Natural Language Processing

Tagging part II

Based on slides from Michael Collins
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 ./O
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

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

\(\mathcal{Y} : \) all tag sequences for \(x\)
Global linear models

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

• 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 an MEMM model for names with two tokens, that are either PERSON or LOC.

• States
  • B-person
  • I-person
  • B-loc
  • I-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:

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

Information from second word is lost
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 \((h, t)\)

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

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

\[
\begin{align*}
\text{if } & w_i = \text{'base'} \text{ and } t = \text{VB} \\
& w_i \text{ ends with 'ing' and } t = \text{VBG} \\
& (t_{i-2}, t_{i-1}, t) = (\text{DT, JJ, VB})
\end{align*}
\]
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

\[ f(x, y) = \sum_i g(h_i, t_i) \]

- 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

\[ L(\theta) = \sum_{m=1}^{M} \log p_{\theta}(y^m|x^m) \]

Gradient for each training point:

\[ \nabla \log p_{\theta}(y^m|x^m) = f(x^m, y^m) - \sum_{y'} p_{\theta}(y'|x^m) f(x^m, y') \]

How to compute the second term?
Learning

• Let’s look at bigram features only

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

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

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

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

• 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 $j$

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

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

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

- Compute with dynamic programming

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

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

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

- Compute with dynamic programming

\[
\alpha(1, y) = \Psi(\ast, y, 1) \\
\text{for } j \in \{2 \ldots m\}, y \in \mathcal{Y}
\]

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

<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></td>
<td>(\alpha(1, y_1) = \Psi(\ast, y_1, 1))</td>
</tr>
<tr>
<td>(y_2)</td>
<td>(\alpha(1, y_2))</td>
<td></td>
<td>(\alpha(1, y_2) = \Psi(\ast, y_2, 1))</td>
</tr>
<tr>
<td>(y_3)</td>
<td>(\alpha(1, y_3))</td>
<td></td>
<td>(\alpha(1, y_3) = \Psi(\ast, y_3, 1))</td>
</tr>
</tbody>
</table>
\[ \alpha \text{ terms} \]

<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>

\[\begin{align*}
\alpha(2, y_1) &= \alpha(1, y_1)\Psi(y_1, y_1, 2) + \alpha(1, y_2)\Psi(y_2, y_1, 2) + \alpha(1, y_3)\Psi(y_3, y_1, 2) \\
&= \Psi(*, y_1, 1)\Psi(y_1, y_1, 2) + \Psi(*, y_2, 1)\Psi(y_2, y_1, 2) + \Psi(*, y_3, 1)\Psi(y_3, y_1, 2) \\
\alpha(2, y_2) &= \Psi(*, y_1, 1)\Psi(y_1, y_2, 2) + \Psi(*, y_2, 1)\Psi(y_2, y_2, 2) + \Psi(*, y_3, 1)\Psi(y_3, y_2, 2) \\
\alpha(2, y_3) &= \Psi(*, y_1, 1)\Psi(y_1, y_3, 2) + \Psi(*, y_2, 1)\Psi(y_2, y_3, 2) + \Psi(*, y_3, 1)\Psi(y_3, y_3, 2)
\end{align*}\]
\( \alpha \text{ terms} \)

- The distributive law works

| \( y_1 \) | \( \alpha(1, y_1) \) | \( \alpha(2, y_1) \) | \( \alpha(3, y_1) \) |
| \( y_2 \) | \( \alpha(1, y_2) \) | \( \alpha(2, y_2) \) |
| \( y_3 \) | \( \alpha(1, y_3) \) | \( \alpha(3, y_3) \) |

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

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

$$\mu(j, a) = \alpha(j, a) \cdot \beta(j, a)$$

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

$$q_j(a, b) = \frac{\mu(j, a, b)}{Z}$$
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

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

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

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

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

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

\[
\begin{align*}
A & B & C & \quad A & D & E \\
\text{abc} & & \quad \text{abe}
\end{align*}
\]

\[
q(x, y) = \alpha \times \prod_{i=1}^{L} q_i(x_i, y_i) \in \{\text{AB, BC, AD, DE}\} + \alpha \times \prod_{i=1}^{L} (x_i, y_i) \in \{\text{aA, bB, cC, bD, eE}\}
\]

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

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

For any definition of \(q(x, y)\):

\[
P_\alpha(ABC|abc) = P_\alpha(A|a) \times P_\alpha(B|A, a, b) \times P_\alpha(C|A, B, a, b, c) \leq P_\alpha(B|A, a, b)
\]

\[
P_\alpha(AD|abc) = P_\alpha(A|a) \times P_\alpha(D|A, a, b) \times P_\alpha(E|A, B, a, b, e) \leq P_\alpha(D|A, a, b)
\]

\[
P_\alpha(ABC|abc) + P_\alpha(AD|abc) \leq P_\alpha(B|A, a, b) + P_\alpha(D|A, a, b) \leq 1
\]

On the other hand, for large enough \(\alpha\):

\[
P_\alpha(ABC|abc) + P_\alpha(AD|abc) > 1
\]
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

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

- All tag decisions are independent!! Works OK for POS tagging
  - 96.97 acc. compared to 97.32 with a linear CRF
  - Works less well for NER tagging. Why?

Ling et al., 2015
Deep learning models for tagging

- Independent tagging
  - 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...i-1}, w, i>\$
  - Predict $t_i$ from history
  - $p(y = t_i | x) \propto \exp(f(x,y) \times \theta) = \text{score}(x,y)$
  - 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/DT important/JJ base/NN

softmax
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

\[ s(x, y) = \sum_{i} \text{NN}(x, i, y_{i-1}, y_i) \]
Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/NN
Class 7
Tagging wrap-up

• Log-linear models:
  
  • Motivation: richer features

  • Use (history, tag) pairs for training

\[
p_\theta(y \mid x) = \frac{e^{f(x,y)^\top \theta}}{\sum_{y' \in \mathcal{Y}} e^{f(x,y')^\top \theta}}
\]
Tagging wrap-up

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

• 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

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

- Decoding:
  - If features don’t decompose:
    - Greedy
    - Beam-search

NNP  VBZ  NN  NNS  CD  NN
Fed  raises  interest  rates  0.5  percent
Tagging wrap-up

• Decoding:
  • If features decompose: Viterbi
Tagging wrap-up

• Globally-normalized models:

  • Motivation:

    • Optimize what you care about

    • More flexibility in defining features (to be seen)

    • Under some conditions more expressive model (label bias)

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

\[\mathcal{Y} : \text{all tag sequences for } x\]
Tagging wrap-up

• Globally-normalized models

• 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 wrap-up

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

• Decoding: if features decompose can still use Viterbi
Tagging wrap-up

- Main difference: 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?
Tagging wrap-up

• 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
Tagging wrap-up

• Deep models
  • Greedy taggers
    • Arbitrary feed-forward network
  • Viterbi taggers
    • Arbitrary feed-forward network over current and limited horizon tags
Neural structured prediction

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

• **Learning**:

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

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

• We don’t the gradient ourselves

• Instead we just need to express log \( p(y \mid x) \) and backpropagation computes the gradients

• So the only thing that is needed is to express log \( p(y|x) \) using differentiable operations

• Basically implement forward-(backward) with a neural network
Neural CRF

\[ \log P(N \mid \text{He eats}) \]

\[ \text{score(He eats|NV)} \]

\[ \alpha(1, N) \psi(2, N, N) \]

\[ \alpha(1, V) \psi(2, V, N) \]

\[ \alpha(1, N) \psi(2, N, V) \]

\[ \alpha(1, V) \psi(2, V, V) \]

\[ \alpha(2, N) \]

\[ \alpha(2, V) \]

\[ z \]

He/N eats/V
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

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

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

\( c_i = [l_i; r_i] \)

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

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

- \( A \) is a learned parameter matrix (#tags\(^2\)).
- SOTA or close to that on POS tagging, NER, and other tasks
<table>
<thead>
<tr>
<th></th>
<th>Tagging</th>
<th>Parsing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Generative</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HMMs</strong></td>
<td></td>
<td><strong>PCFGs</strong></td>
</tr>
<tr>
<td><strong>Greedy</strong></td>
<td></td>
<td><strong>Transition-based parsing</strong></td>
</tr>
<tr>
<td><strong>Log-linear</strong></td>
<td><strong>History-based</strong></td>
<td><strong>Viterbi</strong></td>
</tr>
<tr>
<td><strong>Viterbi</strong></td>
<td></td>
<td><strong>CKY</strong></td>
</tr>
<tr>
<td><strong>Global</strong></td>
<td><strong>forward-backward</strong></td>
<td><strong>Inside-outside</strong></td>
</tr>
</tbody>
</table>