Lecture 4: Structured Prediction Models

Kai-Wei Chang
CS @ UCLA
kw@kwchang.net

Couse webpage: https://uclanlp.github.io/CS269-17/

UCLA ENGINEERING
Computer Science

Previous Lecture

- Binary linear classification
  - Perceptron, SVMs, Logistic regression, Naïve Bayes
  - Output: $y \in \{1, -1\}$
- Multi-class classification
  - Multiclass Perceptron, Multiclass SVM…
  - Output: $y \in \{1,2,3, \ldots K\}$

What we have seen: multiclass

- Reducing multiclass to binary
  - One-against-all & One-vs-one
  - Error correcting codes
  - Extension: Learning to search
- Training a single classifier
  - Multiclass Perceptron: Kesler's construction
  - Multiclass SVMs: Crammer&Singer formulation
  - Multinomial logistic regression
  - Extension: Graphical models

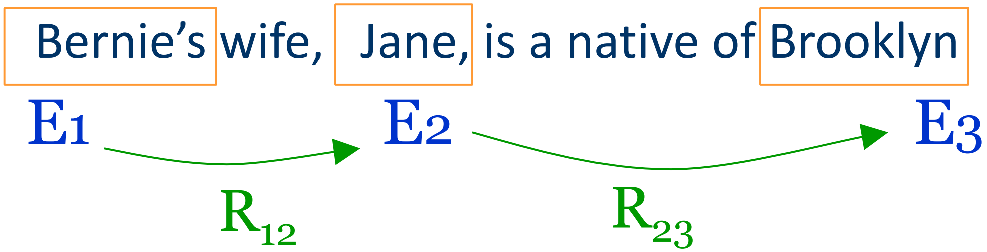
This lecture

- What is structured output?
- Multiclass as structure
- Sequence as structure
- General graph structure

Global decisions

- “Understanding” is a global decision
  - Several local decisions play a role
  - There are mutual dependencies on their outcome.
- Essential to make coherent decisions
  - Joint, Global Inference

Inference with Constraints
[Roth&Yih'04,07,….]



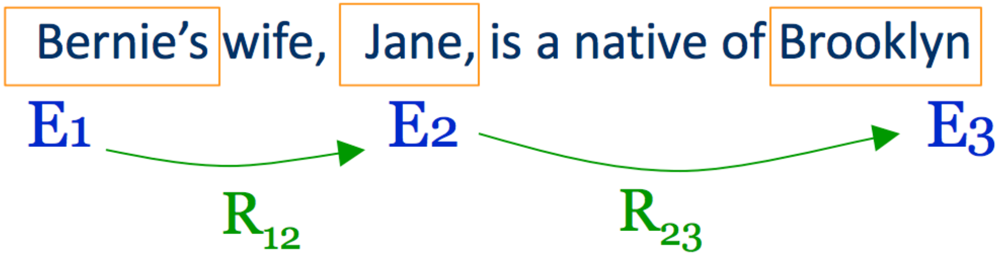
Models could be learned separately/jointly; constraints may come up only at decision time.

Inference with Constraints [Roth&Yih’04,07,….]

other	0.05
per	0.85
loc	0.10

other	0.10
per	0.60
loc	0.30

other	0.05
per	0.50
loc	0.45



Bernie’s wife, Jane, is a native of Brooklyn

irrelevant	0.05
spouse_of	0.45
born_in	0.50

irrelevant	0.10
spouse_of	0.05
born_in	0.85

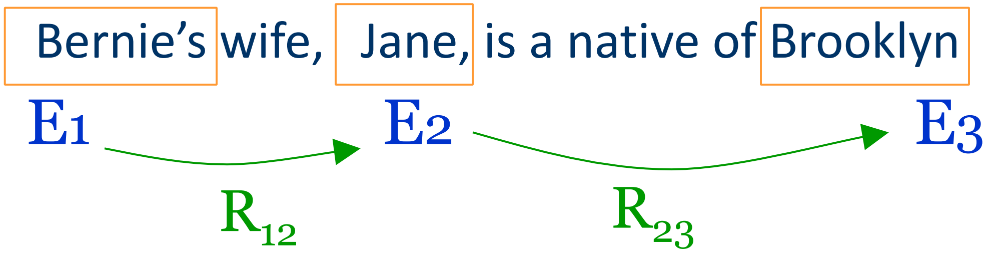
Models could be learned separately/jointly; constraints may come up only at decision time.

Inference with Constraints [Roth&Yih'04,07,….]

other	0.05
per	**0.85**
loc	0.10

other	0.10
per	**0.60**
loc	0.30

other	0.05
per	0.50
loc	**0.45**



Bernie's wife, Jane, is a native of Brooklyn

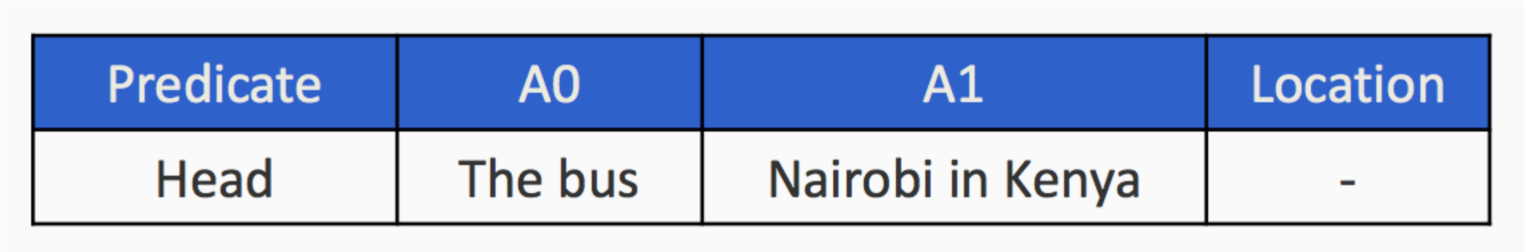
irrelevant	0.05
spouse_of	**0.45**
born_in	0.50

irrelevant	0.10
spouse_of	0.05
born_in	**0.85**

Models could be learned separately/jointly; constraints may come up only at decision time.

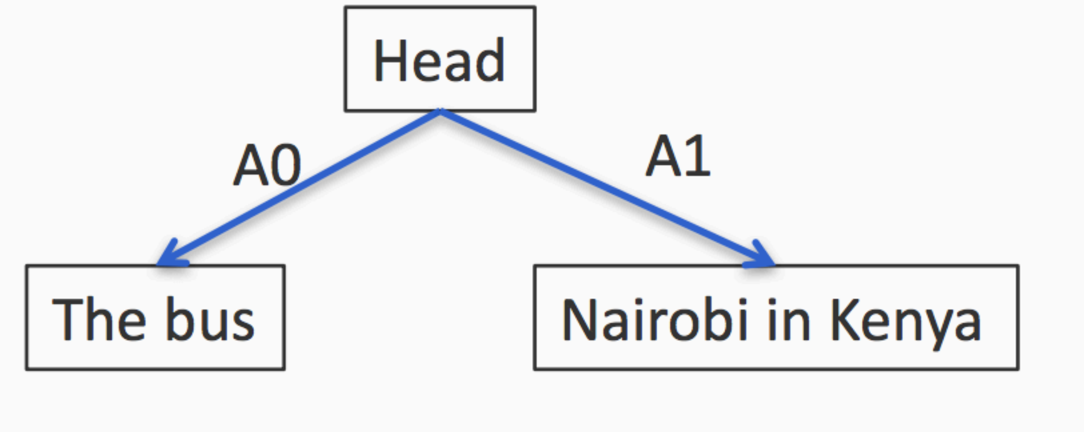
Structured output is…

- A predefine structure



Predicate	A0	A1	Location
Head	The bus	Nairobi in Kenya	-

- Can be represented as a graph



Sequential tagging

- The process of assigning a part-of-speech to each word in a collection (sentence).

WORD	tag
the	**DET**
koala	**N**
put	**V**
the	**DET**
keys	**N**
on	**P**
the	**DET**
table	**N**

Let’s try

Don’t worry! There is no problem with your eyes or computer.

🙞/DT 🙜⑥🙝/NN ⓪⑩/VBZ 🙚🙛🙞⑩⓪⑤🙝/VBG 🙞/DT 🙚🙞⓿/NN 🗐/.

🙞/DT 🙘⑥❹/NN ⓪⑩/VBZ ⑨❶⑤⑤⓪⑤🙝/VBG 🗐/.

🙞/DT 🙙⑥❺/NN ⓪⑩/VBZ ⑩⓪⑤🙝⓪⑤🙝/VBG 🗐/.
🙞/DT 🙛🙞⑦⑦❺/JJ 🙙⓪⑨🙜/NN

What is the POS tag sequence of the following sentence?

🙞 🙛🙞⑦⑦❺ 🙚🙞⓿ ❸🙞⑩ ⑩⓪⑤🙝⓪⑤🙝 🗐

UCLA ENGINEERING
Computer Science

Let’s try

- 🙐/DT 🙓⑥🙖/NN ⓪⑩/VBZ 🙒🙗🙐⑩⓪⑤🙖/VBG 🙐/DT 🙒🙐⓿/NN 🗐/.
  a/DT dog/NN is/VBZ chasing/VBG a/DT cat/NN ./.
- 🙐/DT 🙕⑥❹/NN ⓪⑩/VBZ ⑨❶⑤⑤⓪⑤🙖/VBG 🗐/.
  a/DT fox/NN is/VBZ running/VBG ./.
- 🙐/DT 🙑⑥❺/NN ⓪⑩/VBZ ⑩⓪⑤🙖⓪⑤🙖/VBG 🗐/.
  a/DT boy/NN is/VBZ singing/VBG ./.
- 🙐/DT 🙗🙐⑦⑦❺/JJ 🙑⓪⑨🙓/NN
  a/DT happy/JJ bird/NN
- 🙐 🙗🙐⑦⑦❺ 🙒🙐⓿ ❸🙐⑩ ⑩⓪⑤🙖⓪⑤🙖 🗐
  a happy cat was singing .

How you predict the tags?

- Two types of information are useful
  - Relations between words and tags
  - Relations between tags and tags
    - DT NN, DT JJ NN…
    - Fed in “The Fed” is a Noun because it follows a Determiner

Combinatorial optimization problem

$$\hat{y} = \text{argmax}_{\boldsymbol{y}\in\boxed{\mathcal{Y}}}\, f(\boldsymbol{y}; \boxed{\boldsymbol{w}}, \boxed{\boldsymbol{x}})$$

input

model parameters

output space

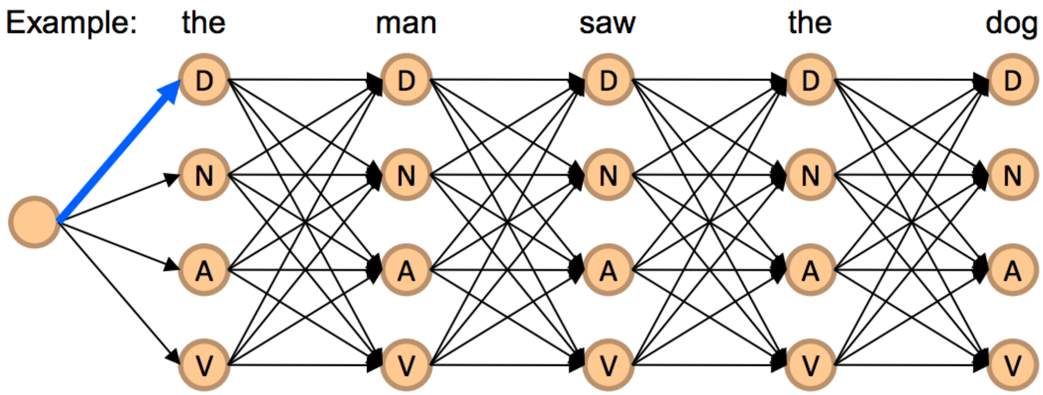
- Inference/Test: given $\boldsymbol{w}, \boldsymbol{x}$, solve argmax
- Learning/Training: find a good $\boldsymbol{w}$

Challenges with structured output

- We cannot train a separate weight vector for each possible inference outcome (why?)
  - For multi-class we train one weight vector for each class
- We cannot enumerate all possible structures for inference
  - Inference for multiclass was easy

Deal with combinatorial output

- Decompose the output into parts that are labeled
- Define a graph to represent
  - how the parts interact with each other
  - These labeled interacting parts are scored; the total score for the graph is the sum of scores of each part
  - an inference algorithm to assign labels to all the parts



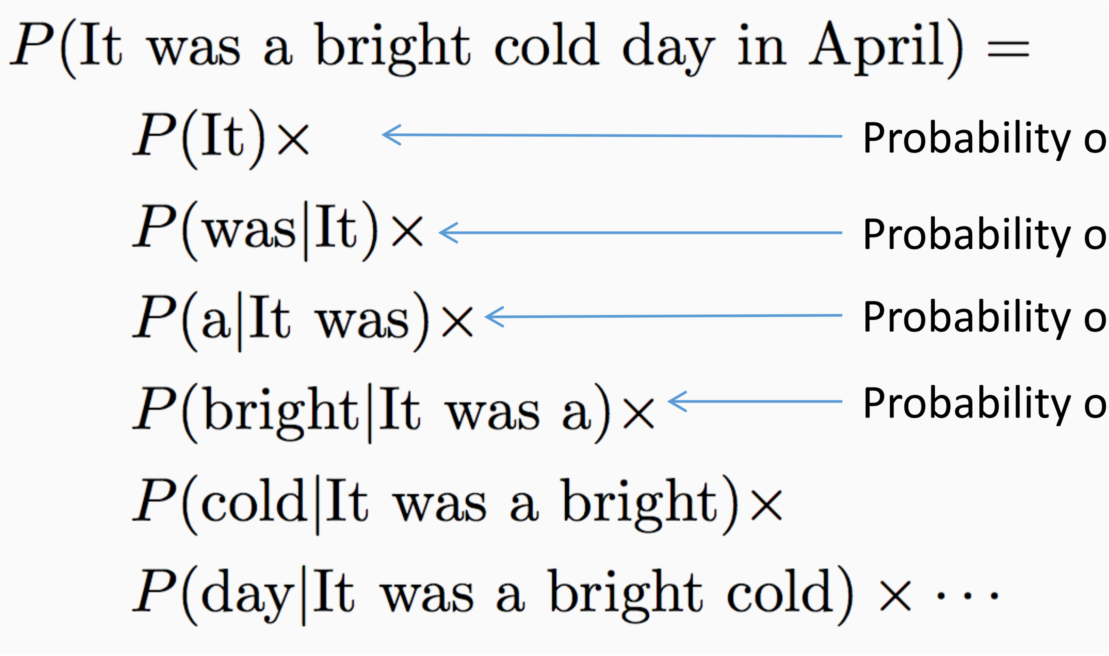
A history-based model

$$P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i | x_1, x_2, \cdots, x_{i-1})$$

- Each token is dependent on all the tokens that came before it
  - Simple conditioning
  - Each $P(x_i$ | …) is a multinomial probability distribution over the tokens

Example: A Language model

It was a bright cold day in April.



$$
\begin{aligned}
&P(\text{It was a bright cold day in April}) = \\
&\qquad P(\text{It})\times \quad \longleftarrow \quad \text{Probability of a word starting a sentence} \\
&\qquad P(\text{was}|\text{It})\times \quad \longleftarrow \quad \text{Probability of a word following “It”} \\
&\qquad P(\text{a}|\text{It was})\times \quad \longleftarrow \quad \text{Probability of a word following “It was”} \\
&\qquad P(\text{bright}|\text{It was a})\times \quad \longleftarrow \quad \text{Probability of a word following “It was a”} \\
&\qquad P(\text{cold}|\text{It was a bright})\times \\
&\qquad P(\text{day}|\text{It was a bright cold}) \times \cdots
\end{aligned}
$$

A history-based model

$$P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i | x_1, x_2, \cdots, x_{i-1})$$

- Each token is dependent on all the tokens that came before it
  - Simple conditioning
  - Each P(x_i | …) is a multinomial probability distribution over the tokens
- What is the problem here?
  - How many parameters do we have?
    - Grows with the size of the sequence!

Solution: Lose the history

Discrete Markov Process

- A system can be in one of K states at a time
- State at time t is x_t
- First-order Markov assumption

  The state of the system at any time is ***independent*** of the full sequence history given the previous state

$$P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i | x_{i-1})$$

  - Defined by two sets of probabilities:
    - Initial state distribution: $P(x_1 = S_j)$
    - State transition probabilities: $P(x_i = S_j \mid x_{i-1} = S_k)$

Example: Another language model

It was a bright cold day in April

$$P(\text{It was a bright cold day in April}) =$$

$$P(\text{It})\times$$

Probability of a word starting a sentence

$$P(\text{was}|\text{It})\times$$

Probability of a word following “It”

$$P(\text{a}|\text{was})\times$$

Probability of a word following “was”

$$P(\text{bright}|\text{a})\times$$

Probability of a word following “a”

$$P(\text{cold}|\text{bright})\times$$

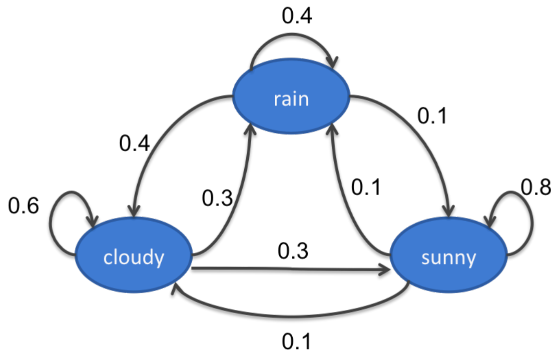
$$P(\text{day}|\text{cold}) \times \cdots$$

If there are K tokens/states, how many parameters do we need? $O(K^2)$

Example: The weather

- Three states: rain, cloudy, sunny

State transitions:



- Observations are Markov chains:

Eg: *cloudy sunny sunny rain*

Probability of the sequence =

P(cloudy) P(sunny|cloudy) P(sunny | sunny) P(rain | sunny)

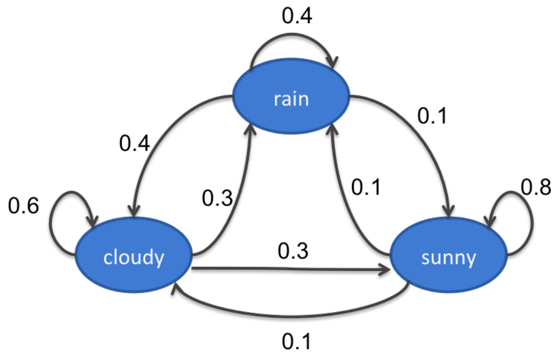
Initial probability

Transition probabilities

Example: The weather

- Three states: rain, cloudy, sunny

State transitions:



- Obse

Eg: *cl*

These probabilities define the model; can find P(any sequence)

Probability of the sequence =

P(cloudy) P(sunny|cloudy) P(sunny | sunny) P(rain | sunny)

Initial probability

Transition probabilities

Outline

- Sequence models
- *Hidden Markov models*
  - Inference with HMM
  - Learning
- Conditional Models and Local Classifiers
- Global models
  - Conditional Random Fields
  - Structured Perceptron for sequences

Hidden Markov Model

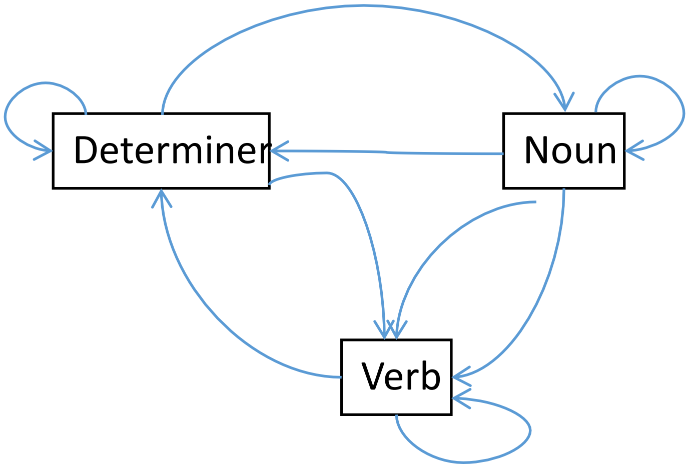
- Discrete Markov Model:
  - States follow a Markov chain
  - *Each state is an observation*

- Hidden Markov Model:
  - States follow a Markov chain
  - States are not observed
  - Each state stochastically emits an observation

Toy part-of-speech example

The Fed raises interest rates

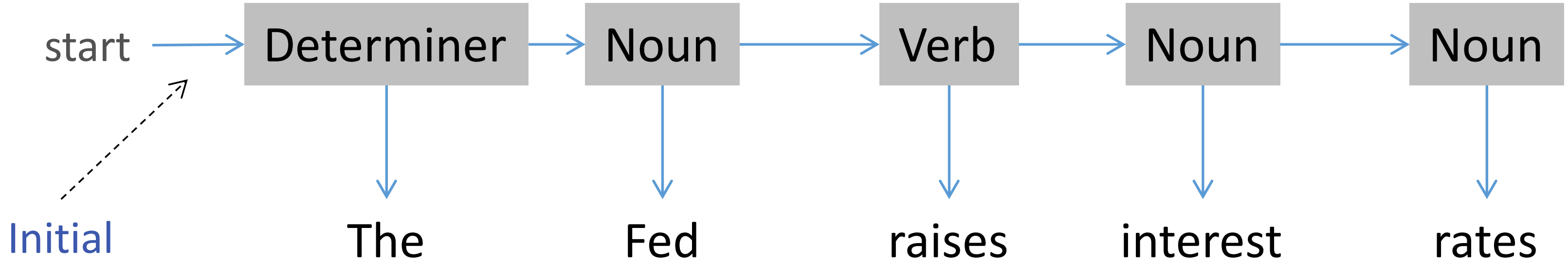
Transitions



Emissions

P(The | Determiner) = 0.5
P(A | Determiner) = 0.3
P(An | Determiner) = 0.1
P(Fed | Determiner) = 0
…

P(Fed| Noun) = 0.001
P(raises| Noun) = 0.04
P(interest| Noun) = 0.07
P(The| Noun) = 0
…



Joint model over states and observations

- Notation
  - Number of states = K, Number of observations = M
  - π: Initial probability over states (K dimensional vector)
  - A: Transition probabilities (K×K matrix)
  - B: Emission probabilities (K×M matrix)

- Probability of states and observations
  - Denote states by y_1, y_2, ⋯ and observations by x_1, x_2, ⋯

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$= \pi_{y_1} \prod_{i=1}^{n-1} A_{y_i, y_{i+1}} \prod_{i=1}^{n} B_{y_i, x_i}$$

Other applications

- Speech recognition
  - Input: Speech signal
  - Output: Sequence of words
- NLP applications
  - Information extraction
  - Text chunking
- Computational biology
  - Aligning protein sequences
  - Labeling nucleotides in a sequence as exons, introns, etc.

Questions?

Three questions for HMMs [Rabiner 1999]

1. Given an observation sequence, x_1, x_2, ⋯ x_n and a model (π, A, B), how to efficiently calculate the probability of the observation?
2. Given an observation sequence, x_1, x_2, ⋯, x_n and a model (π, A, B), how to efficiently calculate the most probable state sequence?

Inference

3. How to calculate (π, A, B) from observations?

Learning

Outline

- Sequence models
- Hidden Markov models
  - *Inference with HMM*
  - Learning
- Conditional Models and Local Classifiers
- Global models
  - Conditional Random Fields
  - Structured Perceptron for sequences

Most likely state sequence

- Input:
  - A hidden Markov model (π, A, B)
  - An observation sequence **x** = (x_1, x_2, ⋯, x_n)
- Output: A state sequence **y** = (y_1, y_2, ⋯, y_n) that corresponds to
  - Maximum *a posteriori* inference (MAP inference)

$$\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, \pi, A, B)$$

- Computationally: combinatorial optimization

MAP inference

- We want $$\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, \pi, A, B)$$

- We have defined

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

- But $P(\mathbf{y}|\mathbf{x}, \pi, A, B) \propto P(\mathbf{x}, \mathbf{y}|\pi, A, B)$
  - And we don’t care about P(**x**) we are maximizing over **y**

- So, $$\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, \pi, A, B) = \arg\max_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}|\pi, A, B)$$

How many possible sequences?

The	Fed	raises	interest	rates
Determiner	Verb	Verb	Verb	Verb
	Noun	Noun	Noun	Noun
1	2	2	2	2

List of allowed tags for each word

In this simple case, 16 sequences (1×2×2×2×2)

Naïve approaches

1. Try out every sequence
   - Score the sequence **y** as P(**y**|**x**, π, A, B)
   - Return the highest scoring one
   - What is the problem?
     - Correct, but slow, $O(K^n)$
2. Greedy search
   - Construct the output left to right
   - For each i, elect the best y_i using y_{i-1} and x_i
   - What is the problem?
     - Incorrect but fast, O(nK)

Solution: Use the independence assumptions

Recall: The first order Markov assumption

The state at token i is only influenced by the previous state, the next state and the token itself

Given the adjacent labels, the others do not matter

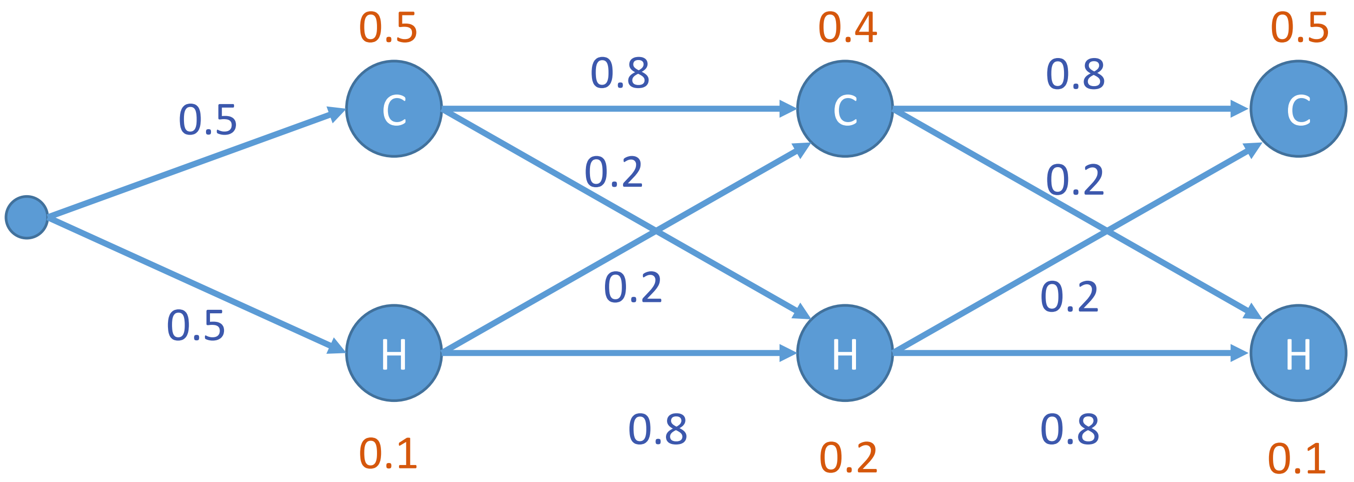
Suggests a recursive algorithm

Jason’s ice cream

#cones

	p(…|C)	p(…|H)	p(…|START)
(1|…)	0.5	0.1	
(2|…)	0.4	0.2	
(3|…)	0.1	0.7	
(C|…)	0.8	0.2	0.5
(H|…)	0.2	0.8	0.5

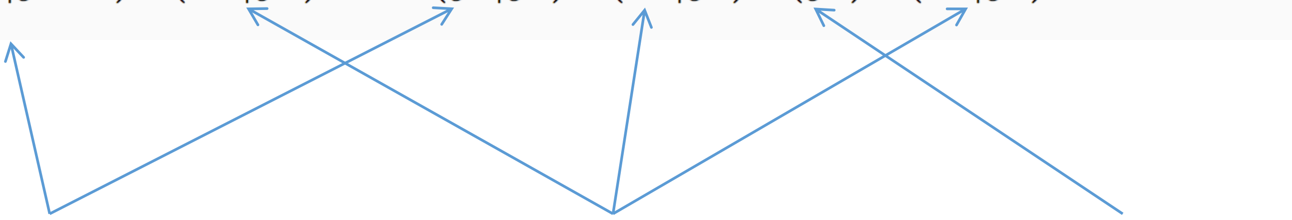
- Best tag sequence for P(”1,2,1”)?



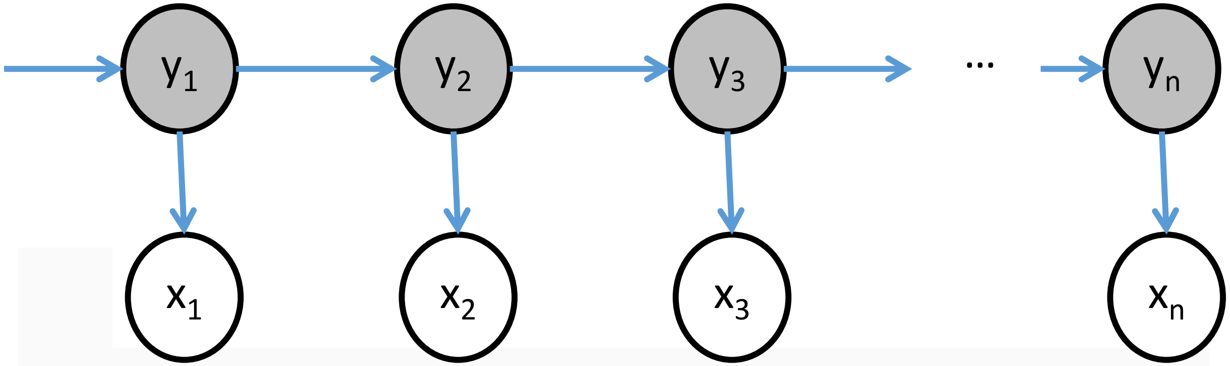
Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$



Transition probabilities Emission probabilities Initial probability



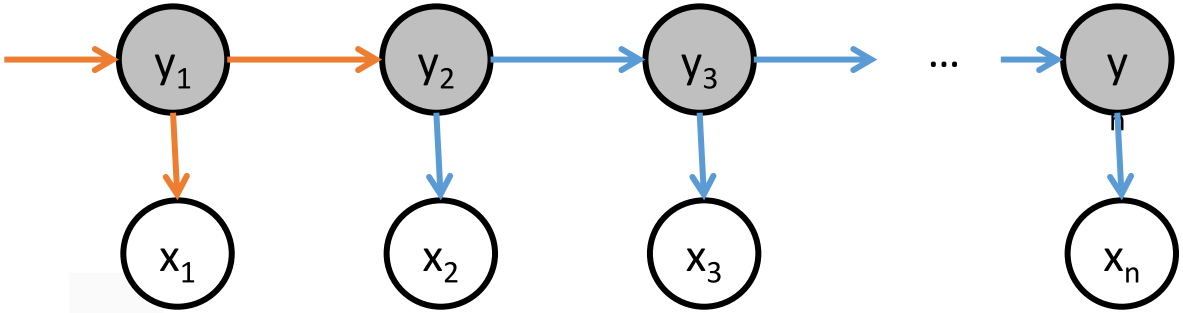
Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1)$$

The only terms that depend on y_1



Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

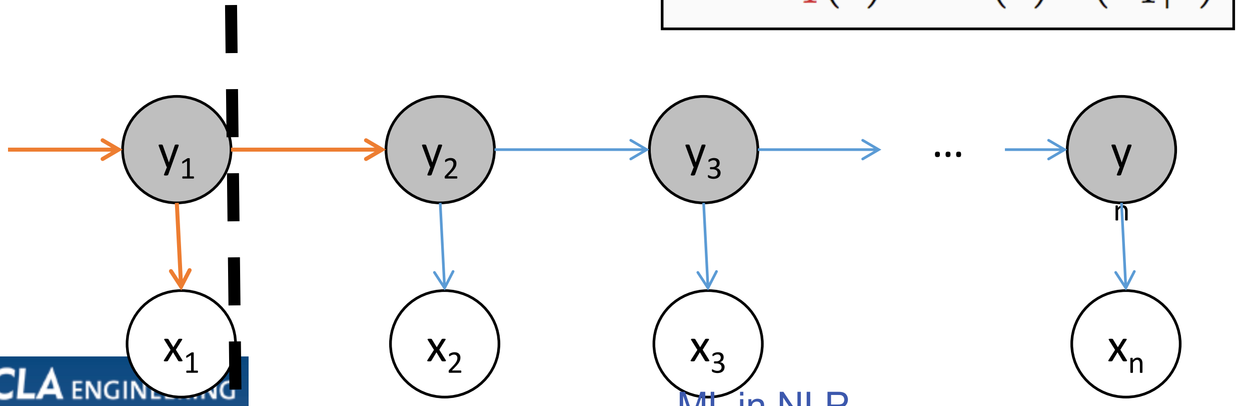
$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1)$$

Abstract away the score for all decisions till here into score

$$\boxed{\text{score}_1(s) = P(s)P(x_1|s)}$$

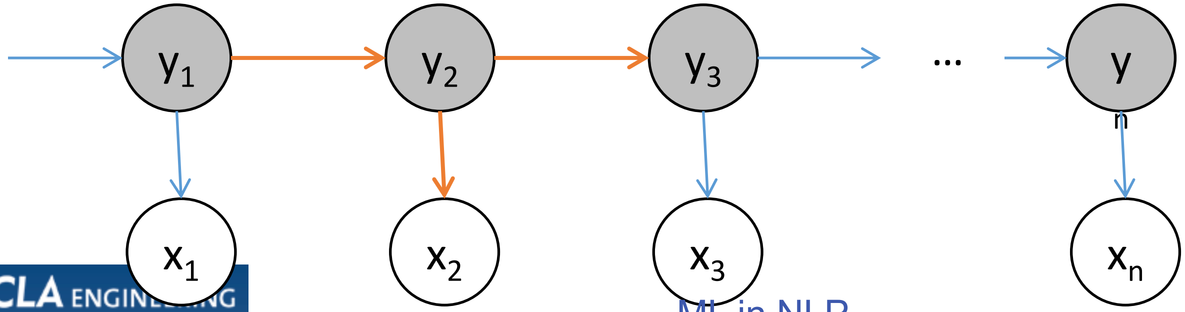


Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\begin{aligned}
&\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1) \\
&= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1) \\
&= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1) P(x_2|y_2) \text{score}_1(y_1) \\
&= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2) P(x_3|y_3) \max_{y_1} P(y_2|y_1) P(x_2|y_2) \text{score}_1(y_1)
\end{aligned}$$

Only terms that depend on y_2



Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

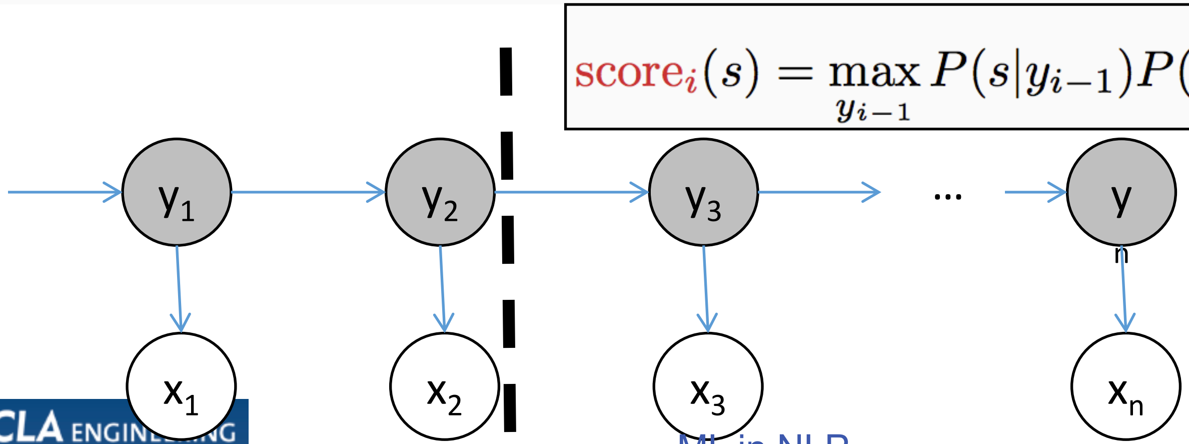
$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1)$$

$$= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3) \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1)$$

$$= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3)\text{score}_2(y_2)$$

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1})$$

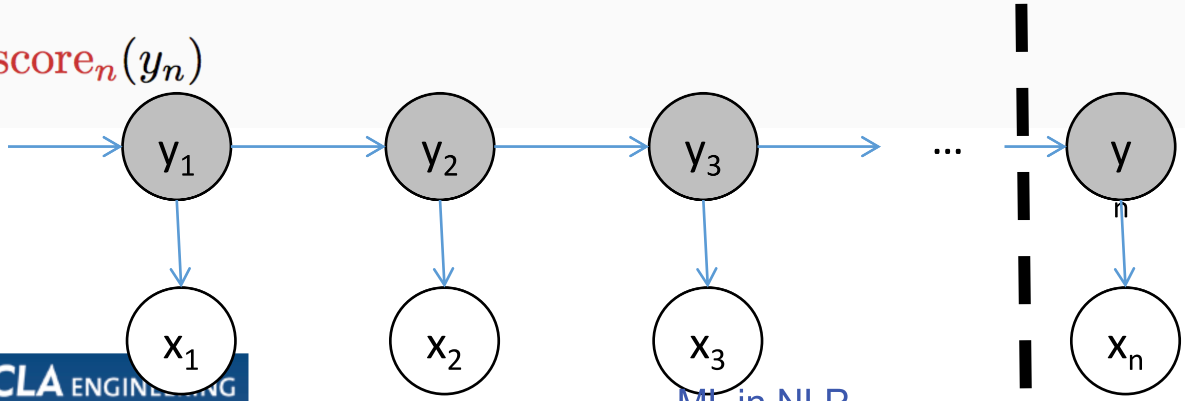


Abstract away the score for all decisions till here into score

Deriving the recursive algorithm

$$P(x_1,x_2,\cdots,x_n,y_1,y_2,\cdots y_n) = P(y_1)\prod_{i=1}^{n-1}P(y_{i+1}|y_i)\prod_{i=1}^{n}P(x_i|y_i)$$

$$\begin{aligned}
&\max_{y_1,y_2,\cdots,y_n} P(y_n|y_{n-1})P(x_n|y_n)\cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)\\
&= \max_{y_2,\cdots,y_n} P(y_n|y_{n-1})P(x_n|y_n)\cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)\\
&= \max_{y_2,\cdots,y_n} P(y_n|y_{n-1})P(x_n|y_n)\cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1)\\
&= \max_{y_3,\cdots,y_n} P(y_n|y_{n-1})P(x_n|y_n)\cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3)\max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1)\\
&= \max_{y_3,\cdots,y_n} P(y_n|y_{n-1})P(x_n|y_n)\cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3)\text{score}_2(y_2)\\
&\vdots\\
&= \max_{y_n} \text{score}_n(y_n)
\end{aligned}$$



Abstract away the score for all decisions till here into score

Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\begin{aligned}
&\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1) \\
&= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1) \\
&= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1) P(x_2|y_2) \text{score}_1(y_1) \\
&= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2) P(x_3|y_3) \max_{y_1} P(y_2|y_1) P(x_2|y_2) \text{score}_1(y_1) \\
&= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2) P(x_3|y_3) \text{score}_2(y_2) \\
&\ \vdots \\
&= \max_{y_n} \text{score}_n(y_n)
\end{aligned}$$

$$\text{score}_1(s) = P(s) P(x_1|s)$$

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1}) P(x_i|s) \text{score}_{i-1}(y_{i-1})$$

Viterbi algorithm

π: Initial probabilities
A: Transitions
B: Emissions

Max-product algorithm for first order sequences

1. Initial: For each state s, calculate

$$\text{score}_1(s) = P(s)P(x_1|s) = \pi_s B_{x_1,s}$$

1. Recurrence: For i = 2 to n, for every state s, calculate

$$\begin{aligned}\text{score}_i(s) &= \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1}) \\ &= \max_{y_{i-1}} A_{y_{i-1},s}B_{s,x_i}\text{score}_{i-1}(y_{i-1})\end{aligned}$$

1. Final state: calculate

$$\max_{\mathbf{y}} P(\mathbf{y},\mathbf{x}|\pi,A,B) = \max_s \text{score}_n(s)$$

This only calculates the max. To get final answer (*argmax*),

- keep track of which state corresponds to the max at each step
- build the answer using these back pointers

Questions?

General idea

- Dynamic programming
  - The best solution for the full problem relies on best solution to sub-problems
  - Memoize partial computation

- Examples
  - Viterbi algorithm
  - Dijkstra’s shortest path algorithm
  - …

Complexity of inference

- Complexity parameters
  - Input sequence length: n
  - Number of states: K
- Memory
  - Storing the table: nK (scores for all states at each position)
- Runtime
  - At each step, go over pairs of states
  - $O(nK^2)$

Outline

- Sequence models
- Hidden Markov models
  - Inference with HMM
  - *Learning*
- Conditional Models and Local Classifiers
- Global models
  - Conditional Random Fields
  - Structured Perceptron for sequences

Learning HMM parameters

- Assume we know the number of states in the HMM
- Two possible scenarios
  1. We are given a data set D = {<$\mathbf{x}_i$, $\mathbf{y}_i$>} of sequences labeled with states

     And we have to learn the parameters of the HMM (π, A, B)

     Supervised learning with complete data
  2. We are given only a collection of sequences D = {$\mathbf{x}_i$}

     And we have to learn the parameters of the HMM (π, A, B)

     Unsupervised learning, with incomplete data

Supervised learning of HMM

- We are given a dataset D = {<$\mathbf{x}_i$, $\mathbf{y}_i$>}
  - Each $\mathbf{x}_i$ is a sequence of observations and $\mathbf{y}_i$ is a sequence of states that correspond to $\mathbf{x}_i$

  Goal: Learn initial, transition, emission distributions (π, A, B)

- How do we learn the parameters of the probability distribution?
  - The maximum likelihood principle

Where have we seen this before?

$$(\hat{\pi}, \hat{A}, \hat{B}) = \max_{\pi, A, B} P(D|\pi, A, B) = \max_{\pi, A, B} \prod_i \boxed{P(\mathbf{x}_i, \mathbf{y}_i | \pi, A, B)}$$

And we know how to write this in terms of the parameters of the HMM

Supervised learning details

$$(\hat{\pi}, \hat{A}, \hat{B}) = \max_{\pi,A,B} P(D|\pi, A, B) = \max_{\pi,A,B} \prod_i P(\mathbf{x}_i, \mathbf{y}_i|\pi, A, B)$$

π, A, B can be estimated separately just by counting

- Makes learning simple and fast

[Exercise: Derive the following using derivatives of the log likelihood. Requires Lagrangian multipliers.]

Number of instances where the first state is s

$$\pi_s = \frac{\text{count}(\text{start} \rightarrow s)}{n}$$

Number of examples

Initial probabilities

$$A_{s',s} = \frac{\text{count}\,(s \rightarrow s')}{\text{count}\,(s)}$$

Transition probabilities

$$B_{s,x} = \frac{\text{count}\begin{pmatrix} s \\ \downarrow \\ x \end{pmatrix}}{\text{count}\,(s)}$$

Emission probabilities

Hidden Markov Models summary

- Predicting sequences
  - As many output states as observations
- Markov assumption helps decompose the score
- Several algorithmic questions
  - Most likely state
  - Learning parameters
    - Supervised, Unsupervised
  - Probability of an observation sequence
    - Sum over all assignments to states, replace max with sum in Viterbi
  - Probability of state for each observation
    - Sum over all assignments to all other states

Outline

- Sequence models
- Hidden Markov models
  - Inference with HMM
  - Learning
- Conditional Models and Local Classifiers
- Global models
  - Conditional Random Fields

Outline

- Sequence models
- Hidden Markov models
  - Inference with HMM
  - Learning
- Conditional Models and Local Classifiers
- Global models
  - Conditional Random Fields

Modeling next-state directly

- Instead of modeling the joint distribution P(**x**, **y**) only focus on P(**y**|**x**)
  - Which is what we care about eventually anyway

- For sequences, different formulations
  - Maximum Entropy Markov Model [McCallum, et al 2000]
  - Projection-based Markov Model [Punyakanok and Roth, 2001]

(other names: discriminative/conditional markov model, …)

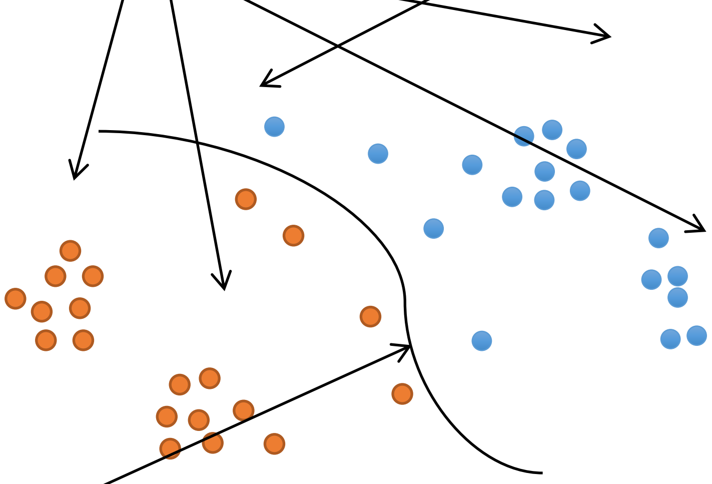
Generative vs Discriminative models

- Generative models
  - learn P(x, y)
  - Characterize how the data is generated (both inputs and outputs)
  - Eg: Naïve Bayes, Hidden Markov Model
- Discriminative models
  - learn P(y | x)
  - Directly characterizes the decision boundary only
  - Eg: Logistic Regression, Conditional models (several names)

Generative vs Discriminative models

- Generative models
  - learn P(x, y)
  - Characterize how the data is generated (both inputs and outputs)
  - Eg: Naïve Bayes, Hidden Markov Model

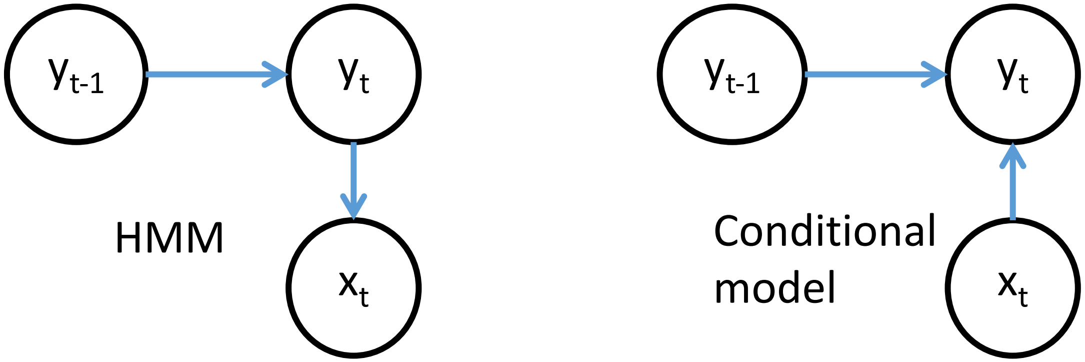
A generative model tries to characterize the distribution of the inputs, a discriminative model doesn't care



- Discriminative models
  - learn P(y | x)
  - Directly characterizes the decision boundary only
  - Eg: Logistic Regression, Conditional models (several names)

Another independence assumption

$$P(y_i|y_{i-1}, y_{i-2}, \cdots, x_i, x_{i-1}, \cdots) = P(y_i|y_{i-1}, x_i)$$



This assumption lets us write the conditional probability of the output as

$$P(\mathbf{y}|\mathbf{x}) = \prod_i P(y_i|y_{i-1}, x_i)$$

Modeling $P(y_i \mid y_{i-1}, x_i)$

- Different approaches possible
  1. Train a *log-linear* classifier
  2. Or, ignore the fact that we are predicting a probability, we only care about maximizing some *score.* Train any classifier (e.g, perceptron algorithm)
- For both cases*:*
  - Use rich features that depend on input and previous state
  - We can increase the dependency to arbitrary neighboring x_i's
    - Eg. Neighboring words influence this words POS tag

Log-linear models for multiclass

Consider multiclass classification

- Inputs: **x**
- Output: **y** ∈ {1, 2, ··· , K}
- Feature representation: ϕ(**x**, **y**)
  - We have seen this before

- Define probability of an input **x** taking a label **y** as

$$P(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \frac{e^{\mathbf{w}^T \phi(\mathbf{x},\mathbf{y})}}{\sum_{y'} e^{\mathbf{w}^T \phi(\mathbf{x},\mathbf{y})}}$$

Interpretation: Score for label, converted to a well-formed probability distribution by exponentiating + normalizing

- A generalization of logistic regression to multiclass

Training a log-linear model

Given a data set D = {<$\mathbf{x}_i$, $\mathbf{y}_i$>}

- Apply the maximum likelihood principle

$$\max_{\mathbf{w}} \sum_i \log P(\mathbf{y}_i | \mathbf{x}_i, \mathbf{w})$$

Here

$$P(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \frac{e^{\mathbf{w}^T \phi(\mathbf{x},\mathbf{y})}}{\sum_{y'} e^{\mathbf{w}^T \phi(\mathbf{x},\mathbf{y})}}$$

- Maybe with a regularizer

$$\max_{\mathbf{w}} -\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_i \log P(\mathbf{y}_i | \mathbf{x}_i, \mathbf{w})$$

Training a log-linear model

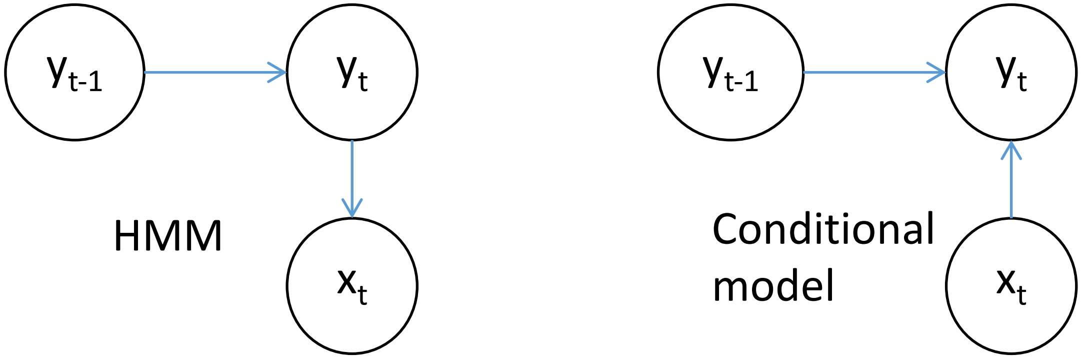
- Gradient based methods to minimize

$$L(\mathbf{w}) = \sum_i \log P(\mathbf{y}_i | \mathbf{x}_i, \mathbf{w})$$

- Usual stochastic gradient descent
  - Initialize $\boldsymbol{w} \leftarrow \mathbf{0}$
  - Iterate through examples for multiple epochs
    - For each example $(\boldsymbol{x}_i\ \boldsymbol{y}_i)$ take gradient step for the loss at that example
      - Update $\boldsymbol{w} \leftarrow \boldsymbol{w}\ - \mathrm{r}_\mathrm{t} \nabla L(\boldsymbol{w}, \boldsymbol{x}_i, \boldsymbol{y}_i)$
  - Return $\boldsymbol{w}$

The next-state model

$$P(y_i | y_{i-1}, y_{i-2}, \cdots, x_i, x_{i-1}, \cdots) = P(y_i | y_{i-1}, x_i)$$



This assumption lets us write the conditional probability of the output as

$$P(\mathbf{y}|\mathbf{x}) = \prod_i \boxed{P(y_i|y_{i-1}, x_i)}$$

We need to learn this function

Modeling ~~$P(y_i \mid y_{i-1}, x_i)$~~ $P(y_i \mid y_{i-1}, \mathbf{x})$

- Different approaches possible
  1. Train a *maximum entropy* classifier

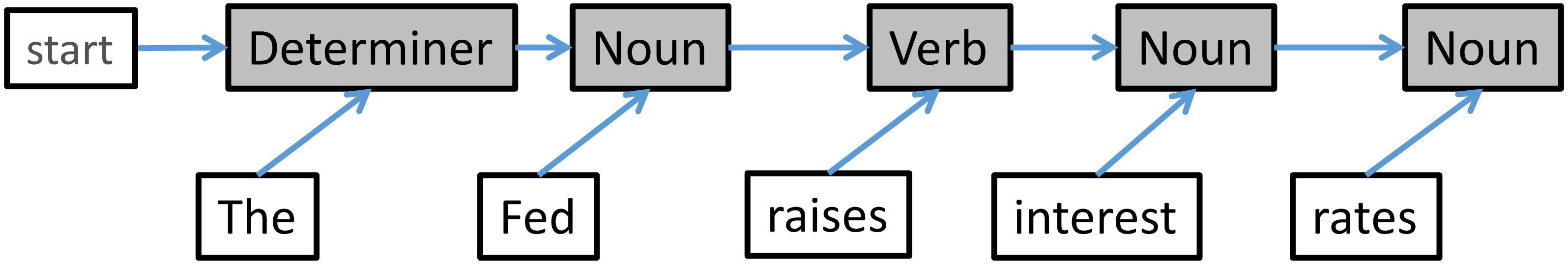
     Basically, multinomial logistic regression

  2. Ignore the fact that we are predicting a probability, we only care about maximizing some *score.* Train any classifier, using say the perceptron algorithm
- For both cases*:*
  - Use rich features that depend on input and previous state
  - We can increase the dependency to arbitrary neighboring x_i's
    - Eg. Neighboring words influence this words POS tag

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

$$P(y_i | y_{i-1}, \mathbf{x}) \propto \exp\left(\mathbf{w}^T \phi(\mathbf{x}, i, y_i, y_{i-1})\right)$$

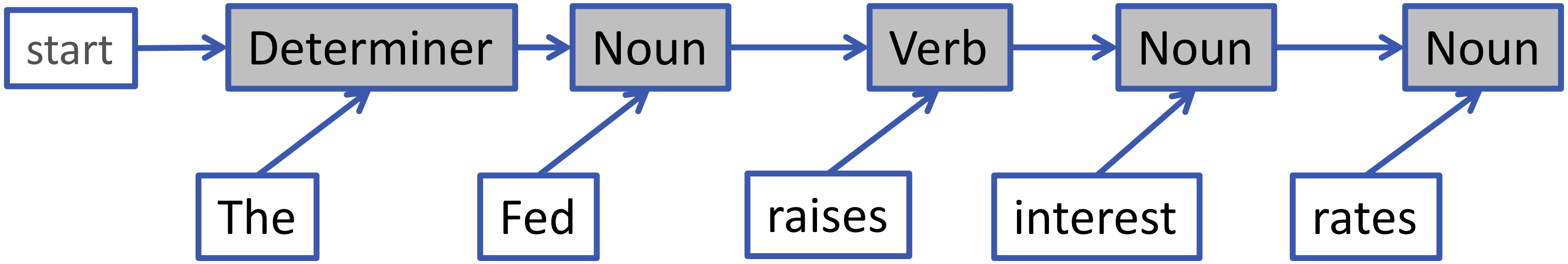


The prediction task:

Using the entire input and the current label, predict the next label

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**) $P(y_i|y_{i-1},\mathbf{x}) \propto \exp\left(\mathbf{w}^T\phi(\mathbf{x},i,y_i,y_{i-1})\right)$



word

Caps

-es

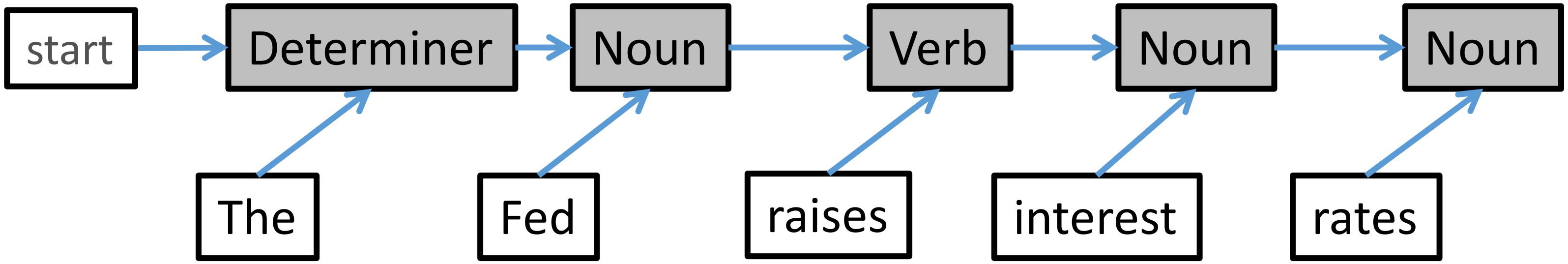
Previous

To model the probability, first, we need to define features for the current classification problem

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

$$P(y_i|y_{i-1},\mathbf{x}) \propto \exp\left(\mathbf{w}^T\phi(\mathbf{x},i,y_i,y_{i-1})\right)$$



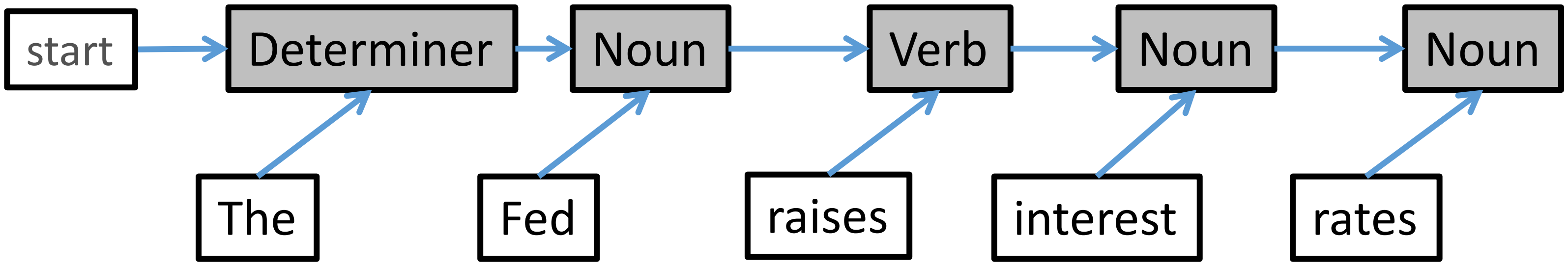
$$\begin{array}{rc} \textit{word} & \left[\begin{array}{c} \text{The} \\ \text{Y} \\ \text{N} \\ \text{start} \end{array}\right] \\ \textit{Caps} & \\ \textit{-es} & \\ \textit{Previous} & \end{array}$$

ϕ(x, 0, start, y_0)

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

$$P(y_i | y_{i-1}, \mathbf{x}) \propto \exp\left(\mathbf{w}^T \phi(\mathbf{x}, i, y_i, y_{i-1})\right)$$



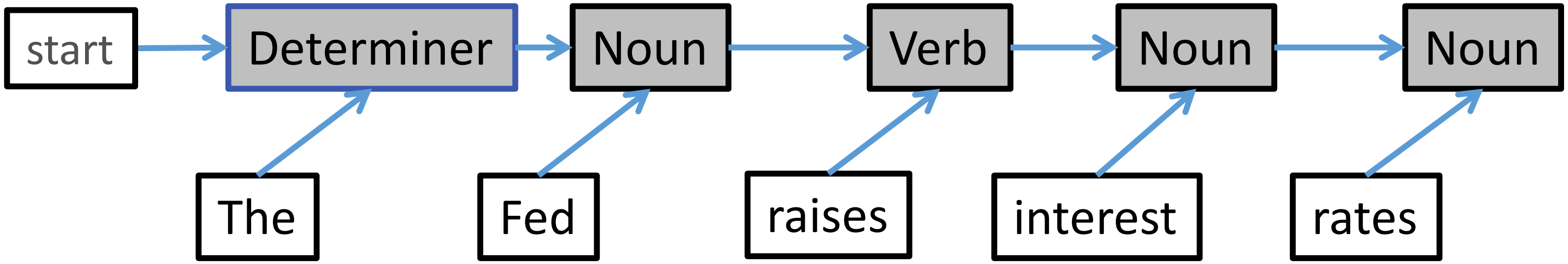
$$\begin{array}{r} \textit{word} \\ \textit{Caps} \\ \textit{-es} \\ \textit{Previous} \end{array} \quad \begin{bmatrix} \text{The} \\ \text{Y} \\ \text{N} \\ \text{start} \end{bmatrix} \quad \begin{bmatrix} \text{Fed} \\ \text{Y} \\ \text{N} \\ \text{Determiner} \end{bmatrix}$$

$\phi(x, 0, \text{start}, y_0)$ $\phi(x, 1, y_{0,}\ y_1)$

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

$$P(y_i | y_{i-1}, \mathbf{x}) \propto \exp\left(\mathbf{w}^T \phi(\mathbf{x}, i, y_i, y_{i-1})\right)$$



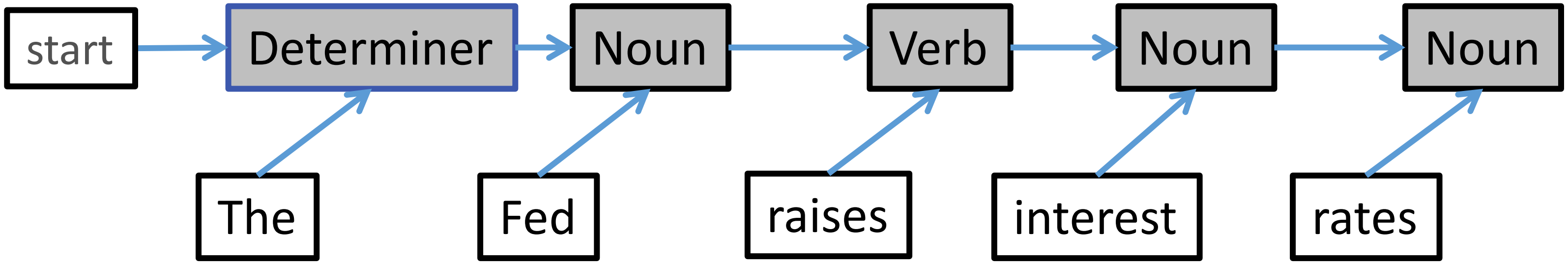
word	The	Fed	raises	interest	rates
Caps	Y	Y	N	N	N
-es	N	N	Y	N	N
Previous	start	Determiner	Noun	Verb	Noun
	$\phi(x, 0, \text{start}, y_0)$	$\phi(x, 1, y_{0,} y_1)$	$\phi(x, 2, y_1, y_2)$	$\phi(x, 3, y_2, y_3)$	$\phi(x, 4, y_3, y_4)$

Can get very creative here

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

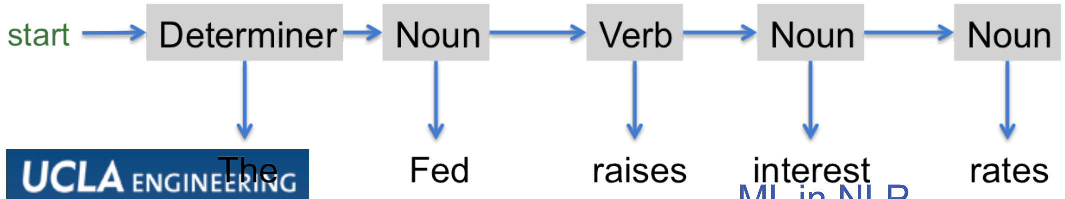
$$P(y_i|y_{i-1},\mathbf{x}) \propto \exp\left(\mathbf{w}^T\phi(\mathbf{x},i,y_i,y_{i-1})\right)$$



word	The	Fed	raises	interest	rates
Caps	Y	Y	N	N	N
-es	N	N	Y	N	N
Previous	start	Determiner	Noun	Verb	Noun
	$\phi(x, 0, \text{start}, y_0)$	$\phi(x, 1, y_{0,}\ y_1)$	$\phi(x, 2, y_1, y_2)$	$\phi(x, 3, y_2, y_3)$	$\phi(x, 4, y_3, y_4)$

Can get very creative here

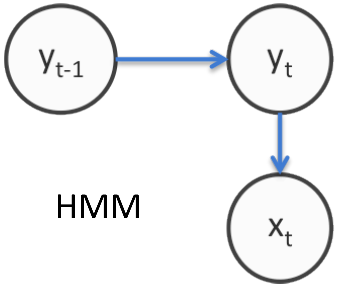
Compare to HMM: Only depends on the word and the previous tag



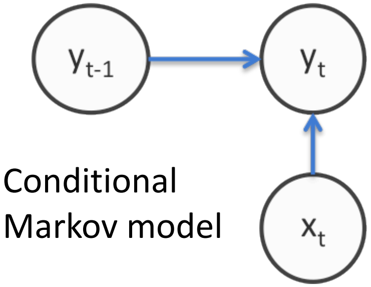
Questions?

Using MEMM

- Training
  - Next-state predictor **locally** as maximum likelihood
    - Similar to any maximum entropy classifier
- Prediction/decoding
  - Modify the Viterbi algorithm for the new independence assumptions



HMM



Conditional Markov model

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1})$$

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1}, x_i)\text{score}_{i-1}(y_{i-1})$$

Generalization: Any multiclass classifier

- **Viterbi decoding**: we only need a score for each decision
  - So far, probabilistic classifiers

- In general, use any learning algorithm to build get a score for the label y_i given y_{i-1} and $\mathbf{x}$
  - Multiclass versions of perceptron, SVM
  - Just like MEMM, these allow arbitrary features to be defined

Exercise: Viterbi needs to be re-defined to work with sum of scores rather than the product of probabilities

Comparison to HMM

What we gain

1. Rich feature representation for inputs
   - ❖ Helps generalize better by thinking about properties of the input tokens rather than the entire tokens
   - ❖ Eg: If a word ends with –es, it might be a present tense verb (such as raises). Could be a feature; HMM cannot capture this

2. Discriminative predictor
   - ❖ Model P(**y** | **x**) rather than P(**y**, x)
   - ❖ *Joint* vs *conditional*

Outline

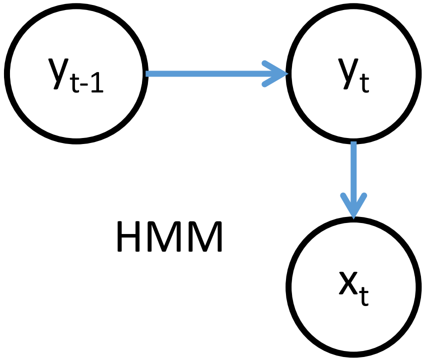
- Sequence models
- Hidden Markov models
  - Inference with HMM
  - Learning
- Conditional Models and Local Classifiers
- Global models
  - Conditional Random Fields
  - Structured Perceptron for sequences

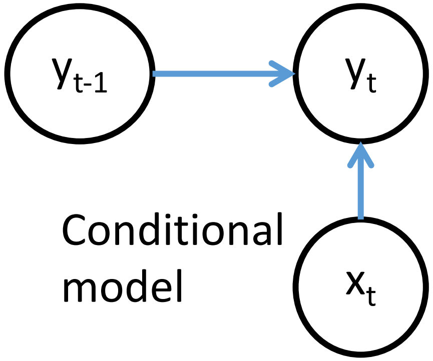
Outline

- Conditional models for predicting sequences
- Log-linear models for multiclass classification
- Maximum Entropy Markov Models
  - The Label Bias Problem

The next-state model for sequences

$$P(y_i|y_{i-1}, y_{i-2}, \cdots, x_i, x_{i-1}, \cdots) = P(y_i|y_{i-1}, x_i)$$





This assumption lets us write the conditional probability of the output as

$$P(\mathbf{y}|\mathbf{x}) = \prod_i P(y_i|y_{i-1}, x_i)$$

We need to train local multiclass classifiers that predicts the next state given the previous state and the input

…local classifiers! Label bias problem

Let’s look at the independence assumption

$$P(y_i|y_{i-1}, y_{i-2}, \cdots, x_i, x_{i-1}, \cdots) = P(y_i|y_{i-1}, x_i)$$

“Next-state” classifiers are locally normalized

…local classifiers! Label bias problem

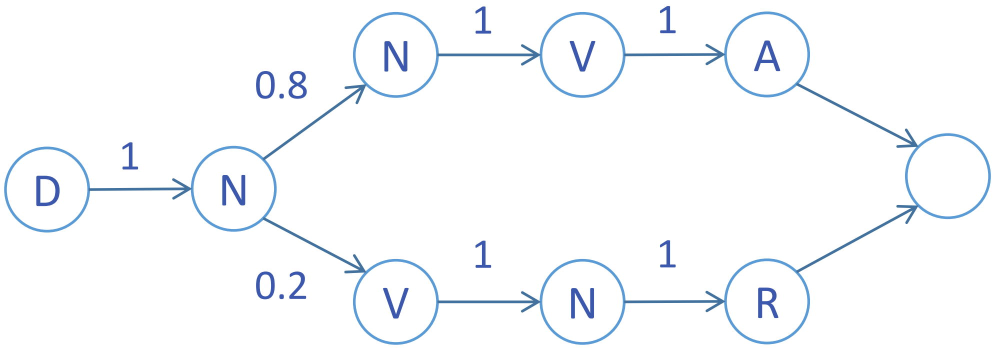
Let’s look at the independence assumption

$$P(y_i|y_{i-1}, y_{i-2}, \cdots, x_i, x_{i-1}, \cdots) = P(y_i|y_{i-1}, x_i)$$

“Next-state” classifiers are locally normalized

Eg: Part-of-speech tagging the sentence

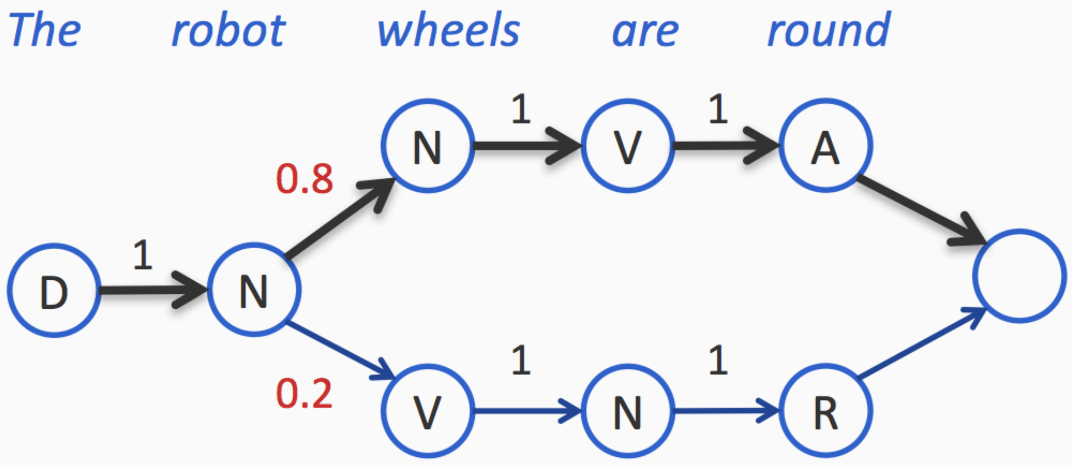
The robot wheels are round



Suppose these are the only state transitions allowed

Example based on [Wallach 2002]

…local classifiers → Label bias problem

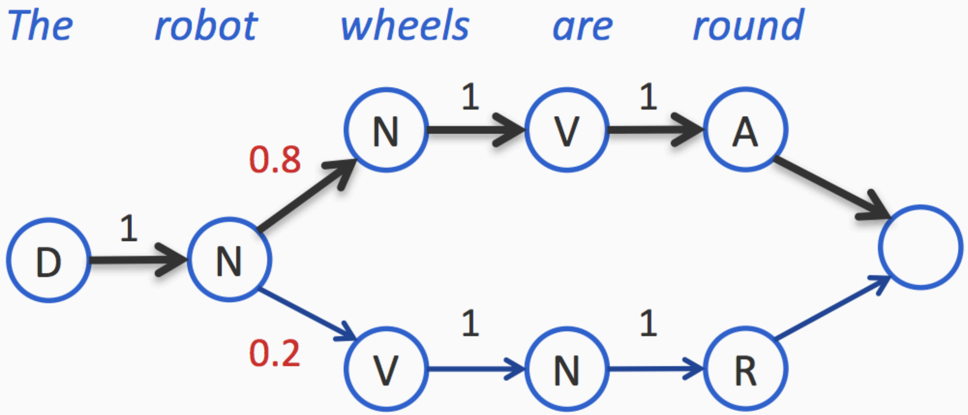


Suppose these are the only state transitions allowed

Option 1: P(D | The) ·
P(N | D, robot) ·
P(N | N, wheels) ·
P(V | N, are) ·
P(A | V, round)

Option 2: P(D | The) ·
P(N | D, robot) ·
P(V | N, wheels) ·
P(N | V, are) ·
P(R| N, round)

…local classifiers → Label bias problem



Suppose these are the only state transitions allowed

Option 1: P(D | The) ·
P(N | D, robot) ·
P(N | N, wheels) ·
~~P(V | N, are)~~ · P(V | N, Fred) ·
P(A | V, round)

Option 2: P(D | The) ·
P(N | D, robot) ·
P(V | N, wheels) ·
~~P(N | V, are)~~ · P(N | V, Fred) ·
P(R| N, round)

The path scores are the same
Even if the word Fred is never observed as a verb in the data, it will be predicted as one
The input Fred does not influence the output at all

Label Bias

- States with a single outgoing transition effectively ignore their input
  - States with lower-entropy next states are less influenced by observations
- Why?
  - Each the next-state classifiers are locally normalized.
  - If a state has fewer next states, each of those will get a higher probability mass
    …and hence preferred
- Side note: Surprisingly doesn’t affect some tasks
  - Eg: part-of-speech tagging

Summary: Local models for Sequences

- Conditional models
- Use rich features in the mode
- Possibly suffer from label bias problem

Outline

- Sequence models
- Hidden Markov models
  - Inference with HMM
  - Learning
- Conditional Models and Local Classifiers
- Global models
  - Conditional Random Fields

Outline

- Sequence models
- Hidden Markov models
  - Inference with HMM
  - Learning
- Conditional Models and Local Classifiers
- Global models
  - *Conditional* Random Fields

So far…

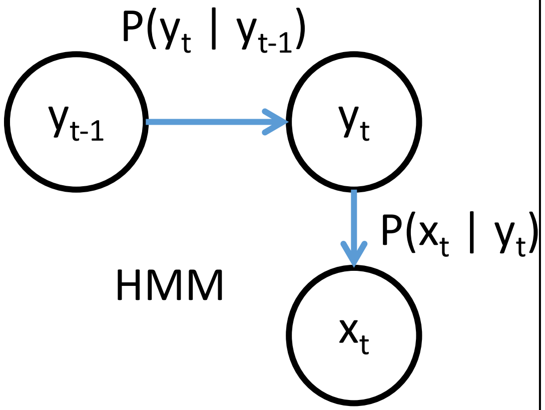
- Hidden Markov models
  - Pros: Decomposition of total probability with tractable
  - Cons: Doesn’t allow use of features for representing inputs
    - Also, generative model
      - not really a downside, but we may get better performance with conditional models if we care only about predictions

- Local, conditional Markov Models
  - Pros: Conditional model, allows features to be used
  - Cons: Label bias problem

Global models

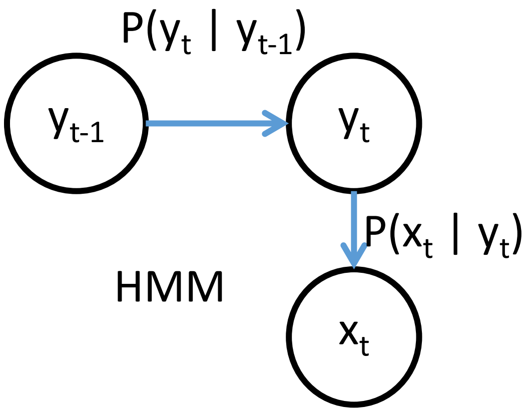
- Train the predictor globally
  - Instead of training local decisions independently
- *Normalize globally*
  - Make each edge in the model undirected
  - Not associated with a probability, but just a “score”
- Recall the difference between local vs. global for multiclass

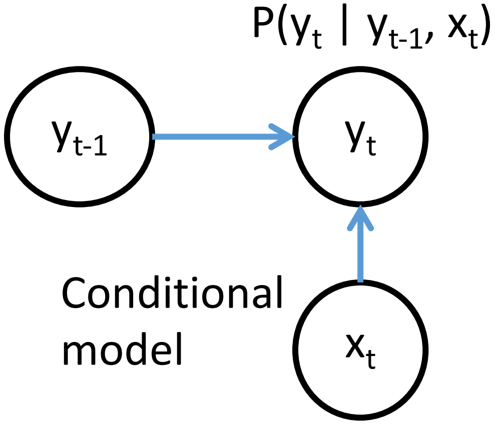
HMM vs. A local model vs. A global model



Generative

HMM vs. A local model vs. A global model



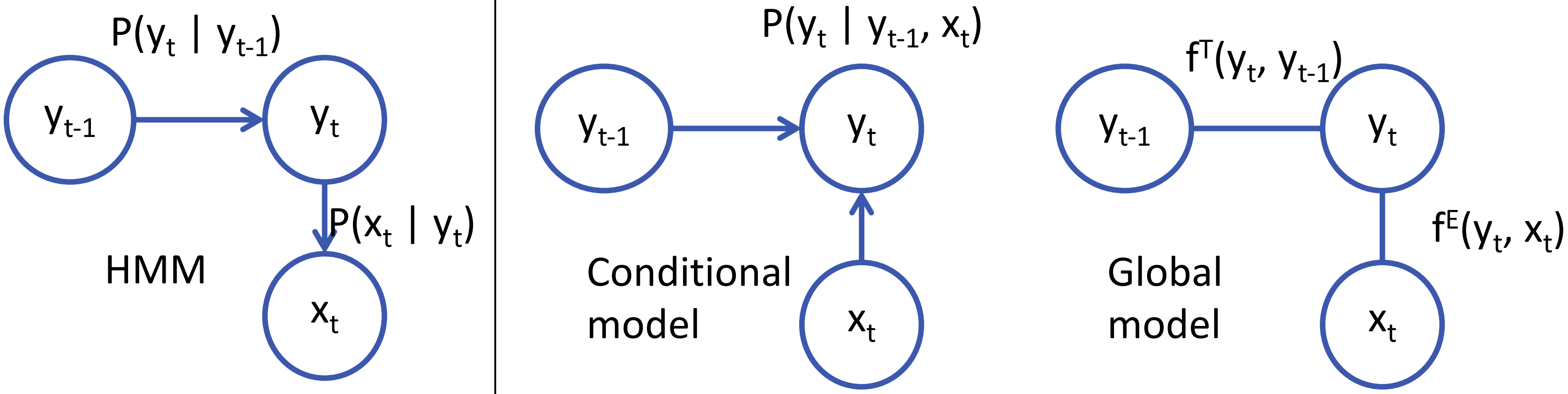


Generative

Discriminative

Local: P is locally normalized to add up to one for each t

HMM vs. A local model vs. A global model



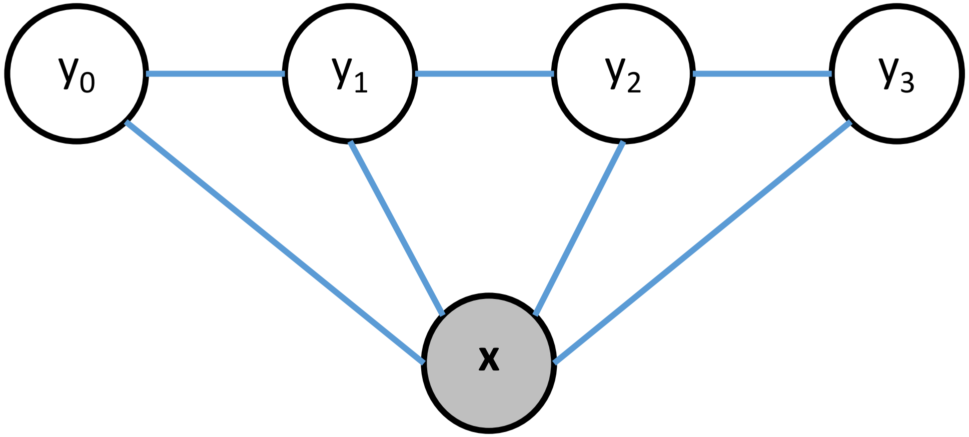
Generative

Discriminative

Local: P is locally normalized to add up to one for each t

Global: The functions f^T and f^E are scores that are not normalized

Conditional Random Field

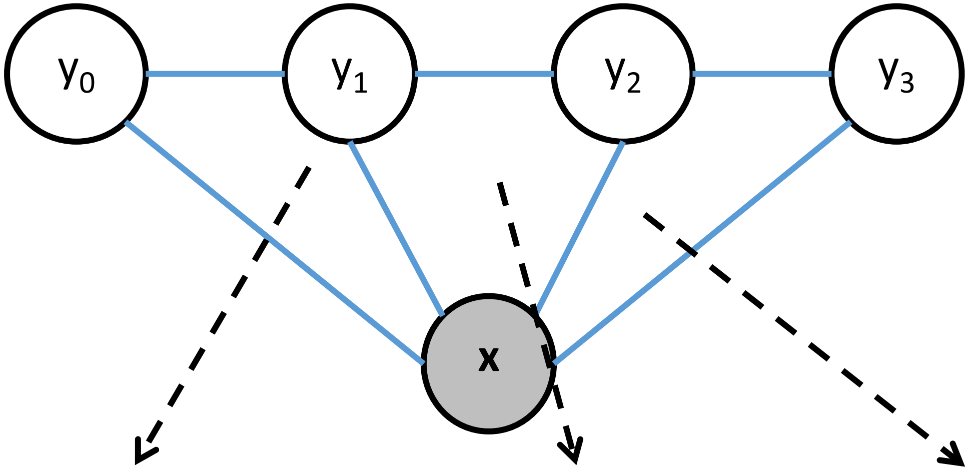


Each node is a random variable

We observe some nodes and the rest are unobserved

The goal: To characterize a probability distribution over the unobserved variables, given the observed

Conditional Random Field



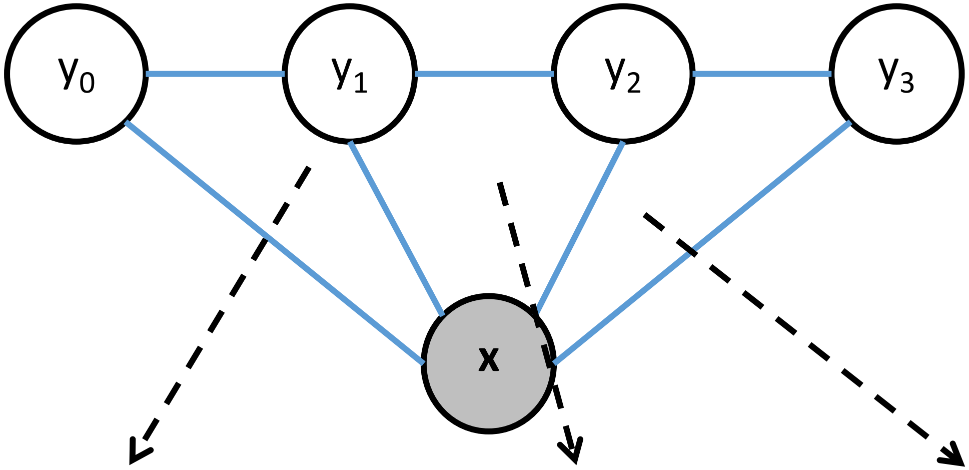
score(**x**, y_0, y_1) **score**(**x**, y_1, y_2) **score**(**x**, y_2, y_3)

Each node is a random variable

We observe some nodes and need to assign the rest

Each ***clique*** is associated with a score

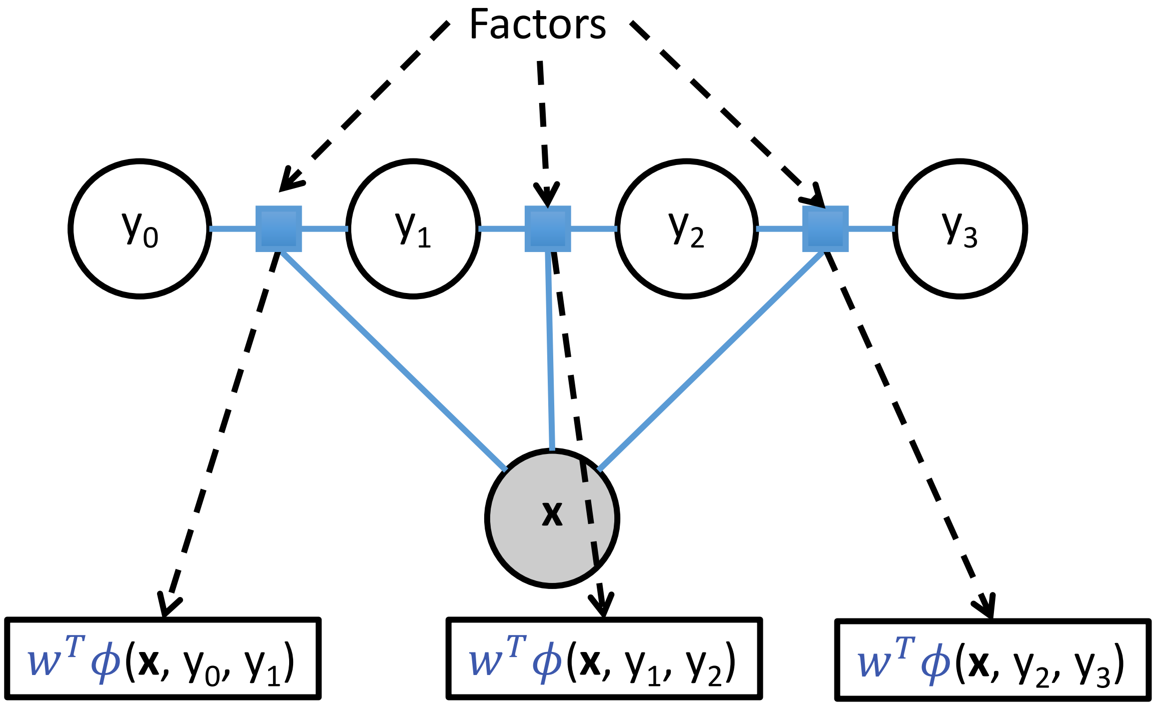
Conditional Random Field



$w^T\phi(\mathbf{x}, y_0, y_1)$ $w^T\phi(\mathbf{x}, y_1, y_2)$ $w^T\phi(\mathbf{x}, y_2, y_3)$

Each node is a random variable
We observe some nodes and need to assign the rest
Each ***clique*** is associated with a score

Conditional Random Field: Factor graph



$w^T\phi(\mathbf{x}, y_0, y_1)$

$w^T\phi(\mathbf{x}, y_1, y_2)$

$w^T\phi(\mathbf{x}, y_2, y_3)$

Each node is a random variable

We observe some nodes and need to assign the rest

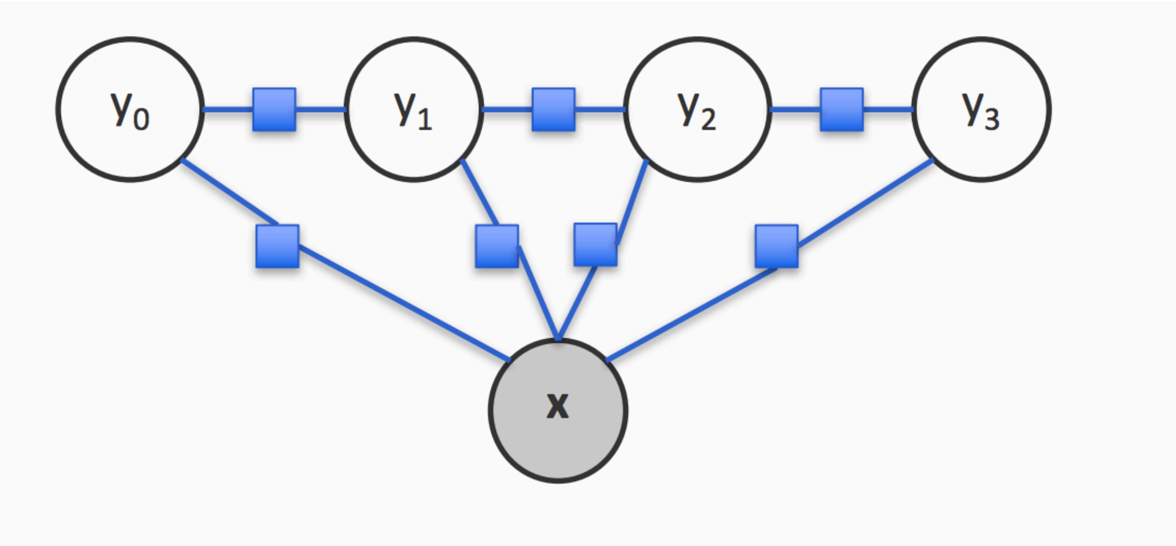
Each ~~clique~~ is associated with a score

factor

Conditional Random Field: Factor graph

A different factorization:

Recall decomposition of structures into parts. Same idea

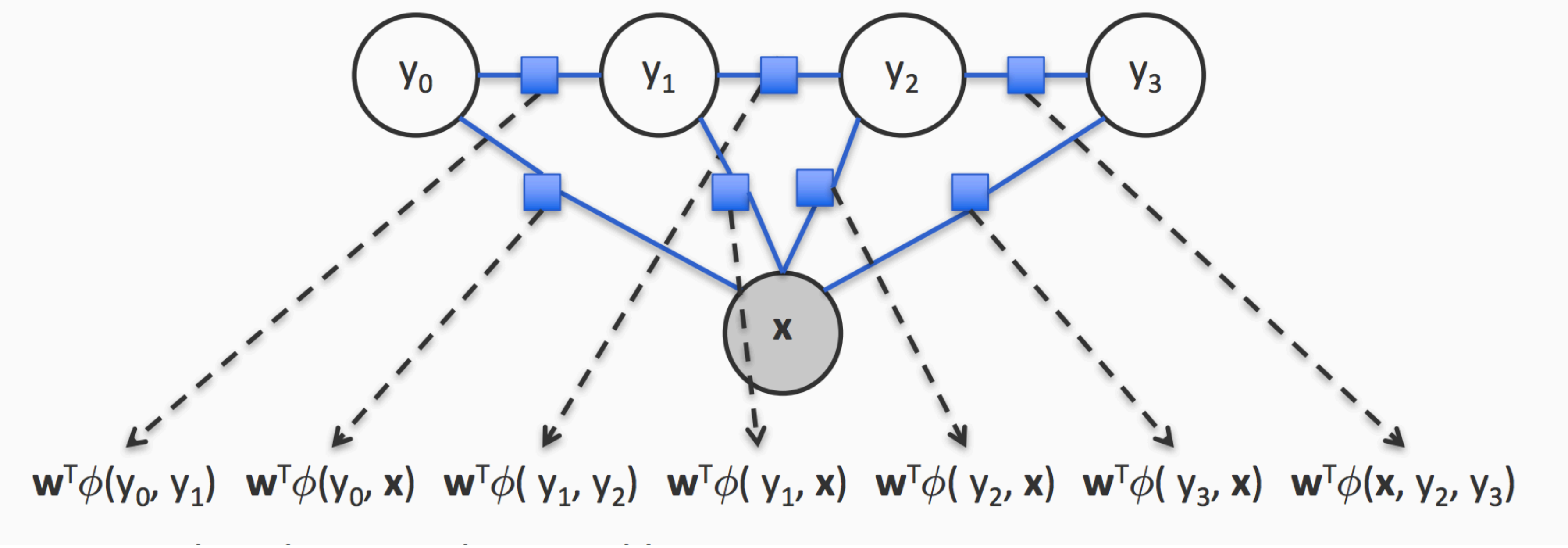


Each node is a random variable

We observe some nodes and need to assign the rest

Each factor is associated with a score

Conditional Random Field: Factor graph



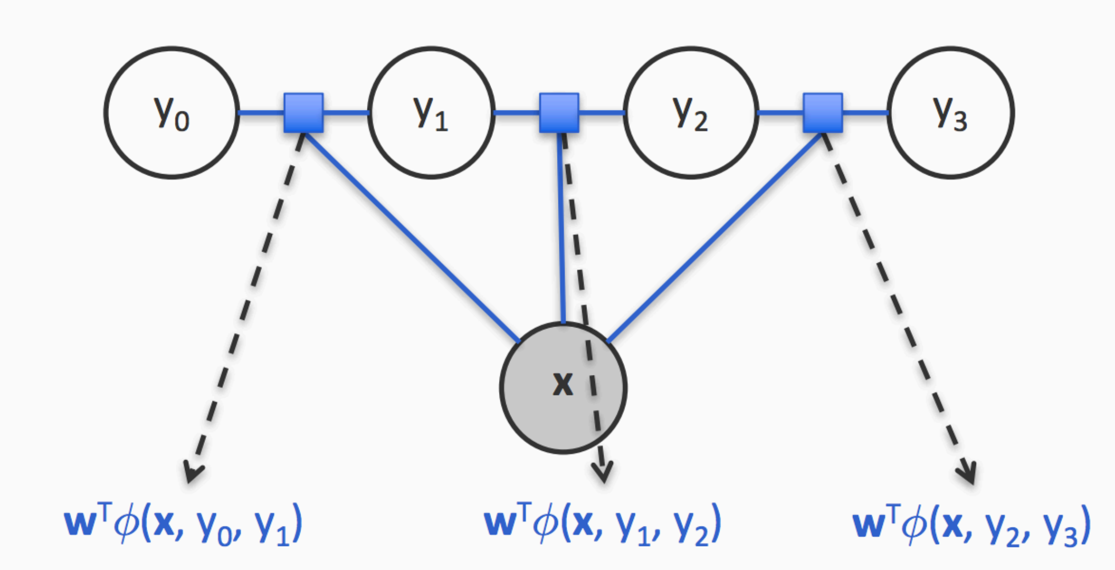
Each node is a random variable
We observe some nodes and need to assign the rest
Each factor is associated with a score

Conditional Random Field for sequences

$$P(\boldsymbol{y} \mid \boldsymbol{x}) = \frac{1}{Z}\prod_i \exp(\boldsymbol{w}^T\phi(\boldsymbol{x}, y_i, y_{i-1}))$$

Z: Normalizing constant, sum over all sequences

$$Z = \sum_{\hat{y}}\prod_i \exp(w^T\phi(\boldsymbol{x}, \hat{y}_i, \hat{y}_{i-1}))$$


y_0
y_1
y_2
y_3
x
$\mathbf{w}^T\phi(\mathbf{x}, y_0, y_1)$
$\mathbf{w}^T\phi(\mathbf{x}, y_1, y_2)$
$\mathbf{w}^T\phi(\mathbf{x}, y_2, y_3)$

CRF: A different view

- Input: **x**, Output: **y**, both sequences (for now)
- Define a feature vector for the **entire** input and output sequence: ϕ(**x**, **y**)
- Define a giant log-linear model, P(**y** | **x**) parameterized by **w**

$$P(\boldsymbol{y} \mid \boldsymbol{x}) = \frac{1}{Z}\prod_i \exp(\boldsymbol{w}^T\phi(\boldsymbol{x}, y_i, y_{i-1})) \propto \exp\left(w^T \sum_i \phi(\boldsymbol{x}, y_i, y_{i-1})\right)$$

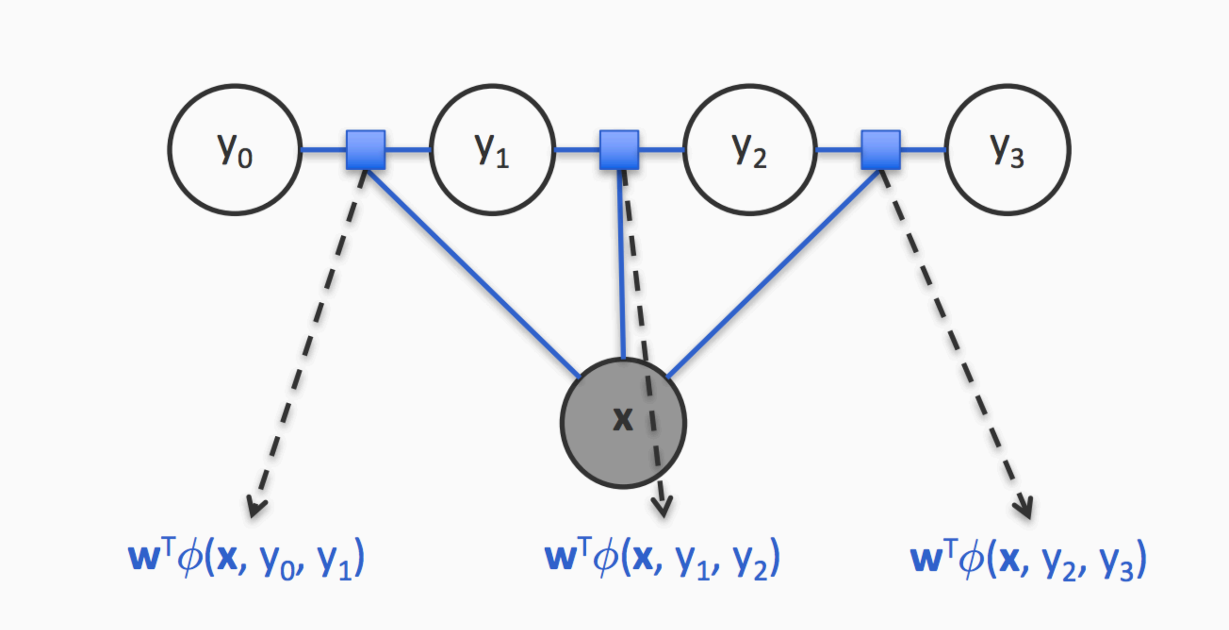
  - Just like any other log-linear model, except
    - Space of **y** is the set of all possible sequences of the correct length
    - Normalization constant sums over all sequences

In an MEMM, probabilities were locally normalized

Global features

The feature function decomposes over the sequence

$$\phi(\mathbf{x}, \mathbf{y}) = \sum_i \phi(\mathbf{x}, y_i, y_{i-1})$$


y_0
y_1
y_2
y_3
$\mathbf{x}$
$\mathbf{w}^T\phi(\mathbf{x}, y_0, y_1)$
$\mathbf{w}^T\phi(\mathbf{x}, y_1, y_2)$
$\mathbf{w}^T\phi(\mathbf{x}, y_2, y_3)$

Prediction

Goal: To predict most probable sequence **y** an input **x**

$$\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}} \exp(\mathbf{w}^T\phi(\mathbf{x}, \mathbf{y})) = \arg\max_{\mathbf{y}} \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y})$$

But the score decomposes as $\mathbf{w}^T\phi(\mathbf{x}, \mathbf{y}) = \sum_i \mathbf{w}^T\phi(\mathbf{x}, y_i, y_{i-1})$

Prediction via Viterbi (with sum instead of product)

$$\text{score}_0(s) = \mathbf{w}^T\phi(\mathbf{x}, y_0, \text{start})$$

$$\text{score}_i(s) = \max_{y_{i-1}} \left(\mathbf{w}^T\phi(\mathbf{x}, y_i, y_{i-1}) + \text{score}_{i-1}(y_{i-1})\right)$$

Training a chain CRF

- Input:
  - Dataset with labeled sequences, D = {<$\mathbf{x}_i$, $\mathbf{y}_i$>}
  - A definition of the feature function
- How do we train?
  - Maximize the (regularized) log-likelihood

$$\max_{\mathbf{w}} -\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_i \log P(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w})$$

Recall: Empirical loss minimization

Training with inference

- Many methods for training
  - Numerical optimization
  - Can use a gradient or hessian based method

$$\max_{\mathbf{w}} -\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_i \log P(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w})$$

- Simple gradient ascent

$$\mathbf{w} \leftarrow \mathbf{w} + \sum_i \left(\phi(\mathbf{x}_i, \mathbf{y}_i) - \sum_{\hat{\mathbf{y}}} P(\hat{\mathbf{y}}|\mathbf{x}_i, \mathbf{w}) \phi(\mathbf{x}_i, \hat{\mathbf{y}}) \right)$$

Training with inference

- Many methods for training
  - Numerical optimization
  - Can use a gradient or hessian based method

$$\max_{\mathbf{w}} -\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_i \log P(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w})$$

- Simple gradient ascent

$$\mathbf{w} \leftarrow \mathbf{w} + \sum_i \left(\phi(\mathbf{x}_i, \mathbf{y}_i) - \sum_{\hat{\mathbf{y}}} P(\hat{\mathbf{y}}|\mathbf{x}_i, \mathbf{w})\phi(\mathbf{x}_i, \hat{\mathbf{y}}) \right)$$

- Training involves inference!
  - A different kind than what we have seen so far
  - Summing over all sequences is just like Viterbi
    - With summation instead of maximization

CRF summary

- An undirected graphical model
  - Decompose the score over the structure into a collection of factors
  - Each factor assigns a score to assignment of the random variables it is connected to
- Training and prediction
  - Final prediction via argmax $w^T \phi(\mathbf{x}, \mathbf{y})$
  - Train by maximum (regularized) likelihood (also need inference)
- Relation to other models
  - Effectively a linear classifier
  - A generalization of logistic regression to structures
  - An instance of Markov Random Field, with some random variables observed (We will see this soon)