CSE 517
Natural Language Processing
Winter2015

Feature Rich Models

Yejin Choi - University of Washington

[Slides from Jason Eisner, Dan Klein, Luke Zettlemoyer]

Feature Rich Models

- **Throw anything you want into the stew**
- Add a bonus for this, a penalty for that, etc.

"11,001 New Features for Statistical Machine Translation", (D. Chiang, K. Knight, and W. Wang), NAACL, 2009. Best Paper Award.

Probabilistic Models

(Unstructured) categorization:

- Naïve Bayes

Structured prediction:

- HMMs
- PCFG Models
- IBM Models

Feature-rich / (Log)-linear Models

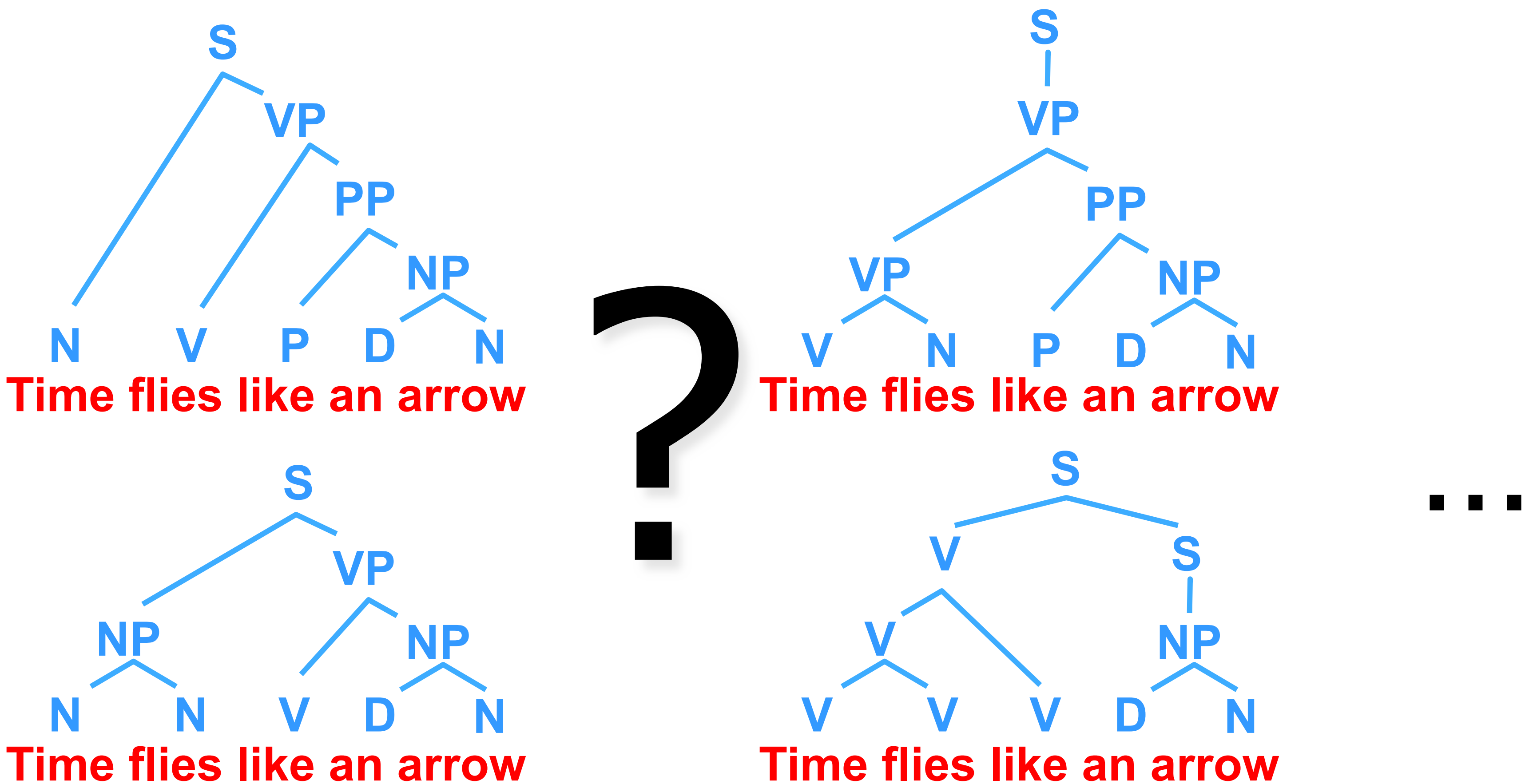
(Unstructured) categorization:

- Perceptron
- Maximum Entropy

Structured prediction:

- Perceptron for Structured Prediction
- MEMM (Maximum Entropy Markov Model)
- CRF (Conditional Random Fields)

Structured Prediction with Perceptrons and CRFs



p(category | message)

goodmail	spam
Reply today to claim your …	Reply today to claim your …
goodmail	spam
Wanna get pizza tonight?	Wanna get pizza tonight?
goodmail	spam
Thx; consider enlarging the …	Thx; consider enlarging the …
goodmail	spam
Enlarge your hidden …	Enlarge your hidden …

p(RHS | LHS)

S → NP VP

S → N VP

S → NP[+wh] V S/V/NP

S → VP NP

S → Det N

S → PP P

...

p(RHS | LHS)

S → NP VP

S → N VP

S → NP[+wh] V S/V/NP

S → VP NP

S → Det N

S → PP P

...

NP → NP VP

NP → N VP

NP → NP CP/NP

NP → VP NP

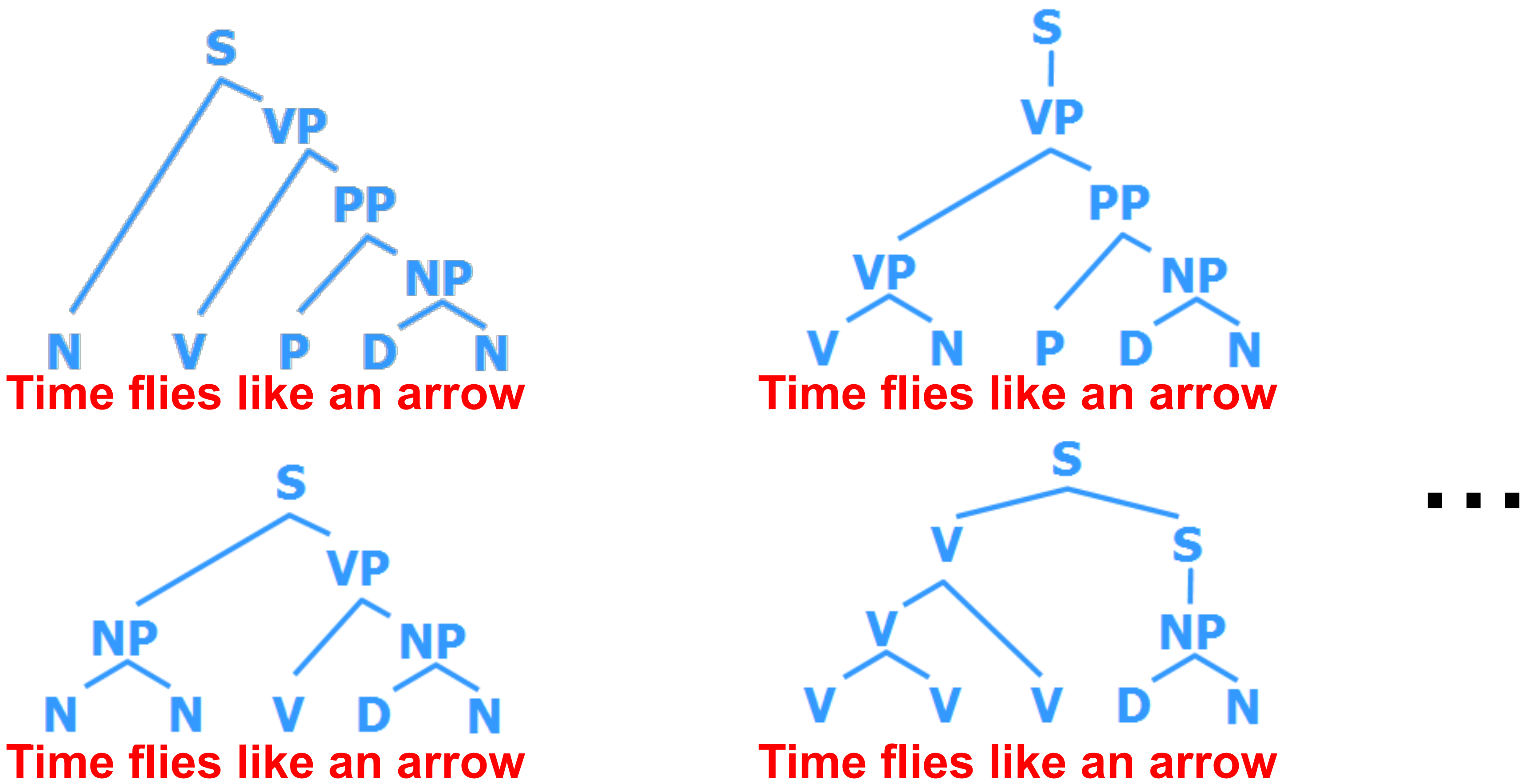
NP → Det N

NP → NP PP

...

⋮

p(parse | sentence)



p(tag sequence | word sequence)

N V P D N
Time flies like an arrow

V N P D N
Time flies like an arrow

N N V D N
Time flies like an arrow

V V V D N
Time flies like an arrow

...

Today's general problem

- Given some input x
- Consider a set of candidate outputs y
  - Classifications for x (small number: often just 2)
  - Taggings of x (exponentially many)
  - Parses of x (exponential, even infinite)
  - Translations of x (exponential, even infinite)
  - …

Structured prediction

- Want to find the "best" y, given x

Scoring by Linear Models

- Given some input x
- Consider a set of candidate outputs y
- Define a scoring function score(x,y)

Linear function: A sum of feature weights (you pick the features!)

Weight of feature k
(learned or set by hand)

$$\text{score}(x, y) = \sum_k \boxed{\theta_k} \boxed{f_k(x, y)}$$

Ranges over all features,
e.g., k=5 (numbered features)
or k="see Det Noun" (named features)

Whether (x,y) has feature k(0 or 1)
Or **how many times** it fires (≥ 0)
Or **how strongly** it fires (real #)

- Choose y that maximizes score(x,y)

Scoring by Linear Models

- Given some input x
- Consider a set of candidate outputs y
- Define a scoring function score(x,y)

  Linear function: A sum of feature weights (you pick the features!)

(learned or set by hand)

$$\mathrm{score}(x, y) = \boxed{\vec{\theta}} \cdot \vec{f}(x, y)$$

This linear decision rule is sometimes called a "perceptron."
It's a "structured perceptron" if it does structured prediction
(number of y candidates is unbounded, e.g., grows with |x|).

- Choose y that maximizes score(x,y)

Probabilistic Models

(Unstructured) categorization:

- Naïve Bayes

Structured prediction:

- HMMs
- PCFG Models
- IBM Models

Feature-rich / (Log)-linear Models

(Unstructured) categorization:

- Perceptron
- Maximum Entropy

Structured prediction:

- Perceptron for Structured Prediction
- MEMM (Maximum Entropy Markov Model)
- CRF (Conditional Random Fields)

Perceptron Training Algorithm

- initialize θ (usually to the zero vector)
- repeat:
  - Pick a training example (x,y)
  - Model predicts y* that maximizes score(x,y*)
  - Update weights by a step of size $\varepsilon > 0$:
    $$\theta = \theta + \varepsilon \cdot (f(x,y) - f(x,y^*))$$

If model prediction was correct (y=y*), θ doesn't change.

So once model predicts all training examples correctly, stop.

If some θ can do the job, this eventually happens!

(If not, θ will oscillate, but the average θ from all steps will settle down. So return that eventual average.)

Perceptron Training Algorithm

- initialize θ (usually to the zero vector)
- repeat:
  - Pick a training example (x,y)
  - Model predicts y* that maximizes score(x,y*)
  - Update weights by a step of size $\varepsilon > 0$:

$$\theta = \theta + \varepsilon \cdot (f(x,y) - f(x,y^*))$$

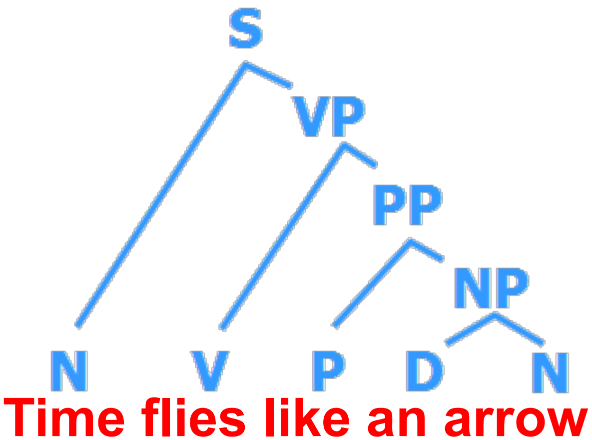
If model prediction was wrong ($y \neq y^*$), then we must have score(x,y) ≤ score(x,y*) instead of > as we want.

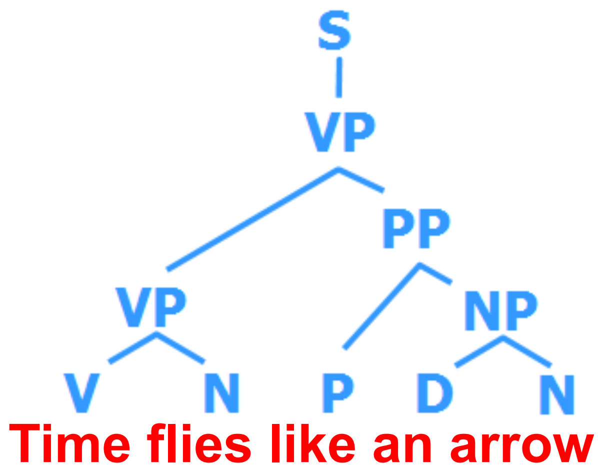
Equivalently, $\theta \cdot f(x,y) \leq \theta \cdot f(x,y^*)$

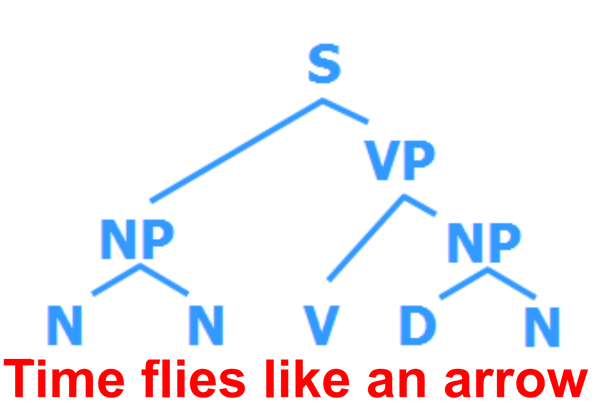
Equivalently, $\theta \cdot (f(x,y) - f(x,y^*)) \leq 0$ but we want it positive.

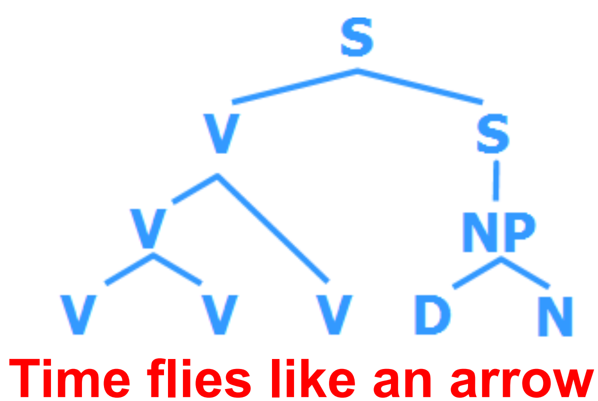
Our update <u>increases</u> it (by $\varepsilon \cdot \| f(x,y) - f(x,y^*) \|^2 \geq 0$)

~~p(parse | sentence)~~ score(sentence, parse)









...

Nuthin' but adding weights

- n-grams: … + log p(w7 | w5, w6) + log p(w8 | w6, w7) + …
- PCFG: log p(NP VP | S) + log p(Papa | NP) + log p(VP PP | VP) …
- HMM tagging: … + log p(t7 | t5, t6) + log p(w7 | t7) + …
- Noisy channel: [log p(source)] + [log p(data | source)]
- Cascade of composed FSTs:

  [log p(A)] + [log p(B | A)] + [log p(C | B)] + …
- Naïve Bayes:

  log p(Class) + log p(feature1 | Class) + log p(feature2 | Class) …

What if our weights were arbitrary real numbers?

Change log p(this | that) to θ(this ; that)

- n-grams: … + log p(w7 | w5, w6) + log p(w8 | w6, w7) + …
- PCFG: log p(NP VP | S) + log p(Papa | NP) + log p(VP PP | VP) …
- HMM tagging: … + log p(t7 | t5, t6) + log p(w7 | t7) + …
- Noisy channel: [log p(source)] + [log p(data | source)]
- Cascade of FSTs:
  [log p(A)] + [log p(B | A)] + [log p(C | B)] + …
- Naïve Bayes:
  log p(Class) + log p(feature1 | Class) + log p(feature2 | Class) …

What if our weights were arbitrary real numbers?

Change log p(this | that) to θ(this ; that)

- n-grams: … + θ(w7 ; w5, w6) + θ(w8 ; w6, w7) + …
- PCFG: θ(NP VP ; S) + θ(Papa ; NP) + θ(VP PP ; VP) …
- HMM tagging: … + θ(t7 ; t5, t6) + θ(w7 ; t7) + …
- Noisy channel: [θ(source)] + [θ(data ; source)]
- Cascade of FSTs:
  [θ(A)] + [θ(B ; A)] + [θ(C ; B)] + …
- Naïve Bayes:
  θ(Class) + θ(feature1 ; Class) + θ(feature2 ; Class) …

In practice, θ is a hash table
Maps from feature name (a string or object) to feature weight (a float)
e.g., θ(NP VP ; S) = weight of the S → NP VP rule, say -0.1 or +1.3

What if our weights were arbitrary real numbers?

Change log p(this | that) to θ(this ; that)

- n-grams: ... + θ(w5 w6 w7) + θ(w6 w7 w8) + ...
- WCFG ~~PCFG:~~ θ(S → NP VP) + θ(NP → Papa) + θ(VP → VP PP) ...
- HMM tagging: ... + θ(t5 t6 t7) + θ(t7 → w7) + ...
- Noisy channel: [θ(source)] + [θ(source, data)]
- Cascade of FSTs:
  [θ(A)] + [θ(A, B)] + [θ(B, C)] + ...
- ~~Naïve Bayes:~~ (multi-class) logistic regression
  θ(Class) + θ(Class, feature 1) + θ(Class, feature2) ...

Finding the best y given x

- At **both** training & test time, given input x, perceptron picks y that maximizes score(x,y)

$$\text{score}(x,y) = \sum_k \theta_k f_k(x,y)$$

- How do we find argmax_y score(x,y)?
  - Easy when only a few candidates y (e.g., text classification)
    - Just try each y in turn.
  - Harder for structured prediction: but you now know how!
    - Find the best string, path, or tree …
    - Viterbi for HMM, CKY for trees, stack decoding for MT
    - Dynamic programming if possible

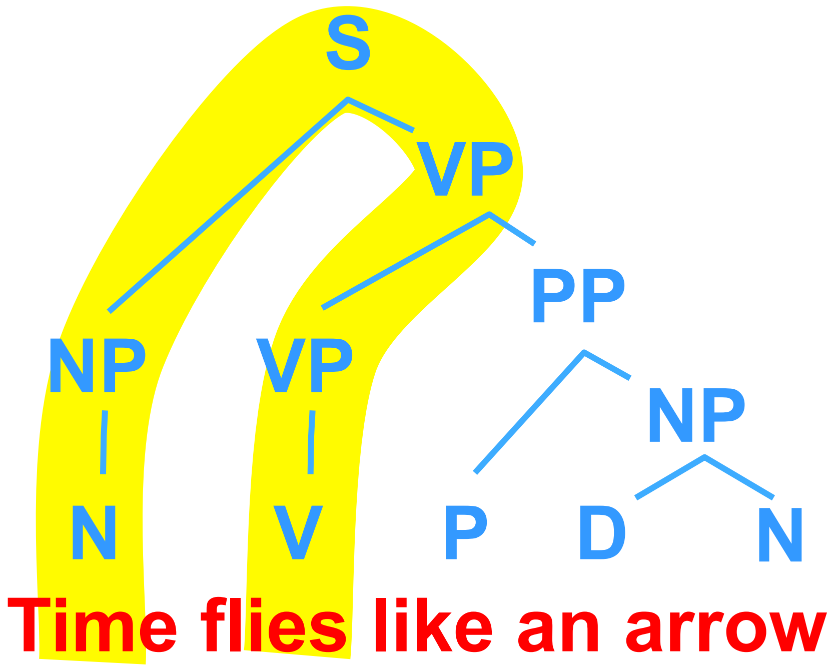
Why would we switch from probabilities to scores?

1. "Discriminative" training (e.g., perceptron) might work better.
   - It tries to optimize weights to actually predict the right y for each x.
   - More important than maximizing log p(x,y) = log p(y|x) + log p(x), as we've been doing in HMMs and PCFGs.
   - Satisfied once the right y wins. The example puts no more pressure on the weights to raise log p(y|x). And <u>never</u> pressures us to raise log p(x).

2. Having more freedom in the weights might help?
   - Now weights can be positive or negative.
   - Exponentiated weights no longer have to sum to 1.
   - <u>But</u> turns out new θ vectors can't do more than the old restricted ones.
     - Roughly, for every WCFG there's an equivalent PCFG.
     - Though it's true a regularizer might favor one of the new ones.

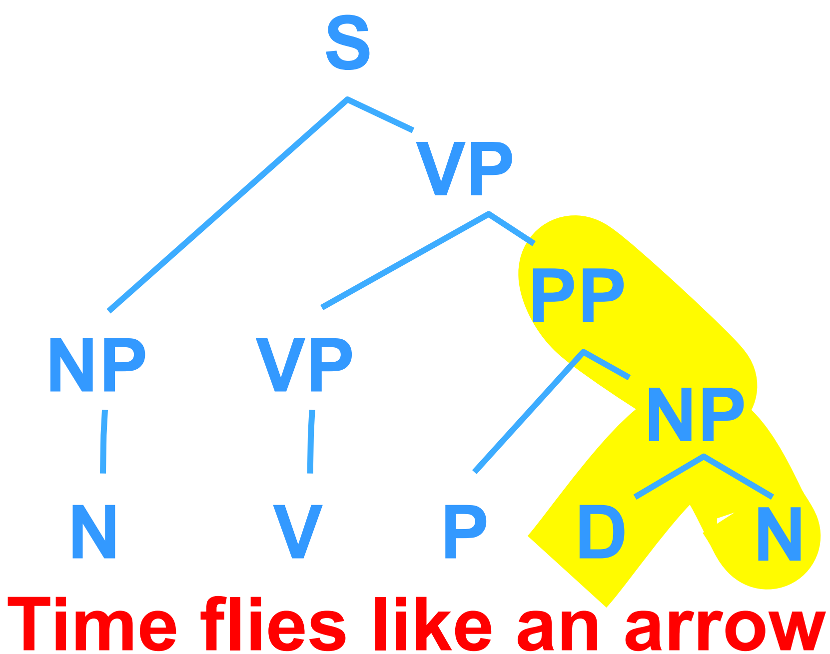
3. We can throw lots more features into the stewpot.
   - Allows model to capture more of the useful predictive patterns!
   - **So, what features can we throw in <u>efficiently</u>?**

Cross-rule substructures



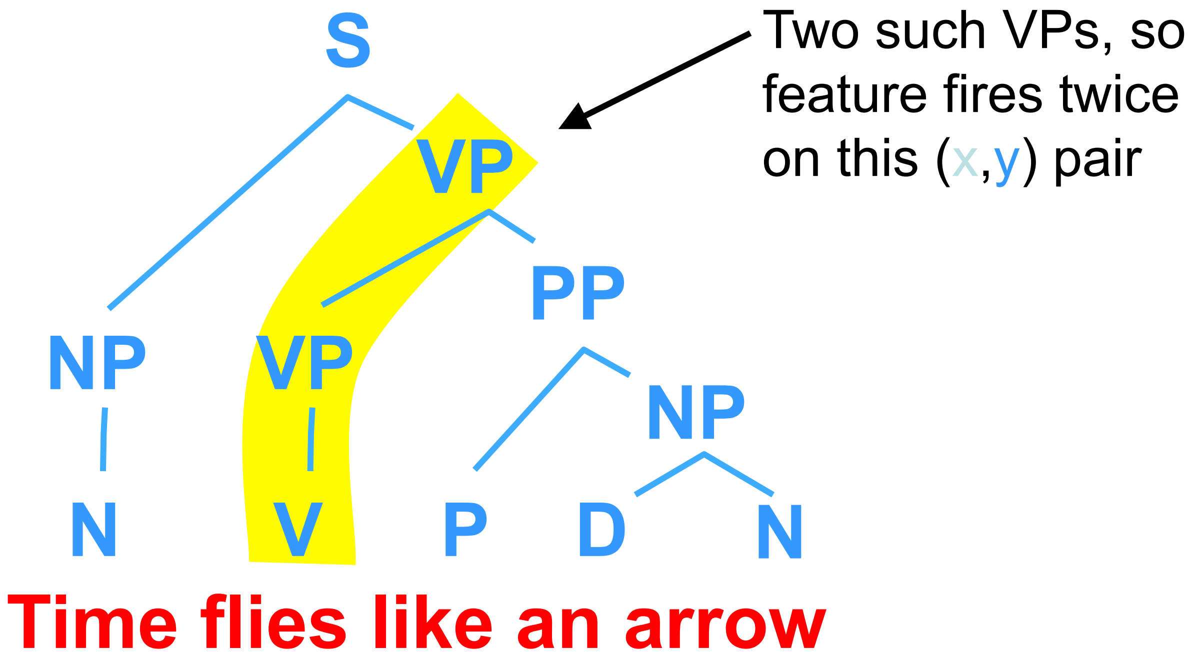
- Count of “flies” as a verb with subject “time”

Cross-rule substructures



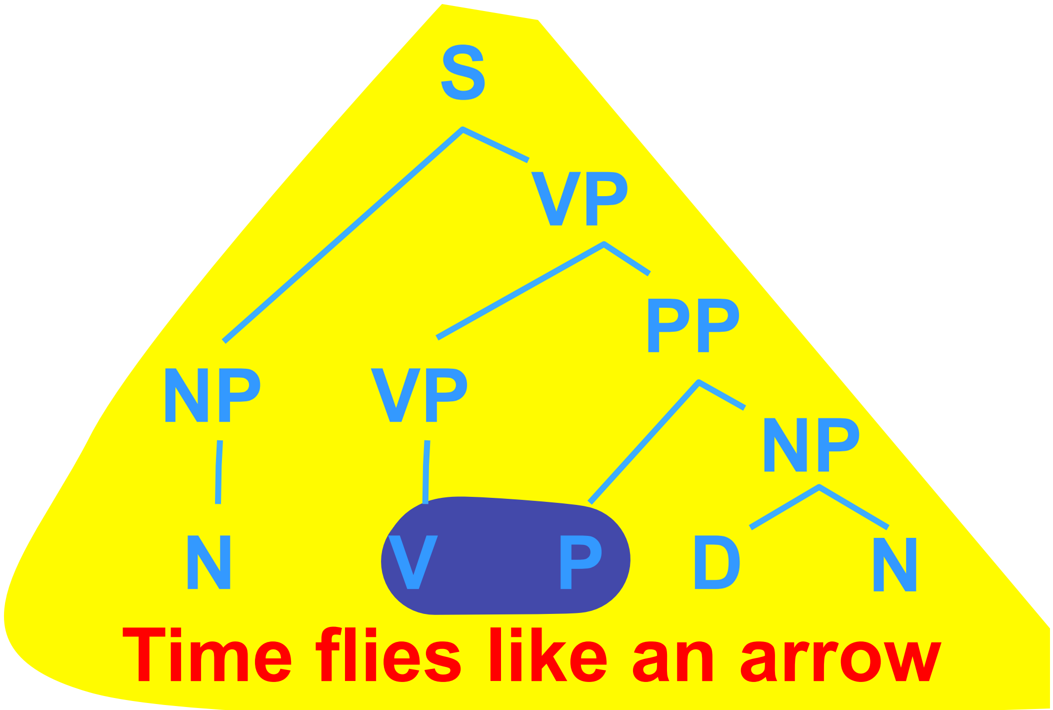
- Count of “flies” as a verb with subject “time”
- Count of NP → D N when the NP is the object of a preposition

Cross-rule substructures



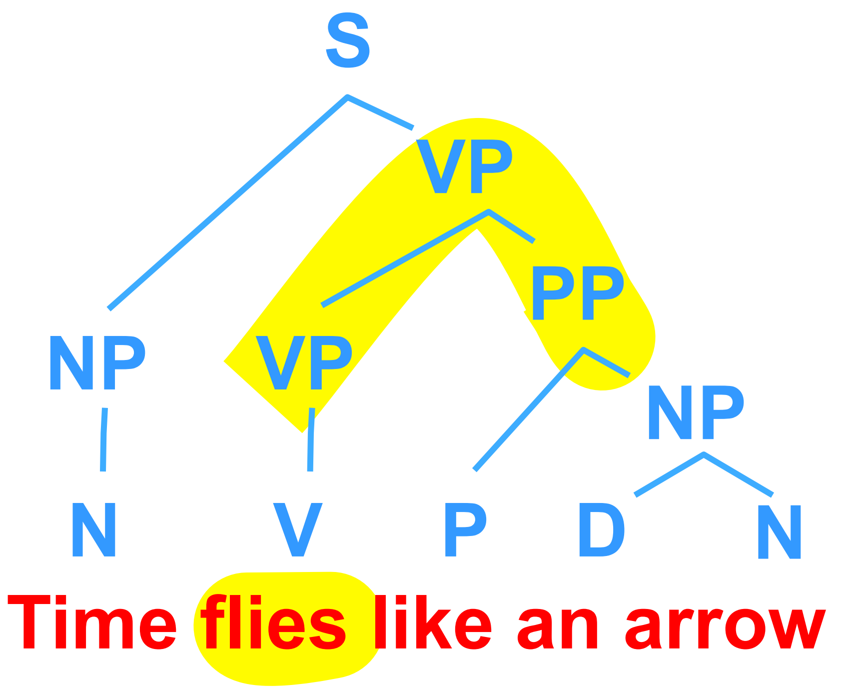
- Count of "flies" as a verb with subject "time"
- Count of NP → D N when the NP is the object of a preposition
- Count of VPs that contain a V

Global features



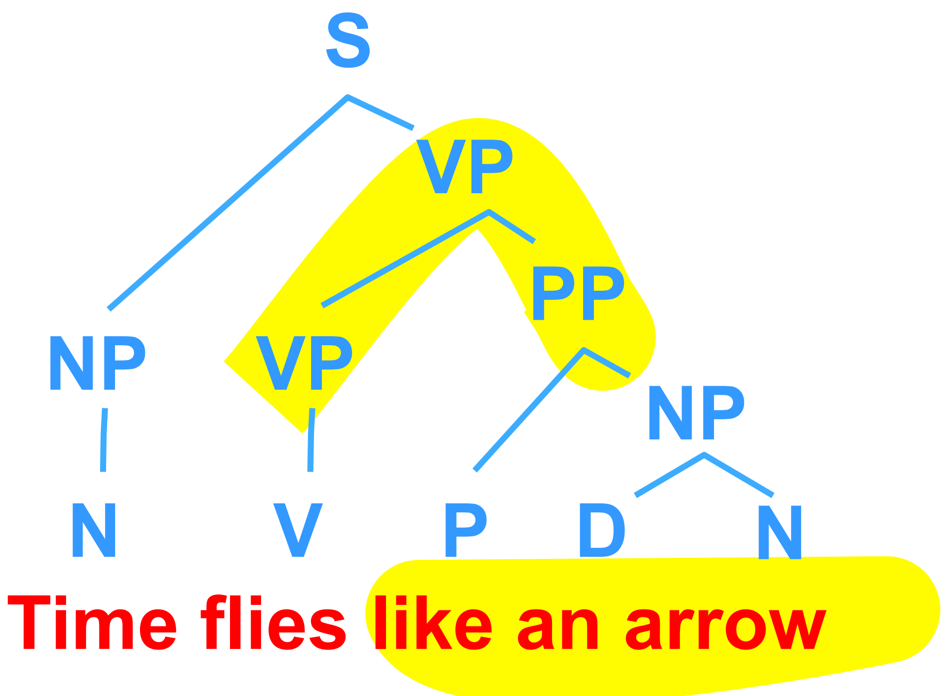
- Count of “NP and NP” when the two NPs have very different size or structure [this feature has weight < 0]
- The number of PPs is even
- The depth of the tree is prime ☺
- Count of the tag bigram V P in the preterminal seq

Context-specific features



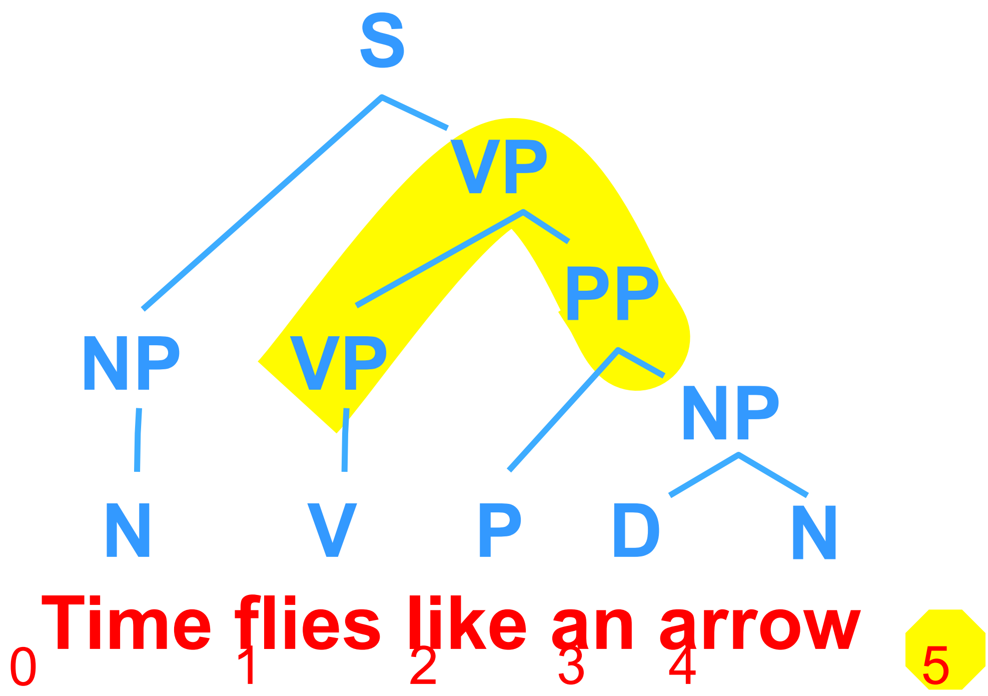
- Count of VP → VP PP whose first word is "flies"

Context-specific features



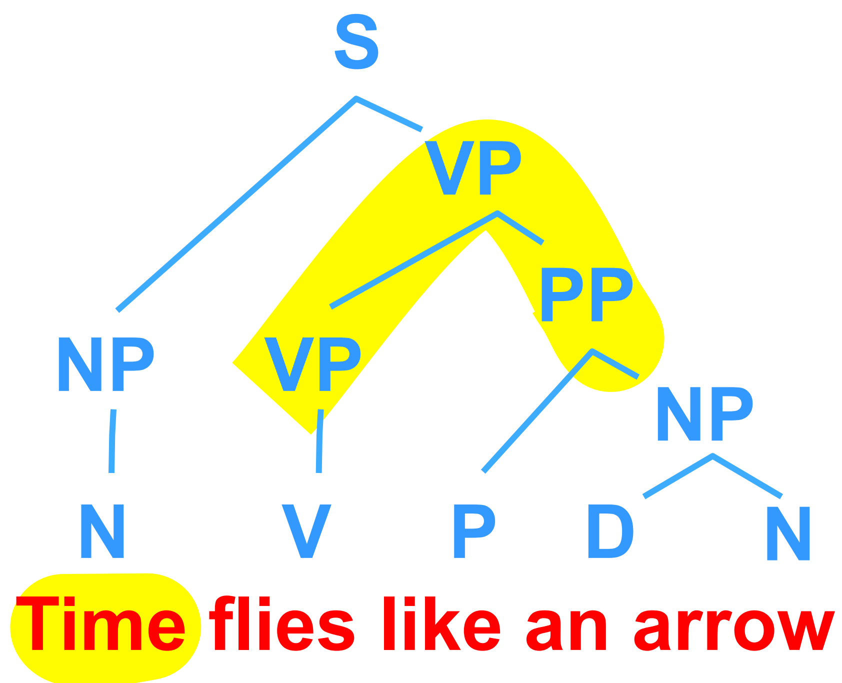
- Count of VP → VP PP whose first word is "flies"
- Count of VP → VP PP whose right child has width 3

Context-specific **features**



- Count of VP → VP PP whose first word is "flies"
- Count of VP → VP PP whose right child has width 3
- Count of VP → VP PP at the end of the input

Context-specific features



- Count of VP → VP PP whose first word is "flies"
- Count of VP → VP PP whose right child has width 3
- Count of VP → VP PP at the end of the input
- Count of VP → VP PP right after a capitalized word

In the case of tagging …

N V P D N

Time flies like an arrow

- Count of tag P as the tag for “like”
- Count of tag P
- Count of tag P in the middle third of the sentence
- Count of tag bigram V P
- Count of tag bigram V P followed by “an”
- Count of tag bigram V P where P is the tag for “like”
- Count of tag bigram V P where both words are lowercase

Overview: POS tagging Accuracies

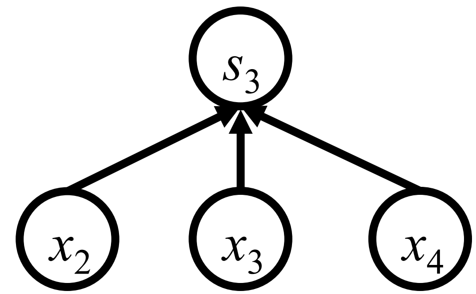
- Roadmap of (known / unknown) accuracies:
  - Most freq tag: ~90% / ~50%
  - Trigram HMM: ~95% / ~55%
  - TnT (HMM++): 96.2% / 86.0%

  - What if feature-rich classifier that predicts each POS tag one at a time?

  - Upper bound: ~98%

What about better features?

- Choose the most common tag
  - 90.3% with a bad unknown word model
  - 93.7% with a good one



- 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.)
  - Phrasal verb in sentence?   put …… __
  - Conjunctions of these things

- Uses lots of features: > 200K

Probabilistic Models

(Unstructured) categorization:

- Naïve Bayes

Structured prediction:

- HMMs
- PCFG Models
- IBM Models

Feature-rich / (Log)-linear Models

(Unstructured) categorization:

- Perceptron
- Maximum Entropy

Structured prediction:

- Perceptron for Structured Prediction
- MEMM (Maximum Entropy Markov Model)
- CRF (Conditional Random Fields)

Maximum Entropy (MaxEnt) Models

- Also known as "Log-linear" Models (*linear if you take log*)

$$P(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}))}{\sum_{\mathbf{y}'} \exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}'))}$$

- The feature vector representation may include redundant and overlapping features

Training MaxEnt Models

- Maximizing the likelihood of the training data incidentally maximizes the entropy (hence "maximum entropy")

$$\mathsf{P}(\mathbf{y}|\mathbf{x},\mathbf{w}) = \frac{\exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}))}{\sum_{\mathbf{y}'} \exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}'))}$$

Make positive

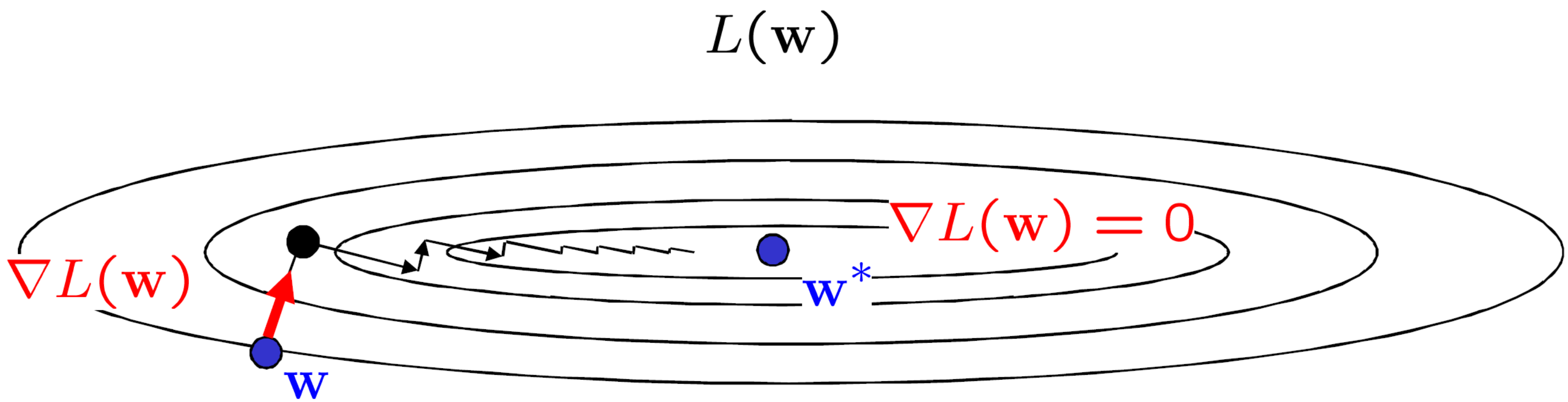
Normalize

- Maximize the (log) conditional likelihood of training data

$$L(\mathbf{w}) = \log \prod_i \mathsf{P}(\mathbf{y}^i|\mathbf{x}^i,\mathbf{w}) = \sum_i \log \left(\frac{\exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i))}{\sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}))} \right)$$

$$= \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right)$$

Convex Optimization for Training



- The likelihood function is convex. (can get global optimum)
- Many optimization algorithms/software available.
  - Gradient ascent (descent), Conjugate Gradient, L-BFGS, etc
- All we need are:

  (1) evaluate the function at current 'w'

  (2) evaluate its derivative at current 'w'

Training MaxEnt Models

$$L(\mathbf{w}) = \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right)$$

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}_n} = \sum_i \left(\mathbf{f}_i(\mathbf{y}^i)_n - \sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}_i) \mathbf{f}_i(\mathbf{y})_n \right)$$

Total count of feature n in correct candidates

Expected count of feature n in predicted candidates

Training with Regularization

$$L(\mathbf{w}) = -k||\mathbf{w}||^2 + \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right)$$

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}_n} = -2k\mathbf{w}_n + \sum_i \left(\mathbf{f}_i(\mathbf{y}^i)_n - \sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}_i)\mathbf{f}_i(\mathbf{y})_n \right)$$

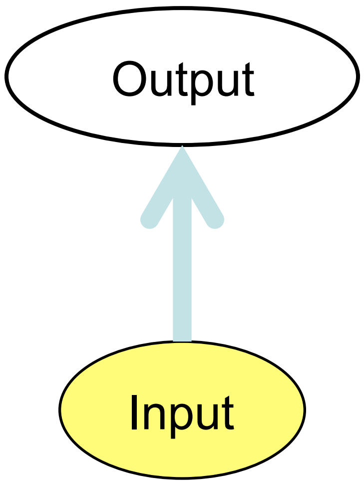
Big weights are bad

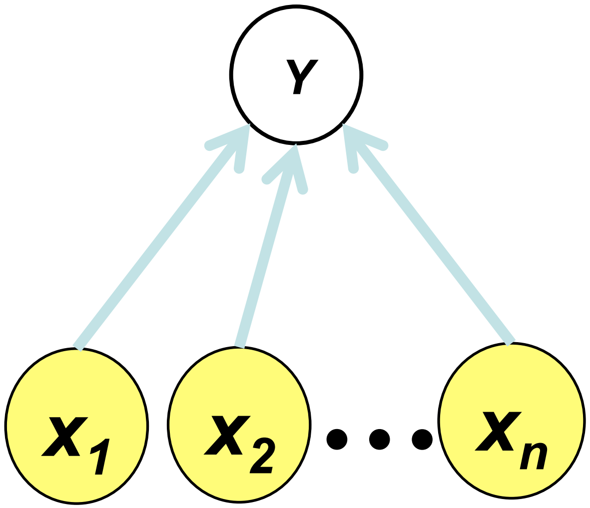
Expected count of feature n in predicted candidates

Total count of feature n in correct candidates

Graphical Representation of MaxEnt

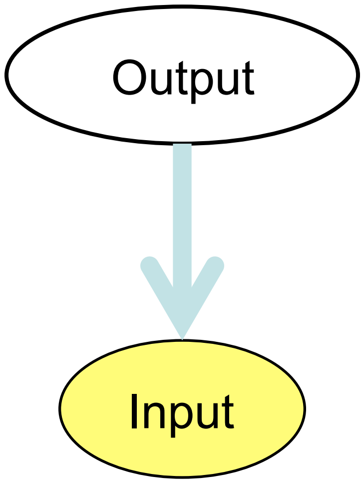
$$P(\mathbf{y}|\mathbf{x},\mathbf{w}) = \frac{\exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}))}{\sum_{\mathbf{y}'} \exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}'))}$$

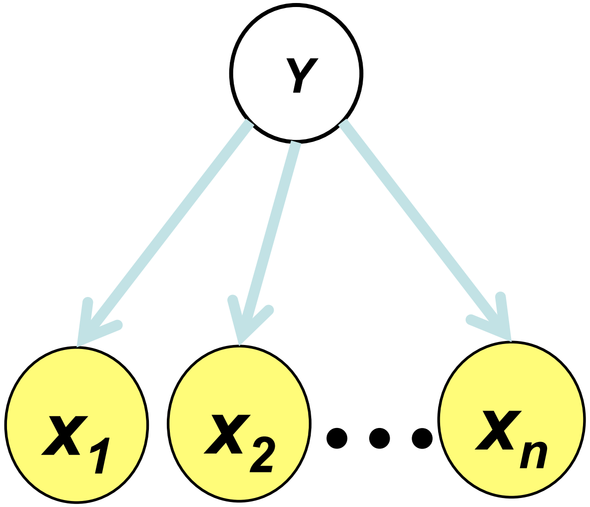




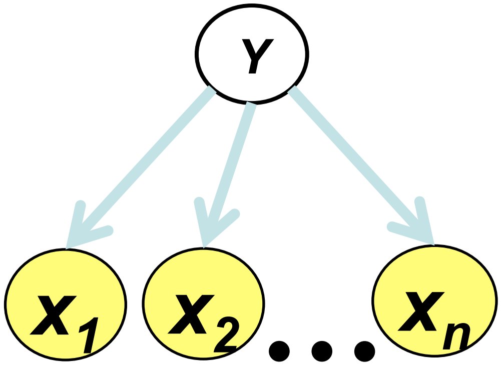
Graphical Representation of **Naïve Bayes**

$$P(X \mid Y) = \prod_{j=1} P(x_j \mid Y)$$

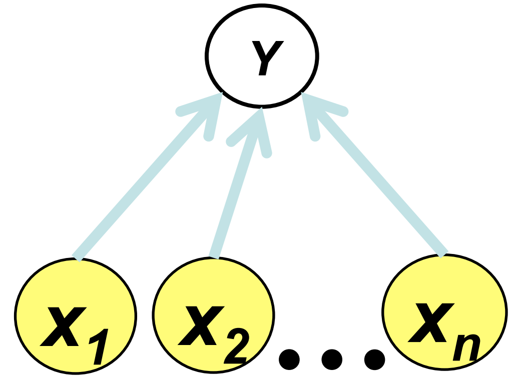




Naïve Bayes



MaxEnt



Naïve Bayes Classifier	Maximum Entropy Classifier
"***Generative***" models ➔ p(input | output) ➔ For instance, for text categorization, P(words | category) ➔ Unnecessary efforts on generating input	"***Discriminative***" models ➔ p(output | input) ➔ For instance, for text categorization, P(category | words) ➔ Focus directly on predicting the output
➔ Independent assumption among input variables: Given the category, each word is generated independently from other words (too strong assumption in reality!) ➔ Cannot incorporate arbitrary/redundant/ overlapping features	➔ By conditioning on the entire input, we don't need to worry about the independent assumption among input variables ➔ Can incorporate arbitrary features: redundant and overlapping features

Overview: POS tagging Accuracies

- Roadmap of (known / unknown) accuracies:
  - Most freq tag: ~90% / ~50%
  - Trigram HMM: ~95% / ~55%
  - TnT (HMM++): 96.2% / 86.0%
  - Maxent $P(s_i|x)$: 96.8% / 86.8%

  - Q: What does this say about sequence models?
  - Q: How do we add more features to our sequence models?

  - Upper bound: ~98%

Probabilistic Models

(Unstructured) categorization:

- Naïve Bayes

Structured prediction:

- HMMs
- PCFG Models
- IBM Models

Feature-rich / (Log)-linear Models

(Unstructured) categorization:

- Perceptron
- Maximum Entropy

Structured prediction:

- Perceptron for Structured Prediction
- MEMM (Maximum Entropy Markov Model)
- CRF (Conditional Random Fields)

MEMM Taggers

- One step up: also condition on previous tags

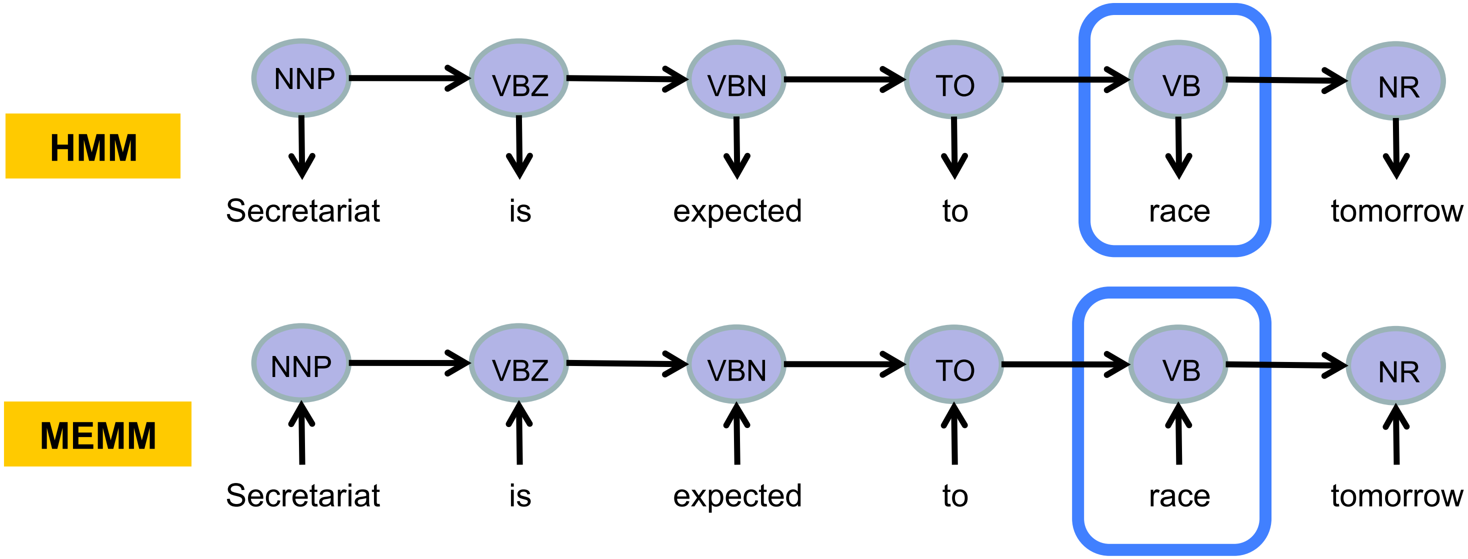
$$p(s_1 \dots s_m | x_1 \dots x_m) = \prod_{i=1}^{m} p(s_i | s_1 \dots s_{i-1}, x_1 \dots x_m)$$

$$= \prod_{i=1}^{m} p(s_i | s_{i-1}, x_1 \dots x_m)$$

  - Train up $p(s_i|s_{i-1},x_1..x_m)$ as a discrete log-linear (maxent) model, then use to score sequences

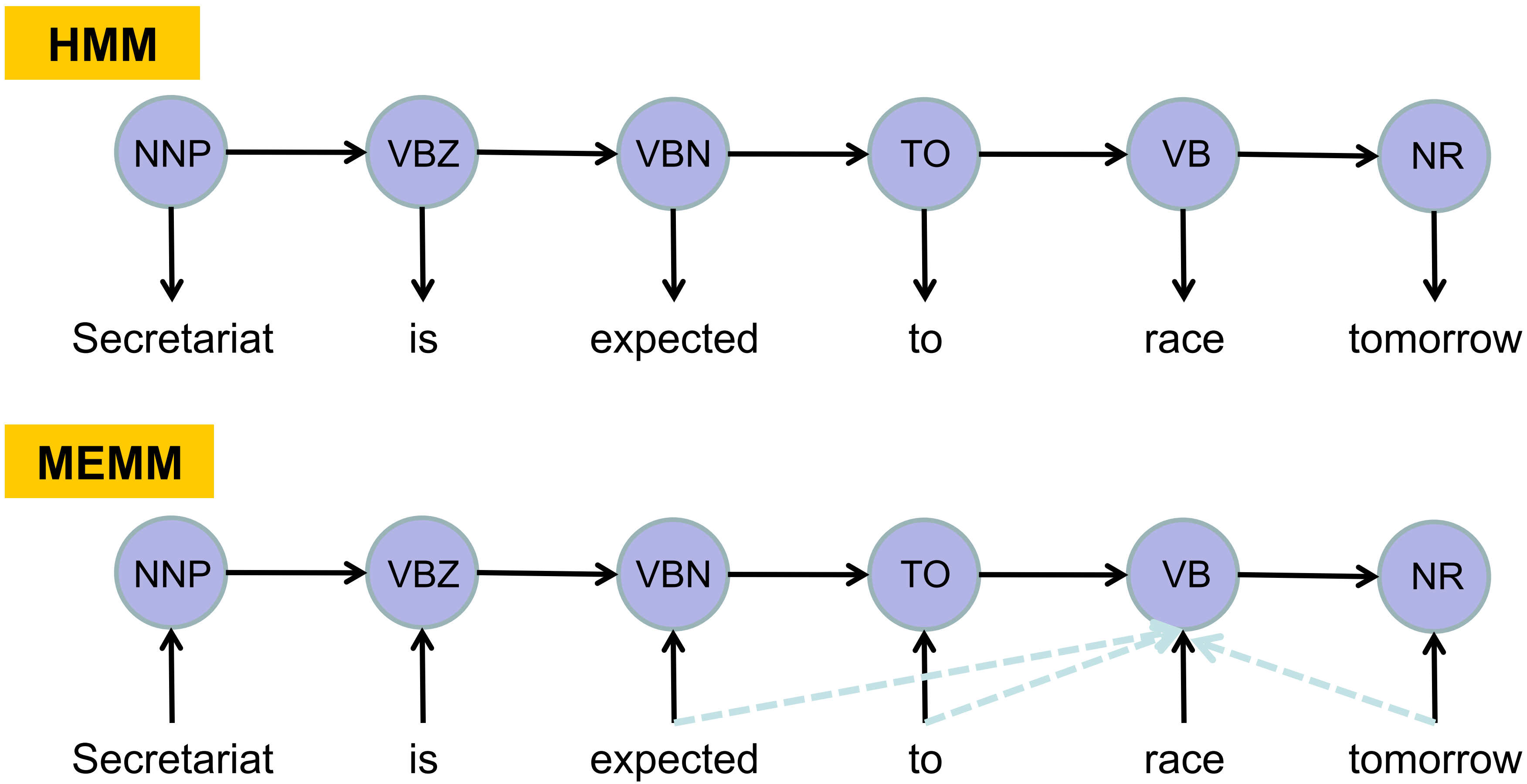
$$p(s_i | s_{i-1}, x_1 \dots x_m) = \frac{\exp\left(w \cdot \phi(x_1 \dots x_m, i, s_{i-1}, s_i)\right)}{\sum_{s'} \exp\left(w \cdot \phi(x_1 \dots x_m, i, s_{i-1}, s')\right)}$$

  - This is referred to as an MEMM tagger [Ratnaparkhi 96]
  - Beam search effective! (Why?)
  - What's the advantage of beam size 1?

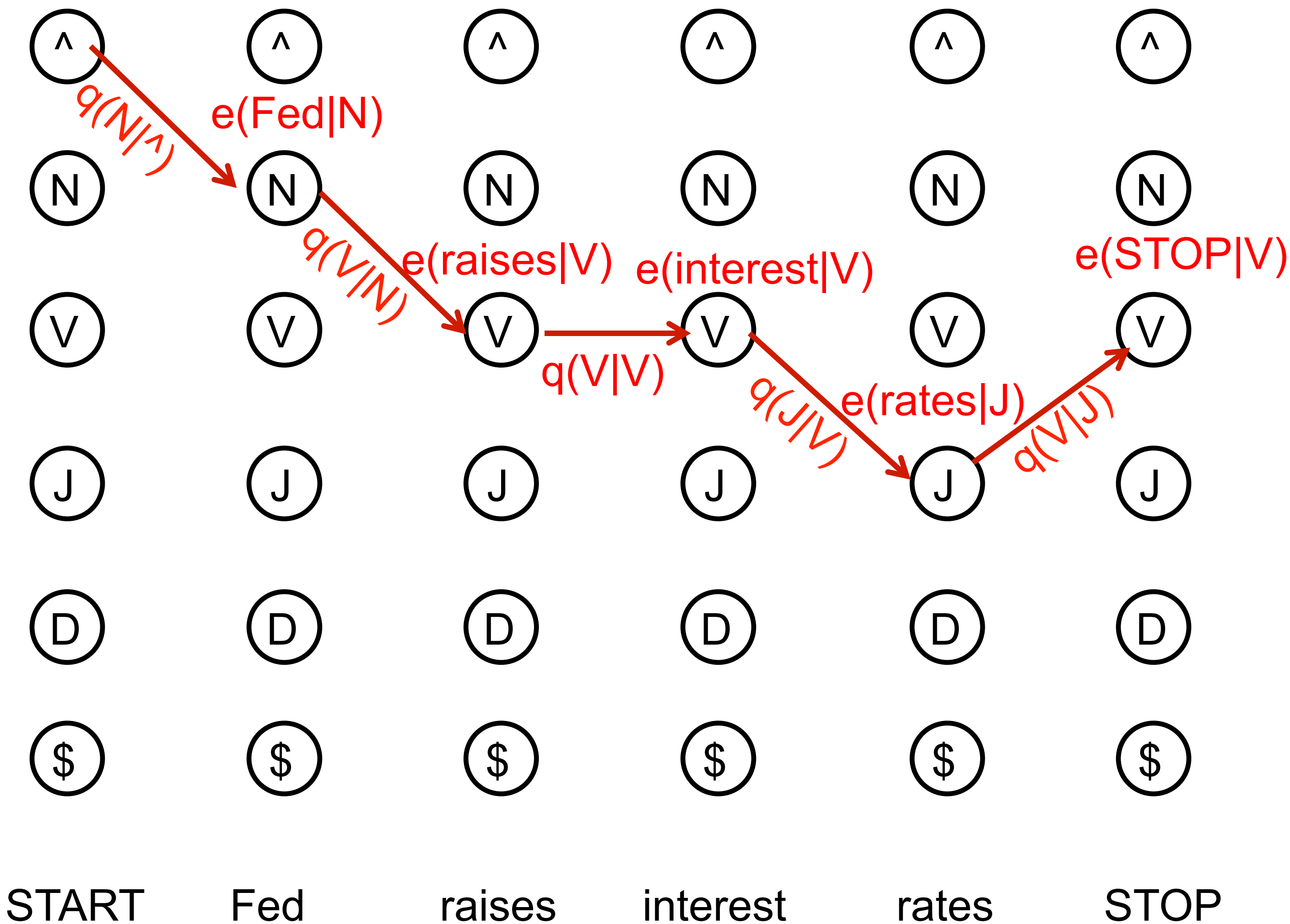


HMM	MEMM
“Generative” models ➔ joint probability **p(words, tags)** ➔“generate” input (in addition to tags) ➔ but we need to predict tags, not words!	“Discriminative” or “Conditional” models ➔ conditional probability **p(tags | words)** ➔“condition” on input ➔ Focusing only on predicting tags
Probability of each slice = emission * transition = p(word_i | tag_i) * p(tag_i | tag_i-1) = ➔ Cannot incorporate long distance features	Probability of each slice = p(tag_i | tag_i-1, word_i) or p(tag_i | tag_i-1, all words) ➔ Can incorporate long distance features

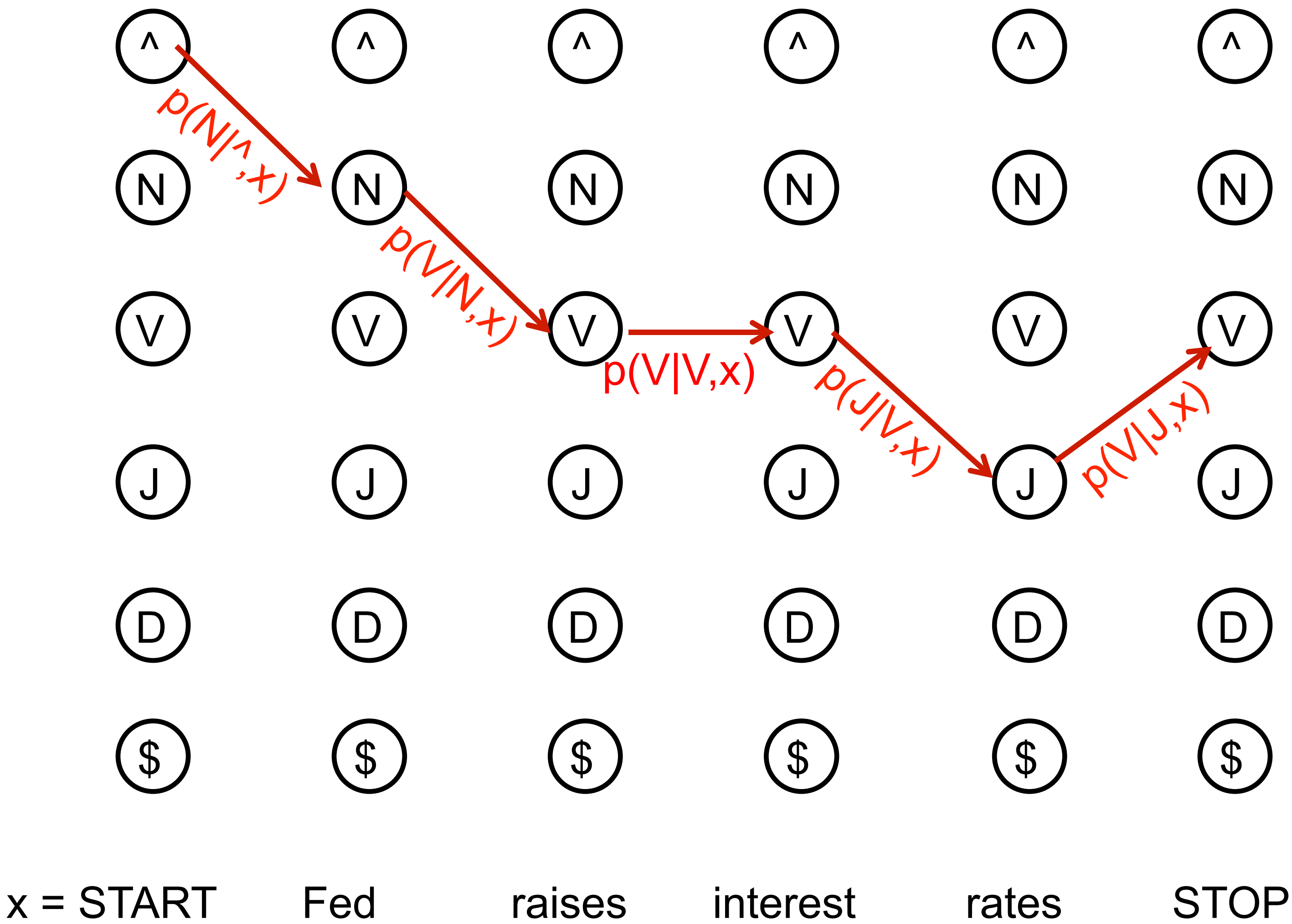
HMM v.s. MEMM


HMM
NNP
VBZ
VBN
TO
VB
NR
Secretariat
is
expected
to
race
tomorrow
MEMM
NNP
VBZ
VBN
TO
VB
NR
Secretariat
is
expected
to
race
tomorrow

The HMM State Lattice / Trellis (repeat slide)



The MEMM State Lattice / Trellis



Decoding: $p(s_1 \ldots s_m | x_1 \ldots x_m) = \prod_{i=1}^{m} p(s_i | s_1 \ldots s_{i-1}, x_1 \ldots x_m)$

- Decoding maxent taggers:
  - Just like decoding HMMs
  - Viterbi, beam search, posterior decoding
- Viterbi algorithm (HMMs):
  - Define $\pi(i,s_i)$ to be the max score of a sequence of length i ending in tag s_i

$$\pi(i, s_i) = \max_{s_{i-1}} e(x_i | s_i) q(s_i | s_{i-1}) \pi(i-1, s_{i-1})$$

- Viterbi algorithm (Maxent):
  - Can use same algorithm for MEMMs, just need to redefine $\pi(i,s_i)$!

$$\pi(i, s_i) = \max_{s_{i-1}} p(s_i | s_{i-1}, x_1 \ldots x_m) \pi(i-1, s_{i-1})$$

Overview: Accuracies

- Roadmap of (known / unknown) accuracies:
  - Most freq tag: ~90% / ~50%
  - Trigram HMM: ~95% / ~55%
  - TnT (HMM++): 96.2% / 86.0%
  - Maxent $P(s_i|x)$: 96.8% / 86.8%
  - MEMM tagger: 96.9% / 86.9%

  - Upper bound: ~98%

Global Discriminative Taggers

- Newer, higher-powered discriminative sequence models
  - CRFs (also perceptrons, M3Ns)
  - Do not decompose training into independent local regions
  - Can be deathly slow to train – require repeated inference on training set
- Differences can vary in importance, depending on task
- However: one issue worth knowing about in local models
  - "**Label bias**" and other **explaining away effects**
  - MEMM taggers' local scores can be near one without having both good "transitions" and "emissions"
  - This means that often evidence doesn't flow properly
  - Why isn't this a big deal for POS tagging?
  - Also: in decoding, condition on predicted, not gold, histories

Probabilistic Models

(Unstructured) categorization:

- Naïve Bayes

Structured prediction:

- HMMs
- PCFG Models
- IBM Models

Feature-rich / (Log)-linear Models

(Unstructured) categorization:

- Perceptron
- Maximum Entropy

Structured prediction:

- Perceptron for Structured Prediction
- MEMM (Maximum Entropy Markov Model)
- CRF (Conditional Random Fields)

Linear Models: Perceptron

[Collins 02]

- The perceptron algorithm
  - Iteratively processes the training set, reacting to training errors
  - Can be thought of as trying to drive down training error
- The (online) perceptron algorithm:
  - Start with zero weights
  - Visit training instances (x_i, y_i) one by one
    - Make a prediction

Sentence: $x = x_1 \ldots x_m$

$$y^* = \arg\max_y w \cdot \phi(x_i, y)$$

Tag Sequence: $y = s_1 \ldots s_m$

    - If correct ($y^* == y_i$): no change, goto next example!
    - If wrong: adjust weights

$$w = w + \phi(x_i, y_i) - \phi(x_i, y^*)$$

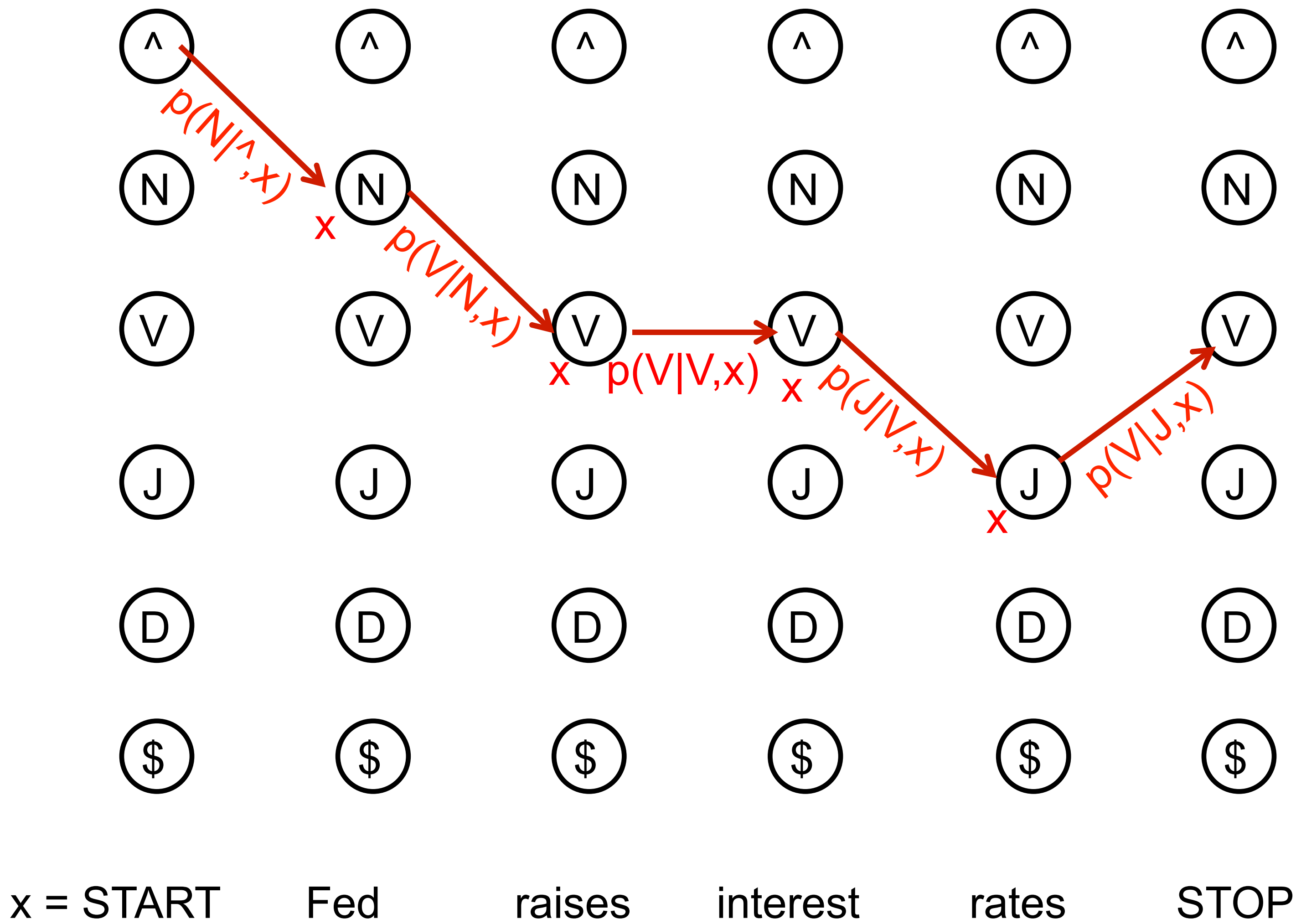
Challenge: How to compute argmax efficiently?

Decoding

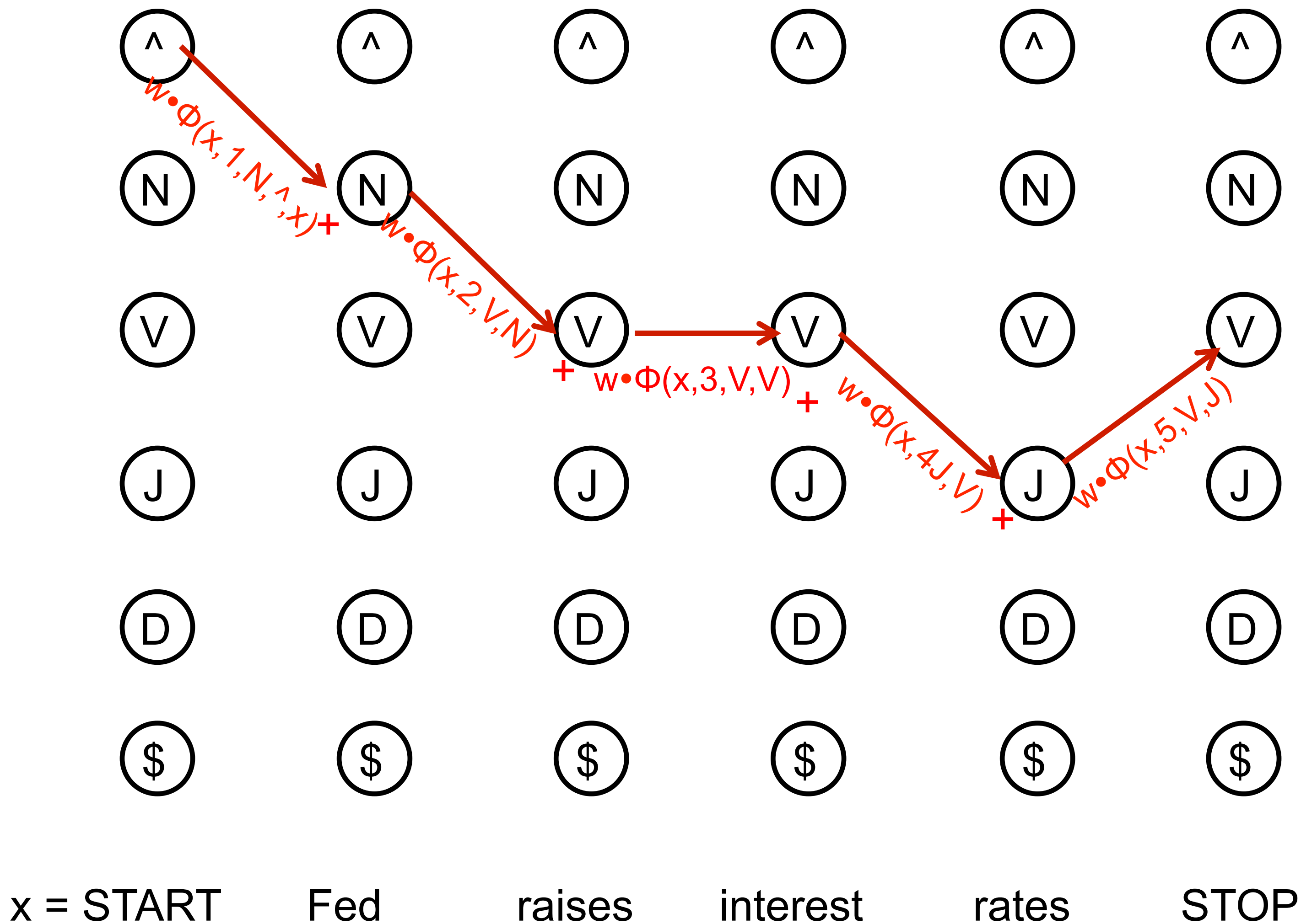
- Linear Perceptron $$s^* = \arg\max_s w \cdot \Phi(x, s) \cdot \theta$$
  - Features must be local, for $x=x_1 \ldots x_m$, and $s=s_1 \ldots s_m$

$$\Phi(x, s) = \sum_{j=1}^{m} \phi(x, j, s_{j-1}, s_j)$$

The MEMM State Lattice / Trellis (repeat)



The Perceptron State Lattice / Trellis



Decoding

- Linear Perceptron $$s^* = \arg\max_s w \cdot \Phi(x,s) \cdot \theta$$
  - Features must be local, for $x=x_1 \ldots x_m$, and $s=s_1 \ldots s_m$

$$\Phi(x,s) = \sum_{j=1}^{m} \phi(x,j,s_{j-1},s_j)$$

  - Define $\pi(i,s_i)$ to be the max score of a sequence of length i ending in tag s_i

$$\pi(i,s_i) = \max_{s_{i-1}} w \cdot \phi(x,i,s_{i-i},s_i) + \pi(i-1,s_{i-1})$$

- Viterbi algorithm (HMMs):

$$\pi(i,s_i) = \max_{s_{i-1}} e(x_i|s_i)q(s_i|s_{i-1})\pi(i-1,s_{i-1})$$

- Viterbi algorithm (Maxent):

$$\pi(i,s_i) = \max_{s_{i-1}} p(s_i|s_{i-1},x_1 \ldots x_m)\pi(i-1,s_{i-1})$$

Overview: Accuracies

- Roadmap of (known / unknown) accuracies:
  - Most freq tag: ~90% / ~50%
  - Trigram HMM: ~95% / ~55%
  - TnT (HMM++): 96.2% / 86.0%
  - Maxent $P(s_i|x)$: 96.8% / 86.8%
  - MEMM tagger: 96.9% / 86.9%
  - Perceptron 96.7% / ??

  - Upper bound: ~98%

Probabilistic Models

(Unstructured) categorization:

- Naïve Bayes

Structured prediction:

- HMMs
- PCFG Models
- IBM Models

Feature-rich / (Log)-linear Models

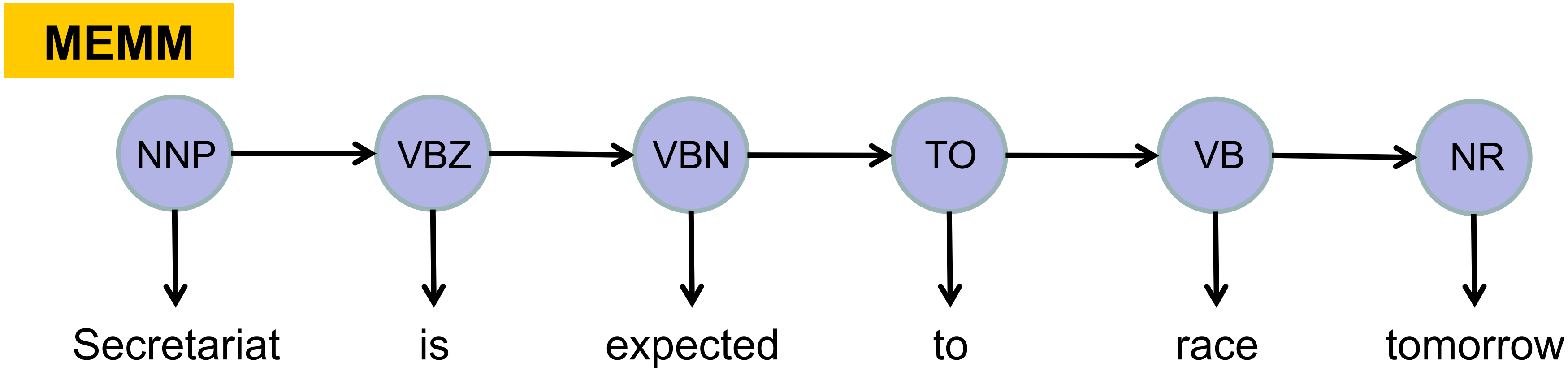
(Unstructured) categorization:

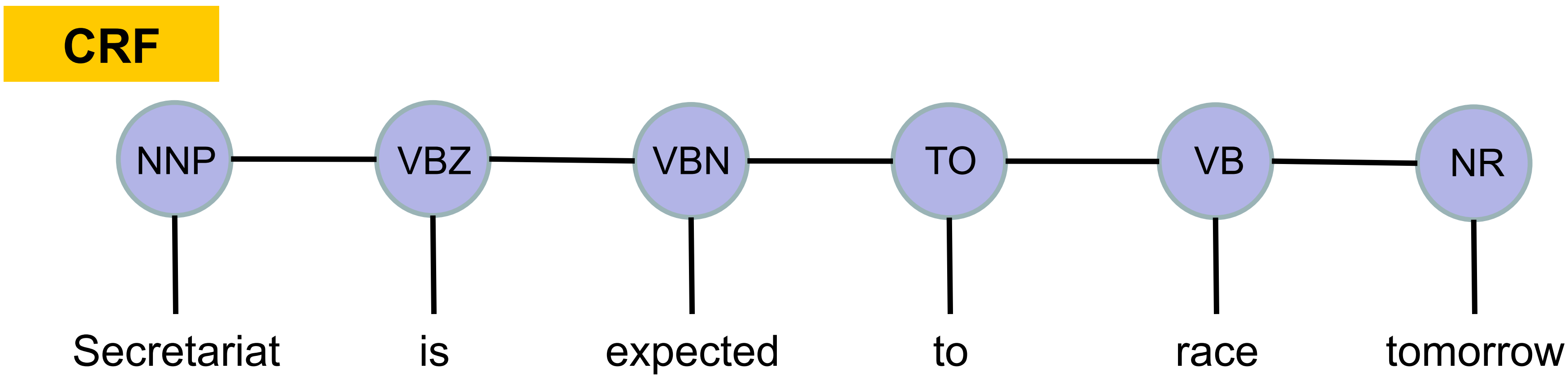
- Perceptron
- Maximum Entropy

Structured prediction:

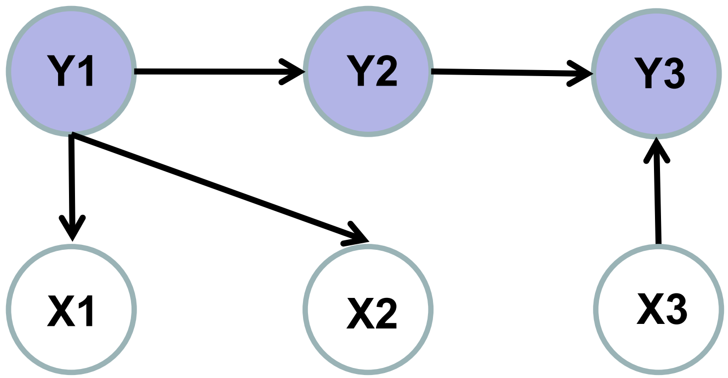
- Perceptron for Structured Prediction
- MEMM (Maximum Entropy Markov Model)
- CRF (Conditional Random Fields)

MEMM v.s. CRF
(Conditional Random Fields)





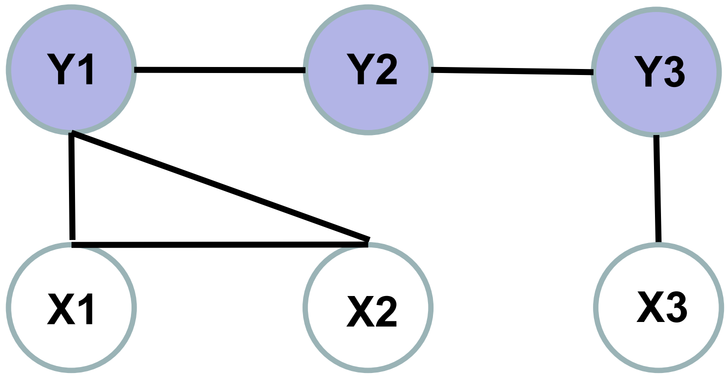
Graphical Models



- Conditional probability for each node
  - e.g. p(Y3 | Y2, X3) for Y3
  - e.g. p(X3) for X3
- Conditional independence
  - e.g. p(Y3 | Y2, X3) = p(Y3 | Y1, Y2, X1, X2, X3)
- Joint probability of the entire graph

  = product of conditional probability of each node

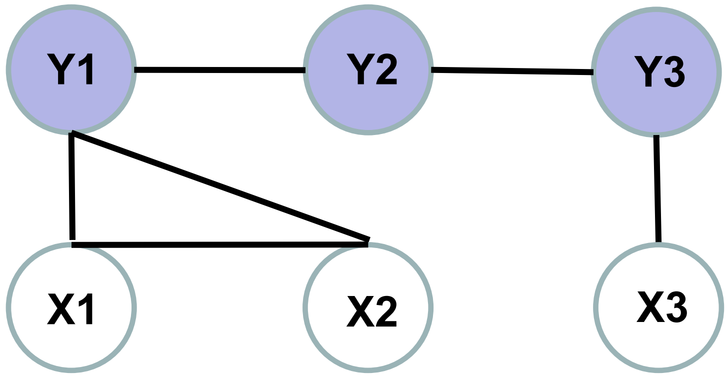
Undirected Graphical Model Basics



- Conditional independence
  - e.g. p(Y3 | all other nodes) = p(Y3 | Y3' neighbor)
- No conditional probability for each node
- Instead, "***potential function***" for each ***clique***
  - e.g. ϕ (X1, X2, Y1) or ϕ (Y1, Y2)
- Typically, log-linear potential functions

  ➔ ϕ (Y1, Y2) = exp Σ_k w_k f_k (Y1, Y2)

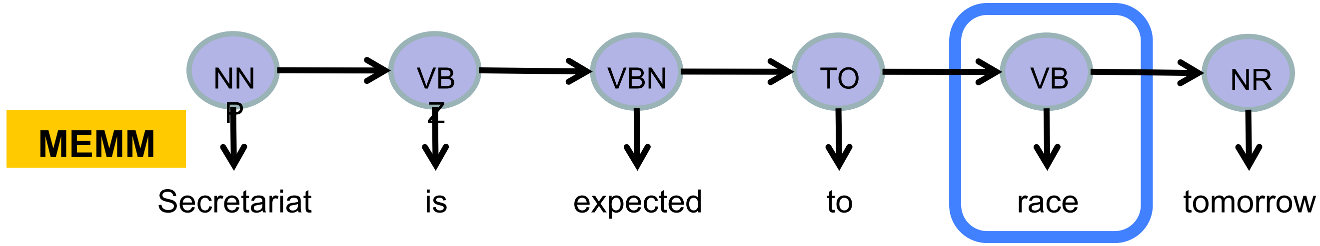
Undirected Graphical Model Basics

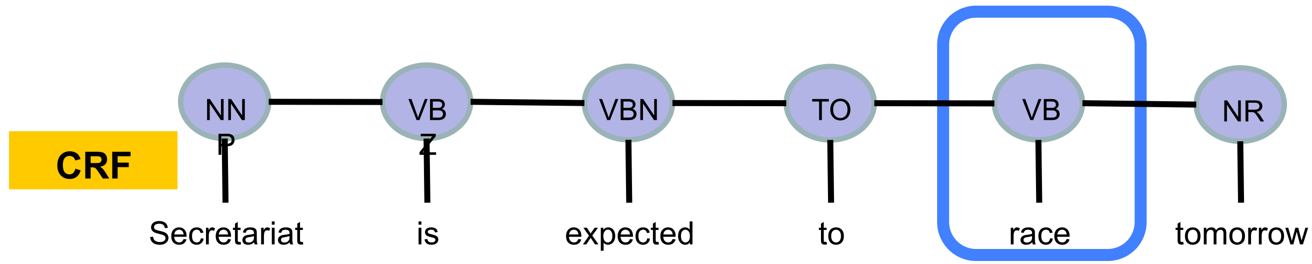


- Joint probability of the entire graph

$$P(\vec{Y}) = \frac{1}{Z} \prod_{\text{clique } C} \varphi(\vec{Y}_C)$$

$$Z = \sum_{\vec{Y}} \prod_{\text{clique } C} \varphi(\vec{Y}_C)$$



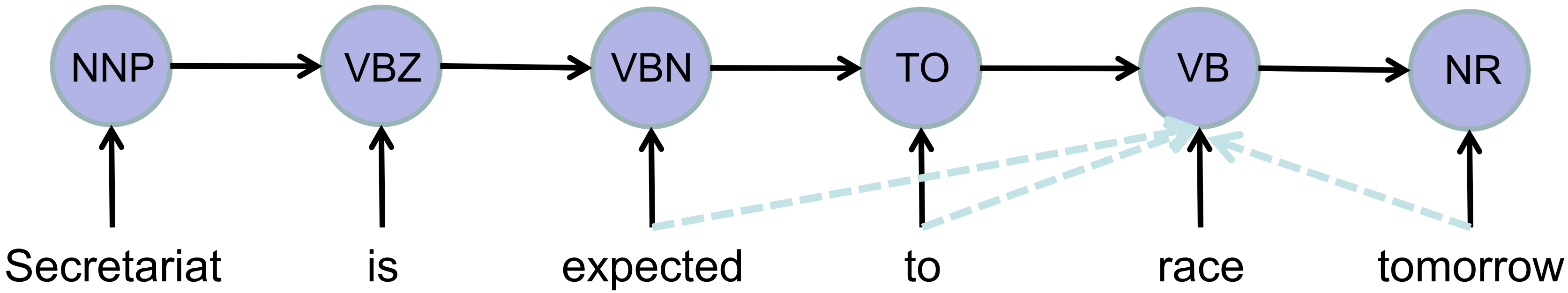


<table>
<tr><th>MEMM</th><th>CRF</th></tr>
<tr><td>Directed graphical model</td><td>Undirected graphical model</td></tr>
<tr><td colspan="2">“Discriminative” or “Conditional” models
➔ conditional probability p(tags | <u>words</u>)</td></tr>
<tr><td>Probability is defined for each slice =

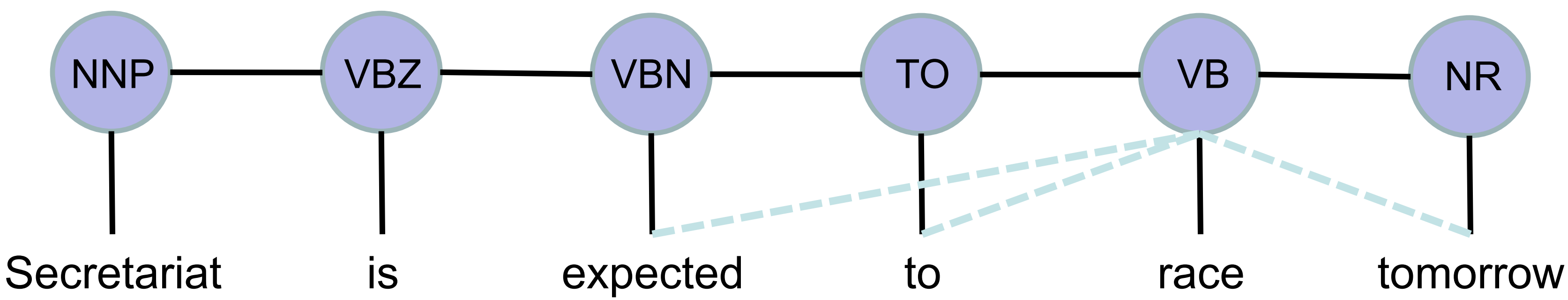
P (tag_i | tag_i-1, word_i)
or
p (tag_i | tag_i-1, all words)</td><td>Instead of probability, potential (energy function) is defined for each slide =
ϕ (tag_i, tag_i-1) * ϕ (tag_i, word_i)
or
ϕ(tag_i, tag_i-1, all words) * ϕ (tag_i, all words)</td></tr>
<tr><td colspan="2">➔ Can incorporate long distance features</td></tr>
</table>

MEMM v.s. CRF

MEMM



CRF



Conditional Random Fields (CRFs)

[Lafferty, McCallum, Pereira 01]

- Maximum entropy (logistic regression)

Sentence: $x=x_1 \ldots x_m$

$$p(y|x;w) = \frac{\exp\left(w \cdot \phi(x,y)\right)}{\sum_{y'} \exp\left(w \cdot \phi(x,y')\right)}$$

Tag Sequence: $y=s_1 \ldots s_m$

- Learning: maximize the (log) conditional likelihood of training data $\{(x_i, y_i)\}_{i=1}^{n}$

$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^{n} \left(\phi_j(x_i, y_i) - \sum_{y} p(y|x_i; w) \phi_j(x_i, y) \right) - \lambda w_j$$

- Computational Challenges?
  - Most likely tag sequence, normalization constant, gradient

Decoding

$$s^* = \arg\max_s p(s|x;w)$$

- CRFs
  - Features must be local, for x=$x_1 \ldots x_m$, and s=$s_1 \ldots s_m$

$$p(s|x;w) = \frac{\exp\left(w \cdot \Phi(x,s)\right)}{\sum_{s'} \exp\left(w \cdot \Phi(x,s')\right)} \quad \Phi(x,s) = \sum_{j=1}^{m} \phi(x, j, s_{j-1}, s_j)$$

$$\arg\max_s \frac{\exp\left(w \cdot \Phi(x,s)\right)}{\sum_{s'} \exp\left(w \cdot \Phi(x,s')\right)} = \arg\max_s \exp\left(w \cdot \Phi(x,s)\right)$$

$$= \arg\max_s w \cdot \Phi(x,s)$$

- Same as Linear Perceptron!!!

$$\pi(i, s_i) = \max_{s_{i-1}} \phi(x, i, s_{i-i}, s_i) + \pi(i-1, s_{i-1})$$

CRFs: Computing Normalization*

$$p(s|x;w) = \frac{\exp\left(w \cdot \Phi(x,s)\right)}{\sum_{s'} \exp\left(w \cdot \Phi(x,s')\right)} \quad \Phi(x,s) = \sum_{j=1}^{m} \phi(x,j,s_{j-1},s_j)$$

$$\sum_{s'} \text{exp}\left(w \cdot \Phi(x,s')\right) = \sum_{s'} \text{exp}\left(\sum_{j} w \cdot \phi(x,j,s_{j-1},s_j)\right)$$

$$= \sum_{s'} \prod_{j} \text{exp}\left(w \cdot \phi(x,j,s_{j-1},s_j)\right)$$

Define norm(i,s_i) to sum of scores for sequences ending in position i

$$norm(i,y_i) = \sum_{s_{i-1}} \text{exp}\left(w \cdot \phi(x,i,s_{i-1},s_i)\right) norm(i-1,s_{i-1})$$

- Forward Algorithm! Remember HMM case:

$$\alpha(i,y_i) = \sum_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \alpha(i-1,y_{i-1})$$

  - Could also use backward?

CRFs: Computing Gradient*

$$p(s|x;w) = \frac{\exp\left(w \cdot \Phi(x,s)\right)}{\sum_{s'} \exp\left(w \cdot \Phi(x,s')\right)} \qquad \Phi(x,s) = \sum_{j=1}^{m} \phi(x,j,s_{j-1},s_j)$$

$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^{n} \left(\Phi_j(x_i,s_i) - \sum_{s} p(s|x_i;w)\Phi_j(x_i,s) \right) - \lambda w_j$$

$$\sum_{s} p(s|x_i;w)\Phi_j(x_i,s) = \sum_{s} p(s|x_i;w) \sum_{j=1}^{m} \phi_k(x_i,j,s_{j-1},s_j)$$

$$= \sum_{j=1}^{m} \sum_{a,b} \sum_{s:s_{j-1}=a,s_b=b} p(s|x_i;w)\phi_k(x_i,j,s_{j-1},s_j)$$

- Need forward and backward messages

See notes for full details!

Overview: Accuracies

- Roadmap of (known / unknown) accuracies:
  - Most freq tag: ~90% / ~50%
  - Trigram HMM: ~95% / ~55%
  - TnT (HMM++): 96.2% / 86.0%
  - Maxent $P(s_i|x)$: 96.8% / 86.8%
  - MEMM tagger: 96.9% / 86.9%
  - Perceptron 96.7% / ??
  - CRF (untuned) 95.7% / 76.2%

  - Upper bound: ~98%

Cyclic Network

[Toutanova et al 03]

- Train two MEMMs, multiple together to score
- And be very careful
  - Tune regularization
  - Try lots of different features
  - See paper for full details

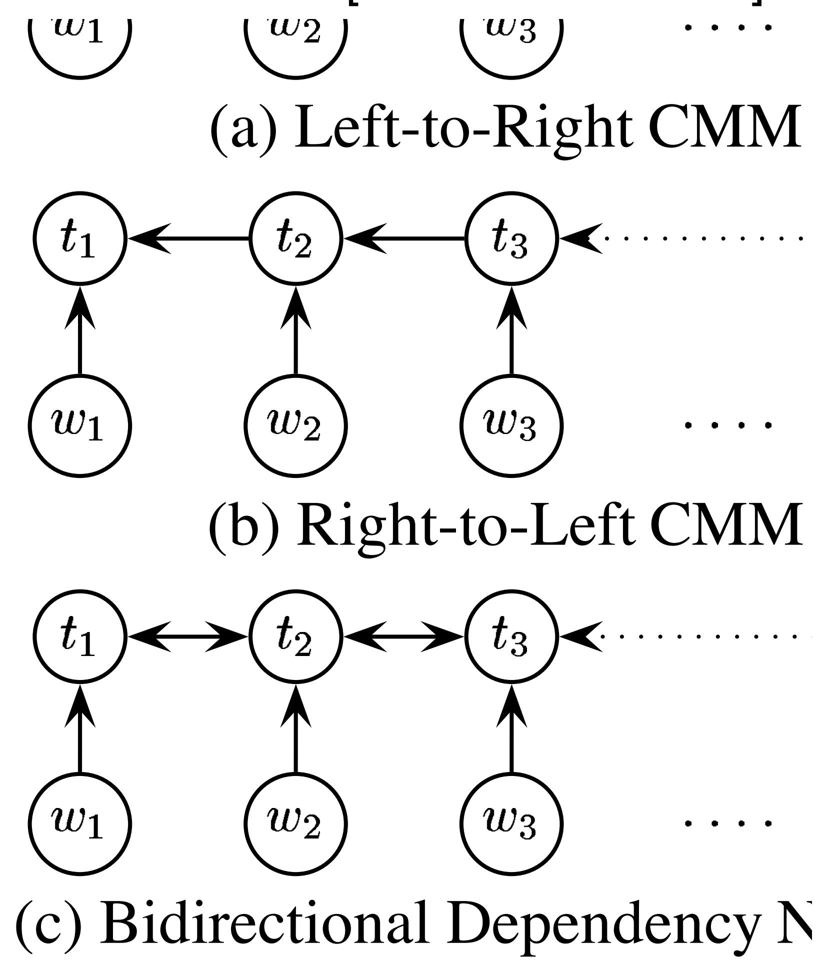


Figure 1: Dependency networks: (a) the (star
first-order CMM, (b) the (reversed) right-to-l
the bidirectional dependency network.

Overview: Accuracies

- Roadmap of (known / unknown) accuracies:
  - Most freq tag: ~90% / ~50%
  - Trigram HMM: ~95% / ~55%
  - TnT (HMM++): 96.2% / 86.0%
  - Maxent $P(s_i|x)$: 96.8% / 86.8%
  - MEMM tagger: 96.9% / 86.9%
  - Perceptron 96.7% / ??
  - CRF (untuned) 95.7% / 76.2%
  - Cyclic tagger: 97.2% / 89.0%
  - Upper bound: ~98%