CSE 447/547 Natural Language Processing Winter 2020

Feature Rich Models (Log Linear Models)

Yejin Choi

University of Washington

[Many slides from Dan Klein, Luke Zettlemoyer]

Structure in the output variable(s)?

What is the input representation?

	No Structure	Structured Inference
Generative models (classical probabilistic models)	Naïve Bayes	HMMs PCFGs IBM Models
Log-linear models (discriminatively trained feature-rich models)	Perceptron Maximum Entropy Logistic Regression	MEMM CRF
Neural network models (representation learning)	Feedforward NN CNN	RNN LSTM GRU …

Feature Rich Models

- Throw anything (features) you want into the stew (the model)

- Log-linear models
- Often lead to great performance.

(sometimes even a best paper award) "11,001 New Features for Statistical Machine Translation", D. Chiang, K. Knight, and W. Wang, NAACL, 2009.

Why want richer features?

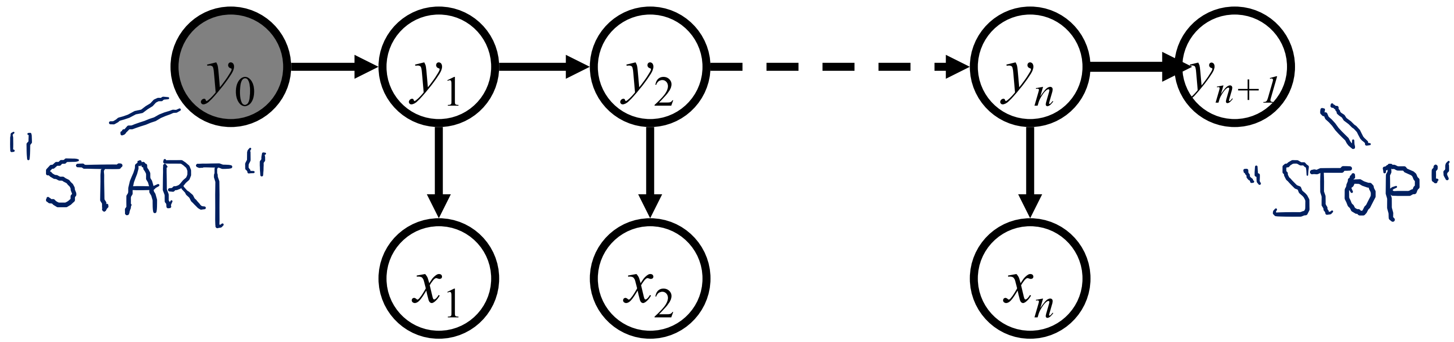
- POS tagging: more information about the context?
  - Is previous word "the"?
  - Is previous word "the" and the next word "of"?
  - Is previous word capitalized and the next word is numeric?

  - Is there a word "program" within [-5,+5] window?
  - Is the current word part of a known idiom?
  - Conjunctions of any of above?

- Desiderata:
  - Lots and lots of features like above: > 200K
  - No independence assumption among features
- Classical probability models, however
  - Permit very small amount of features
  - Make strong independence assumption among features

HMMs: P(tag sequence|sentence)

- We want a model of sequences y and observations x

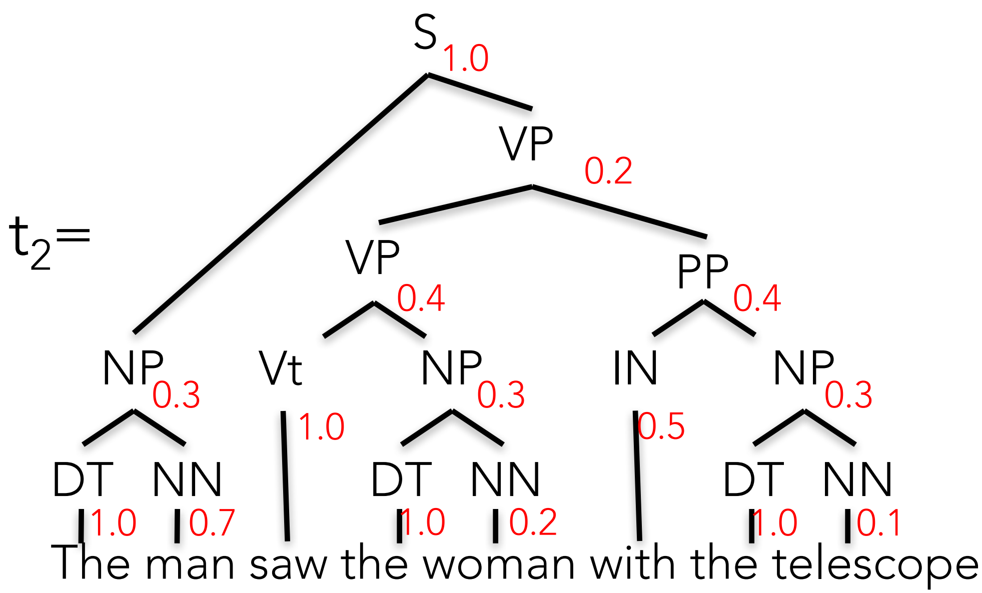


$$p(x_1 \ldots x_n, y_1 \ldots y_{n+1}) = q(\text{STOP}|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

where y_0=START and we call q(y'|y) the transition distribution and e(x|y) the emission (or observation) distribution.

- Assumptions:
  - Tag/state sequence is generated by a markov model
  - Words are chosen independently, conditioned only on the tag/state
  - These are totally broken assumptions: why?

PCFGs: P(parse tree|sentence)



$p(t_s)$=1.8*0.3*1.0*0.7*0.2*0.4*1.0*0.3*1.0*0.2*0.4*0.5*0.3*1.0*0.1

- Probability of a tree t with rules

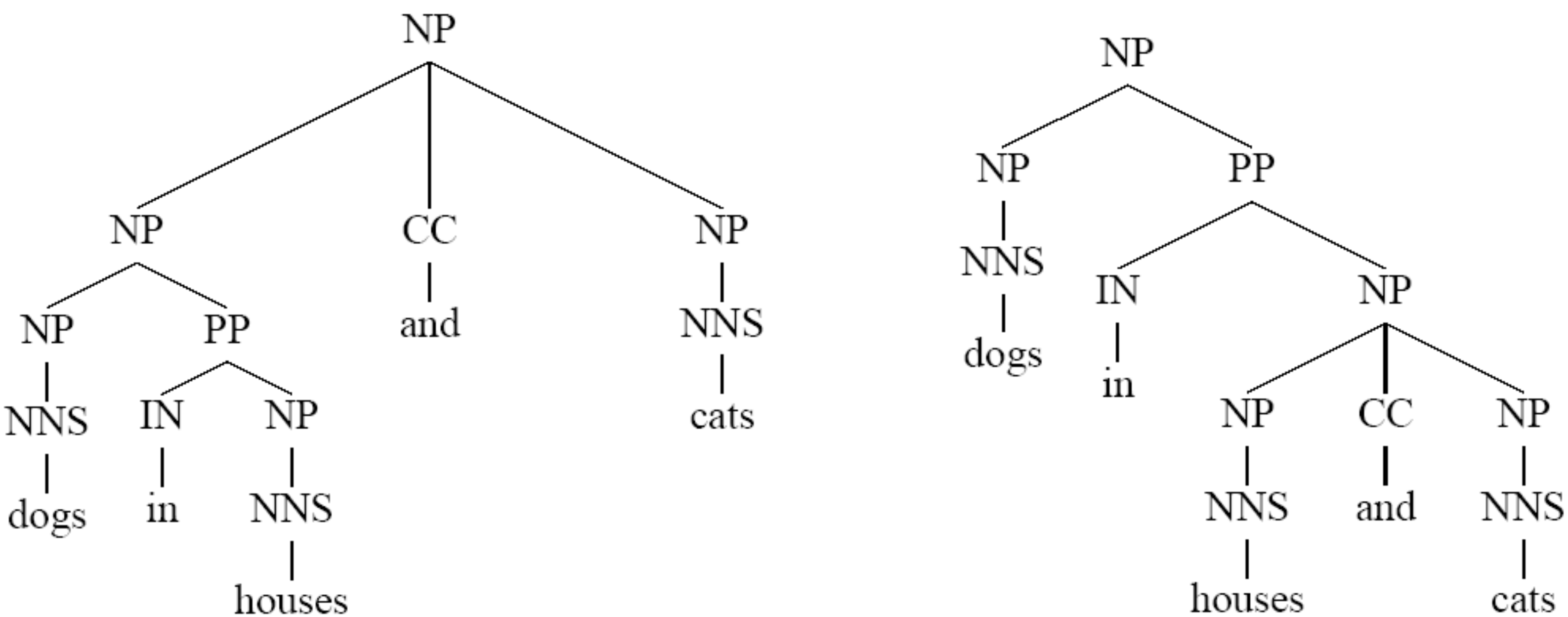
$$\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_n \rightarrow \beta_n$$

is

$$p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)$$

where $q(\alpha \rightarrow \beta)$ is the probability for rule $\alpha \rightarrow \beta$.

Rich features for long range dependencies



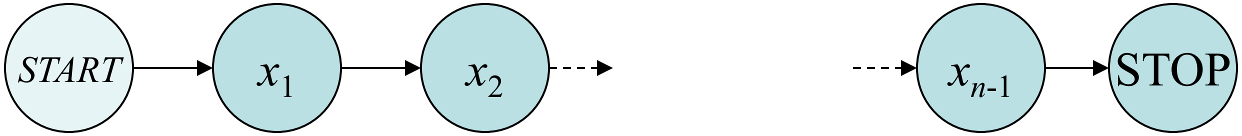
- What's different between basic PCFG scores here?
- What (lexical) correlations need to be scored?

LMs: P(text)

$$p(x_1 ... x_n) = \prod_{i=1}^{n} q(x_i | x_{i-1}) \quad \text{where} \sum_{x_i \in \mathcal{V}^*} q(x_i | x_{i-1}) = 1$$

$$x_0 = \text{START} \quad \& \quad \mathcal{V}^* := \mathcal{V} \cup \{\text{STOP}\}$$

- Generative process: (1) generate the very first word conditioning on the special symbol START, then, (2) pick the next word conditioning on the previous word, then repeat (2) until the special word STOP gets picked.
- Graphical Model:



- Subtleties:
  - If we are introducing the special START symbol to the model, then we are making the assumption that the sentence always starts with the special start word START, thus when we talk about $p(x_1 ... x_n)$ it is in fact $p(x_1 ... x_n | x_0 = \text{START})$
  - While we add the special STOP symbol to the vocabulary $\mathcal{V}^*$, we do not add the special START symbol to the vocabulary. Why?

Internals of probabilistic models: nothing but adding log-prob

- LM: … + 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)]
- Naïve Bayes:
  log p(Class) + log p(feature1 | Class) + log p(feature2 | Class) …

arbitrary scores instead of log probs?

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

- LM: … + Φ (w7 ; w5, w6) + Φ (w8 ; w6, w7) + …
- PCFG: Φ (NP VP ; S) + Φ (Papa ; NP) + Φ (VP PP ; VP) …
- HMM tagging: … + Φ (t7 ; t5, t6) + Φ (w7 ; t7) + …

- Noisy channel: [Φ (source)] + [Φ (data ; source)]
- Naïve Bayes:
  Φ (Class) + Φ (feature1 ; Class) + Φ (feature2 ; Class) …

arbitrary scores instead of log probs?

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

- LM: ... + Φ (w7 ; w5, w6) + Φ (w8 ; w6, w7) + ...
- PCFG: Φ (NP VP ; S) + Φ (Papa ; NP) + Φ (VP PP ; VP) ...
- ~~HMM~~ tagging: ... + Φ (t7 ; t5, t6) + Φ (w7 ; t7) + ...

MEMM or CRF

- Noisy channel: [Φ (source)] + [Φ (data ; source)]
- ~~Naïve Bayes~~:

Φ (Class) + Φ (feature1 ; Class) + Φ (feature2 ; Class) ...

logistic regression / max-ent

Running example: POS tagging

- Roadmap of (known / unknown) accuracies:
- Strawman baseline:
  - Most freq tag: ~90% / ~50%
- Generative models:
  - Trigram HMM: ~95% / ~55%
  - TnT (HMM++): 96.2% / 86.0% (with smart UNK'ing)
- Feature-rich models?

  - Upper bound: ~98%

Structure in the output variable(s)?

What is the input representation?

	No Structure	Structured Inference
Generative models (classical probabilistic models)	Naïve Bayes	HMMs PCFGs IBM Models
Log-linear models (discriminatively trained feature-rich models)	Perceptron Maximum Entropy Logistic Regression	MEMM CRF
Neural network models (representation learning)	Feedforward NN CNN	RNN LSTM GRU …

Rich features for rich contextual information

- Throw in various features about the context:
  - f1 := Is previous word “the” and the next word “of”?
  - f2 := Is previous word capitalized and the next word is numeric?
  - f3 := Frequencies of “the” within [-15,+15] window?
  - f4 := Is the current word part of a known idiom?

given a sentence “the blah … the truth of … the blah ”
Let’s say x = “truth” above, then

f(x) := (f1, f2, f3, f4)
f(truth) = (true, false, 3, false)
=>
f(x) = (1, 0, 3, 0)

Rich features for rich contextual information

- Throw in various features about the context:
  - f1 := Is previous word "the" and the next word "of"?
  - f2 := …
- You can also define features that look at the output 'y'!
  - f1_N := Is previous word "the" and the next tag is "N"?
  - f2_N := …
  - f1_V := Is previous word "the" and the next tag is "V"?
  - …. (replicate all features with respect to different values of y)

f(x) := (f1, f2, f3, f4)

f(x,y) := (f1_N, f2_N, f3_N, f4_N,
f1_V, f2_V, f3_V, f4_V,
f1_D, f2_D, f3_D, f4_D,
….)

Rich features for rich contextual information

- You can also define features that look at the output 'y'!
  - f1_N := Is previous word "the" and the next tag is "N"?
  - f2_N := …
  - f1_V := Is previous word "the" and the next tag is "V"?
  - …. (replicate all features with respect to different values of y)

given a sentence "the blah … the truth of … the blah "
Let's say x = "truth" above, and y = "N", then

f(truth) = (true, false, 3, false)

f(x,y) := (f1_N, f2_N, f3_N, f4_N,
f1_V, f2_V, f3_V, f4_V,
f1_D, f2_D, f3_D, f4_D,
….)

f(truth, N) = ?

Rich features for rich contextual information

- Throw in various features about the context:
  - f1 := Is previous word "the" and the next word "of"?
  - f2 := Is previous word capitalized and the next word is numeric?
  - f3 := Frequencies of "the" within [-15,+15] window?
  - f4 := Is the current word part of a known idiom?
- You can also define features that look at the output 'y'!
  - f1_N := Is previous word "the" and the next tag is "N"?
  - f1_V := Is previous word "the" and the next tag is "V"?

- You can also take any conjunctions of above.

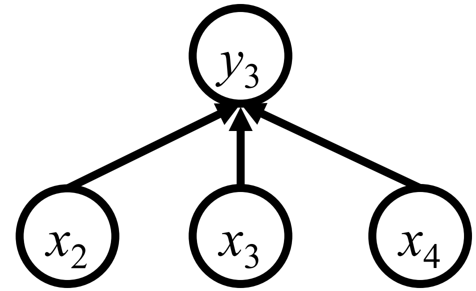
$$f(x, y) = [0, 0, 0, 1, 0, 0, 0, 0, 3, 0.2, 0, 0, ...]$$

- Create a very long feature vector with dimensions often >200K
- Overlapping features are fine – no independence assumption among features

Goals of this Class

- How to construct a feature vector f(x)
- How to extend the feature vector to f(x,y)
- How to construct a probability model using any given f(x,y)
- How to learn the parameter vector w for MaxEnt (log-linear) models
- Knowing the key differences between MaxEnt and Naïve Bayes
- How to extend MaxEnt to sequence tagging

Maximum Entropy (MaxEnt) Models



- Output: y
  - One POS tag for one word (at a time)
- Input: x (any words in the context)
  - Represented as a feature vector f(x, y)
- Model parameters: w
- Make probability using **SoftMax** function:
- Also known as "**Log-linear**" Models (*linear if you take log*)

$$p(y|x) = \frac{\exp(w \cdot f(x, y))}{\sum_{y'} \exp(w \cdot f(x, y'))}$$

Make positive! (numerator)

Normalize! (denominator)

Training MaxEnt Models

- Make probability using SoftMax function

$$p(y|x) = \frac{\exp(w \cdot f(x,y))}{\sum_{y'} \exp(w \cdot f(x,y'))}$$

- Training:
  - maximize log likelihood of training data $\{(x^i, y^i)\}_{i=1}^{n}$

$$L(w) = \log \prod_i p(y^i|x^i) = \sum_i log \frac{\exp(w \cdot f(x^i, y^i))}{\sum_{y'} \exp(w \cdot f(x^i, y'))}$$

  - which also incidentally maximizes the entropy (hence "maximum entropy")

Training MaxEnt Models

- Make probability using SoftMax function

$$p(y|x) = \frac{\exp(w \cdot f(x,y))}{\sum_{y'} \exp(w \cdot f(x,y'))}$$

- Training:
  - maximize log likelihood

$$L(w) = \log \prod_i p(y^i|x^i) = \sum_i log \frac{\exp(w \cdot f(x^i,y^i))}{\sum_{y'} \exp(w \cdot f(x^i,y'))}$$

$$= \sum_i \left(w \cdot f(x^i,y^i) - \log \sum_{y'} \exp(w \cdot f(x^i,y')) \right)$$

Training MaxEnt Models

$$L(w) = \sum_i \left(w \cdot f(x^i, y^i) - \log \sum_{y'} \exp(w \cdot f(x^i, y')) \right)$$

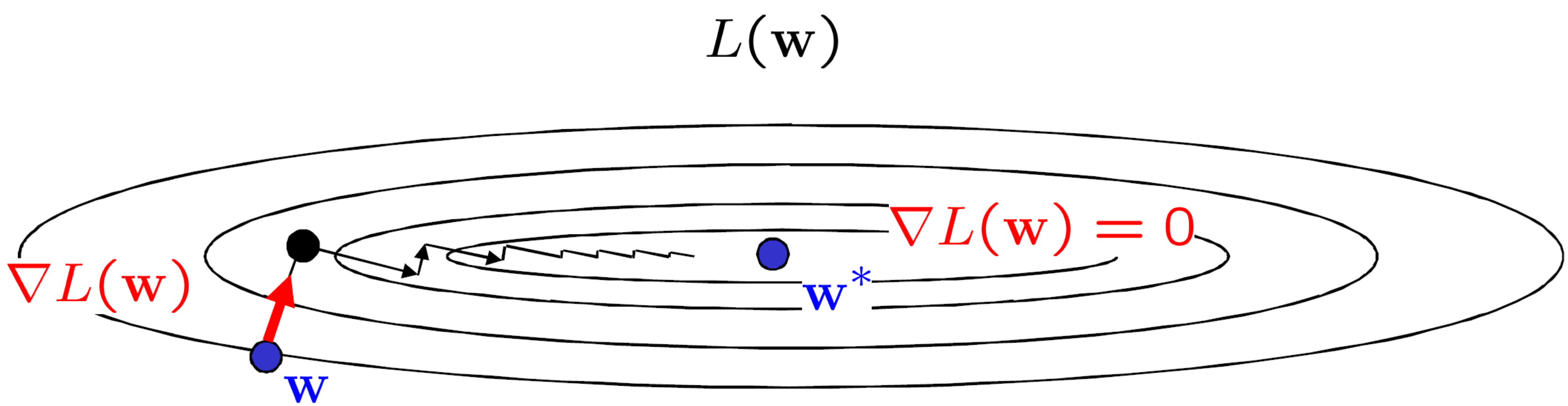
Take partial derivative for each w_k in the weight vector w:

$$\frac{\partial L(w)}{\partial w_k} = \sum_i \left(f_k(x^i, y^i) - \sum_{y'} p(y'|x^i) f_k(x^i, y') \right)$$

Total count of feature k with respect to the correct predictions

Expected count of feature k with respect to the predicted output

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'

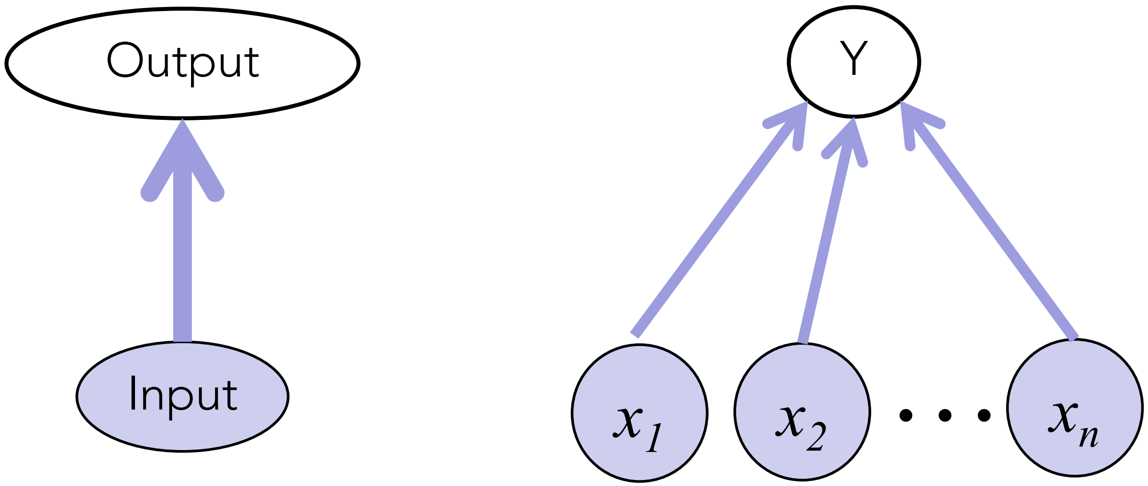
  (2) evaluate its derivative at current 'w'

Goals of this Class

- How to construct a feature vector f(x)
- How to extend the feature vector to f(x,y)
- How to construct a probability model using any given f(x,y)
- How to learn the parameter vector w for MaxEnt (log-linear) models
- Knowing the key differences between MaxEnt and Naïve Bayes
- How to extend MaxEnt to sequence tagging

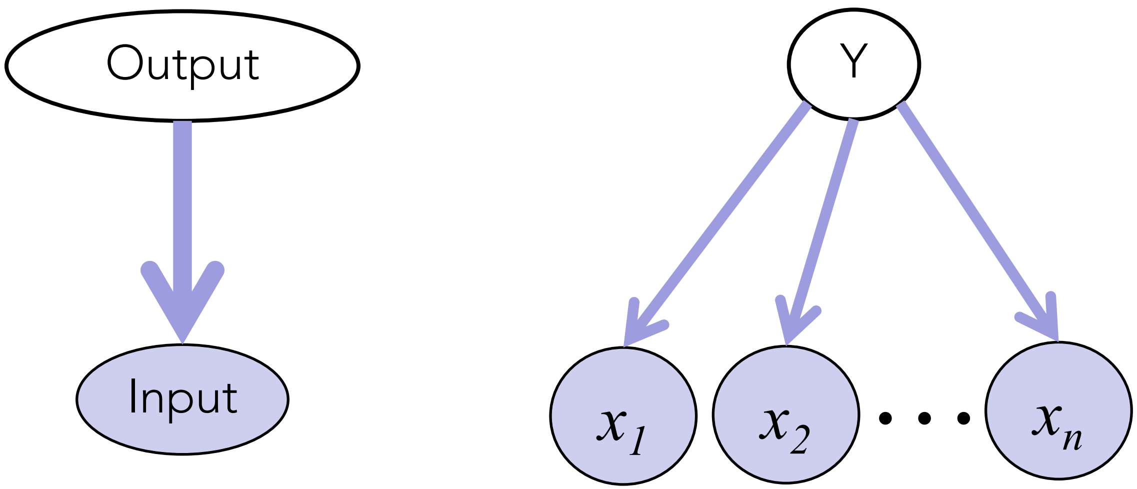
Graphical Representation of MaxEnt

$$p(y|x) = \frac{\exp(w \cdot f(x, y))}{\sum_{y'} \exp(w \cdot f(x, y'))}$$

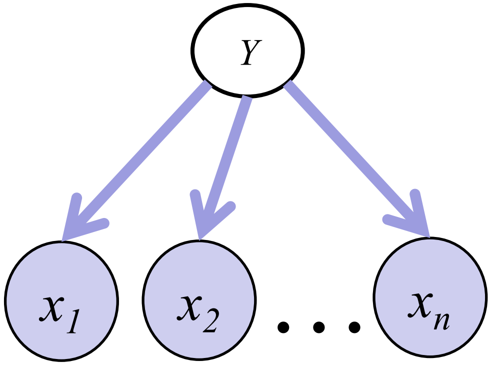


Graphical Representation of Naïve Bayes

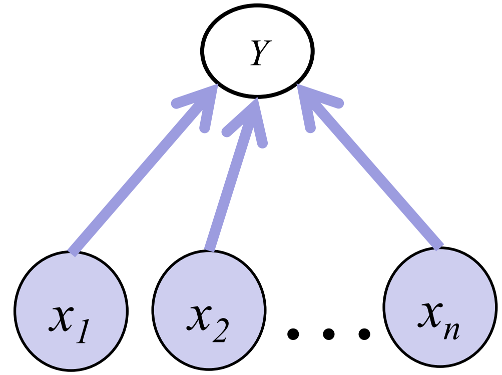
$$p(x|y) = \prod_j p(x_j|y)$$



Naïve Bayes



MaxEnt



Naïve Bayes Classifier	Maximum Entropy Classifier
"Generative" models ➔ p(<u>input</u> | output) ➔ For instance, for text categorization, P(words | category) ➔ Unnecessary efforts on generating input	"Discriminative" models ➔ p(output | <u>input</u>) ➔ 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!)	➔ By conditioning on the entire input, we don't need to worry about the independent assumption among input variables
➔ Cannot incorporate arbitrary/redundant/overlapping features	➔ 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's missing in MaxEnt compared to HMM?

  - Upper bound: ~98%

Structure in the output variable(s)?

What is the input representation?

	No Structure	Structured Inference
Generative models (classical probabilistic models)	Naïve Bayes	HMMs PCFGs IBM Models
Log-linear models (discriminatively trained feature-rich models)	Perceptron Maximum Entropy Logistic Regression	MEMM CRF
Neural network models (representation learning)	Feedforward NN CNN	RNN LSTM GRU …

Goals of this Class

- How to construct a feature vector f(x)
- How to extend the feature vector to f(x,y)
- How to construct a probability model using any given f(x,y)
- How to learn the parameter vector w for MaxEnt (log-linear) models
- Knowing the key differences between MaxEnt and Naïve Bayes
- How to extend MaxEnt to sequence tagging

MEMM Taggers

- One step up: also condition on previous tags

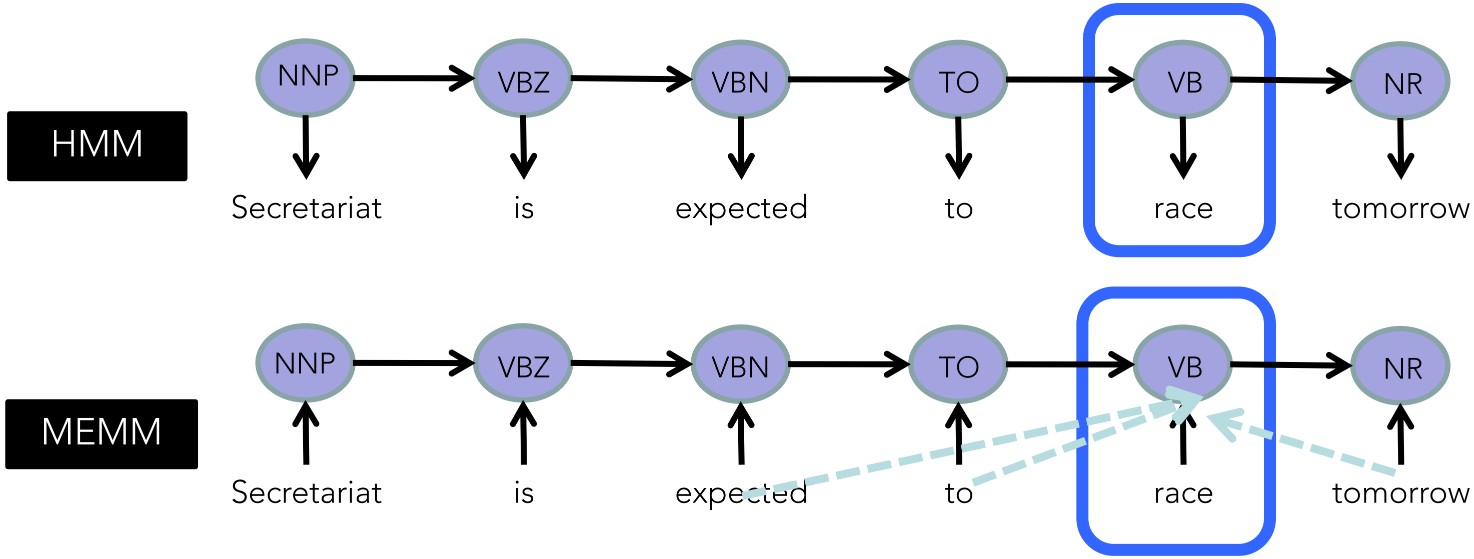
$$p(s_1 \dots s_m | x_1 \dots x_m) = \prod_{i=1}^{m} p(s_i | s_1 \dots s_{i-1}, x_1 \dots x_m)$$

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

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

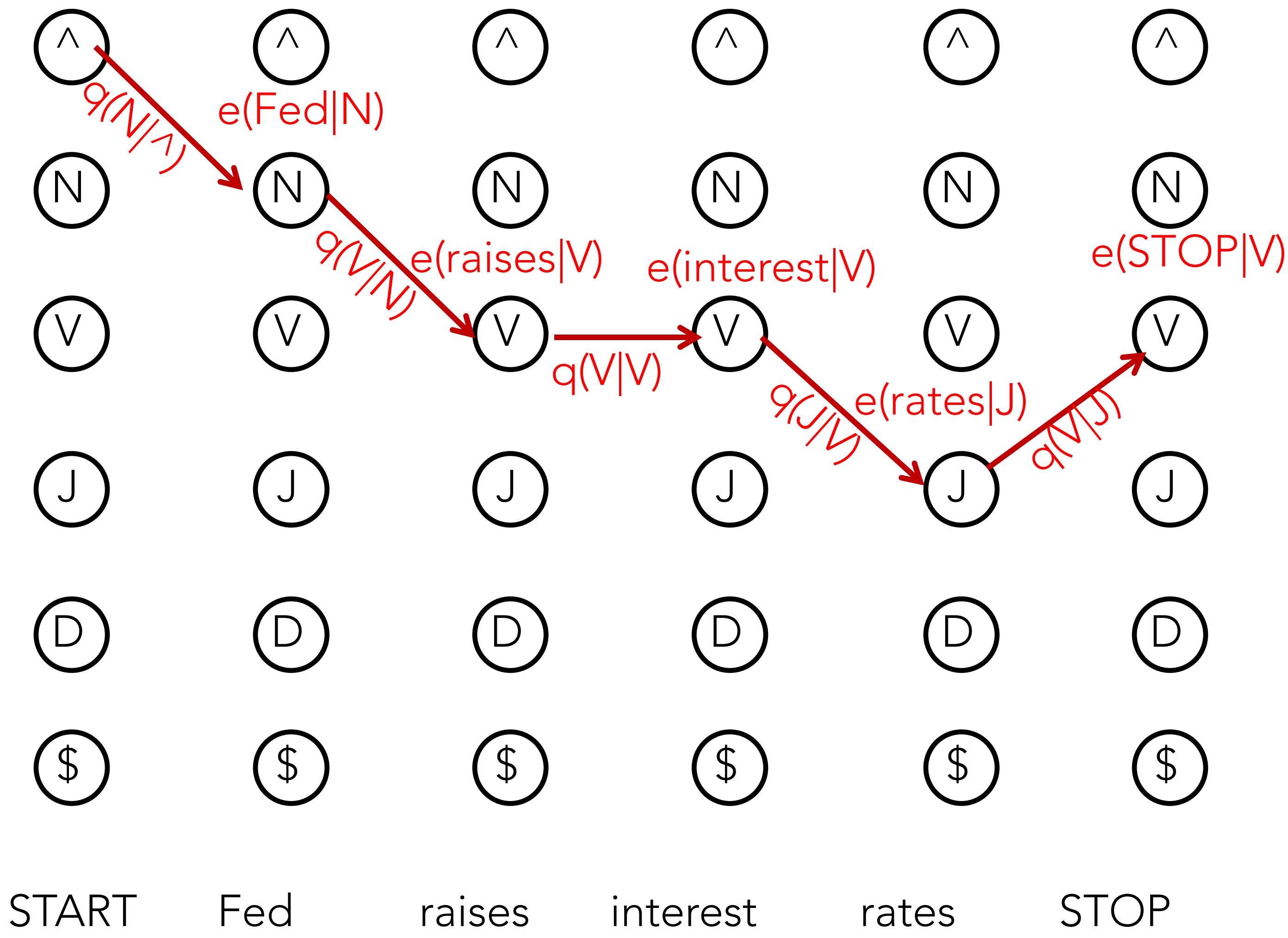
$$p(s_i | s_{i-1}, x_1 \dots x_m) = \frac{\exp\left(w \cdot \phi(x_1 \dots x_m, i, s_{i-1}, s_i)\right)}{\sum_{s'} \exp\left(w \cdot \phi(x_1 \dots x_m, i, s_{i-1}, s')\right)}$$

  - This is referred to as an MEMM tagger [Ratnaparkhi 96]

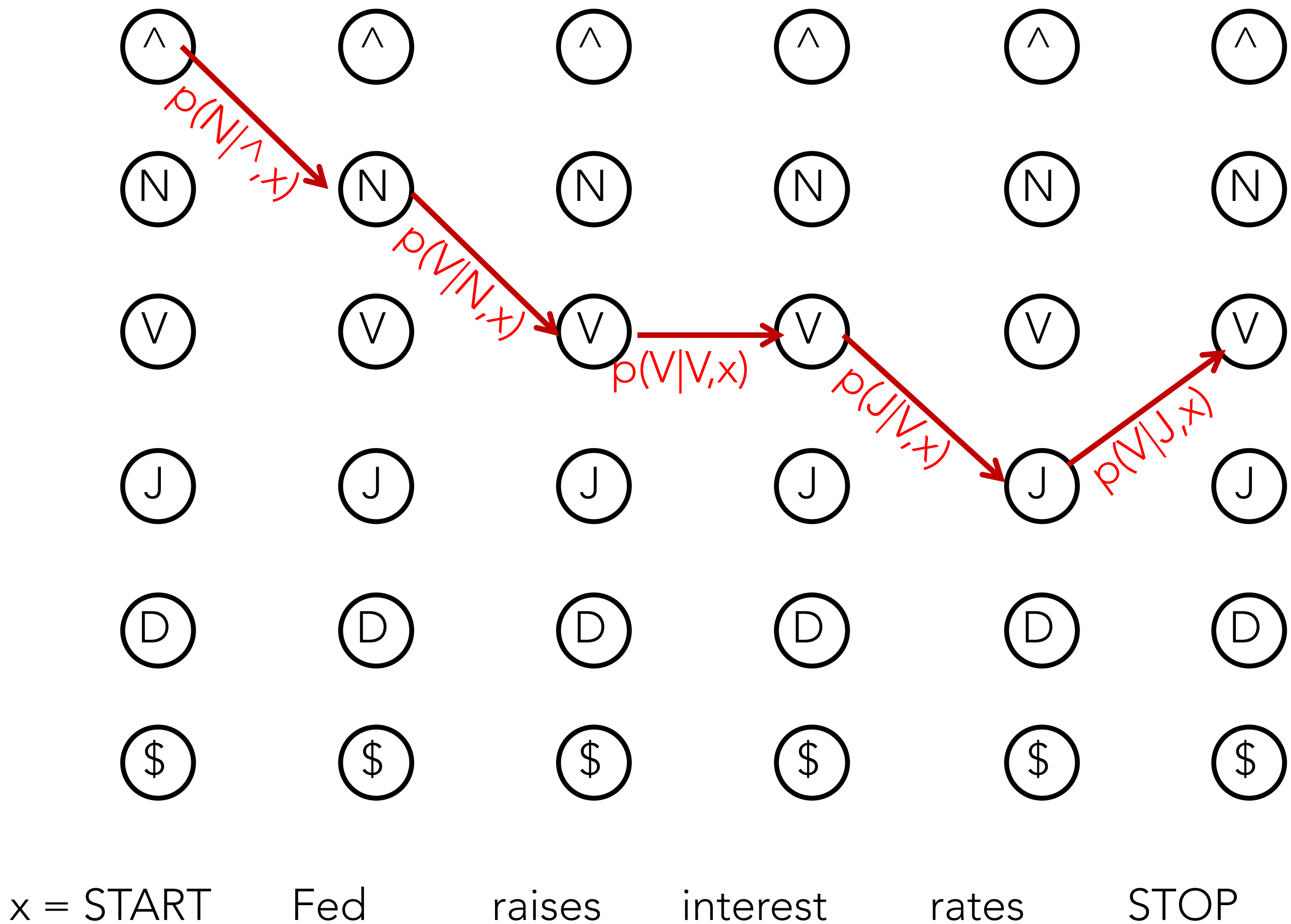


HMM	MEMM
"Generative" models ➔ joint probability p(words, tags) ➔ "generate" input (in addition to tags) ➔ but we need to predict tags, not words!	"Discriminative" or "Conditional" models ➔ conditional probability p(tags | words) ➔ "condition" on input ➔ Focusing only on predicting tags
Probability of 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

The HMM State Lattice / Trellis (repeat slide)



The MEMM State Lattice / Trellis



Decoding:

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

- 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%

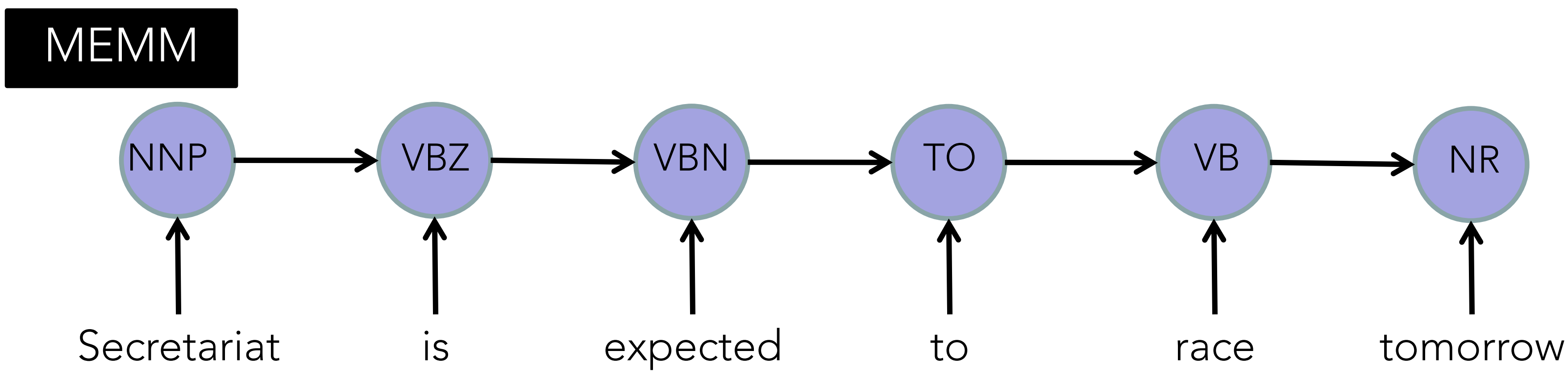
  - Upper bound: ~98%

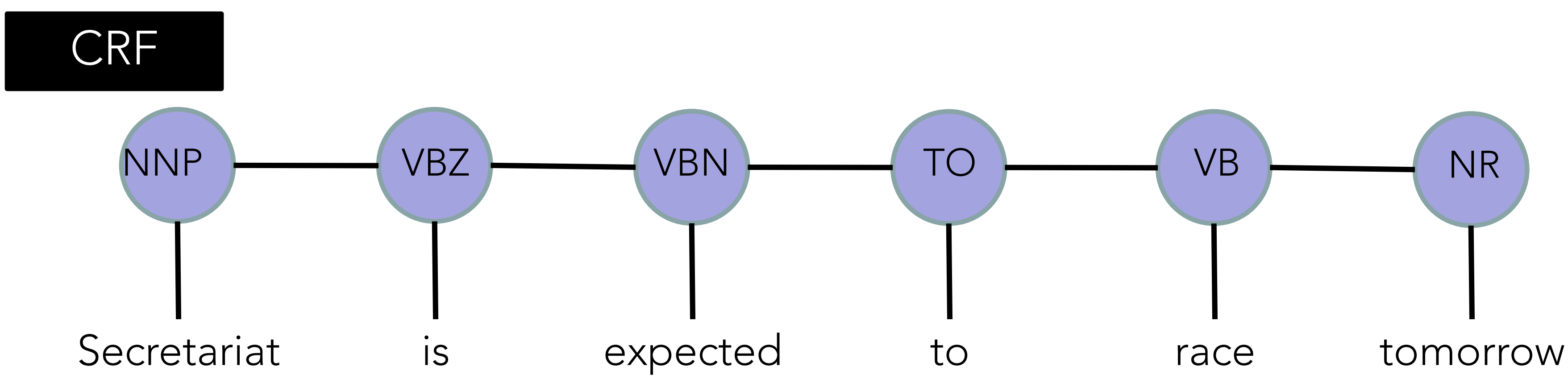
Structure in the output variable(s)?

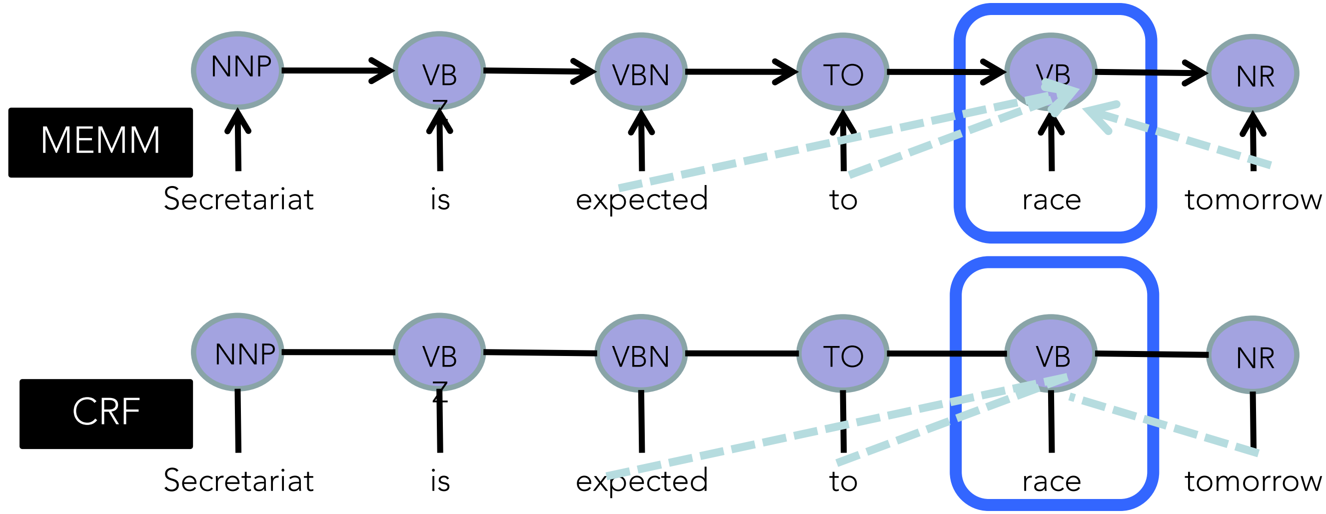
What is the input representation?

	No Structure	Structured Inference
Generative models (classical probabilistic models)	Naïve Bayes	HMMs PCFGs IBM Models
Log-linear models (discriminatively trained feature-rich models)	Perceptron Maximum Entropy Logistic Regression	MEMM CRF
Neural network models (representation learning)	Feedforward NN CNN	RNN LSTM GRU …

MEMM v.s. CRF
(Conditional Random Fields)







MEMM	CRF
Directed graphical model	Undirected graphical model
"Discriminative" or "Conditional" models ➔ conditional probability p(tags | <u>words</u>)	
Probability is defined for each slice = P (tag_i | tag_i-1, word_i) or p (tag_i | tag_i-1, all words)	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)
➔ Can incorporate long distance features	

Conditional Random Fields (CRFs)

[Lafferty, McCallum, Pereira 01]

- Maximum entropy (logistic regression)

Sentence: $x=x_1 \ldots x_m$

Tag Sequence: $s=s_1 \ldots s_m$

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

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

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

- 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 $s=s_1 \ldots s_m$

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

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

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

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

- Viterbi recursion

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

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*

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

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

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

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

- Need forward and backward messages

See notes for full details!

Overview: Accuracies

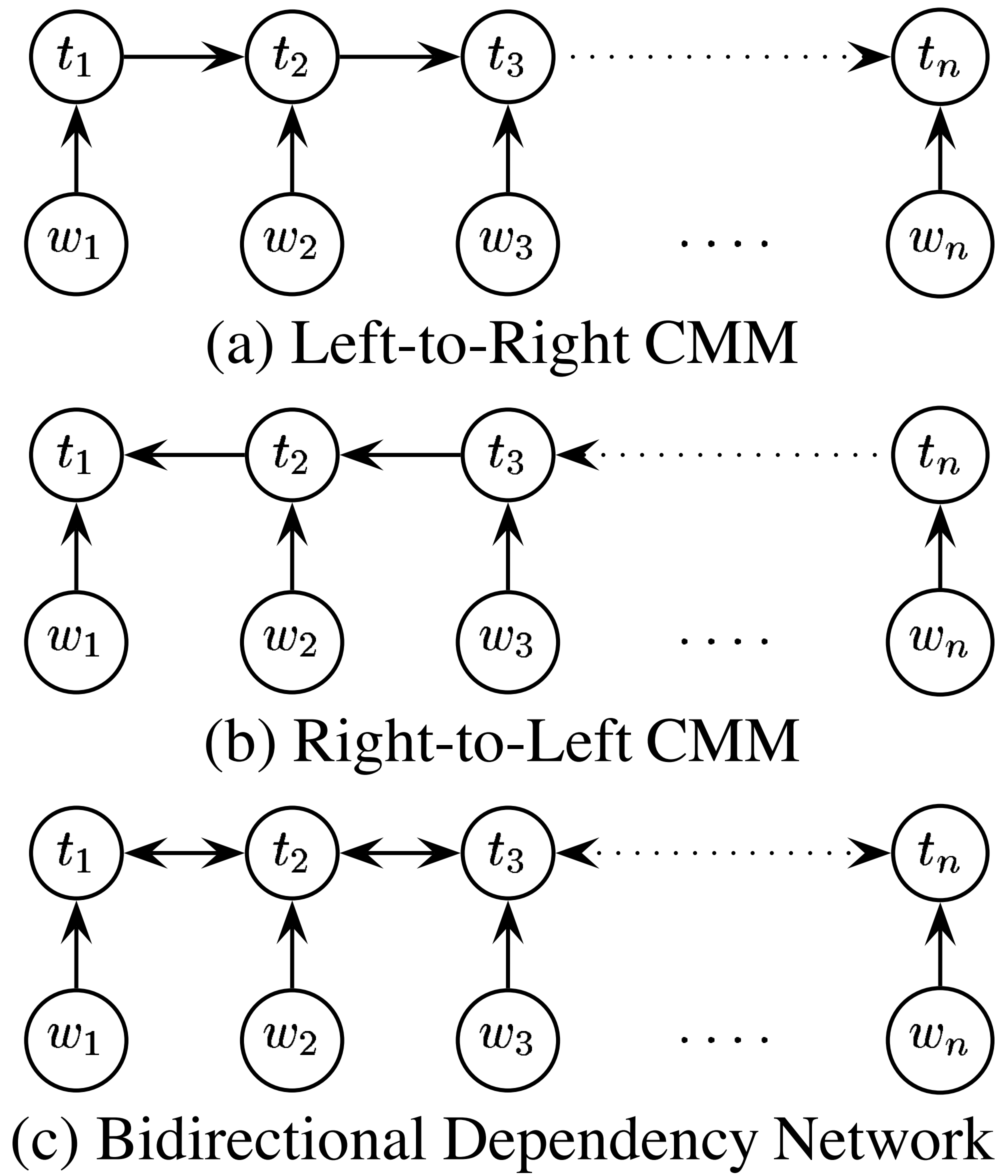
- Roadmap of (known / unknown) accuracies:
  - Most freq tag: ~90% / ~50%
  - Trigram HMM: ~95% / ~55%
  - TnT (HMM++): 96.2% / 86.0%
  - Maxent $P(s_i|x)$: 96.8% / 86.8%
  - MEMM tagger: 96.9% / 86.9%
  - CRF (untuned) 95.7% / 76.2%

  - 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



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%

- Locally normalized models
  - HMMs, MEMMs
  - Local scores are probabilities
  - However: one issue 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?
- Globally normalized models
  - Local scores are arbitrary scores
  - Conditional Random Fields (CRFs)
  - Slower to train (structured inference at each iteration of learning)
  - Neural Networks (global training w/o structured inference)

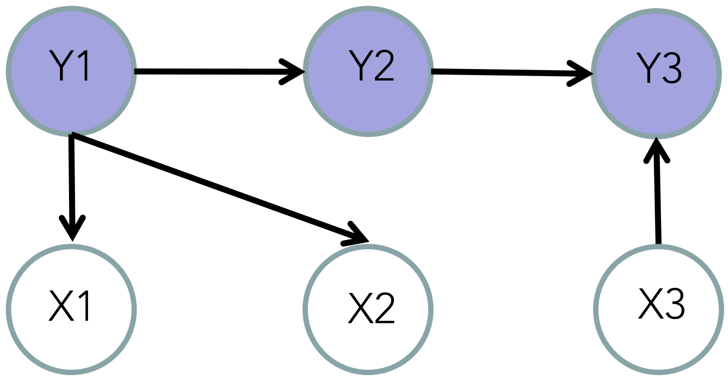
Structure in the output variable(s)?

What is the input representation?

	No Structure	Structured Inference
Generative models (classical probabilistic models)	Naïve Bayes	HMMs PCFGs IBM Models
Log-linear models (discriminatively trained feature-rich models)	Perceptron Maximum Entropy Logistic Regression	MEMM CRF
Neural network models (representation learning)	Feedforward NN CNN	RNN LSTM GRU …

Supplementary Material

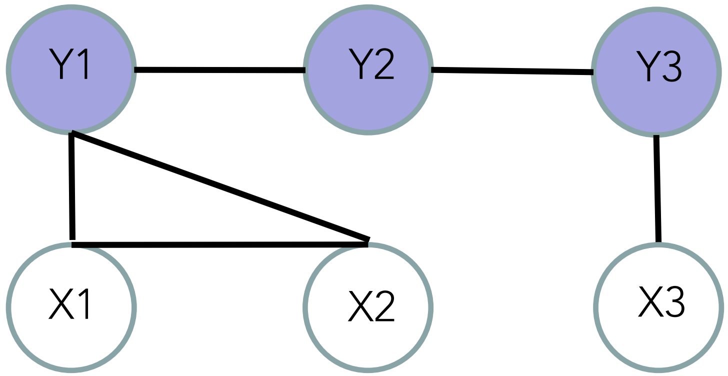
Graphical Models



- 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

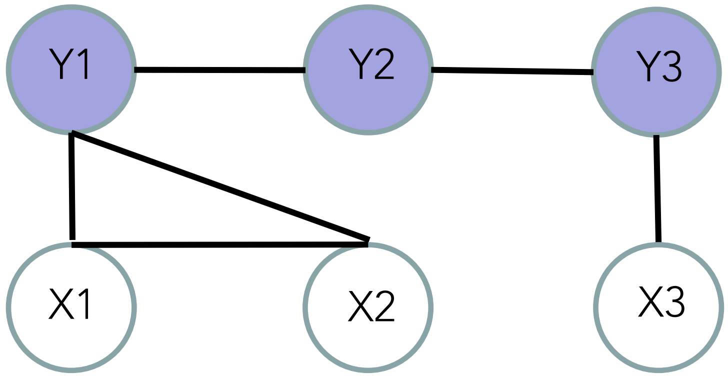
Undirected Graphical Model Basics



- 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

➔ $\phi (Y1, Y2) = \exp \Sigma_k \; w_k \, f_k (Y1, Y2)$

Undirected Graphical Model Basics



- Joint probability of the entire graph

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

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