can follow modals though!)

Estimating Transitions

- Similar to Naive Bayes estimation: maximum likelihood solution = normalized counts (with smoothing) read off supervised data
- \( P(\text{tag} | \text{NN}) = (0.5 <S>, 0.5 \text{NNS}) \)
- How to smooth?
  - One method: smooth with unigram distribution over tags
    \[ P(\text{tag} | \text{tag}_{-1}) = (1 - \lambda) \hat{P}(\text{tag} | \text{tag}_{-1}) + \lambda \hat{P}(\text{tag}) \]
    \( \hat{P} \) = empirical distribution (read off from data)

Emissions in POS Tagging

- Emissions \( P(x | y) \) capture the distribution of words occurring with a given tag
  - \( P(\text{word} | \text{NN}) = (0.05 \text{ person}, 0.04 \text{ official}, 0.03 \text{ government}, 0.03 \text{ market} ...) \)
- When you compute the posterior for a given word’s tags, the distribution favors tags that are more likely to generate that word
Estimating Emissions

- **Fed raises interest rates 0.5 percent**
  - P(word | NN) = (0.5 interest, 0.5 percent) — hard to smooth!

- Can interpolate with distribution looking at word shape
  - P(word shape | tag) (e.g., P(capitalized word of len >= 8 | tag))

- Alternative: use Bayes’ rule
  \[
P(\text{word} | \text{tag}) = \frac{P(\text{tag} | \text{word}) P(\text{word})}{P(\text{tag})}
\]

- Fancy techniques from language modeling, e.g. look at type fertility — P(tag | word) is flatter for some kinds of words than for others

- P(word | tag) can be a log-linear model — we’ll see this in a few lectures

Inference in HMMs

- **Input** \(x = (x_1, …, x_n)\)  
  **Output** \(y = (y_1, …, y_n)\)

- Inference problem: \(\arg\max_y P(y | x) = \arg\max_y \frac{P(y, x)}{P(x)}\)

- Exponentially many possible \(y\) here!

- Solution: dynamic programming (possible because of Markov structure!)

- Many neural sequence models depend on entire previous tag sequence, need to use approximations like beam search
Forward-Backward Algorithm

- Compute marginal distributions $P(y_i = s | x)$
- Replace max with + everywhere, also run backward pass
  $\text{forward}_2(s) \text{backward}_2(s) = P(x, y_2 = s)$
  $P(y_2 = s | x) \propto \text{forward}_2(s) \text{backward}_2(s)$ ← i.e. normalize by $P(x)$
- Be careful not to double-count $P(x_2 | y_2)$ when combining these!
- Store everything as log probabilities to avoid underflow

1. Initial: For each state $s$, calculate
   \[ \text{score}_1(s) = P(s) P(x_1 | s) = \pi_s B_{x_1,s} \]

2. Recurrence: For $i = 2$ to $n$, for every state $s$, calculate
   \[ \text{score}_i(s) = \max_{y_{i-1}} P(s | y_{i-1}) P(x_i | s) \text{score}_{i-1}(y_{i-1}) = \max_{y_{i-1}} A_{y_{i-1}, s} B_{s, x_i} \text{score}_{i-1}(y_{i-1}) \]

3. Final state: calculate
   \[ \max_y P(y, x | \pi, A, B) = \max_s \text{score}_n(s) \]

This only calculates the max. To get final answer (argmax),
- keep track of which state corresponds to the max at each step
- build the answer using these back pointers

HMM POS Tagging

- Most frequent tag: ~90% accuracy
- Trigram HMM: ~95% accuracy / 55% on unknown words
- TnT tagger (tuned) HMM: 96.2% accuracy / 86.0% on unknown words
- Logistic regression $P(t | w)$: 93.7% / 82.6% (*only* at current word)
- State-of-the-art (BiLSTM-CRFs): 97.5% / 89%+
Errors

<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>VBD</th>
<th>VBN</th>
<th>VBP</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>JJ</td>
<td>177</td>
<td>56</td>
<td>25</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>108</td>
<td>488</td>
</tr>
<tr>
<td>NN</td>
<td>244</td>
<td>103</td>
<td>132</td>
<td>110</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>129</td>
</tr>
<tr>
<td>NNP</td>
<td>107</td>
<td>106</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>137</td>
</tr>
<tr>
<td>NNPS</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>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>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>0</td>
<td>65</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>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>323</td>
</tr>
<tr>
<td>VB</td>
<td>17</td>
<td>64</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</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>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>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>
</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>
</tr>
</tbody>
</table>

Remaining Errors

- Lexicon gap (word not seen with that tag in training) 4.5%
- Unknown word: 4.5%
- Could get right: 16% (many of these involve parsing!)
- Difficult linguistics: 20%

They set up absurd situations, detached from reality

Underspecified / unclear, gold standard inconsistent / wrong: 58%

Other Languages

- Universal POS tagset (~12 tags), cross-lingual model works as well as tuned CRF using external resources

Next Time

- CRFs: feature-based discriminative models
- Structured SVM for sequences
- NER