

# Statistical Learning for Text Data Analytics

## Sequence Labeling and Structured Output Learning: HMM and Conditional Models and Local Classifiers

Yangqiu Song

Hong Kong University of Science and Technology

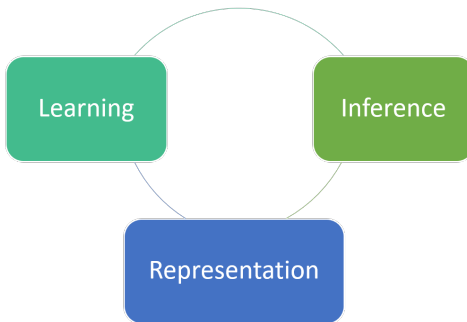
*yqsong@cse.ust.hk*

Spring 2018

\*Contents are based on materials created by Vivek Srikumar, Dan Roth, Xiaojin (Jerry) Zhu, Chris Manning

- Vivek Srikumar. CS 6355 Structured Prediction. <https://svivek.com/teaching/structured-prediction/spring2018/>
- Dan Roth. CS546: Machine Learning and Natural Language . <http://12r.cs.uiuc.edu/~danr/Teaching/CS546-16/>
- Xiaojin (Jerry) Zhu. CS 769: Advanced Natural Language Processing. <http://pages.cs.wisc.edu/~jerryzhu/cs769.html>
- Chris Manning. CS 224N/Ling 237. Natural Language Processing. <https://web.stanford.edu/class/cs224n/>

# Course Topics



- **Representation**: language models, word embeddings, topic models
- **Learning**: supervised learning, semi-supervised learning, **sequence models**, deep learning, optimization techniques
- **Inference**: constraint modeling, joint inference, search algorithms

NLP applications: tasks introduced in Lecture 1

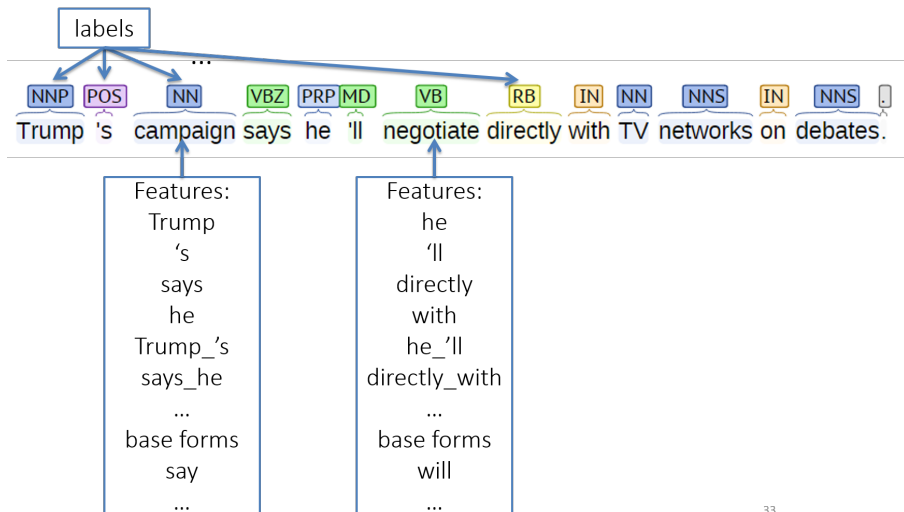
## 1 Hidden Markov Models

- Representation
- Learning
- Inference

## 2 Conditional Models and Local Classifiers

- Conditional Models for Predicting Sequences
- Log-linear Models for Multiclass Classification
- Maximum Entropy Markov Models
  - The Label Bias Problem

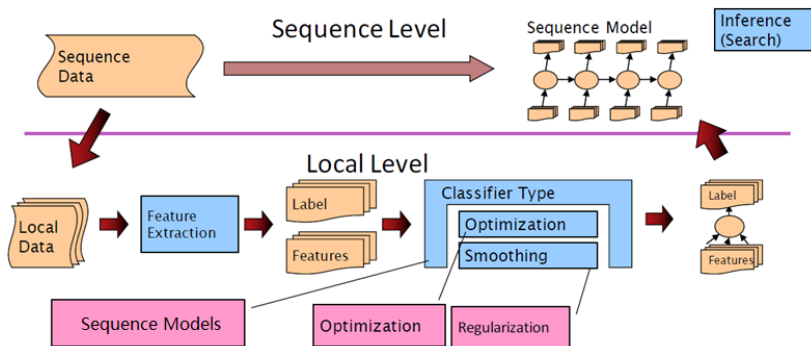
# Classification Problem



33

# The General Framework of Training and Testing

- Analogous to classification



# Label and Feature Dependencies

- Current label may depend on the previous one
  - Fed in “The Fed” is a Noun because it follows a Determiner
  - Fed in “I fed the..” is a Verb because it follows a Pronoun
- Sometimes more difficult: “I/PN can/MD can/VB a/DT can/NN.”
- Two kinds of information incorporated in learning:
  - Some tag sequences are more likely than others. For instance, DT JJ NN is quite common, while DT JJ VBP is unlikely. (“a new book”)
  - A word may have multiple possible POS, but some are more likely than others, e.g., “flour” is more often a noun than a verb
- The question is:
  - Given a word sequence

$$\mathbf{x}_{1:N} \doteq \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N,$$

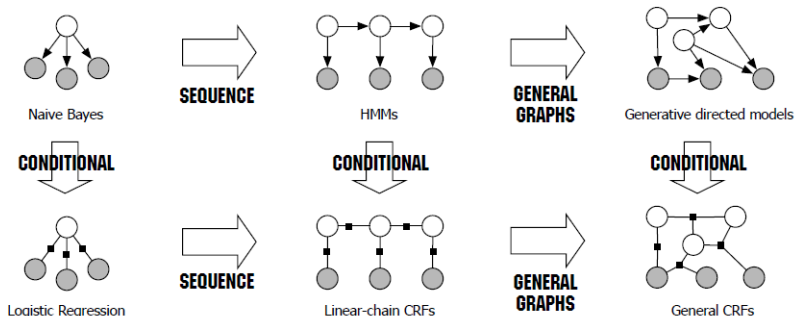
how do we compute the most likely POS sequence

$$y_{1:N} \doteq y_1, y_2, \dots, y_N$$

- One method is to use a Hidden Markov Model

# Classifiers Feasible for Sequence Labeling

- Generative
  - Naive Bayes
  - Hidden Markov model (HMM)
- Discriminative models
  - Maximum entropy, logistic regression
  - Maximum Entropy Markov Model (MEMM)
  - Conditional random field (CRF)



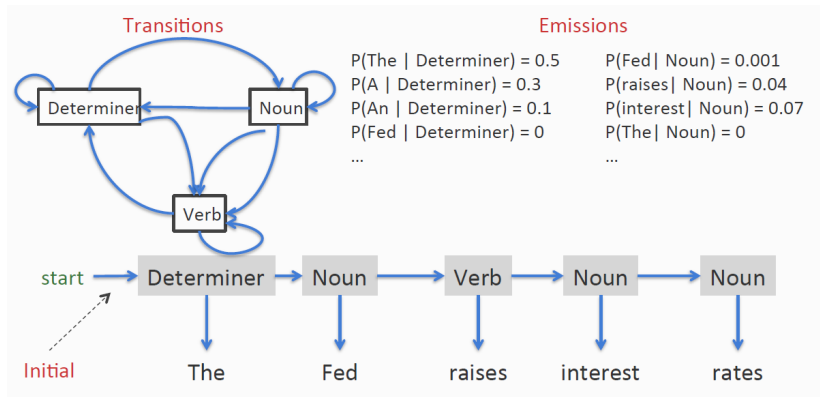


# Hidden Markov Model

- 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

# A Toy Part-of-Speech Example

- Sentence “The Fed raises interest rates”



# Most Likely State Sequence

- Input:
  - A hidden Markov model  $\Theta = \{\pi, \mathbf{A}, \Phi\}$
  - An observation sequence  $\mathbf{x}_{1:N}$
- Output: A state sequence  $y_{1:N}$  that corresponds to

$$\arg \max_{y_{1:N}} P(y_{1:N} | \mathbf{x}_{1:N}, \Theta)$$

- This is maximum a posteriori inference (MAP inference)
- Computationally a combinatorial optimization problem

# MAP Inference

- We want  $\arg \max_{y_{1:N}} P(y_{1:N} | \mathbf{x}_{1:N}, \Theta)$
- Note that  $P(y_{1:N} | \mathbf{x}_{1:N}, \Theta) \propto P(y_{1:N}, \mathbf{x}_{1:N} | \Theta)$ 
  - And we don't care about  $P(\mathbf{x}_{1:N})$  since we are maximizing over  $y_{1:N}$
- So

$$\arg \max_{y_{1:N}} P(y_{1:N} | \mathbf{x}_{1:N}, \Theta) = \arg \max_{y_{1:N}} P(y_{1:N}, \mathbf{x}_{1:N} | \Theta)$$

- We have defined

$$P(\mathbf{x}_{1:N}, y_{1:N} | \Theta) = P(y_1 | \pi) P(\mathbf{x}_1 | y_1, \Phi) \prod_{n=2}^N P(y_n | y_{n-1}, \mathbf{A}) P(\mathbf{x}_n | y_n, \Phi)$$

- We omit the parameters for the ease of derivation

$$P(\mathbf{x}_{1:N}, y_{1:N}) = P(y_1) P(\mathbf{x}_1 | y_1) \prod_{n=2}^N P(y_n | y_{n-1}) P(\mathbf{x}_n | y_n)$$

# How Many Possible Sequences?

|                                    |              |              |              |              |
|------------------------------------|--------------|--------------|--------------|--------------|
| The                                | Fed          | raises       | interest     | rates        |
| List of allowed tags for each word |              |              |              |              |
| Determiner                         | Verb<br>Noun | Verb<br>Noun | Verb<br>Noun | Verb<br>Noun |
| 1                                  | 2            | 2            | 2            | 2            |

- In this simple case, we have 16 candidate sequences

$$(1 \times 2 \times 2 \times 2 \times 2)$$

# How Many Possible Sequences?

- Output: one state per observation  $y_n = s_k$

| Observations                                | $x_1$ | $x_2$ | ... | $x_n$ |
|---|-------|-------|-----|-------|
| List of allowed states for each observation |       |       |     |       |
|   | $s_1$ | $s_1$ | ... | $s_1$ |
|   | $s_2$ | $s_2$ |     | $s_2$ |
|   | $s_3$ | $s_2$ |     | $s_3$ |
|   | .     | .     |     | .     |
|   | .     | .     |     | .     |
|   | $s_K$ | $s_K$ |     | $s_K$ |

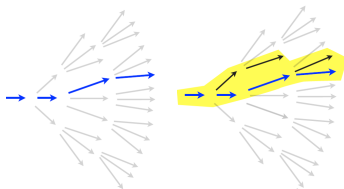
- We have  $K^n$  possible sequences to consider in  
 $\arg \max_{y_{1:N}} P(y_{1:N}, \mathbf{x}_{1:N} | \Theta)$

- Try out every sequences
  - Score the sequence  $y_{1:N}$  using  $P(y_{1:N}, \mathbf{x}_{1:N} | \Theta)$
  - Return the highest scoring one
  - Correct but slow  $O(K^N)$
- Greedy search
  - Construct the output left to right
  - For each  $n$ , elect the best  $y_n$  using  $y_{n-1}$  and  $\mathbf{x}_n$
  - Incorrect but fast,  $O(NK)$

# Beam Search

- Beam inference

- At each position keep the top k complete sequences
- Extend each sequence in each local way
- The extensions compete for the k slots at the next position



(a) Greedy      (b) Beam Search

- Advantages

- Fast; beam sizes of 3-5 are almost as good as exact inference in many cases
- Easy to implement (no dynamic programming required)

- Disadvantage

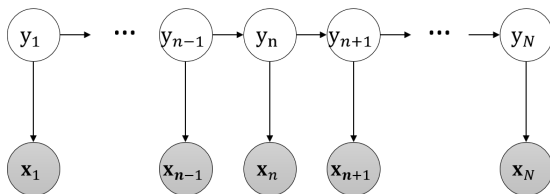
- Inexact: the globally best sequence can fall off the beam



# Optimal Solution: General Idea

- Dynamic programming
  - The best solution for the full problem relies on the best solution to the sub-problem
  - Memorize partial computation
- Examples
  - Viterbi algorithm
  - Dijkstra's shortest path algorithm
  - MDP value iteration
  - ...

# Deriving the Recursion



$$\max_{y_{1:N}} P(\mathbf{x}_{1:N}, y_{1:N}) = \max_{y_{1:N}} P(y_1)P(\mathbf{x}_1|y_1) \prod_{n=1}^N P(y_n|y_{n-1})P(\mathbf{x}_n|y_n)$$

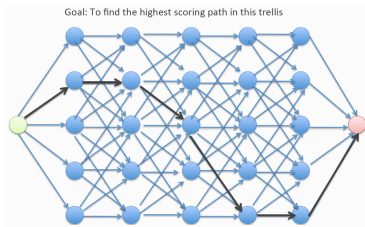
We reorganize it as

$$\max_{y_{1:N}} P(\mathbf{x}_N|y_N)P(y_N|y_{N-1}) \cdot \dots \cdot P(\mathbf{x}_2|y_2)P(y_2|y_1) \cdot P(\mathbf{x}_1|y_1)P(y_1)$$

# Deriving the Recursion

$$\begin{aligned} & \max_{y_{1:N}} P(\mathbf{x}_N|y_N)P(y_N|y_{N-1}) \cdot \dots \cdot P(\mathbf{x}_2|y_2)P(y_2|y_1) \cdot P(\mathbf{x}_1|y_1)P(y_1) \\ = & \max_{y_{2:N}} P(\mathbf{x}_N|y_N)P(y_N|y_{N-1}) \cdot \dots \cdot \max_{y_1} P(\mathbf{x}_2|y_2)P(y_2|y_1) \cdot P(\mathbf{x}_1|y_1)P(y_1) \\ = & \max_{y_{2:N}} P(\mathbf{x}_N|y_N)P(y_N|y_{N-1}) \cdot \dots \cdot \max_{y_1} P(\mathbf{x}_2|y_2)P(y_2|y_1) \cdot \text{score}_1(y_1) \\ = & \max_{y_{3:N}} P(\mathbf{x}_N|y_N)P(y_N|y_{N-1}) \cdot \dots \cdot \max_{y_2} P(\mathbf{x}_3|y_3)P(y_3|y_2) \\ & \cdot \max_{y_1} P(\mathbf{x}_2|y_2)P(y_2|y_1) \cdot \text{score}_1(y_1) \\ = & \max_{y_{3:N}} P(\mathbf{x}_N|y_N)P(y_N|y_{N-1}) \cdot \dots \cdot \max_{y_2} P(\mathbf{x}_3|y_3)P(y_3|y_2) \cdot \text{score}_2(y_2) \\ = & \dots \\ = & \max_{y_N} \text{score}_N(y_N) \end{aligned}$$

where we have  $\text{score}_n(y_n) = \max_{y_{n-1}} P(y_n|y_{n-1})P(\mathbf{x}_n|y_n)\text{score}_{n-1}(y_{n-1})$



# 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
  - $O(NK^2)$

# Summary of Viterbi Inference

- Viterbi inference
  - Dynamic programming or memoization
  - Requires small window of state influence (e.g., past two states are relevant)
- Advantage
  - Exact: the global best sequence is returned
- Disadvantage
  - Harder to implement long-distance state-state interactions (but beam inference tends not to allow long-distance resurrection of sequences anyway)

- Predicting sequences
  - As many output states as observations
- Markov assumption helps decompose the score
- Several algorithmic questions
  - Most likely state
  - Learning parameters: supervised, unsupervised (posterior, sum-product algorithm)
  - Probability of an observation sequence: sum over all assignments of states; replace max with sum in Viterbi
  - Inference: Viterbi (or max-product algorithm)

- Sequence Models
- Hidden Markov Models
  - Representation
  - Learning
  - Inference
- Conditional Models and Local Classifiers
- Global Models
  - Conditional Random Fields
  - Structured Perceptron for sequences
  - Structural SVM

## 1 Hidden Markov Models

- Representation
- Learning
- Inference

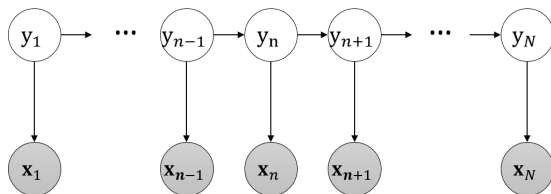
## 2 Conditional Models and Local Classifiers

- Conditional Models for Predicting Sequences
- Log-linear Models for Multiclass Classification
- Maximum Entropy Markov Models
  - The Label Bias Problem

\*Contents are based on materials created by Vivek Srikumar, Dan Roth



# HMM Recap



- The joint probability is

$$P(\mathbf{x}_{1:N}, y_{1:N} | \Theta) = P(y_1 | \pi) P(\mathbf{x}_1 | y_1, \Phi) \prod_{n=2}^N P(y_n | y_{n-1}, \mathbf{A}) P(\mathbf{x}_n | y_n, \Phi)$$

- Training via maximum likelihood (supervised learning)

$$\Theta = \{\pi, \mathbf{A}, \Phi\} = \arg \max_{\Theta} \prod_i P(\mathbf{x}_{1:N}^{(i)}, y_{1:N}^{(i)} | \Theta) = P(\mathbf{x}_{1:N}^{(i)}, y_{1:N}^{(i)} | \Theta)$$

where  $\mathbf{x}_{1:N}^{(i)}, y_{1:N}^{(i)}$  is the  $i$ -th example (sequence)

- In the training phase, we are optimizing joint likelihood of the input and the output for training

$$P(\mathbf{x}_{1:N}, y_{1:N} | \Theta)$$

- In the test phase, we are trying to find a state sequence  $y_{1:N}$  that corresponds to

$$\arg \max_{y_{1:N}} P(y_{1:N} | \mathbf{x}_{1:N}, \Theta)$$

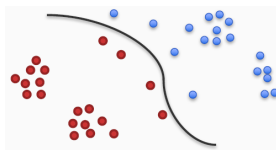
- Question:
  - Why not directly optimize this conditional likelihood instead in training phase?

# Modeling Next-state Directly

- Instead of modeling the joint distribution  $P(\mathbf{x}_{1:N}, y_{1:N})$ , we only focus on  $P(y_{1:N}|\mathbf{x}_{1:N})$ 
  - 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, ...

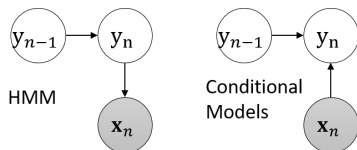
# Generative vs Discriminative Models

- Generative models
  - Learn  $P(\mathbf{x}, y)$
  - Characterize how the data is generated (both inputs and outputs)
  - E.g., Naive Bayes, Hidden Markov Model
- Discriminative models
  - Learn  $P(y|\mathbf{x})$
  - Directly characterize the decision boundary only
  - E.g., Logistic Regression, Conditional models (several names)
- A generative model tries to characterize the distribution of the inputs, while a discriminative model doesn't care



# Another Independence Assumption

$$P(y_n | y_{n-1}, y_{n-2}, \dots, \mathbf{x}_n, \mathbf{x}_{n-1}, \mathbf{x}_{n-2}, \dots) = P(y_n | y_{n-1}, \mathbf{x}_n)$$



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

$$P(y_{1:N} | \mathbf{x}_{1:N}) = \prod_n P(y_n | y_{n-1}, \mathbf{x}_n)$$

- Compared to HMM  $P(y_{1:N} | \mathbf{x}_{1:N}, \Theta) \propto P(y_{1:N}, \mathbf{x}_{1:N} | \Theta)$ 
  - where we don't care about  $P(\mathbf{x}_{1:N})$  since we are maximizing over  $y_{1:N}$
  - We don't even need to model  $P(\mathbf{x}_n | y_n)$  here
  - Very similar to logistic regression vs. naive Bayes

# Modeling $P(y_n|y_{n-1}, \mathbf{x}_n)$

- Different approaches possible
  - Train a maximum entropy classifier
  - Or, 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  $\mathbf{x}_n$ 's
    - E.g., Neighboring words influence this words POS tag

## 1 Hidden Markov Models

- Representation
- Learning
- Inference

## 2 Conditional Models and Local Classifiers

- Conditional Models for Predicting Sequences
- **Log-linear Models for Multiclass Classification**
- Maximum Entropy Markov Models
  - The Label Bias Problem

# Detour: Log-linear Models for Multiclass

- Consider multiclass classification
  - Input:  $\mathbf{x} \in \mathbb{R}^d$
  - Output:  $y \in \{1, 2, \dots, K\}$
  - Feature representation:  $\phi(\mathbf{x}, y)$ 
    - We have seen this before
- Define probability of an input  $\mathbf{x}$  taking a label  $y$  as

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

- A generalization of logistic regression to multiclass

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



# Training a Log-linear Model

- Given a data set  $\{\mathbf{x}^{(i)}, y^{(i)}\}$  (to be consistent here we use superscript to denote instance ids)
  - Apply the maximum likelihood principle

$$\max_{\mathbf{w}} \prod_i P(y^{(i)} | \mathbf{x}^{(i)}, \mathbf{w})$$

where

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

- With a regularizer

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

- (Stochastic) gradient based methods to train  $\mathbf{w}$
- Log-linear = Maximum Entropy distribution with feature constraints

## 1 Hidden Markov Models

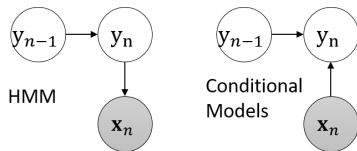
- Representation
- Learning
- Inference

## 2 Conditional Models and Local Classifiers

- Conditional Models for Predicting Sequences
- Log-linear Models for Multiclass Classification
- **Maximum Entropy Markov Models**
  - The Label Bias Problem

# The Next-state Model

$$P(y_n | y_{n-1}, y_{n-2}, \dots, \mathbf{x}_n, \mathbf{x}_{n-1}, \mathbf{x}_{n-2}, \dots) = P(y_n | y_{n-1}, \mathbf{x}_n)$$



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

$$P(y_{1:N} | \mathbf{x}_{1:N}) = \prod_n P(y_n | y_{n-1}, \mathbf{x}_n)$$

# Modeling $P(y_n|y_{n-1}, \mathbf{x}_n)$

- Different approaches possible
  - Train a maximum entropy classifier
  - Or, 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  $\mathbf{x}_n$ 's
    - E.g., Neighboring words influence this words POS tag

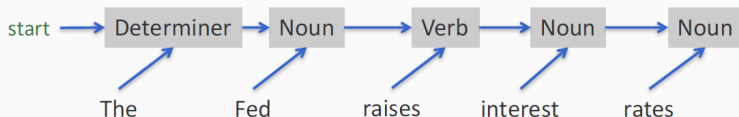
# Modeling $P(y_n|y_{n-1}, \mathbf{x}_n)$ $P(y_n|y_{n-1}, \mathbf{x}_{1:N})$

- Different approaches possible
  - Train a maximum entropy classifier
    - Basically, a multinomial logistic regression
  - Or, 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  $\mathbf{x}_n$ 's
    - E.g., Neighboring words influence this words POS tag

# Maximum Entropy Markov Model (MEMM)

Goal: Compute  $P(y_{1:N}|\mathbf{x}_{1:N}, \mathbf{w}) = \prod_n P(y_n|y_{n-1}, \mathbf{x}_{1:N})$  where

$$P(y_n|y_{n-1}, \mathbf{x}_{1:N}) \propto \exp(\mathbf{w}^\top \phi(\mathbf{x}, n, y_n, y_{n-1}))$$

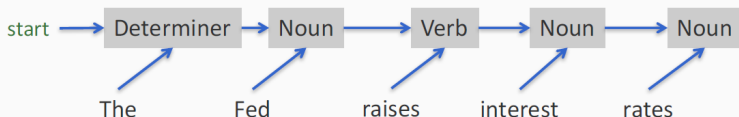


The prediction task: Using the entire input and the current label, predict the next label

# Maximum Entropy Markov Model (MEMM)

Goal: Compute  $P(y_{1:N}|\mathbf{x}_{1:N}, \mathbf{w}) = \prod_n P(y_n|y_{n-1}, \mathbf{x}_{1:N})$  where

$$P(y_n|y_{n-1}, \mathbf{x}_{1:N}) \propto \exp(\mathbf{w}^\top \phi(\mathbf{x}, n, y_n, y_{n-1}))$$



*word*

*Caps*

*-es*

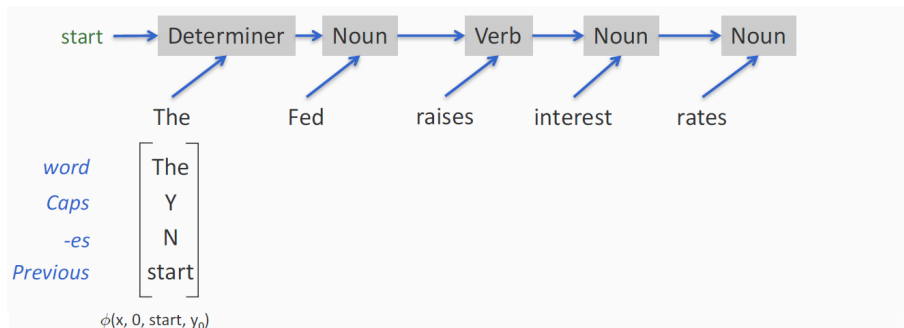
*Previous*

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

# Maximum Entropy Markov Model (MEMM)

Goal: Compute  $P(y_{1:N}|\mathbf{x}_{1:N}, \mathbf{w}) = \prod_n P(y_n|y_{n-1}, \mathbf{x}_{1:N})$  where

$$P(y_n|y_{n-1}, \mathbf{x}_{1:N}) \propto \exp(\mathbf{w}^\top \phi(\mathbf{x}, n, y_n, y_{n-1}))$$

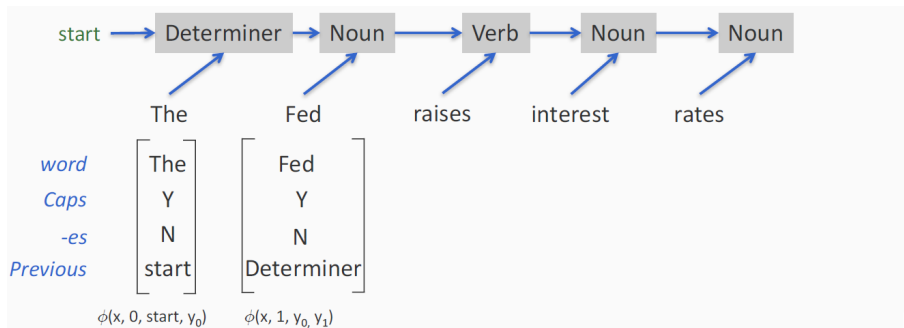




# Maximum Entropy Markov Model (MEMM)

Goal: Compute  $P(y_{1:N}|\mathbf{x}_{1:N}, \mathbf{w}) = \prod_n P(y_n|y_{n-1}, \mathbf{x}_{1:N})$  where

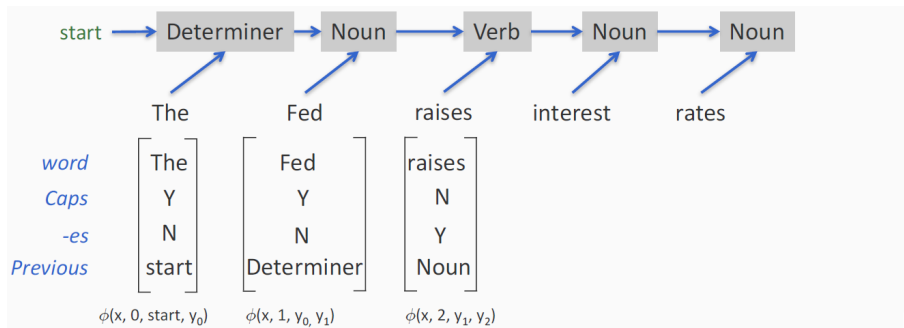
$$P(y_n|y_{n-1}, \mathbf{x}_{1:N}) \propto \exp(\mathbf{w}^\top \phi(\mathbf{x}, n, y_n, y_{n-1}))$$



# Maximum Entropy Markov Model (MEMM)

Goal: Compute  $P(y_{1:N}|\mathbf{x}_{1:N}, \mathbf{w}) = \prod_n P(y_n|y_{n-1}, \mathbf{x}_{1:N})$  where

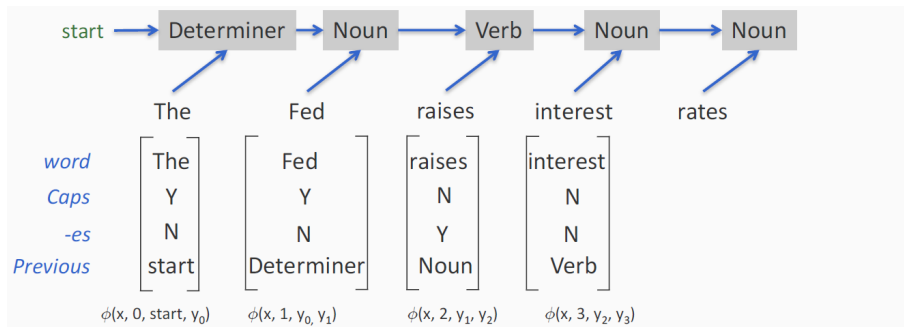
$$P(y_n|y_{n-1}, \mathbf{x}_{1:N}) \propto \exp(\mathbf{w}^\top \phi(\mathbf{x}, n, y_n, y_{n-1}))$$



# Maximum Entropy Markov Model (MEMM)

Goal: Compute  $P(y_{1:N}|\mathbf{x}_{1:N}, \mathbf{w}) = \prod_n P(y_n|y_{n-1}, \mathbf{x}_{1:N})$  where

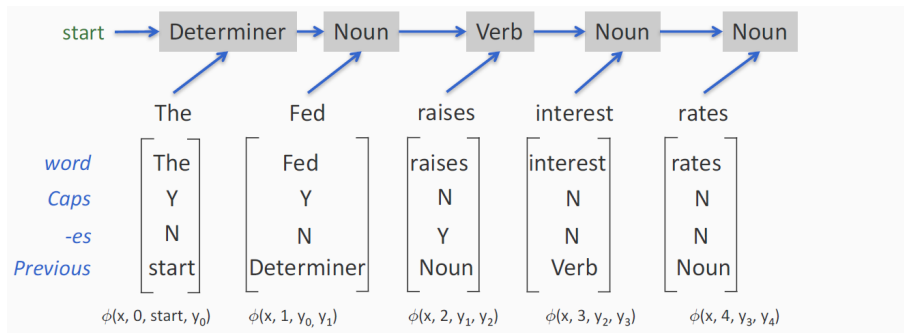
$$P(y_n|y_{n-1}, \mathbf{x}_{1:N}) \propto \exp(\mathbf{w}^\top \phi(\mathbf{x}, n, y_n, y_{n-1}))$$



# Maximum Entropy Markov Model (MEMM)

Goal: Compute  $P(y_{1:N}|\mathbf{x}_{1:N}, \mathbf{w}) = \prod_n P(y_n|y_{n-1}, \mathbf{x}_{1:N})$  where

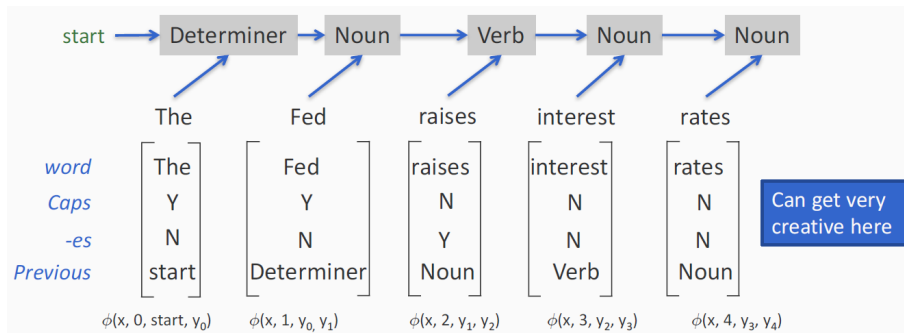
$$P(y_n|y_{n-1}, \mathbf{x}_{1:N}) \propto \exp(\mathbf{w}^\top \phi(\mathbf{x}, n, y_n, y_{n-1}))$$



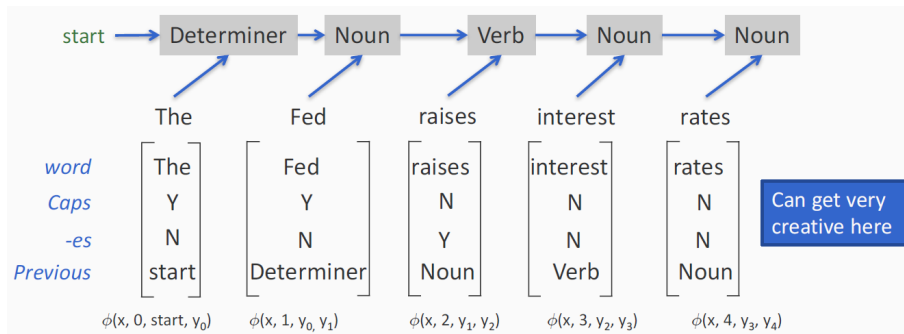
# Maximum Entropy Markov Model (MEMM)

Goal: Compute  $P(y_{1:N}|\mathbf{x}_{1:N}, \mathbf{w}) = \prod_n P(y_n|y_{n-1}, \mathbf{x}_{1:N})$  where

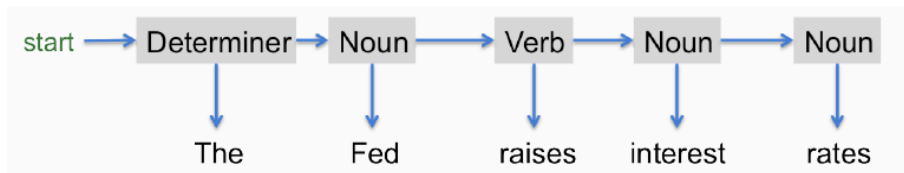
$$P(y_n|y_{n-1}, \mathbf{x}_{1:N}) \propto \exp(\mathbf{w}^\top \phi(\mathbf{x}, n, y_n, y_{n-1}))$$



# Compare MEMM and HMM

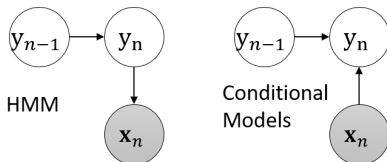


HMM: Only depends on the word and the previous tag



# 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



- In HMM, we use

$$\text{score}_n(y_n) = \max_{y_{n-1}} P(y_n|y_{n-1})P(\mathbf{x}_n|y_n)\text{score}_{n-1}(y_{n-1})$$

- In MEMM, we use

$$\text{score}_n(y_n) = \max_{y_{n-1}} P(y_n|y_{n-1}, \mathbf{x}, n)\text{score}_{n-1}(y_{n-1})$$

# 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  $y_n$  given  $y_{n-1}$  and  $\mathbf{x}$ 
  - Multiclass versions of perceptron, SVM
  - Just like MEMM, these allow arbitrary features to be defined
- Viterbi needs to be re-defined to work with sum of scores rather than the product of probabilities



# Comparison to HMM

## What we gain

- Rich feature representation for inputs
  - Helps generalize better by thinking about properties of the input tokens rather than the entire tokens
  - E.g., If a word ends with es, it might be a present tense verb (such as raises). Could be a feature; HMM cannot capture this
- Discriminative predictor
  - Model  $P(y|\mathbf{x})$  rather than  $P(y, \mathbf{x})$
  - Joint vs conditional

## 1 Hidden Markov Models

- Representation
- Learning
- Inference

## 2 Conditional Models and Local Classifiers

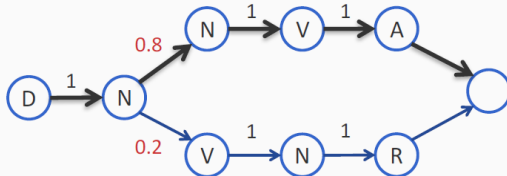
- Conditional Models for Predicting Sequences
- Log-linear Models for Multiclass Classification
- Maximum Entropy Markov Models
  - The Label Bias Problem

# But ... Local Classifiers → Label Bias Problem

- Recall: the independence assumption (“Next-state” classifiers are locally normalized)

$$P(y_n | y_{n-1}, y_{n-2}, \dots, \mathbf{x}_n, \mathbf{x}_n | y_{n-1}, \mathbf{x}_{n-2}, \dots) = P(y_n | y_{n-1}, \mathbf{x}_n)$$

*The robot wheels are round*



Suppose these are the only state transitions allowed

Option 1:  $P(D \mid \text{The}) \cdot$

$P(N \mid D, \text{robot}) \cdot$   
 $P(N \mid N, \text{wheels}) \cdot$   
 $P(V \mid N, \text{are}) \cdot$   
 $P(A \mid V, \text{round})$

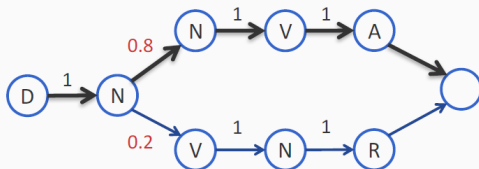
Option 2:  $P(D \mid \text{The}) \cdot$

$P(N \mid D, \text{robot}) \cdot$   
 $P(V \mid N, \text{wheels}) \cdot$   
 $P(N \mid V, \text{are}) \cdot$   
 $P(R \mid N, \text{round})$

# But ... Local Classifiers → Label Bias Problem

- The robot wheels Fred round

*The robot wheels are round*



Suppose these are the only state transitions allowed

Option 1:  $P(D \mid \text{The}) \cdot$

$P(N \mid D, \text{robot}) \cdot$

$P(N \mid N, \text{wheels}) \cdot$

~~$P(V \mid N, \text{are}) \cdot$~~   $P(V \mid N, \text{Fred}) \cdot$

$P(A \mid V, \text{round})$

Option 2:  $P(D \mid \text{The}) \cdot$

$P(N \mid D, \text{robot}) \cdot$

$P(V \mid N, \text{wheels}) \cdot$

~~$P(N \mid V, \text{are}) \cdot$~~   $P(N \mid V, \text{Fred}) \cdot$

$P(R \mid N, \text{round})$

- 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

- States with a single outgoing transition effectively ignore their input
  - States with lower-entropy next states are less influenced by observations
- Why?
  - Because 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
- Surprisingly doesn't affect some tasks
  - E.g., POS tagging

# Summary: Local Models for Sequences

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

- McCallum, A., Freitag, D., and Pereira, F. C. N. (2000). Maximum entropy markov models for information extraction and segmentation. In *ICML*, pages 591–598.
- Punyakanok, V. and Roth, D. (2001). The use of classifiers in sequential inference. *CoRR*, cs.LG/0111003.