Natural Language Processing

Tagging part II

Based on slides from Michael Collins
Welcome back!

• Hope you enjoyed elections + passover

• Last time we started to talk about tagging
Part-of-speech tagging

**Input:**
Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

**Output:**
Profits/N soared/V at/P Boeing/N Co./N ,/, easily/Adv topping/V forecasts/N on/P Wall/N Street/N ,/, as/P their/POSS CEO/N Alan/N Mulally/N announced/V first/Adj quarter/N results/N ./.  

N: noun  
V: verb  
P: preposition  
Adv: adverb  
Adj: adjective  
...
NER tagging (reminder)

**Input:**
Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

**Output:**
Profits/O soared/O at/O Boeing/B-org Co./I-org ,/O easily/O topping/O forecasts/O on/O Wall/B-loc Street/I-loc ,/O as/O their/O CEO/O Alan/B-per Mulally/I-per announced/O first/O quarter/O results/O.
HMMs (Reminder)

\[ p(x_1, \ldots, x_n, y_1 \ldots, y_n, \text{STOP}) = \]

\[ q(\text{STOP} \mid y_{n-1}, y_n) \times \prod_{i=1}^{n} q(y_i \mid y_{i-2}, y_{i-1}) \cdot e(x_i \mid y_i) \]

\[ y_0 = y_{-1} = * \]
Learning (Reminder)

\[ q(y_i \mid y_{i-2}, y_{i-1}) = \lambda_1 \cdot \frac{c(y_{i-2}, y_{i-1}, y_i)}{c(y_{i-2}, y_{i-1})} + \]
\[ \lambda_2 \cdot \frac{c(y_{i-1}, y_i)}{c(y_{i-1})} + \]
\[ \lambda_3 \cdot \frac{c(y_i)}{M} \]

\[ \sum_i \lambda_i = 1, \quad \lambda_i \geq 0 \]

\[ e(x \mid y) = \frac{c(x, y)}{c(y)} \]
Viterbi Decoding (Reminder)

Input: a sentence $x_1, \ldots, x_n$, parameters $q, e$, and tag set $S$

Base case: $\pi(0, \ast, \ast) = 1$

Definition: $S_{-1} = S_0 = \{\ast\}, S_k = S$ for $k \in \{1 \ldots n\}$

Algorithm:

for $k \in \{1 \ldots n\}, u \in S_{k-1}, v \in S_k$:

$$\pi(k, u, v) = \max_{w \in S_{k-2}} \pi(k-1, w, u) \times q(v | w, u) \times e(x_k | v)$$

$$bp(k, u, v) = \arg \max_{w \in S_{k-2}} \pi(k-1, w, u) \times q(v | w, u) \times e(x_k | v)$$

$$(y_{n-1}, y_n) = \arg \max_{u, v} (\pi(n, u, v) \times q($$STOP$ | u, v)$$

for $k = (n - 2) \ldots 1, y_k = bp(k + 2, y_{k+1}, y_{k+2})$

return $y_1, \ldots, y_n$
Log-linear “feature-rich” models (Reminder)

• **Input**: domain $X$, Input label set $Y$

  • $(x,y)$ is not (sentence, tag sequence), but (history, label)

• **Feature function**:
  \[ f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^m \]

• **Parameters**:
  \[ \theta \in \mathbb{R}^m \]

• **Output**:
  \[ p(y \mid x) = \frac{e^{f(x,y) \top \theta}}{\sum_{y' \in \mathcal{Y}} e^{f(x,y') \top \theta}} \]
Parameter estimation in log-linear models
Maximum likelihood

• Input: training examples

\[ \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}, \quad (x^{(i)}, y^{(i)}) \in \mathcal{X} \times \mathcal{Y} \]

• Objective function:

\[
L(\theta) = \sum_{i=1}^{n} \log p_\theta(y^{(i)} | x^{(i)}) = \sum_{i=1}^{n} f(x^{(i)}, y^{(i)})^\top \theta - \sum_{i=1}^{n} \log \sum_{y' \in \mathcal{Y}} e^{f(x^{(i)}, y')^\top \theta}
\]

\[ \theta_{\text{ML}} = \arg \max_\theta L(\theta) \]
Optimization

- L is a concave function (sum of concave functions)
- Can find optimal point with gradient-based methods
Computing the gradient

• Let’s look at one example

\[
\frac{\partial L^{(i)}(\theta)}{\theta_k} = f_k(x^{(i)}, y^{(i)}) - \frac{1}{\sum_{z' \in \mathcal{Y}} e^{f_k(x^{(i)}, z')^\top \theta}} \cdot \frac{\partial (\sum_{y' \in \mathcal{Y}} e^{f_k(x^{(i)}, y')^\top \theta})}{\partial \theta_k} \\
= f_k(x^{(i)}, y^{(i)}) - \frac{\sum_{y' \in \mathcal{Y}} e^{f_k(x^{(i)}, y')^\top \theta} \cdot f_k(x^{(i)}, y')}{\sum_{z' \in \mathcal{Y}} e^{f_k(x^{(i)}, z')^\top \theta}} \\
= f_k(x^{(i)}, y^{(i)}) - \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') \cdot \frac{e^{f_k(x^{(i)}, y')^\top \theta}}{\sum_{z' \in \mathcal{Y}} e^{f_k(x^{(i)}, z')^\top \theta}} \\
= f_k(x^{(i)}, y^{(i)}) - \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') \cdot p_\theta(y' | x) \\
= f_k(x^{(i)}, y^{(i)}) - \mathbb{E}_{y' \sim p_\theta} [f_k(x, y')] \\
\]

• Add empirical count and subtract expected counts under the model
Regularization

- Say the word “base” appears exactly twice in the training data and it is tagged as Vt in both times

\[ f_{100}(x, y) = \begin{cases} 
1 & \text{if } w_i = \text{“base”} \text{ and } y = \text{Vt} \\
0 & \text{otherwise}
\end{cases} \]

- The gradient with respect to \( \theta_{100} \) are zero when:

\[
\sum_{i=1}^{n} f_{100}(x^{(i)}, y^{(i)}) = \sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} p_{\theta}(y' \mid x) \cdot f_{100}(x^{(i)}, y')
\]

- This happens when \( \theta_{100} \rightarrow \infty \): will not affect any other training example ==\> bad generalization

- We will always tag base as Vt for any history
L2 regularization

\[ L(\theta) = \sum_{i=1}^{n} \log p_{\theta}(y^{(i)} | x^{(i)}) + \lambda \cdot ||\theta||^2 \]

- Penalize large parameter values
- Empirically this allows using millions of features
Example for language modeling

• With n-gram features with no regularization:
  • The solution is exactly the ML estimate we saw in generative models

• With regularization
  • Empirical performance is at least as good as discounting methods
Log-linear model for tagging

• Input is a sentence $w$ and label is a tag sequence $t$

• We define a log-linear model $p(t_i \mid w, t_{-1}...t_{i-1})$

\[
p(t_1, \ldots, t_n \mid w_1, \ldots, w_n, t_{-1}, t_0) = p(t_1 \mid t_{-1}, t_0, w_1, \ldots, w_n) \\
\quad \times p(t_2 \mid t_{-1}, t_0, t_1, w_1, \ldots, w_n) \\
\quad \vdots \\
\quad \times \ldots p(t_n \mid t_{-1}, \ldots, t_{n-1}, w_1, \ldots, w_n) \\
= \prod_{i=1}^{n} p(t_i \mid t_{-1}, \ldots, t_{i-1}, w_1, \ldots, w_n)
\]
Independence assumption

• Markov assumption again:

\[ p(t_1, \ldots, t_n \mid w_1, \ldots, w_n, t_{-1}, t_0) = \prod_{i=1}^{n} p(t_i \mid t_{-1}, \ldots, t_{i-1}, w_1, \ldots, w_n) \]

\[ = \prod_{i=1}^{n} p(t_i \mid t_{i-2}, t_{i-1}, w_1, \ldots, w_n) \]

• Question: what will we lose if we don’t make this assumption?
  
  • Answer: exact decoding

• This is called a **locally normalized** log-linear model

• Why?
Example

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/?? from which Spain expanded its empire into the rest of the Western Hemisphere.

• A history $x$ is a 4-tuple $<t_{i-2}, t_{i-1}, w, i>$

• A label $y$ is $t_i$.
  
  \[
  \begin{align*}
  t_{i-2} & : \text{DT} \\
  t_{i-1} & : \text{JJ} \\
  w & : \text{Hispaniola \ldots \ Hemisphere} \\
  i & : 6 \\
  t_i & : \text{NN}
  \end{align*}
  \]
Training

• Given training set of sentences and tag sequences $w^{(j)}, t^{(j)} \ j = 1...n$

• Build a training set of the form $x^{(i)}, y^{(i)}$ by constructing all history/label pairs.

• Maximize the L2-regularized maximum-likelihood estimate

• Gradient is the difference between the empirical expected feature vector and model expected feature vector

\[
L(\theta) = \sum_{i=1}^{n} \log p_{\theta}(y^{(i)} \mid x^{(i)}) + \lambda \cdot \|\theta\|^2
\]
Decoding
Greedy Decoding

• Greedy decoding

  • For $i = 1 \ldots n$

    • Choose $t_i$ with maximal probability

• This failed in generative models, but works pretty well here. Why?

  • Because the model is more expressive

• Complexity?

• Can we drop some independence assumptions?
Beam search

- Beam search
  - Keep a set of $K$ current hypotheses
  - For $i = 1 \ldots n$
    - Consider all continuations $t_i$ of current set of hypotheses $t_1 \ldots t_{i-1}$ and score them
  - Keep the top $K$

Fed raises interest rates 0.5 percent
Viterbi

• After training the log-linear model we obtain:

\[ p_{\theta}(y \mid x) = p_{\theta}(t_i \mid t_{i-2}, t_{i-1}, w, i) \]

• Our goal in decoding is to solve:

\[ \arg \max_{t_1, \ldots, t_n} \prod_{i=1}^{n} p(t_i \mid t_{i-2}, t_{i-1}, w, i) \]

• Solution: Features depend on local hidden variables and so we can use Viterbi
Viterbi

• Define the probability of a tag prefix

\[ r(t_1, \ldots, t_k) = \prod_{i=1}^{k} p_{\theta}(t_i | t_{i-2}, t_{i-1}, w, i) \]

• Define the dynamic programming chart/table:

  • Maximum probability for a tag sequence of length \( k \) ending in \( u, v \)

\[ \pi(k, u, v) = \max_{t_1, \ldots, t_{k-2}} r(t_1, \ldots, t_k) \]
Viterbi

**Definition:** $S_k$ is the set of possible tags in position

**Base:** $\pi(0, *, *) = 1$

for all $k \in \{1 \ldots n\}$, for all $u \in S_{k-1}, v \in S_k$:

$$\pi(k, u, v) = \max_{t \in S_{k-2}} \pi(k-1, t, u) \times q(v \mid t, u, w, k)$$

- **Correctness:** again any sequence ending in triples $t, u, w$ in position $k$, must go through the highest probability sequence ending in $t, u$ in position $k-1$

- **Space and time complexity:** just like HMMs
  - Time $O(n|S|^3)$, Space $O(n|S|^2)$
Add backpointers

**Definition:** $S_k$ is the set of possible tags in position $k$.

**Base:** $\pi(0, *, *) = 1$

for all $k \in \{1 \ldots n\}$, for all $u \in S_{k-1}, v \in S_k$:

$$\pi(k, u, v) = \max_{t \in S_{k-2}} \pi(k - 1, t, u) \times q(v | t, u, w, k)$$

$$bp(k, u, v) = \arg \max_{t \in S_{k-2}} \pi(k - 1, t, u) \times q(v | t, u, w, k)$$

$$(y_{n-1}, y_n) = \arg \max_{u, v} \pi(n, u, v)$$

for $k = (n - 2) \ldots 1$, $y_k = bp(k + 2, y_{k+1}, y_{k+2})$
Ratnaparkhi, 1996

- Feature templates for
  - word/tag pairs
  - prefixes <5 / tag pairs
  - suffixes <5 / tag pairs
  - tag trigrams
  - tag bigrams
  - tag unigrams
  - word/tag pairs for previous word
  - word/tag pairs for subsequent word
Greedy vs. Viterbi

- What are the advantages of using greedy?
  - Faster
  - Feature function not constrained to the previous two tags

- What are the advantages of Viterbi
  - Global inference
  - Does not suffer from error propagation at test time
## Greedy vs. Viterbi

<table>
<thead>
<tr>
<th>Language</th>
<th>Source</th>
<th># Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>PADT/CoNLL07 (Hajič et al., 2004)</td>
<td>21</td>
</tr>
<tr>
<td>Basque</td>
<td>Basque3LB/CoNLL07 (Aduriz et al., 2003)</td>
<td>64</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>BTB/CoNLL06 (Simov et al., 2002)</td>
<td>54</td>
</tr>
<tr>
<td>Catalan</td>
<td>CESS-ECE/CoNLL07 (Martí et al., 2007)</td>
<td>54</td>
</tr>
<tr>
<td>Chinese</td>
<td>Penn Chinese/Treebank 6.0 (Palmer et al., 2007)</td>
<td>24</td>
</tr>
<tr>
<td>Chinese</td>
<td>Sinica/CoNLL07 (Chen et al., 2003)</td>
<td>294</td>
</tr>
<tr>
<td>Czech</td>
<td>PDT/CoNLL07 (Böhmová et al., 2003)</td>
<td>63</td>
</tr>
<tr>
<td>Danish</td>
<td>DDT/CoNLL06 (Kromann et al., 2003)</td>
<td>25</td>
</tr>
<tr>
<td>Dutch</td>
<td>Alpino/CoNLL06 (Van der Beck et al., 2002)</td>
<td>12</td>
</tr>
<tr>
<td>English</td>
<td>PennTreebank (Marcus et al., 1993)</td>
<td>45</td>
</tr>
<tr>
<td>French</td>
<td>FrenchTreebank (Abeillé et al., 2003)</td>
<td>30</td>
</tr>
<tr>
<td>German</td>
<td>Tiger/CoNLL06 (Brants et al., 2002)</td>
<td>54</td>
</tr>
<tr>
<td>German</td>
<td>Negra (Skut et al., 1997)</td>
<td>54</td>
</tr>
<tr>
<td>Greek</td>
<td>GDT/CoNLL07 (Prokopidis et al., 2005)</td>
<td>38</td>
</tr>
<tr>
<td>Hungarian</td>
<td>Szeged/CoNLL07 (Csendes et al., 2005)</td>
<td>43</td>
</tr>
<tr>
<td>Italian</td>
<td>ISST/CoNLL07 (Montemagni et al., 2003)</td>
<td>28</td>
</tr>
<tr>
<td>Japanese</td>
<td>Verbmobil/CoNLL06 (Kawata and Bartels, 2000)</td>
<td>80</td>
</tr>
<tr>
<td>Japanese</td>
<td>Kyoto4.0 (Kurohashi and Nagao, 1997)</td>
<td>42</td>
</tr>
<tr>
<td>Korean</td>
<td>Sejong (<a href="http://www.sejong.or.kr">http://www.sejong.or.kr</a>)</td>
<td>187</td>
</tr>
<tr>
<td>Portuguese</td>
<td>Floresta Sintáctica/CoNLL06 (Afonso et al., 2002)</td>
<td>22</td>
</tr>
<tr>
<td>Russian</td>
<td>SynTagRus-RNC (Boguslavsky et al., 2002)</td>
<td>11</td>
</tr>
<tr>
<td>Slovenian</td>
<td>SDT/CoNLL06 (Džeroski et al., 2006)</td>
<td>20</td>
</tr>
<tr>
<td>Spanish</td>
<td>Ancora-Cast3LB/CoNLL06 (Civit and Martí, 2004)</td>
<td>47</td>
</tr>
<tr>
<td>Swedish</td>
<td>Talbanken05/CoNLL06 (Nivre et al., 2006)</td>
<td>41</td>
</tr>
<tr>
<td>Turkish</td>
<td>METU-Sabanci/CoNLL07 (Oflazer et al., 2003)</td>
<td>31</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
294^2 &= 86436 \\
45^2 &= 2045 \\
11^2 &= 121
\end{align*}

\[
\begin{align*}
294^4 &= 7471182096 \\
45^4 &= 4100625 \\
11^4 &= 14641
\end{align*}

[Petrov et al. 2012]
## HMM vs. MEMM (POS tags)

<table>
<thead>
<tr>
<th></th>
<th>Token-level accuracy</th>
<th>OOV accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>96.46%</td>
<td>85.86%</td>
</tr>
<tr>
<td>MEMM</td>
<td>96.96%</td>
<td>91.29%</td>
</tr>
</tbody>
</table>
MEMM vs. HMM

- McCallum et al. (2000) compared the two in a task of FAQ segmentation
  - Modeling word generation is hard
  - Tags: head, question, answer

```xml
<head>X-NNTP-Poster: NewsHound v1.33
<head>
<head>Archive-name: acorn/faq/part2
<head>Frequency: monthly
<head>
<question>2.6) What configuration of serial cable should I use
<answer>
<answer> Here follows a diagram of the necessary connections
<answer>programs to work properly. They are as far as I know t
<answer>agreed upon by commercial comms software developers fo
<answer>
<answer> Pins 1, 4, and 8 must be connected together inside
<answer>is to avoid the well known serial port chip bugs. The
```
MEMM

• Current tag is $t$, previous tag is $t’$ and
  • Line begins with number/punctuation/wh-word…
  • Line contains …
  • Line ends with …
  • Line indentation is
Token HMM

• Generate each word in the sentence independently from other words given the tag seems bad

<question>2.6) What configuration of serial cable should I use

\[ p(\text{“2.6) What configuration…”} | \text{question}) = p(\text{“2.6”) } | \text{question}) \times p(\text{“What”} | \text{question}) \times p(\text{“configuration”} | \text{question}) \times \ldots \]
Feature HMM

• Replace words with features and then generate:

\[
p("2.6) What configuration of serial cable should I use\) = \\
p(begins-with-number \mid \text{question}) \times \\
p(wh-word \mid \text{question}) \times \\
p(contains-alphanum \mid \text{question}) \times \\
\ldots
\]
# FQA results

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>ME-stateless</td>
<td>0.038</td>
<td>0.362</td>
</tr>
<tr>
<td>Token HMM</td>
<td>0.276</td>
<td>0.140</td>
</tr>
<tr>
<td>Feature HMM</td>
<td>0.413</td>
<td>0.529</td>
</tr>
<tr>
<td>MEMM</td>
<td>0.867</td>
<td>0.681</td>
</tr>
</tbody>
</table>

- **Precision**: number of correct segments predicted divided by total number of segments predicted
- **Recall**: Number of correct segments predicted divided by total number of true segments

Rich overlapping features help a lot!
# Model zoo

<table>
<thead>
<tr>
<th></th>
<th>Tagging</th>
<th>Parsing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generative</td>
<td>HMMs</td>
<td>PCFGs</td>
</tr>
<tr>
<td>Log-linear</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greedy</td>
<td>Transition-based parsing</td>
<td></td>
</tr>
<tr>
<td>Locally normalized</td>
<td>Viterbi</td>
<td>CKY</td>
</tr>
<tr>
<td>Globally normalized</td>
<td>forward-backward</td>
<td>Inside-outside</td>
</tr>
</tbody>
</table>

And then also deep learning variants!
Summary

• Decompose a tag sequence to a sequence of “decisions”

• Train a log-linear model for making local decisions

• Decode with Viterbi
  • Or greedy
  • Or beam search
Globally-normalized models
Globally-normalized models

• Why do we decompose to a sequence of decisions?

• Can we directly estimate the probability of an entire sequence

\[ p_\theta(y \mid x) = \frac{\exp(f(x, y)\top \theta)}{\sum_{y' \in \mathcal{Y}} \exp(f(x, y')\top \theta)} \]

\[ \mathcal{Y} : \text{all tag sequences for } x \]
Global linear models

• Motivations:
  • Optimize directly what you care about
  • Flexibility in feature function
  • Avoid the label bias problem

• Question:
  • How to compute the denominator?
Label Bias Problem

- We are normalizing at each time step

- So a fixed probability mass must be distributed at each step

- So this favors states that have low entropy output distribution

- We are conditioning on the input not generating it, so once we are “stuck” in a low entropy state we ignore the input
Label Bias Problem

- 0 2 2 2 2 “seems” like the right choice
- But 0 1 1 1 1 has higher probability (0.3 vs. $0.7^4 = 0.24$)
- This is since once you choose state “1” you are stuck there and will predict “1” regardless of the input
Label bias problem

- Consider a history-based model for names with two tokens, that are either PERSON or LOC.

  - States
    - b-person
    - e-person
    - b-loc
    - e-loc
    - other
Label bias problem

corpus:

*Harvey Ford*
   (person 9 times, location 1 time)
*Harvey Park*
   (location 9 times, person 1 time)
*Myrtle Ford*
   (person 9 times, location 1 time)
*Myrtle Park*
   (location 9 times, person 1 time)

Second word provides good information
Label bias problem

Conditional probabilities:

\[
\begin{align*}
    p(\text{b-person} \mid \text{other}, w = \text{Harvey}) &= 0.5 \\
    p(\text{b-locn} \mid \text{other}, w = \text{Harvey}) &= 0.5 \\
    p(\text{b-person} \mid \text{other}, w = \text{Myrtle}) &= 0.5 \\
    p(\text{b-locn} \mid \text{other}, w = \text{Myrtle}) &= 0.5 \\
    p(\text{e-person} \mid \text{b-person}, w = \text{Ford}) &= 1 \\
    p(\text{e-person} \mid \text{b-person}, w = \text{Park}) &= 1 \\
    p(\text{e-locn} \mid \text{b-locn}, w = \text{Ford}) &= 1 \\
    p(\text{e-locn} \mid \text{b-locn}, w = \text{Park}) &= 1
\end{align*}
\]

Information from second word is lost
Label bias problem

• Prefer states that have low entropy next-state distributions
Global linear models

• Main differences:
  • Normalize once over entire space
  • We don’t look at a history of decisions
  • Define features over **full** structure
    • More flexible!
      • We will see this in parsing
  • But we directly consider the exponential space
Global linear models

- Contain three components
  - A feature function $f$ mapping a pair $(x,y)$ to a feature vector $f(x,y)$
  - A generating function $GEN$ enumerating all candidate outputs
  - A parameter vector $\theta$

$$F(x) = \arg \max_y f(x,y)^\top \theta$$
Tagging: GEN

• Inputs: sentences $x = w_1 \ldots w_n$

• Tag set: $T$

• $GEN(X) = T^n$

• In other setups:
  • All parse trees
  • Top-K parse trees produced by weaker model
  • All translations
Tagging: feature function

- A **history** \( h \) is a 4-tuple \( <t_{i-2}, t_{i-1}, w, i> \)
- \( t_{i-2}, t_{i-1} \): previous two tags
- \( w \): input sentence
- \( i \): index of word being tagged

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/?? from which Spain expanded its empire into the rest of the Western Hemisphere.

- \( t_{i-2}, t_{i-1} \): DT JJ
- \( w \): Hispaniola, …, Hemisphere
- \( i \): 6
Local features

- For every history/tag pair \((h, i, t)\)

\[ g_s(h, i, t) \text{ for } s = 1 \ldots , d : \]

local features for tagging decision \(t\) in position \(i\) given history \(h\):

\[
\begin{align*}
  g_{100}(h, t) & \begin{cases} 1 & w_i = \text{‘base’} \text{ and } t = \text{VB} \\ 0 & \text{otherwise} \end{cases} \\
  g_{101}(h, t) & \begin{cases} 1 & w_i \text{ ends with ‘ing’} \text{ and } t = \text{VBG} \\ 0 & \text{otherwise} \end{cases} \\
  g_{102}(h, t) & \begin{cases} 1 & (t_{i-2}, t_{i-1}, t) = (\text{DT, JJ, VB}) \\ 0 & \text{otherwise} \end{cases}
\end{align*}
\]
Multiple local features

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/NN

<table>
<thead>
<tr>
<th>History</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>*</td>
<td>NNP</td>
</tr>
<tr>
<td>NNP</td>
<td>RB</td>
</tr>
<tr>
<td>RB</td>
<td>VB</td>
</tr>
<tr>
<td>VB</td>
<td>DT</td>
</tr>
<tr>
<td>DT</td>
<td>JJ</td>
</tr>
</tbody>
</table>
Global feature function

- Global feature function
  \[ f(x, y) = \sum_{i} g(h, i, t) \]

- Features are now counts, rather than binary

  - How many times a word ending in ‘ing’ was tagged as VBG
Global log-linear model

\[ p_\theta(y \mid x) = \frac{\exp(f(x, y)^\top \theta)}{\sum_{y' \in \text{GEN}(x)} \exp(f(x, y')^\top \theta)} \]

\[ = \frac{\exp(\sum_i g(x, i, y_{i-2}, y_{i-1}, y_i)^\top \theta)}{\sum_{y' \in \text{GEN}(x)} \exp(\sum_i g(x, i, y'_{i-2}, y'_{i-1}, y'_i)^\top \theta)} \]
Decoding

- As long as the feature function decomposes locally we can still apply Viterbi

- Otherwise we can do greedy/beam decoding
Decoding

- Find the structure that maximizes the dot linear score

$$\arg \max_y p_\theta(y \mid x) = \arg \max_y \frac{\exp(f(x, y)^\top \theta)}{\sum_{y' \in \text{GEN}(x)} \exp(f(x, y')^\top \theta)}$$

$$= \arg \max_y \exp(f(x, y)^\top \theta)$$

$$= \arg \max_y f(x, y)^\top \theta$$

$$= \arg \max_y \sum_i g(x, i, y_{i-2}, y_{i-1}, y_i)^\top \theta$$
Viterbi still works!

- Dependencies did not change

**Definition:** $\mathcal{Y}_i$ is the set of possible tags in position $i$

**Base:** $\pi(1, *, y) = g(x, 1, *, *, y)^\top \theta$

for all $i \in \{2 \ldots n\}$, for all $u \in \mathcal{Y}_{i-1}, v \in \mathcal{Y}_i$:

$$\pi(i, u, v) = \max_{t \in \mathcal{Y}_{i-2}} \pi(j - 1, t, u) + g(x, i, t, u, v)^\top \theta$$
Learning

\[ L(\theta) = \sum_i \log p_\theta(y_i \mid x_i) \]

\[ \nabla L(\theta)_i = f(x_i, y_i) - \sum_{y'} p_\theta(y' \mid x) f(x, y') \]

How to compute the second term?
Learning

• Let’s look at bigram features only

\[
\sum_y p_\theta(y \mid x) f(x, y) = \sum_y \sum_i p_\theta(y \mid x) g(x, i, y_{i-1}, y_i)
\]

\[
= \sum_i \sum_{a,b} \sum_{y:y_{i-1}=a, y_i=b} p_\theta(y \mid x) g(x, i, y_{i-1}, y_i)
\]

\[
= \sum_i \sum_{a,b} g(x, i, a, b) \sum_{y:y_{i-1}=a, y_i=b} p_\theta(y \mid x)
\]

\[
= \sum_i \sum_{a,b} g(x, i, a, b) q_i(a, b)
\]

• q terms are the probability that the tag sequence has a and b in positions i-1, i
Learning

• If we compute the $q$ terms efficiently then we can compute the gradients and learn.

• The $q$ terms are computed with a dynamic programming algorithm called *forward-backward*.

• Also used in unsupervised parameter estimation for HMMs.

• Similar to Viterbi.
Forward-backward

• Definitions

\[ \Psi(y', y, j) = \exp(g(x, j, y', y)^\top \theta) \]

\[ \Psi(y_1, \ldots, y_m) = \prod_{j=1}^{m} \Psi(y_{j-1}, y_j, j) \]

\[ = \prod_{j=1}^{m} \exp(g_j(x, j, y_{j-1}, y_j)^\top \theta) \]

\[ = \exp(\sum_{j=1}^{m} g_j(x, j, y_{j-1}, y_j)^\top \theta) \]

\[ p(y_1, \ldots y_m \mid x) = \frac{\Psi(y_1, \ldots, y_m)}{\sum_{z_1, \ldots, z_m} \Psi(z_1, \ldots, z_m)} \]
Forward-backward

• What will be computed?

\[ Z = \sum_{y_1, \ldots, y_m} \Psi(y_1, \ldots, y_m) \]

\[ \mu(j, a) = \sum_{y_1, \ldots, y_m: y_j = a} \Psi(y_1, \ldots, y_m) \]

\[ \mu(j, a, b) = \sum_{y_1, \ldots, y_m: y_j = a, y_{j+1} = b} \Psi(y_1, \ldots, y_m) \]

\[ q_j(a, b) = \frac{\mu(j, a, b)}{Z} \]
Forward-backward

• What will be computed?

Sum of the scores of all paths that end at tag $y$ in position $j$

$$\alpha(j, y) = \sum_{y_1, \ldots, y_j: y_j = y} \prod_{k=1}^{j} \Psi(y_{k-1}, y_k, k)$$

Sum of the scores of all paths start at tag $y$ in position $j$

$$\beta(j, y) = \sum_{y_j, \ldots, y_m: y_j = y} \prod_{k=j}^{m} \Psi(y_k, y_{k+1}, k)$$
\( \alpha \) terms

- Compute with dynamic programming

\[
\alpha(1, y) = \Psi(\ast, y, 1)
\]

for \( j \in \{2 \ldots m\}, y \in \mathcal{Y} \)

\[
\alpha(j, y) = \sum_{y' \in \mathcal{Y}} \alpha(j - 1, y') \times \Psi(y', y, j)
\]

\[
\begin{array}{c|c|c|c}
  & 1 & 2 & 3 \\
\hline
y_1 & \alpha(1, y_1) &  &  \\
\hline
y_2 & \alpha(1, y_2) &  &  \\
\hline
y_3 & \alpha(1, y_3) &  &  \\
\end{array}
\]

\[
\begin{align*}
\alpha(1, y_1) &= \Psi(\ast, y_1, 1) \\
\alpha(1, y_2) &= \Psi(\ast, y_2, 1) \\
\alpha(1, y_3) &= \Psi(\ast, y_3, 1)
\end{align*}
\]
\[ \alpha \text{ terms} \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_1 )</td>
<td>( \alpha(1, y_1) )</td>
<td>( \alpha(2, y_1) )</td>
<td></td>
</tr>
<tr>
<td>( y_2 )</td>
<td>( \alpha(1, y_2) )</td>
<td>( \alpha(2, y_2) )</td>
<td></td>
</tr>
<tr>
<td>( y_3 )</td>
<td>( \alpha(1, y_3) )</td>
<td>( \alpha(3, y_3) )</td>
<td></td>
</tr>
</tbody>
</table>

\[
\alpha(2, y_1) = \alpha(1, y_1)\Psi(y_1, y_1, 2) + \alpha(1, y_2)\Psi(y_2, y_1, 2) + \alpha(1, y_3)\Psi(y_3, y_1, 2)
\]

\[
= \Psi(\ast, y_1, 1)\Psi(y_1, y_1, 2) + \Psi(\ast, y_2, 1)\Psi(y_2, y_1, 2) + \Psi(\ast, y_3, 1)\Psi(y_3, y_1, 2)
\]

\[
\alpha(2, y_2) = \Psi(\ast, y_1, 1)\Psi(y_1, y_2, 2) + \Psi(\ast, y_2, 1)\Psi(y_2, y_2, 2) + \Psi(\ast, y_3, 1)\Psi(y_3, y_2, 2)
\]

\[
\alpha(2, y_3) = \Psi(\ast, y_1, 1)\Psi(y_1, y_3, 2) + \Psi(\ast, y_2, 1)\Psi(y_2, y_3, 2) + \Psi(\ast, y_3, 1)\Psi(y_3, y_3, 2)
\]
### α terms

- The distributive law works

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>$\alpha(1, y_1)$</td>
<td>$\alpha(2, y_1)$</td>
<td>$\alpha(3, y_1)$</td>
</tr>
<tr>
<td>$y_2$</td>
<td>$\alpha(1, y_2)$</td>
<td>$\alpha(2, y_2)$</td>
<td></td>
</tr>
<tr>
<td>$y_3$</td>
<td>$\alpha(1, y_3)$</td>
<td>$\alpha(3, y_3)$</td>
<td></td>
</tr>
</tbody>
</table>

\[
\alpha(3, y_1) = \alpha(2, y_1) \Psi(y_1, y_1, 3) + \alpha(2, y_2) \Psi(y_2, y_1, 3) + \alpha(2, y_3) \Psi(y_3, y_1, 3) \\
= \Psi(\ast, y_1, 1) \Psi(y_1, y_1, 2) \Psi(y_1, y_1, 3) + \Psi(\ast, y_2, 1) \Psi(y_2, y_1, 2) \Psi(y_1, y_1, 3) + \Psi(\ast, y_3, 1) \Psi(y_3, y_1, 2) \Psi(y_1, y_1, 3) \\
+ \Psi(\ast, y_1, 1) \Psi(y_1, y_2, 2) \Psi(y_2, y_1, 3) + \Psi(\ast, y_2, 1) \Psi(y_2, y_2, 2) \Psi(y_2, y_1, 3) + \Psi(\ast, y_3, 1) \Psi(y_3, y_2, 2) \Psi(y_2, y_1, 3) \\
+ \Psi(\ast, y_1, 1) \Psi(y_1, y_3, 2) \Psi(y_3, y_1, 3) + \Psi(\ast, y_2, 1) \Psi(y_2, y_3, 2) \Psi(y_3, y_1, 3) + \Psi(\ast, y_3, 1) \Psi(y_3, y_3, 2) \Psi(y_3, y_1, 3)
\]
\( \beta \) terms

\[
\beta(m, y) = 1
\]
for \( j \in \{m - 1 \ldots 1\} \), \( y \in \mathcal{Y} \)

\[
\beta(j, y) = \sum_{y' \in \mathcal{Y}} \beta(j + 1, y') \times \Psi(y, y', j)
\]

• Use distributive property again to get final result

\[
Z = \sum_{y \in \mathcal{Y}} \alpha(m, y)
\]

\[
\mu(j, a) = \alpha(j, a) \cdot \beta(j, a)
\]

\[
\mu(j, a, b) = \alpha(j, a) \cdot \Psi(a, b, j + 1) \cdot \beta(j + 1, a)
\]

\[
q_j(a, b) = \frac{\mu(j, a, b)}{Z}
\]
Complexity

• If length of sentence is $m$, then $m|Y|^2$

• We can compute the gradient

• We can apply SGD

• We can learn

• **Limitation**: decomposable features
Structured perceptron

- Similar to CRFs, but needs only Viterbi
- For every training example \((x,y)\)
  - Find best \(y'\) according to model (Viterbi)
  - If different from gold \(y\)
    - Update weights: add features of \(y\) and subtract features of \(y'\)
- Regularization is needed (averaged perceptron)
Global vs. Local models

\[ p_L(y_i \mid x, y_1 \ldots y_{i-1} \mid x) = \frac{\exp(s(x, y_1 \ldots y_i))}{Z_L()} \]

\[ Z_L(x, y_1 \ldots, y_{i-1}) = \sum_t \exp(s(x, y_1 \ldots y_{i-1}, t)) \]

\[ p_L(y_1 \ldots, y_n \mid x) = \frac{\exp \sum_{i=1}^n s(x, y_1 \ldots y_i)}{\prod_{i=1}^n Z_L(x, y_1 \ldots, y_{i-1})} \]

\[ p_G(y_1 \ldots, y_n \mid x) = \frac{\exp \sum_{i=1}^n s(x, y_1 \ldots y_i)}{Z_G(x)} \]

\[ Z_G(x) = \sum_{y_1 \ldots y_n} \exp \sum_{i=1}^n s(x, y_1 \ldots, y_i) \]

It can be shown that some distribution can only be expressed with \( p_G \).
Training data:

\[
\begin{align*}
&ABC \\
&\text{abc} \\
&ADE \\
&\text{abe}
\end{align*}
\]

\[
g(x, y, \ldots, y, i) = \alpha \prod (x_i, y_i) \in \{AB, BC, AD, DE\} + \alpha \prod (x_i, y_i) \in \{aa, bb, cc, dd, ee\}
\]

\[
P_\epsilon(ABC | abc) = \frac{\exp(5\alpha)}{\sum \exp(5\alpha) + \exp(4\alpha) + \ldots} = \text{softmax}(5\alpha)
\]

\[
\lim_{\alpha \to 0} P_\epsilon(ABC | abc) = 1
\]

- For any definition of \(g(x, y, \ldots, y, i) = g(x, i, y, i, \ldots, y, i)\):

\[
\begin{align*}
P_L(ABC | abc) & = P_L(A | a) \times P_L(B | A, a, b) \times P_L(C | A, B, a, b, c) \\
P_L(ADE | ade) & = P_L(A | a) \times P_L(D | A, a, b) \times P_L(E | A, B, a, b, e)
\end{align*}
\]

\[
P_L(ABC | abc) + P_L(ADE | abc) \leq P_L(B | A, a, b) + P_L(D | A, a, b) \leq 1
\]

on the other hand for large enough \(\alpha\):

\[
P_\epsilon(ABC | abc) + P_\epsilon(ADE | abc) > 1
\]
Graphical models

- Viterbi and forward-backward are instances of max-product and sum-product algorithms for inference in graphical models

- Advanced ML class

HMM

MEMM

CRF

\[ p(y) = \frac{1}{Z} \prod_{i=1}^{n} \Psi(x, y_{i-1}, y_i) \]
Deep learning models for tagging
Deep learning model

• We have a more powerful learning model

• Can we make things
  • better?
  • simpler?
Simple POS-tagging model

\[ x_i : \text{one-hot rep. for word } i \]
\[ e_i = W^{\text{emb}} x_i : \text{word embedding} \]
\[ h^f_i = \text{LSTM}(h_{i-1}, e_i) \]
\[ h^b_i = \text{LSTM}(h_{i+1}, e_i) \]
\[ c_i = \sigma(W[h^f_i; h^b_i] + b) \]
\[ p(y_i | x) = \text{softmax}(W^s c_i + b) \]
\[ L(\theta) = -\sum_i \log p(y_i = y^* | x) \]

- All tag decisions are independent!! Works OK for POS tagging
  - 96.97 acc. compared to 97.32 with a linear CRF
  - Doesn’t work for NER tagging. Why?
Deep learning models for tagging

- Independent tagging (saw last class)
  - No feature decomposition: BiLSTM passes information around

- How to neuralize the models we saw?
  - Greedy tagger
  - Viterbi tagger
  - CRF
  - BiLSTM-CRF
Greedy taggers

• A **history** $x$ is a 4-tuple $<t_1 \ldots i-1, w, i>$

  • Predict $t_i$ from history

• $p(t_i \mid x) \propto \exp(f(x,t_i) \times \theta) = \text{score}(x,t_i)$

• Just replace the log-linear function with a non-linear neural network

• Need to define a neural network that reads the history and produces a label
Greedy tagger

Hispaniola/NNP quickly/RB became/VB an/DTD important/JJ base/NN

softmax

NN VB ... DT
Viterbi tagger

\[ s(x, y) = f(x, y)^\top \theta = \sum_i g(x, i, y_{i-1}, y_i)^\top \theta \]

\[ = s(x, 1, y_1, y_2) + s(x, 2, y_2, y_3) + s(x, 3, y_3, y_4) \]

- Replace linear score with non-linear neural network

\[ s(x, y) = \sum_i \text{NN}(x, i, y_{i-1}, y_i) \]

You mean CRF?
Viterbi tagger

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/NN

softmax

Hispaniola   quickly   became   an   Important   DT   JJ   base
Neural structured prediction

• What has changed?

• **Decoding**: As long as the neural network scores decompose we can still apply Viterbi

• **Learning**:

\[
p_\theta(y \mid x) = \frac{\exp(\sum_i \text{NN}_\theta(x, i, y_{i-1}, y_i))}{\sum_{y'} \exp(\sum_i \text{NN}_\theta(x, i, y'_{i-1}, y'_i))}
\]

\[
L^{(i)}(\theta) = -\log p(y^{(i)} \mid x^{(i)})
\]
Neural CRF

$\text{He/N eats/V}$

$\log P(N, V | \text{He eats})$

$\text{score(He eats|NV)}$

$\alpha(2, N)$

$\alpha(2, V)$

$\alpha(1, N)\psi(2, N, N)$

$\alpha(1, V)\psi(2, V, N)$

$\alpha(1, N)\psi(2, N, V)$

$\alpha(1, V)\psi(2, V, V)$

$\alpha(1, N)$

$\alpha(1, V)$

$z$
Neural structured prediction

- How do we compute the denominator (partition function)?
  - Implement dynamic programming as part of the neural network
  - All dynamic programming operations are differentiable (+ and x)

- In structured perceptron things are slightly simpler
  - Compute the best structure (outside of the network)
  - Define the loss: \[ \max(0, \text{score}(x, y) - \max_{y'} \text{score}(x, y')) \]
Bi-LSTM CRF for NER

$x_i$ : one-hot rep. for word $i$

$e_i = W^\text{emb} x_i$ : word embedding

$l_i = \text{LSTM}(l_{i-1}, e_i)$

$r_i = \text{LSTM}(r_{i+1}, e_i)$

$c_i = [l_i; r_i]$  

$p_i = W^{(\text{proj})} \sigma(Wc_i + b)$

$s(x, y) = \sum_{i=1}^{n} p_{i,y} + \sum_{i=0}^{n} A_{y_i,y_{i+1}}$

- $A$ is a learned parameter matrix ($\#\text{tags}^2$).

- SOTA or close to that on POS tagging, NER, and other tasks
State-of-the-art

• To outperform state-of-the-art
  • Regularization techniques
  • Character-based information
  • Pre-training with a lot of data
More?

• Interactions in this Bi-LSTM CRF are pretty weak. Can we make them stronger?

• A network that predicts a distribution over the tag space (latent) and then reads those distributions and computes a new distribution

• Loss is over both distributions

• Most general: A network that reads (x,y) and produces a score s(x,y)

• But then unclear how to find the best y
# Summary

<table>
<thead>
<tr>
<th></th>
<th>Tagging</th>
<th>Parsing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Generative</strong></td>
<td>HMMs</td>
<td>PCFGs</td>
</tr>
<tr>
<td><strong>Greedy</strong></td>
<td></td>
<td>Transition-based parsing</td>
</tr>
<tr>
<td><strong>History-based</strong></td>
<td>Viterbi</td>
<td>CKY</td>
</tr>
<tr>
<td><strong>Global</strong></td>
<td>forward-backward</td>
<td>Inside-outside</td>
</tr>
</tbody>
</table>