Sequence Methods

Part 2: Discriminative Models

Maximum Entropy Markov Taggers

- MEMMs (Maximum Entropy Markov Models) are discriminative, log-linear models for sequence labeling:

$$p(y_1 \ldots y_m | x_1 \ldots x_m) = \prod_{i=1}^{m} p(y_i | y_1 \ldots y_{i-1}, x_1 \ldots x_m)$$
$$= \prod_{i=1}^{m} p(y_i | y_{i-1}, x_1 \ldots x_m)$$

$$p(y_i | y_{i-1}, x_1 \ldots x_m) = \frac{e^{\,w \cdot f\,(x_1 \ldots x_m, i, y_{i-1}, y_i)}}{\sum_{y'} e^{\,w \cdot f\,(x_1 \ldots x_m, i, y_{i-1}, y')}}$$

Feature/History Representation

- We have some input domain $\mathcal{X}$, and a finite label set $\mathcal{Y}$. Aim is to provide a conditional probability $p(y \mid x)$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

- A **feature** is a function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$
(Often **binary features** or **indicator functions**
$f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$).

- Say we have m features f_k for $k = 1 \ldots m$
$\Rightarrow$ A **feature vector** $f(x, y) \in \mathbb{R}^m$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

Feature/History Representation

Alternative notation:
f is a feature function from *"histories"* and target label

t : the current tag
t$_{-1}$,*t*$_{-2}$*:* previous tags

A trigram MEMM

- $\mathcal{X}$ is the set of all possible histories of form $\langle t_{-2}, t_{-1}, x_{[1:n]}, i\rangle$
- $\mathcal{Y} = \{\text{NN, NNS, Vt, Vi, IN, DT}, \ldots\}$
- We have m features $f_k : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ for $k = 1 \ldots m$

For example:

$$f_1(h,t) = \begin{cases} 1 & \text{if current word } x_i \text{ is base and } t = \texttt{Vt} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(h,t) = \begin{cases} 1 & \text{if current word } x_i \text{ ends in } \texttt{ing} \text{ and } t = \texttt{VBG} \\ 0 & \text{otherwise} \end{cases}$$

$$\ldots$$

$$f_1(\langle \text{JJ, DT, } \langle \text{ Hispaniola, } \ldots \rangle, 6\rangle, \text{Vt}) = 1$$
$$f_2(\langle \text{JJ, DT, } \langle \text{ Hispaniola, } \ldots \rangle, 6\rangle, \text{Vt}) = 0$$

Feature Set in Ratnaparkhi's MEMM Tagger

- Word/tag features for all word/tag pairs, e.g.,

$$f_{100}(h,t) = \begin{cases} 1 & \text{if current word } x_i \text{ is base and } t = \texttt{Vt} \\ 0 & \text{otherwise} \end{cases}$$

- Spelling features for all prefixes/suffixes of length ≤ 4, e.g.,

$$f_{101}(h,t) = \begin{cases} 1 & \text{if current word } x_i \text{ ends in } \texttt{ing} \text{ and } t = \texttt{VBG} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{102}(h,t) = \begin{cases} 1 & \text{if current word } x_i \text{ starts with } \texttt{pre} \text{ and } t = \texttt{NN} \\ 0 & \text{otherwise} \end{cases}$$

"A Maximum Entropy Model for Part-Of-Speech Tagging" / Ratnaparkhi, 1996

Feature Set in Ratnaparkhi's MEMM Tagger

- Contextual Features, e.g.,

$$f_{103}(h,t) = \begin{cases} 1 & \text{if } \langle t_{-2}, t_{-1}, t\rangle = \langle \text{DT, JJ, Vt}\rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{104}(h,t) = \begin{cases} 1 & \text{if } \langle t_{-1}, t\rangle = \langle \text{JJ, Vt}\rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{105}(h,t) = \begin{cases} 1 & \text{if } \langle t\rangle = \langle \text{Vt}\rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{106}(h,t) = \begin{cases} 1 & \text{if previous word } x_{i-1} = \textit{the} \text{ and } t = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{107}(h,t) = \begin{cases} 1 & \text{if next word } x_{i+1} = \textit{the} \text{ and } t = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$

"A Maximum Entropy Model for Part-Of-Speech Tagging" / Ratnaparkhi, 1996

Full Feature Set in Ratnaparkhi's Tagger

- We can come up with practically any questions (*features*) regarding history/tag pairs.
- For a given history $h \in \mathcal{X}$, each label in $\mathcal{Y}$ is mapped to a different feature vector

$$
\begin{aligned}
f(\langle \text{JJ, DT, } \langle \text{ Hispaniola, } \ldots \rangle, 6 \rangle, \text{Vt}) &= 1001011001001100110 \\
f(\langle \text{JJ, DT, } \langle \text{ Hispaniola, } \ldots \rangle, 6 \rangle, \text{JJ}) &= 0110010101011110010 \\
f(\langle \text{JJ, DT, } \langle \text{ Hispaniola, } \ldots \rangle, 6 \rangle, \text{NN}) &= 0001111101001100100 \\
f(\langle \text{JJ, DT, } \langle \text{ Hispaniola, } \ldots \rangle, 6 \rangle, \text{IN}) &= 0001011011000000010
\end{aligned}
$$

"A Maximum Entropy Model for Part-Of-Speech Tagging" / Ratnaparkhi, 1996

MLE in MEMMs

- The conditional likelihood of the training data:
  - Assume N training examples (sequences) are available $(x^{(i)},y^{(i)})$
  - Superscript is the example index (i), subscript is the token index

$$Pr(y^{(1)},\ldots,y^{(N)}|x^{(1)},\ldots,x^{(N)}) = \prod_{i=1}^{N}\prod_{j=1}^{n(i)} Pr(y_j^{(i)}|y_{j-1}^{(i)},x^{(i)}) = \prod_{i=1}^{N}\prod_{j=1}^{n(i)} \frac{exp(f(j,y_j^{(i)},y_{j-1}^{(i)},x^{(i)})^T\cdot w)}{Z(y_{j-1}^{(i)};w)}$$

- The partition function: (locally normalized)

$$Z(y_{j-1}^{(i)};w) = \sum_{y_j} exp(f(j,y_j,y_{j-1}^{(i)},x^{(i)})^T\cdot w)$$

MLE in MEMMs

- Taking the log, we get the log-likelihood we wish to optimize:

$$LL(w) = \sum_{i=1}^{N} \sum_{j=1}^{n(i)} \left[f(j, y_j^{(i)}, y_{j-1}^{(i)}, x^{(i)})^T \cdot w - log(Z(y_{j-1}^{(i)})) \right]$$

- The gradient for *w* is given by simple derivation: (*T* is the set of labels)

$$\frac{\partial LL}{\partial w_k} = \sum_{i=1}^{N} \sum_{j=1}^{n(i)} \left[f_k(j, y_j^{(i)}, y_{j-1}^{(i)}, x^{(i)}) - \sum_{y' \in T} Pr(y_j = y' | y_{j-1}^{(i)}, x^{(i)}) \cdot f_k(j, y_j = y', y_{j-1}^{(i)}, x^{(i)}) \right]$$

Inference in MEMMs

- Define n to be the length of the sentence
- Define

$$r(t_1 \ldots t_k) = \prod_{i=1}^{k} q(t_i | t_{i-2}, t_{i-1}\ x^{w}_{[1:n]}, i)$$

- Define a dynamic programming table

$\pi(k, u, v)$ $=$ maximum probability of a tag sequence ending in tags u, v at position k

that is,

$$\pi(k, u, v) = \max_{\langle t_1, \ldots, t_{k-2} \rangle} r(t_1 \ldots t_{k-2}, u, v)$$

Inference in MEMMs (Viterbi)

Base case:

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

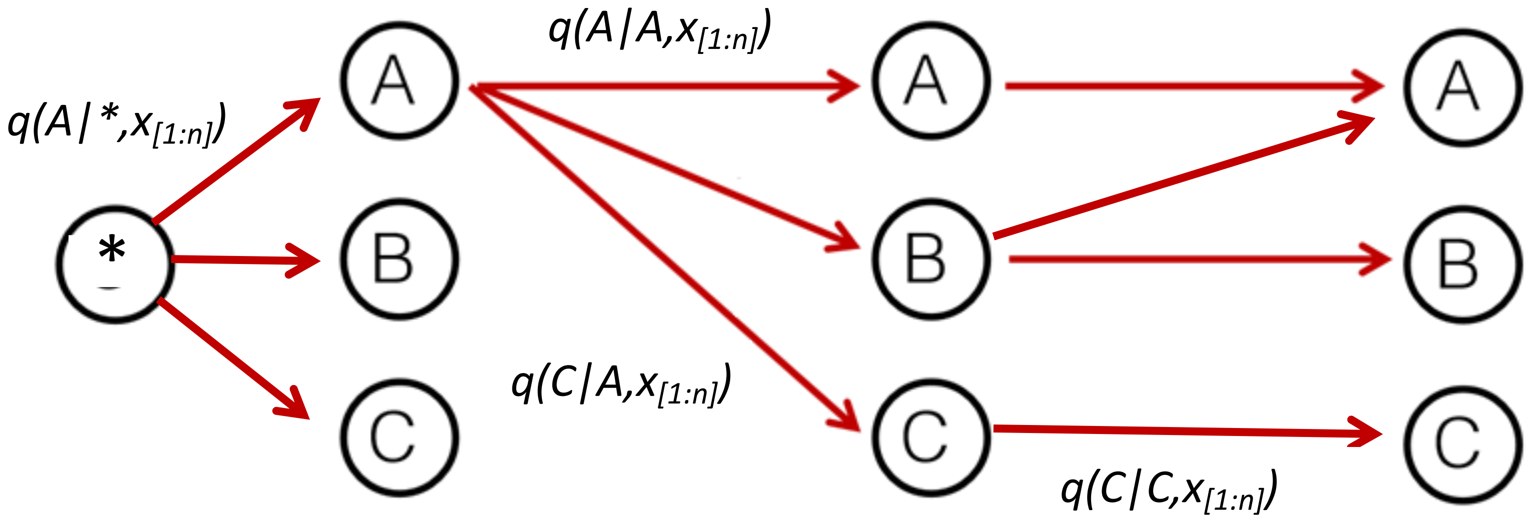
Recursive definition:

For any $k \in \{1 \ldots n\}$, for any $u \in \mathcal{S}_{k-1}$ and $v \in \mathcal{S}_k$:

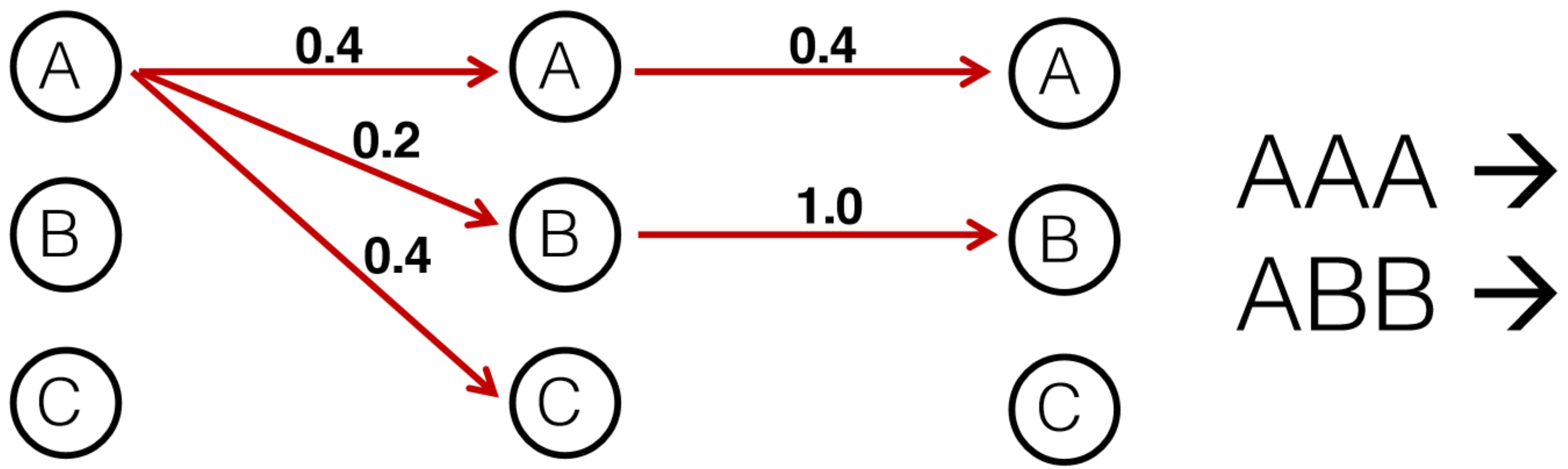
$$\pi(k, u, v) = \max_{t \in \mathcal{S}_{k-2}} \left(\pi(k-1, t, u) \times q(v | t, u, x_{[1:n]}, k) \right)$$

where $\mathcal{S}_k$ is the set of possible tags at position k

Reminder:
The Paths on a Grid Representation

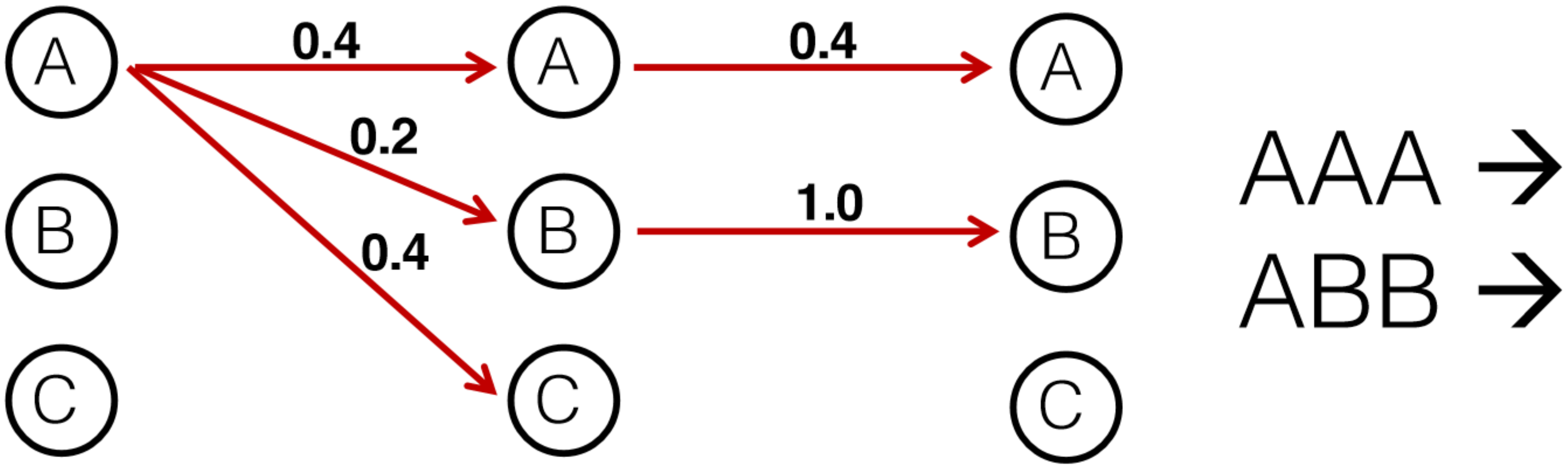


The Label Bias



- We have so far considered locally-normalized models
  - Namely, the scores (probability term) of each transition must sum up to 1
- This assumption may lead to a bias towards labels that have high-probability transitions, even if rarely observed

The Label Bias



- This happens in real-life settings
  - For instance, in NER, consider a label which is rare, but tends to have long names (e.g., the “continue” category of nobility names or books by E. Lockhart)

Books by E. Lockhart

The Boyfriend List: 15 Guys, 11 Shrink Appointments, 4 Ceramic Frogs and Me, Ruby Oliver (Ruby Oliver, #1)

The Boy Book: A Study of Habits and Behaviors, Plus Techniques for Taming Them (Ruby Oliver, #2)

The Treasure Map of Boys: Noel, Jackson, Finn, Hutch, Gideon—and me, Ruby Oliver (Ruby Oliver, #3)

Real Live Boyfriends: Yes. Boyfriends, Plural. If My Life Weren't Complicated, I Wouldn't Be Ruby Oliver

The Boyfriend Quartet: 15 Boys, 43 Lists, 120 Footnotes, and Too Many Panic

Attacks to Count, All in Four Novels about Ruby Oliver

Globally-normalized Models

- Newer, higher-powered discriminative sequence models
- Do not decompose training into independent local regions
  - Thus avoiding the label bias
- Often take a very long time to train - require repeated inference on training set

Sequence Conditional Random Fields (CRFs)

- CRFs are similar to MEMMs, but the normalization is global.
- This solves the label bias: the scores of the next state given the current one need not sum up to 1

$$P(y|x) = \frac{\prod_{j=1}^{n(i)} e^{f(j,y_j,y_{j-1},x)^T \cdot w}}{Z(x;w)}$$

$$Z(x;w) = \sum_{y} \prod_{j=1}^{n(i)} e^{f(j,y_j,y_{j-1},x)^T \cdot w}$$

Training in CRF

- Given training examples *($x^{(1)},x^{(2)},\ldots,x^{(N)}$)* and labels *($y^{(1)},y^{(2)},\ldots,y^{(N)}$)*, we define the conditional log-likelihood:

$$Pr(y^{(1)},\ldots,y^{(N)}|x^{(1)},\ldots,x^{(N)}) = \prod_{i=1}^{N} Pr(y^{(i)}|x^{(i)}) = \prod_{i=1}^{N} \frac{\prod_{j=1}^{n(i)} exp(f(j,y_j^{(i)},y_{j-1}^{(i)},x^{(i)})^T \cdot w)}{Z(x^{(i)};w)}$$

$$Z(x^{(i)};w) = \sum_{y} \prod_{j=1}^{n} exp(f(j,y_j,y_{j-1},x^{(i)})^T \cdot w)$$

Estimating *w*

- We use maximum likelihood :

$$LL(w) = \sum_{i=1}^{N} \Big[\sum_{j=1}^{n(i)} f(j, y_j^{(i)}, y_{j-1}^{(i)}, x^{(i)})^T \cdot w - log(Z(x^{(i)}; w)) \Big]$$

- No close formula. We use gradient-based methods:

$$\frac{\partial LL}{\partial w_k} = \sum_{i=1}^{N} \Big[\sum_{j=1}^{n(i)} f_k(j, y_j^{(i)}, y_{j-1}^{(i)}, x^{(i)}) - \boxed{\sum_{y \in T^{n(i)}} Pr(y|x^{(i)}) \cdot \sum_{j=1}^{n(i)} f_k(j, y_j, y_{j-1}, x^{(i)}) \Big]}$$

CRFs: Gradient Computation

- The second term is more tricky to compute, as it sums over an exponential number of elements:

$$\sum_{y \in T^{n(i)}} Pr(y|x^{(i)}) \cdot \sum_{j=1}^{n(i)} f_k(j, y_j, y_{j-1}, x^{(i)}) =$$

$$\sum_{j=1}^{n(i)} \sum_{y \in T^{n(i)}} Pr(y|x^{(i)}) \cdot f_k(j, y_j, y_{j-1}, x^{(i)}) =$$

$$\sum_{j=1}^{n(i)} \sum_{y_j, y_{j-1}} \boxed{Pr(y_j, y_{j-1}|x^{(i)})} \cdot f_k(j, y_j, y_{j-1}, x^{(i)})$$

CRFs: Gradient Computation

- The marginals require dynamic programming to compute: (we should compute them for every pair of values for y_j and y_{j-1}

$$Pr(y_j, y_{j-1} | x^{(i)})$$

- We can solve this with dynamic programming

CRFs: Forward-Backward Algorithm

- Goal (for every *a,b*): $$Pr(y_j = a, y_{j-1} = b | x^{(i)}) = \sum_{y \in T^{n(i)} : y_j = a, y_{j-1} = b} Pr(y|x^{(i)})$$

<u>Approach:</u> dynamic programming

$$M_i(y, y') = exp(f(i, y', y, x)^T \cdot w)$$

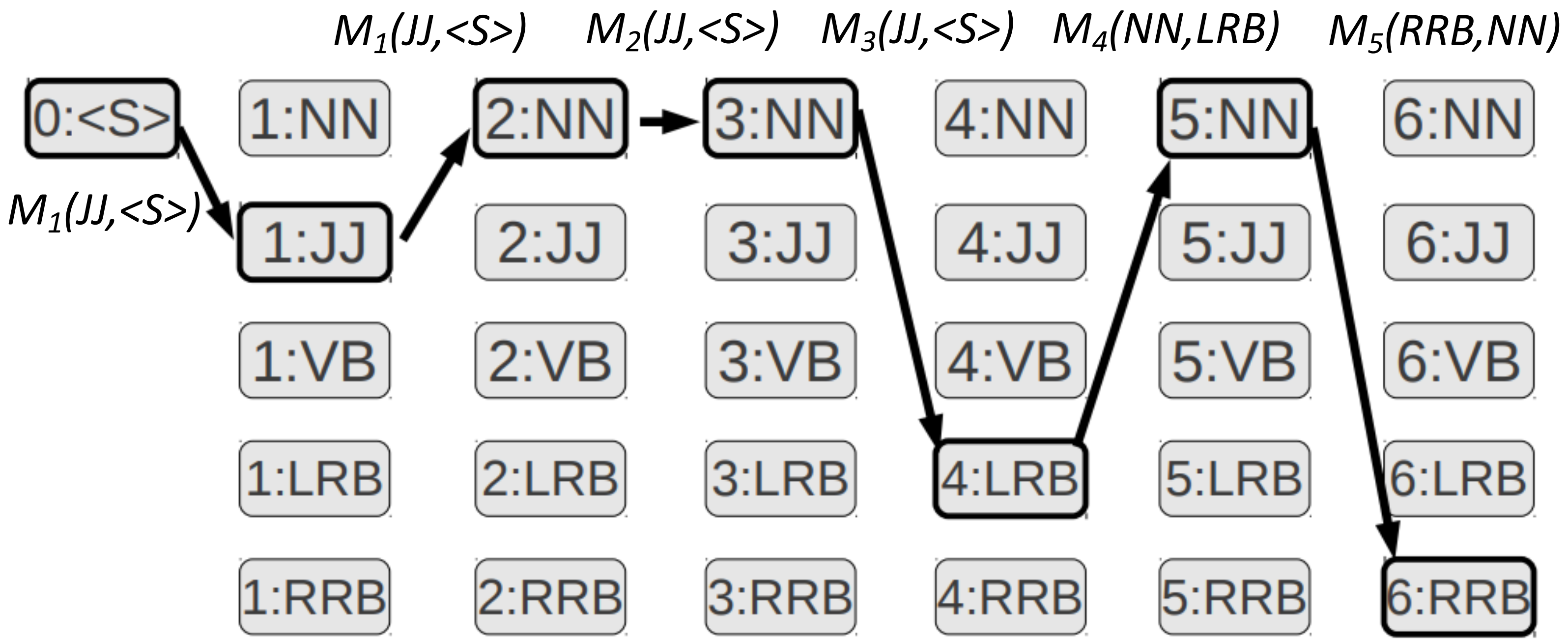
- Define:

$$\alpha_i(y) = \sum_{y_1, .., y_{i-1} = y} \prod_k M_k(y_{k-1}, y_k)$$

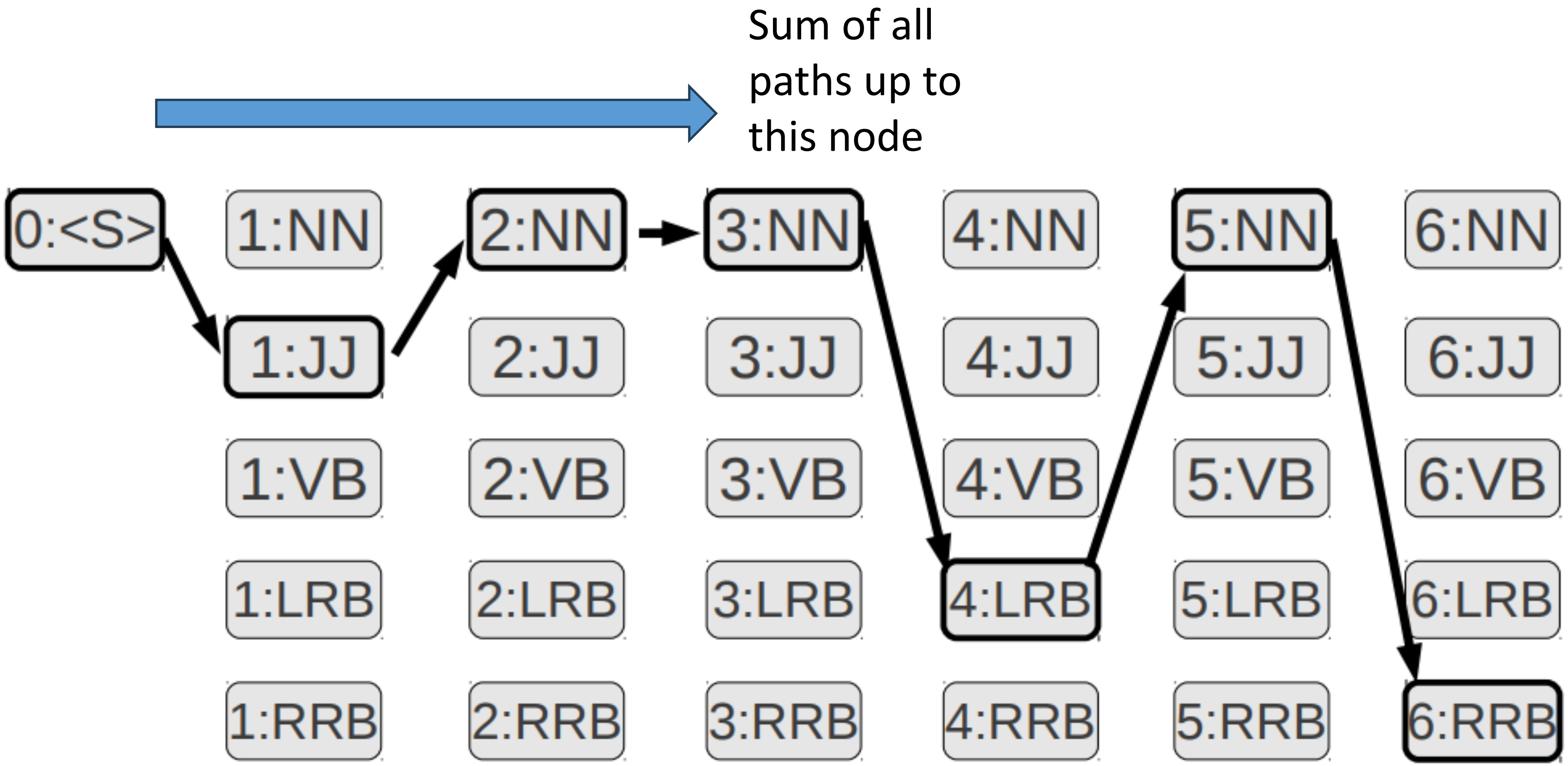
$$\beta_i(y) = \sum_{y_n, .., y_i = y} \prod_k M_k(y_k, y_{k+1})$$

Forward-Backward Algorithm in Viterbi Trellis

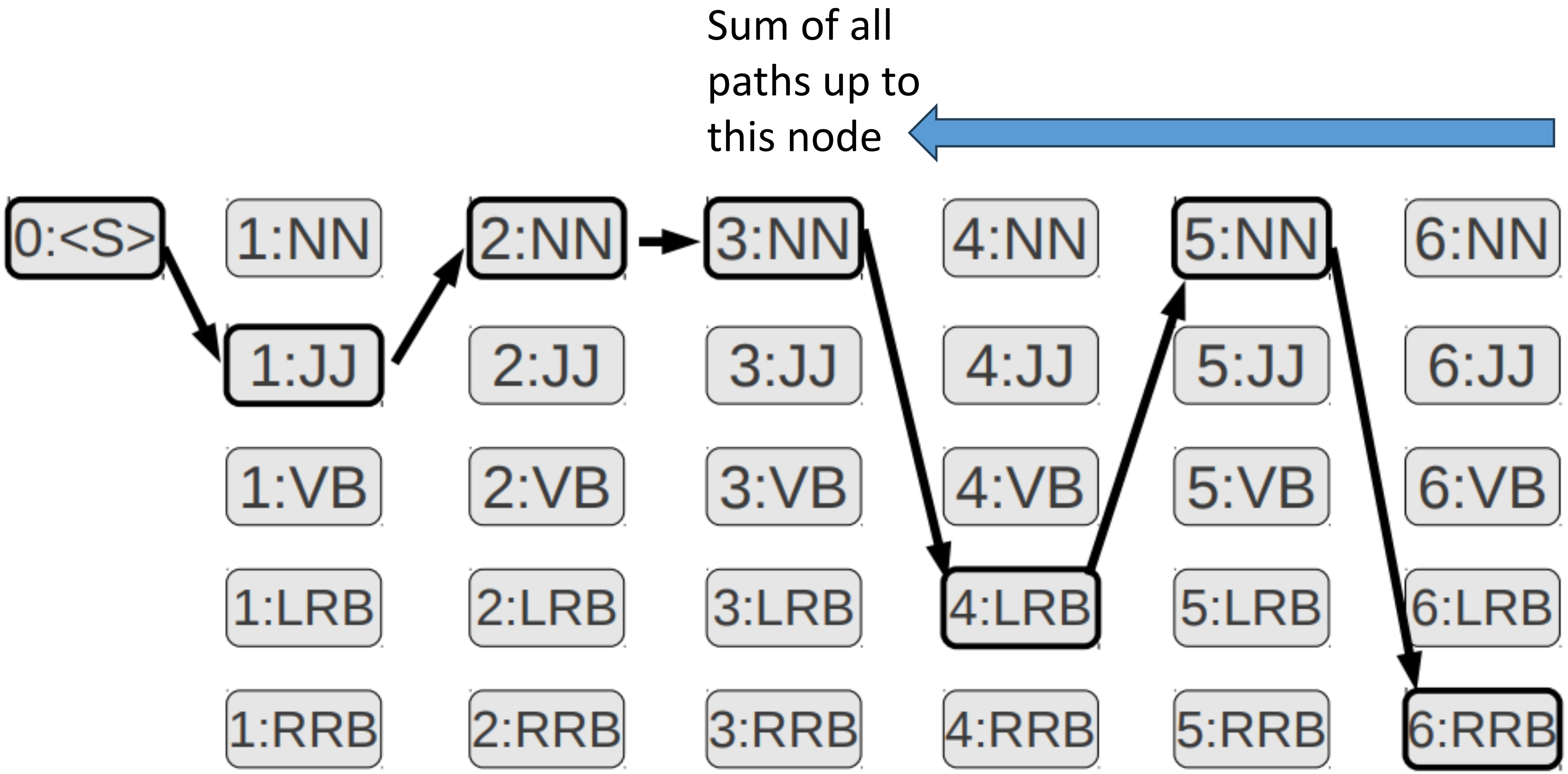
- The marginal $Pr(y_j, y_{j-1}|x^{(i)})$ is the sum over all paths that have the edge (y_{j-1}, y_j) between states j-1 and j
- Edge weights between y and y' is $M_i(y, y')$



Forward Sum (α)



Backward Sum (β)



CRFs: Forward-Backward Algorithm

- The parameters can be computed using dynamic programming with these formulas (for *i*=1, computing α, same for β with *i*=n)

$$\alpha_i(y) = \sum_{y'} M_i(y', y)\alpha_{i-1}(y') \qquad \beta_i(y) = \sum_{y'} M_{i+1}(y, y')\beta_{i+1}(y')$$

- The marginal can now be computed as:

$$Pr(y_j, y_{j-1}|x^{(i)}) = \frac{M_j(y_j, y_{j-1}) \cdot \beta_j(y_j) \cdot \alpha_j(y_{j-1})}{\boxed{Z(x^{(i)})}}$$

CRFs: Forward-Backward Algorithm

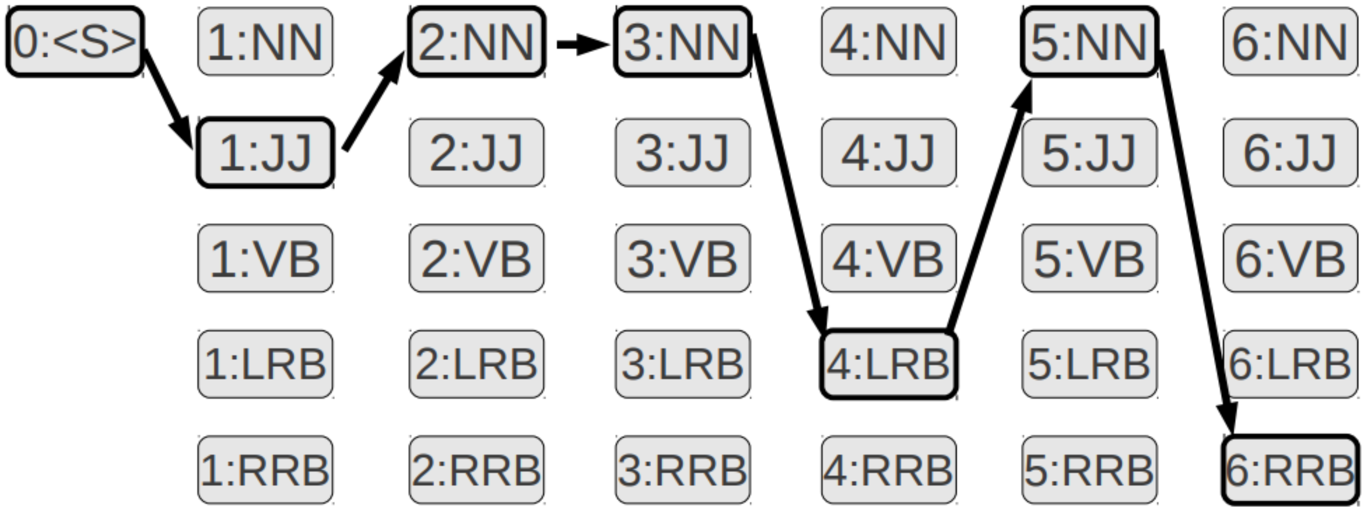
- Computing *Z(x)* can be done by summing the numerator over all possible values for y_j *and* y_{j-1}
  - A practical solution since we need to compute the marginals for every value of y_j and y_{j-1} anyway
- It can also be done directly by employing dynamic programming, or using the computed $\alpha_{n+1}(y)$:

$$Z(x) = \sum_{y} \alpha_{n+1}(y)$$

CRFs: Inference

- Inference again uses the Viterbi algorithm

- The same idea as with MEMMs. Edge weights between a pair of tags *y* and *y'* is $M_i(y, y')$



Some Accuracies on English Supervised POS Tagging

- Rough accuracies: (overall accuracy/unknown words)
  - Most frequent tag: ~90% / ~50%
  - Trigram HMM: ~95% / ~55%
  - TnT – Trigram HMM with better smoothing and handling of unknown words (Brants, 2000): 96.7% / 85.5%
  - Local MaxEnt: 96.8% / 86.8%
  - MEMM tagger: 96.8% / 86.9%
  - Structured Perceptron: 97.1% (we'll talk about this algorithm later in the course)
  - Cyclic tagger (multiplying two MEMMs) / CRFs: 97.2% / 89.0%
  - Inter-annotator agreement: ~98%

Domain Effects

- Accuracies degrade outside of domain
  - Up to triple error rate
  - Usually make the most errors on the things you care about in the domain (e.g. protein names)
- Open questions
  - How to effectively exploit unlabeled data from a new domain (what could we gain?)
  - How to best incorporate domain lexica in a principled way (e.g. UMLS specialist lexicon, ontologies)

Other Applications of CRFs

- CRFs have been used extensively in NLP, Vision and Computational Biology
- A few references: (#citations is a tricky measure, but the paper that introduced CRFs has about 13K of them..)
  - Named Entity Recognition (e.g., identifying protein names in biology papers)
  - Chinese Word Segmentation
  - Prediction of pitch accents
  - Semantic role labeling
  - Shallow Parsing
  - Syntactic Parsing

See section 2.7 in this CRF tutorial:

http://homepages.inf.ed.ac.uk/csutton/publications/crftut-fnt.pdf