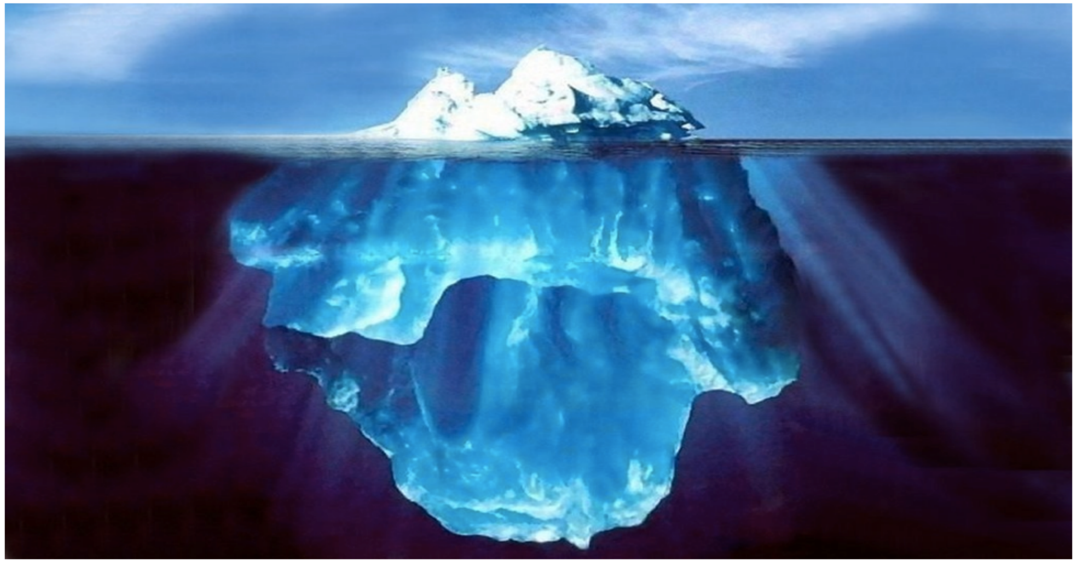
ML4NLP *Local* and *Global* Training for Structure Prediction



Dan Goldwasser

Purdue University

dgoldwas@purdue.edu

CS 590NLP

Structure Prediction

Language is highly structured.

We read words in the context of other words.

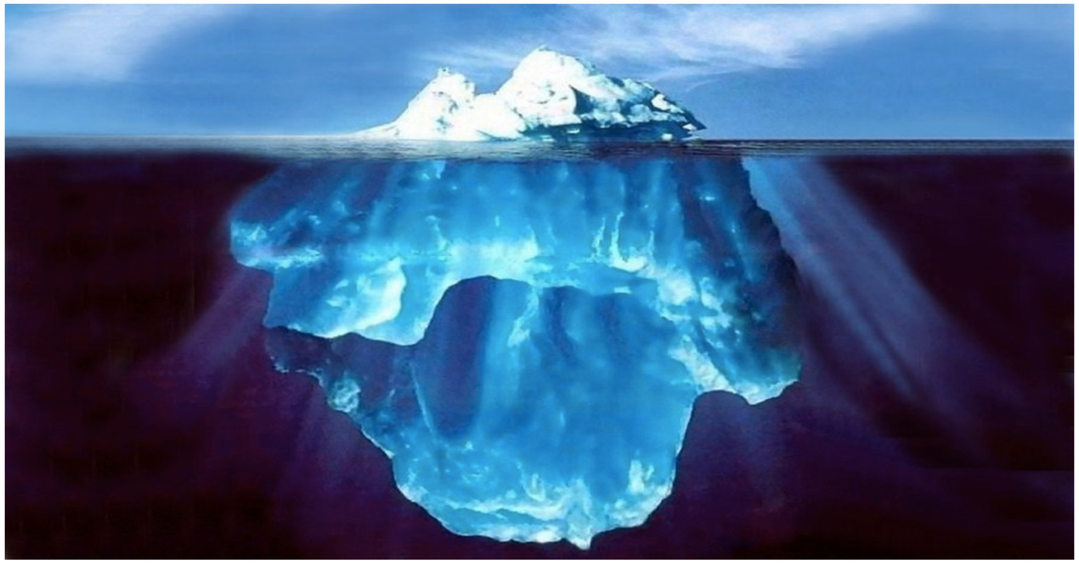
We make long range inferences, in order to decode language meaning.
e.g., *"<u>John</u> at lunch, <u>he</u> later went to sleep"*

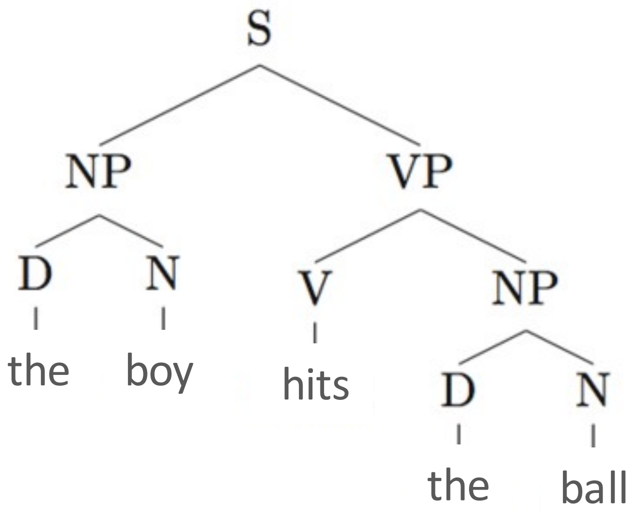
How can we account for structural dependencies when learning and making predictions?

NL Structures: *the 10,000 feet View*

- ***Tip of the iceberg***

Natural language, expressed in words, is just a surface representation underlying a complex structure





*A core concept is **Inference** : generalizing the notion of **classification***

Structure Prediction

Let’s look at a couple of examples…

- Part of speech Tagging
- Named Entity Recognition
- Parsing
- Information extraction
- Co-reference Resolution

How would you categorize the structural dependencies for each task?

Simple example: **Sequence labeling**

- Input: A sequence of *tokens* (e.g., *words*)
- Output: A sequence of labels of same length as input

Notable example: *Part- of – speech* (PoS) tagging:

-Given a sentence, find the PoS tags of all the words

	The	*Fed*	*raises*	*interest*	*rates*
	Determiner	Noun	Verb	Noun	Noun
Other possible output tags for words:		Verb		Verb	Verb

Decoding (prediction)

Given an observation sequence and an HMM, we need to find the **optimal state sequence**:

$$\arg\max_{y} p(x_1, .., x_n | y_1, ..., y_n) p(y_1 ..., y_n)$$

- How can we find it?
  - ***Combinatorial optimization problem***

Basic idea: $\arg\max_{y} \prod_{i=1}^{n} p(x_i|y_i) \prod_{i=1}^{n} p(y_i|y_{i-1})$

Independence assumptions lead to an algorithmic solution!

Viterbi Algorithm

Definitions:

n : length of input, S_k : possible symbols at position *k*

Truncated version of the probability *(defined over k long sequences, k<n)*

$$r(y_1, .., y_k) = \prod_{i=1}^{k} p(y_i|y_{i-1}) \prod_{i=1}^{k} p(x_i|y_i)$$

DP table:

$$\pi(k, v) = max_{(y_1,...,y_k;y_k=v)} r(y_1, ..., y_k)$$
$$\text{max probability tag sequence of size k ending with v}$$

Recursive definition of DP table:

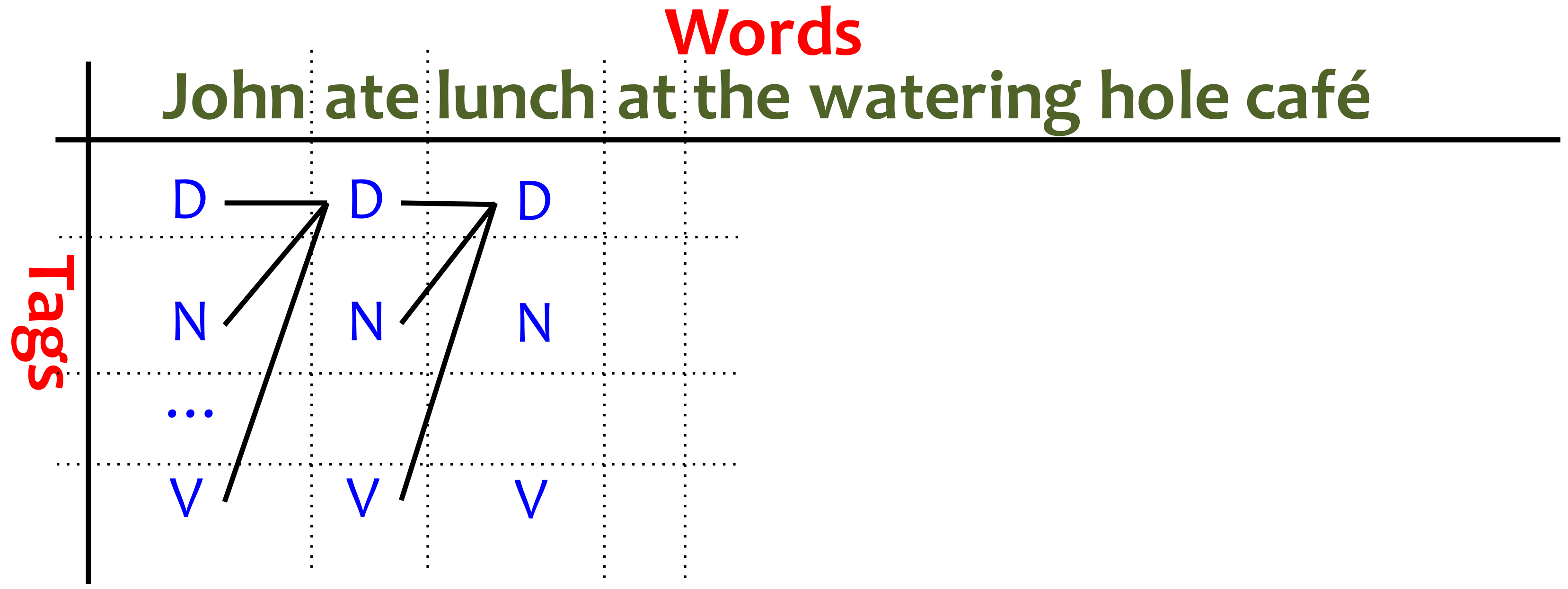
$$\pi(k, v) = \max_{u \in S_k - 1} (\pi(k-1, u) \times p(v|u) \times p(x_k|v))$$

Viterbi: DP Table

$$\pi(k,v) = max_{(y_1,\ldots,y_k;y_k=v)} r(y_1,\ldots,y_k)$$

max probability tag sequence of size k ending with v

$$\pi(k,v) = \max_{u \in S_k - 1} (\pi(k-1,u) \times p(v|u) \times p(x_k|v))$$



$$\pi(3,D) = max_{(y_1,\ldots,y_3;y_3=D)} r(y_1,\ldots,y_3)$$

The Viterbi Algorithm

```
$Input$: a sequence $x_1, .., x_n$ ,
        parameters: $p(s|u),\ p(x|s)\ \forall s, u \in S$

$Initialization$: $\pi(0, \epsilon) = 1$
  (Note: $\epsilon$ is just a start symbol)

For $k = 1..n$
 For $v \in S_k$
   $\pi(k, v) = \max_{u \in S_{k-1}} (\pi(k-1, u) \times p(v|u) \times p(x_k|v))$

Return $max_{u \in S_n} (\pi(n, u) \times p(\sigma|u))$
  (Note: $\sigma$ is just an end symbol)
```

Note:
We augment the set of tags with *start* symbol and compute parameters for these symbols

What is the run time complexity of Viterbi?

What does this algorithm return?

We are interested in the optimal sequence!

Solution: small modification to the algorithm, maintain a list of backpointers

In practice: smoothing is required!

Parameter Estimation

$$p(y_i|y_{i-1})\; p(x_i|y_i)$$

Two terms:

$$p(y_i|y_{i-1})$$

(transitions probabilities)

$$p(NN|DET) = \frac{count(NN, DET)}{count(DET)}$$

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

$$p(x_i|y_i)$$

(emission probabilities)

$$p(\text{“}watering\text{”}|NN) = \frac{count(\text{“}watering\text{”}, NN)}{count(NN)}$$

$$B_{s,x} = \frac{\text{count}\left(\begin{array}{c} s \\ \downarrow \\ x \end{array}\right)}{\text{count}\,(s)}$$

Initial state probability

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

Generative vs. Discriminative

Hmm: *Model for the joint probability of (x,y)*

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

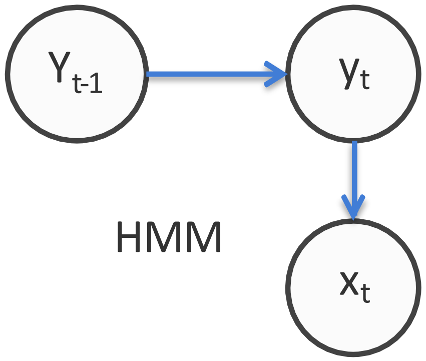
At prediction time we care about the probability of **output given the input**

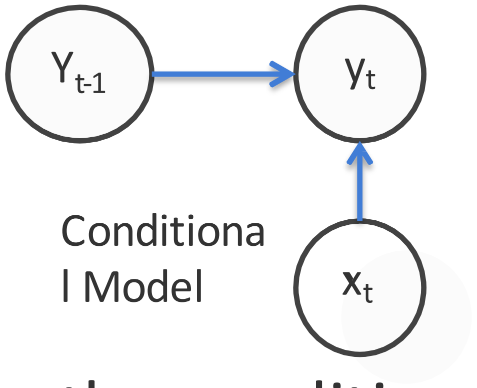
Why not directly optimize this conditional likelihood instead?

- Instead of modeling the joint distribution P(**x**, **y**) only focus on P(**y**|**x**)
  - Where have we seen it before?
    - How can we extend this model to sequences?
  - Maximum Entropy Markov Model [McCallum, et al 2000]

Conditional Models

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





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 learn this function

Learning Conditional Models

- **Advantages:**
  - Use rich features that depend on input and previous state
  - We can extract rich features from the entire input sequence
- **Learning algorithms:**
  - *Probabilistic*: Logistic regressions (=Max Entropy)
  - We can also use any multiclass classifier
    - Perceptron, SVM,..

Log-Linear Models for Multiclass Classification

Consider multiclass classification

- Input: x, output: y (multiclass 1,..,k)
- Feature representation: $\Phi(x,y)$
  - *Joint feature function (input + output)*

- Conditional probability:

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

- ***A generalization of logistic regression to multiclass***

Training is straightforward

Training: maximum likelihood

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

For logistic Regression:

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

Regularized version:

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

Gradient Based optimization

- **Gradient based methods**
  - using gradient of $L(\mathbf{w}) = \sum_i \log P(\mathbf{y}_i | \mathbf{x}_i, \mathbf{w})$
- Simple approach

```
1.    Initialize w =
ti 2.         For t =
1, 21...  Update w = w + $a_t$ r $\nabla$L(w)
3.    Return w
```

A vector, whose j^th element is the derivative of L with $\mathbf{w}_j$.

$$\frac{\partial}{\partial \mathbf{w}_j} L(\mathbf{w}) = \sum_i \left(\phi_j(\mathbf{x}_i, \mathbf{y}_i) - \sum_y P(\mathbf{y} | \mathbf{x}_i, \mathbf{w}) \phi_j(\mathbf{x}_i, \mathbf{y}) \right)$$

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 (log-linear)*** classifier

  2. ***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 x_i's
    - Eg. Neighboring words influence this words POS tag

MEMM: Max Entropy Markov Model

$$\begin{tikzcd}
\text{start} \arrow[r] & \text{Determiner} \arrow[r] & \text{Noun} \arrow[r] & \text{Verb} \arrow[r] & \text{Noun} \arrow[r] & \text{Noun} \\
& \textbf{The} \arrow[u] & \textbf{Fed} \arrow[u] & \textbf{raises} \arrow[u] & \textbf{interest} \arrow[u] & \textbf{rates} \arrow[u]
\end{tikzcd}$$

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

Using MEMM

- **Training**
  - *Train next--state predictor **locally** as maximum likelihood*
    - **Similar to any Log--linear model**

In general, any algorithm can be used to score y_i given y_{i-1} and $\mathbf{x}$

- Pick your favorite multiclass classifier

- **Prediction/Decoding**
  - *Modify the Viterbi algorithm for the new independence assumptions*

Question: ***How would you modify Viterbi when using non--probabilistic classifiers****?*

Example based on [Wallach 2002]

Label bias

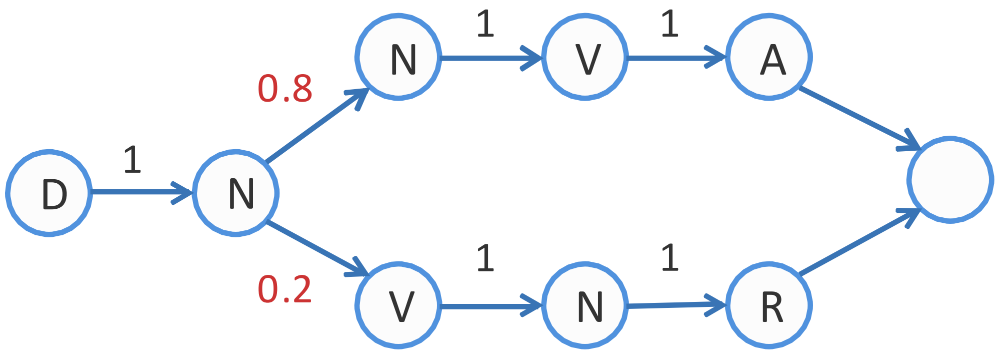
MEMM independence assumption

Learning local "next–state" classifiers

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

Example:

The robot wheels are round

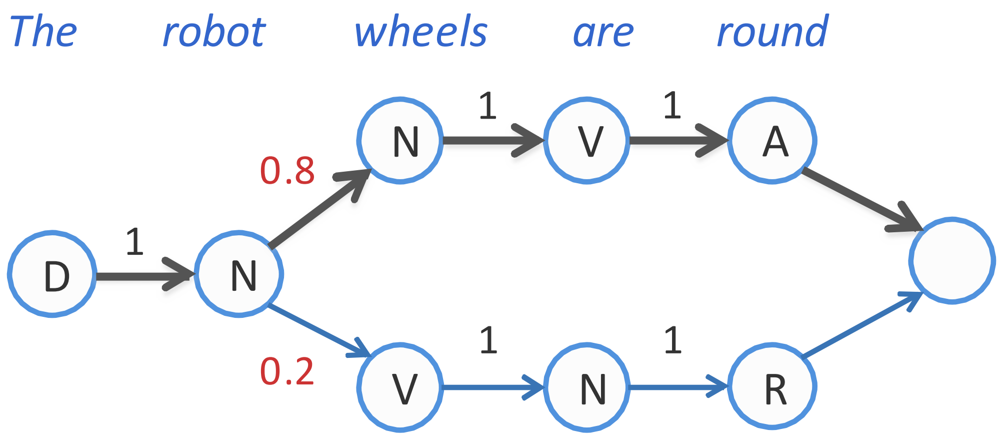


Suppose these are the only state transition allowed

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)

Label bias



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

-- regrdless of the change in inputs!

Label Bias

- States with a single outgoing transition effectively ignore their input
  - States with fewer transitions will dominate the result

- Why?
  - **Each next-state classifier is normalized locally**
  - If a state has fewer next states, each of those will get a higher probability mass
    - …and hence preferred

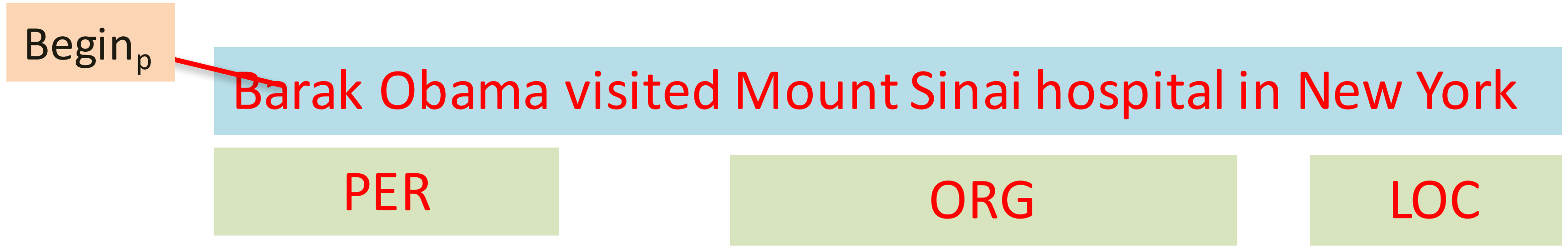
- *Side note: Surprisingly doesn't affect some tasks*
  - Eg: POS tagging

Can you define NER as a MEMM?

Named entity recognition (NER)

Identify mentions of named entity in text

People (PER), places (LOC) and organizations (ORG)



Conceptually two considerations:

- Entity **boundary (Begin, Inside ,Outside)**
- Entity **type (PER,ORG,LOC)**

Our Labels:

Begin_p,Begin_L,Begin_o
Inside_p,Inside_L,Inside_o
Outside

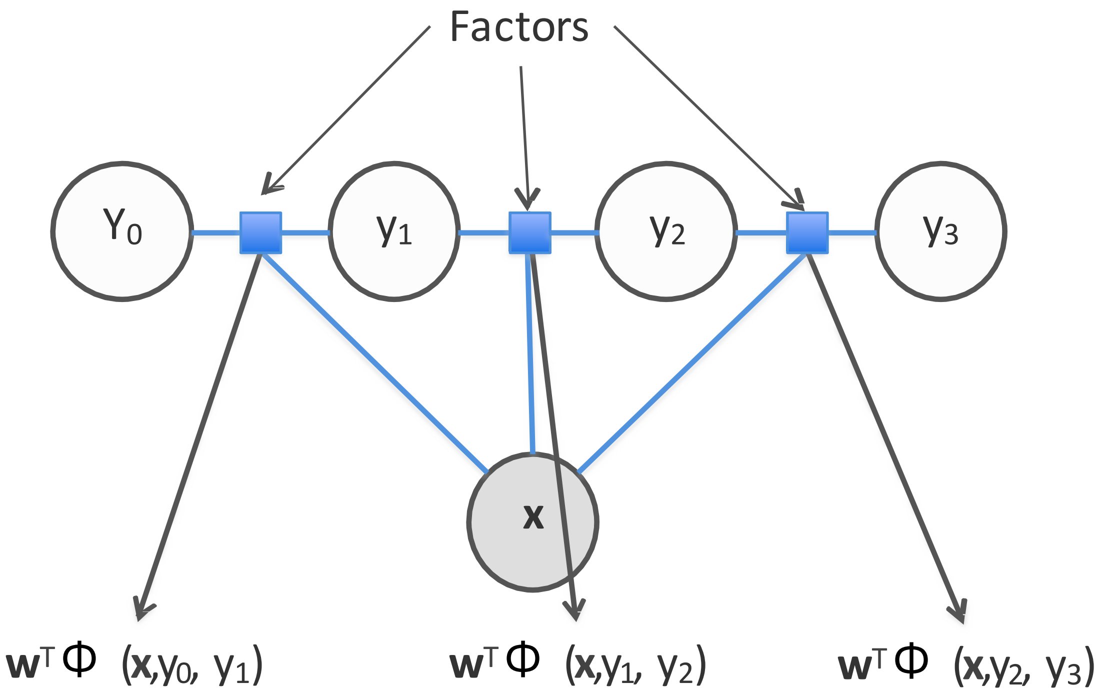
Given a sentence, are consecutive predictions independent of each other?

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*

based on V. Srikumar slides

Conditional Random Field: Factor graph



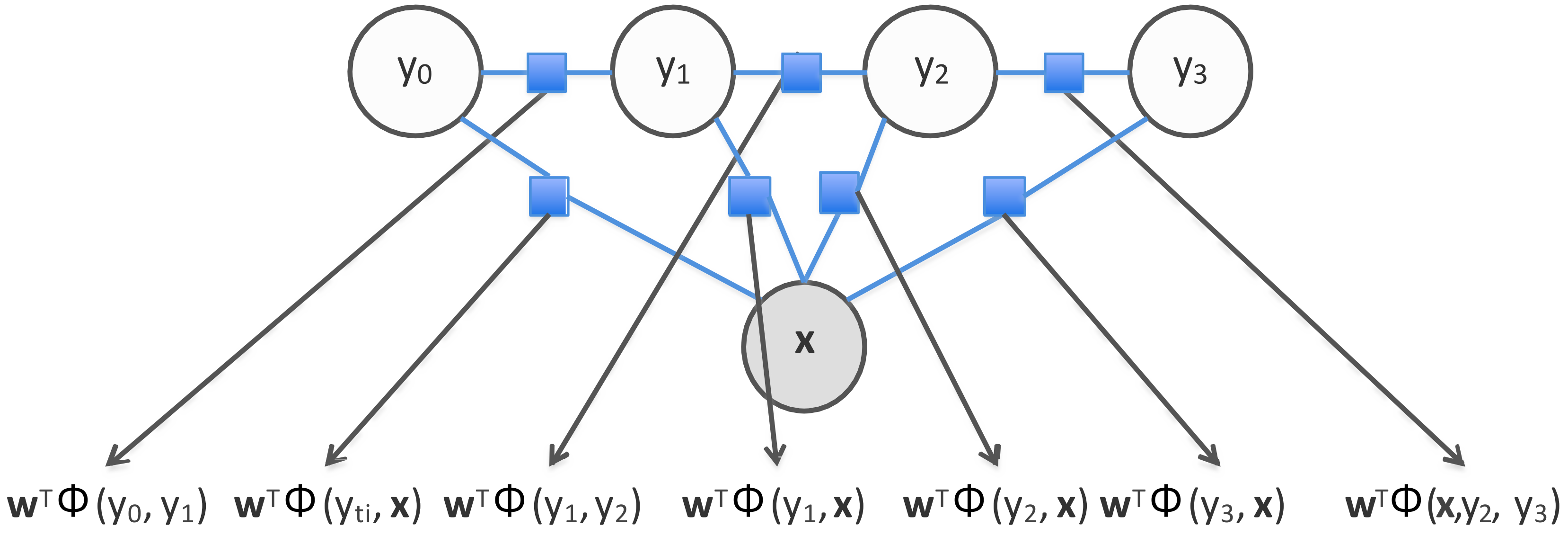
Each node is a random variable

We observe some nodes and need to assign the rest

Arbitrary features, as with local conditional models

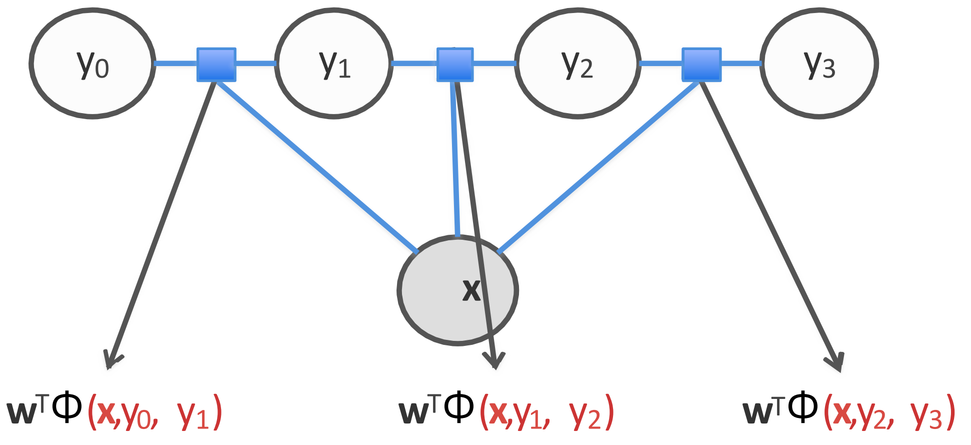
Each factor is associated with a score

Conditional Random Field: Factor graph


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

A different factorization: Recall decomposition of structures into parts.

Conditional Random Field for sequences



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

Assign a (conditional) probability to the **entire** sequence

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

Z: Normalizing constant, sum over all sequences

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

CRF: as a log-linear model over structures

- **Input**: x, **Output**: y,
  - **both sequences** (for now, in general can be more complex structures)
- Given a feature vector for the **entire** input and output sequence: Φ**(x,y)**
- Define a log-linear model, P(**y** | **x**) parameterized by **w**

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

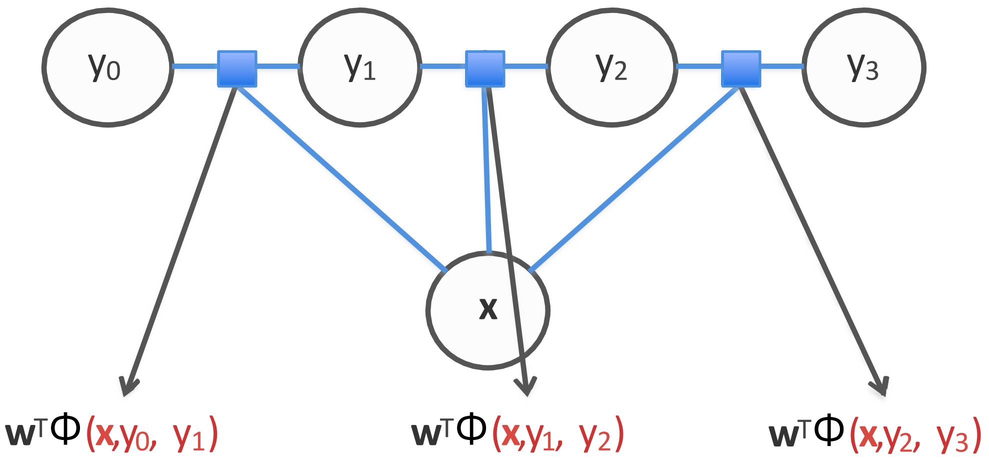
Similar to the log linear model we saw, but now –

- Space of **y** is the set of **all possible sequences** (length n)
- **Normalization constant sums over all sequences**

Global features

The feature function decomposes over the sequence

Aggregates all active features into a global representation



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

Prediction

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

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

Since the score decomposes:

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

➔ Prediction via Viterbi:

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

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

Note: modified definition of Viterbi, uses sum of scores, instead of products of probabilities

Training a chain CRF

- **Training a CRF**
  - *Maximize the (regularized) log-likelihood*

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

Training with inference

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

Many optimization methods:

- Numerical optimization
  - **Stochastic gradient ascent can also work very well**

Simple gradient ascent

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

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

- ***Training involves inference!***
  - Summing over all sequences is just like Viterbi
    - *With summation instead of maximization*

This is an instance of a repeating idea in training global models: ***Training requires inference***

CRF Summary

CRF: Assign condit. probability to **entire** sequence

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

Z: Normalizing constant, sum over all sequences

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

Can also be view as a log-linear model over sequences,

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

Note the feature vector in this case (see below) is defined over the entire sequence

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

With this view, the (regularized) learning objective is:

This is an instance of a big idea in training global models: ***Inference-based Training***

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

A smooth+convex objective, we can use gradient based method:

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

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

- **Relation to other models**
  - **A generalization of logistic regression to structures**
  - Effectively a linear classifier
    - **We'll look at others – perceptron, SVM for structure prediction**
  - An instance of Markov Random Field, with some random variables observed

Big Ideas

Generative vs. Discriminative models

Generative:

learn P(x, y)

Characterize "data generation"

Naïve Bayes, HMM

Discriminative:

learn P(y | x)

Conditional models (focus on decision boundary)

Probabilistic (Logistic Regression) or *not* (SVM,..)

Big Ideas

Local vs. Global models

Local:

Training: ***local*** *next-state classifiers*

Prediction: optimize **global** objective (e.g., using *Viterbi*)

MEMM (or use a non-probabilistic classifier)

Global:

Training: ***Global*** *sequence (or structure) predictor*

"Inference based training"

All global learners require inference at training time!

Prediction: optimize **global** objective (e.g., using *Viterbi*)

CRF, Structured Perceptron, Structured SVM,..

Let's look at a few more global models

In the beginning there was HMM

We derived a discriminative algorithm for learning sequences

- Local: MEMM
- Global: CRF

Today:

Let's derive another simple algorithm

- Recall: naïve Bayes is a linear model
- *Can we say the same about naïve Bayes for sequences?*

NB is a linear classifier, **so is HMM..**

- Consider the HMM

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

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

Indicators: I_z = 1 if z is true; else 0

$$\log P(\mathbf{y},\mathbf{x}) = \sum_i \sum_{s,s'} \log P(y_i = s|y_{i-1} = s')I_{y_i=s \text{ and } y_{i-1}=s'} + \sum_i \sum_s \log P(y_i = s)I_{y_i=s}$$

- Or equivalently

$$\log P(\mathbf{y},\mathbf{x}) = \sum_{s,s'} \log P(s|s')\mathrm{count}(s' \rightarrow s) + \sum_i \sum_s \log P(y_i = s)I_{y_i=s}$$

This is a linear function!

→ Log P terms are the weights; counts and indicators are features

→ Can be written as a $w^T\Phi(\mathbf{x},\mathbf{y})$,and use a different feature set

NB is a linear classifier, **so is HMM..**

HMM is a linear classifier

- Can be written in the form: $w^T\Phi(\mathbf{x},\mathbf{y})$
- We can add other features beyond omission/transition features, as long as the output can still be decomposed

Difference from traditional HMM:

Inference: Viterbi calculates a the maximal score (rather than probabilities) i.e., max $w^T\Phi(x, y)$

Learning: Use other different learning algorithms (not necessarily probabilistic)

- If we need a probabilistic interpretation, we could always normalize
- *We can effectively just focus on the score of y for a particular x*
- **Train a discriminative model!**

Structured Perceptron algorithm

```
Given a training set D = {(x,y)}
1. Initialize $\mathbf{w} \in \mathbf{R}^n$
2. For Iteration= 1 … T:
    1. For each training example (x, y) ∈ D:
        1. Predict $y' = \text{argmax}_{y'} \mathbf{w}^T\Phi(x, y')$
        2. If $y \neq y'$, update $\mathbf{w} \leftarrow \mathbf{w} + \text{learningRate}\,(\Phi(x,y) - \Phi(x, y'))$
3. Return w
```

T is a hyperparameter to the algorithm

Inference based training

Note: update only on error! (perceptron is mistake driven)
Promote y (gold structure) and demote y' (predicted structure)

Prediction: $\text{argmax}_y \mathbf{w}^T\Phi(x, y)$

Structured Perceptron algorithm

Extension of binary perceptron for the structured case

- Similar guarantees; if data is linearly separable
- **Good idea**: limit the number of iterations
  - Tune as a hyperparameter; usually few are enough

You can add stability by **averaging** the models perceptron produces:

- Maintain a counter for each weight vector seen during training.
- Increase the counter at each iteration (regardless if an update occurred)
  - Return weighted average of all weight vectors.

Structured Perceptron with averaging

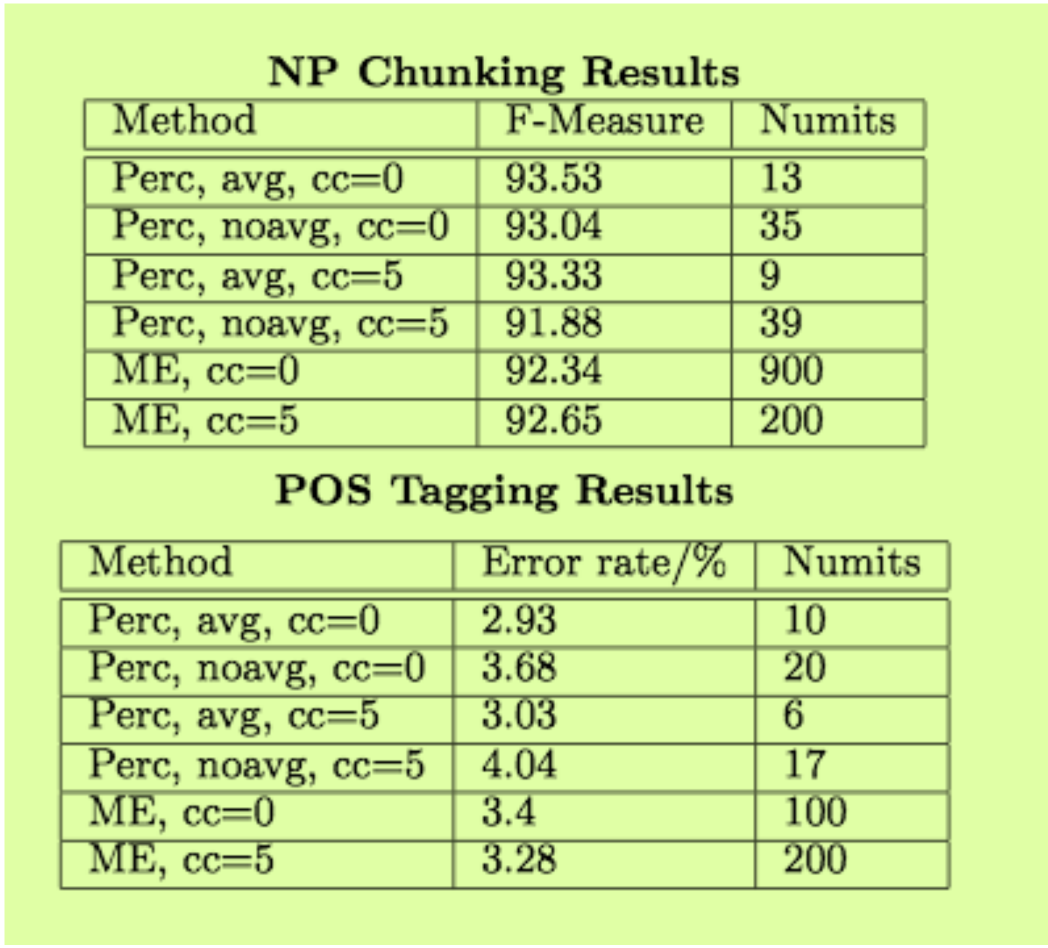
```
Given a training set D = {(x,y)}
1. Initialize  $\mathbf{w} \in \mathbf{R}^n$,  $\mathbf{a} \in \mathbf{R}^n$
2. For epoch = 1 … T:
    1. For each training example $(\mathbf{x}, \mathbf{y}) \in D$:
        1. Predict $\mathbf{y}' = \mathrm{argmax}_{y'}\, \mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y}')$
        2. If $\mathbf{y} \neq \mathbf{y}'$, update $\mathbf{w} \leftarrow \mathbf{w} + \mathrm{learningRate}\,(\Phi(\mathbf{x},\mathbf{y}) - \Phi(\mathbf{x},\mathbf{y}'))$
        3. Set $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{w}$
3. Return $\mathbf{a}/(n\mathrm{T})$
```

Structure Perceptron for Tagging

Use a global feature vector

$$\phi_{1000}(h,t) = \begin{cases} 1 & \text{if current word } w_i \text{ is the} \\ & \text{and } t = \text{DT} \\ 0 & \text{otherwise} \end{cases}$$

Compare a local model (MEMM) with a global model trained using the structured perceptron algorithm



NP Chunking Results

Method	F-Measure	Numits
Perc, avg, cc=0	93.53	13
Perc, noavg, cc=0	93.04	35
Perc, avg, cc=5	93.33	9
Perc, noavg, cc=5	91.88	39
ME, cc=0	92.34	900
ME, cc=5	92.65	200

POS Tagging Results

Method	Error rate/%	Numits
Perc, avg, cc=0	2.93	10
Perc, noavg, cc=0	3.68	20
Perc, avg, cc=5	3.03	6
Perc, noavg, cc=5	4.04	17
ME, cc=0	3.4	100
ME, cc=5	3.28	200

Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms Michael Collins. ACL 2002

CRF and Structured Perceptron are not that different!

stochastic gradient descent update for CRF

- For a training example ($\mathbf{x}_i$, $\mathbf{y}_i$)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_t \left(\phi(\mathbf{x}_i, \mathbf{y}_i) - E_{\mathbf{y}}[\phi(\mathbf{x}_i, \mathbf{y})]\right)$$

Structured perceptron

Intuition: CRF: Expectation ("soft max")
S. Perceptron: max

- For a training example ($\mathbf{x}_i$, $\mathbf{y}_i$)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_t \left(\phi(\mathbf{x}_i, \mathbf{y}_i) - \phi(\mathbf{x}_i, \hat{\mathbf{y}})\right)$$

$\hat{\mathbf{y}}$ Is the result of argmax

Large Margin Structure Prediction

Recall the Cost-Sensitive Multi-Class SVM

Learning:

$$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \; l(\mathbf{w};(\mathbf{x}^n, y^n))$$

$$l(\mathbf{w};(\mathbf{x}^n, y^n)) =$$

$$\max_{y' \in Y} \Delta(y, y') - \langle \mathbf{w}, \phi(\mathbf{x}^n, y^n)\rangle + \langle \mathbf{w}, \phi(\mathbf{x}^n, y')\rangle$$

Prediction:

$$f(x) = \text{argmax}_{y \in \mathcal{Y}} \;\; \langle w, \phi(x, y)\rangle$$

What would we need to change?

Structured Prediction

The key difference is when computing:

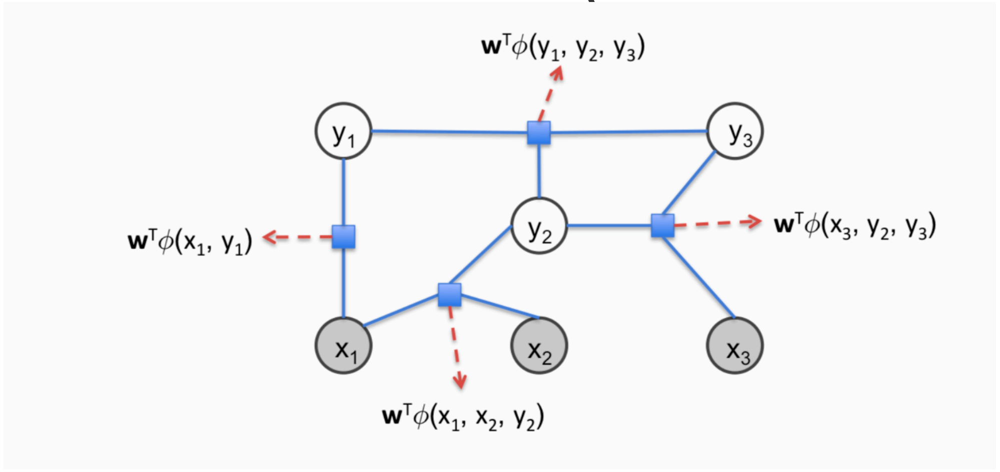
$$f(x) = \text{argmax}_{y \in \mathcal{Y}} \ \langle w, \phi(x, y) \rangle$$

We are predicting a vector instead of a single value.

- Also – adapt the distance function Δ for structure
- *Popular choice: hamming distance*

Structure SVM – Take 1

Suppose we have a structure (defined as a factor graph)



$\mathbf{w}^T\phi(y_1, y_2, y_3)$
y_1
y_3
y_2
$\mathbf{w}^T\phi(x_1, y_1)$
$\mathbf{w}^T\phi(x_3, y_2, y_3)$
x_1
x_2
x_3
$\mathbf{w}^T\phi(x_1, x_2, y_2)$

Each factor (or **part**) associated with a feature function

The feature vector for the entire structure is defined by summing up the features for the parts

$$\phi(\mathbf{x}, \mathbf{y}) = \sum_{p \in \text{parts}(\mathbf{x})} \phi_p(\mathbf{x}, y_p)$$

Structure SVM – **Take 1**

Given some training data, we want to enforce the following:

For each example:

The annotated structure (**y**$_i$) gets the highest score among all structures (*structured Perceptron*)

$$\mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}_i) \geq \mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}) + 1, \quad \forall \mathbf{y}$$

Structure SVM – **Take 1**

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

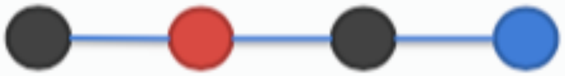
Maximize margin by minimizing the norm of **w**

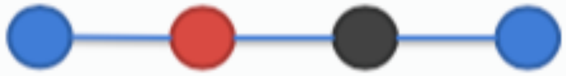
$$\text{s.t.} \quad \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i) \geq \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + 1 \quad \forall(\mathbf{x}_i,\mathbf{y}_i) \in D, \forall \mathbf{y} \neq \mathbf{y}_i$$

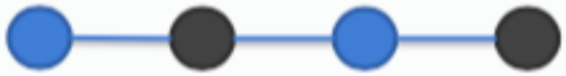
Score for gold output

Score for other output

Problem


Gold structure


Other structure A: Only one mistake


Other structure B: Fully incorrect

Structure B has is more wrong, but this formulation will be happy if *both* A & B are scored one less than gold!

Structure SVM – **Take 2**

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

Maximize margin by minimizing the norm of **w**

$$\text{s.t.} \quad \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i) \geq \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + 1 \quad \forall(\mathbf{x}_i,\mathbf{y}_i) \in D, \forall \mathbf{y} \neq \mathbf{y}_i$$

Score for gold output

Score for other output

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

Hamming distance between structures

$$\text{s.t.} \quad \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i) \geq \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + \Delta(\mathbf{y},\mathbf{y}_i) \quad \forall(\mathbf{x}_i,\mathbf{y}_i) \in D, \forall \mathbf{y}$$

Structure SVM – **Take 2**

Hamming distance
between structures

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{s.t.} \quad \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i) \geq \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + \Delta(\mathbf{y},\mathbf{y}_i) \quad \forall(\mathbf{x}_i,\mathbf{y}_i) \in D, \forall \mathbf{y}$$

Intuition:

➔Structures that are close to the true structure (according to their Hamming distance) can get a close score

➔Structures that are very different should be further apart

Problem?

Structure SVM – **Take 3**

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{s.t.} \quad \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i) \geq \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + \Delta(\mathbf{y},\mathbf{y}_i) \quad \forall(\mathbf{x}_i,\mathbf{y}_i)\in D, \forall\mathbf{y}$$

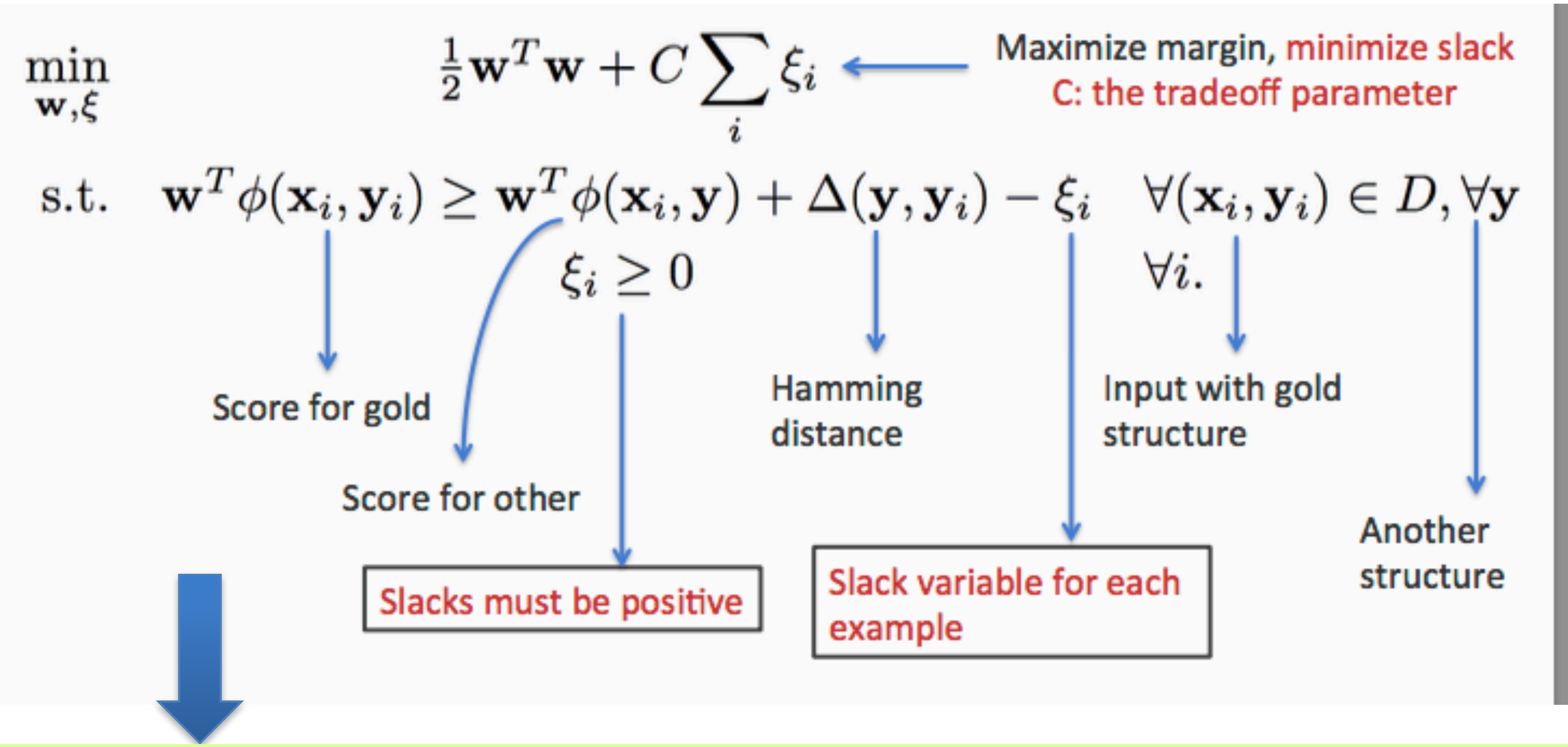
Also minimize total slack

Slack variable for each example, **must be positive**

$$\min_{\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i$$

$$\text{s.t.} \quad \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i) \geq \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + \Delta(\mathbf{y},\mathbf{y}_i) - \xi_i \quad \forall(\mathbf{x}_i,\mathbf{y}_i)\in D, \forall\mathbf{y}$$

Structure SVM – **Take 3**



$$\min_{\mathbf{w},\xi} \quad \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i$$

$$\text{s.t.} \quad \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i) \geq \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + \Delta(\mathbf{y},\mathbf{y}_i) - \xi_i \quad \forall(\mathbf{x}_i,\mathbf{y}_i)\in D, \forall \mathbf{y}$$

$$\xi_i \geq 0 \qquad \forall i.$$

Maximize margin, minimize slack
C: the tradeoff parameter
Score for gold
Score for other
Hamming distance
Input with gold structure
Another structure
Slacks must be positive
Slack variable for each example

$$\min_{\mathbf{w}} \quad \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max_{\mathbf{y}}\left(\mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + \Delta(\mathbf{y},\mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i)\right)$$

Sub gradient for S-SVM

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max_{\mathbf{y}} \left(\mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}_i)\right)$$

Solve the max: $y' = \max_{\mathbf{y}} \left(\mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}_i)\right)$

Notice the difference!

This is known as **loss-augmented inference**

Question: how would you tweak Viterbi?

The sub-gradient is:

$$\mathbf{w} + C\left(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\right)$$

SGD for S-SVM

At each step go down the (sub) gradient:

$$\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \left(\mathbf{w} + C\left(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\right)\right)$$

What happens if y’ = y ?

Equivalent algorithm:

If **y’**=**y**:

$$\mathbf{w} \leftarrow (1 - \gamma_t)\mathbf{w}$$

Otherwise:

$$\mathbf{w} \leftarrow (1 - \gamma_t)\mathbf{w} + \gamma_t \left(C\left(\phi(\mathbf{x}_i, \mathbf{y}_i) - \phi(\mathbf{x}_i, \mathbf{y}')\right)\right)$$

SGD for S-SVM

```
Given a training set D = $\{(\mathbf{x}_i,\mathbf{y}_i)\}$
Initialize $\mathbf{w} = 0 \in \Re^n$
1. For epoch = 1 … T:
    1. Shuffle data
    2. For each training example $(\mathbf{x}_i, \mathbf{y}_i) \in D$:
        1. Let $\mathbf{y}' = \text{argmax}_{\mathbf{y}}\ \mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}_i, \mathbf{y}) - \mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}_i)$
        2. If y' = y: shink $w \leftarrow w(1- \gamma_t)$
        3. Else: update $w \leftarrow w(1- \gamma_t) + \gamma_t C\,(\phi(\mathbf{x}_i, \mathbf{y}_i) - \phi(\mathbf{x}_i, \mathbf{y}'))$
2. Return w
```

(loss-augmented) Inference step

Model update

Open Questions

So far we have seen global and local training regimes

- Local classifier: learn next state functions
- Global classifier: optimize a global objective

Pushing this idea forward: *can we learn global models over different structure types?*

- E.g., a single model for PoS, chunking, NER and parsing

Can we always assume that everything is annotated?

How can we move beyond sequences?

Structured Learning with missing Information

Paraphrase Detection

Consider the following prediction problem:

Given two sentences, determine if the sentences have the same meaning.

The US president met the British PM in London today

Mary gave the ball to John earlier today

The US and British heads of state met in London at 5 PM.

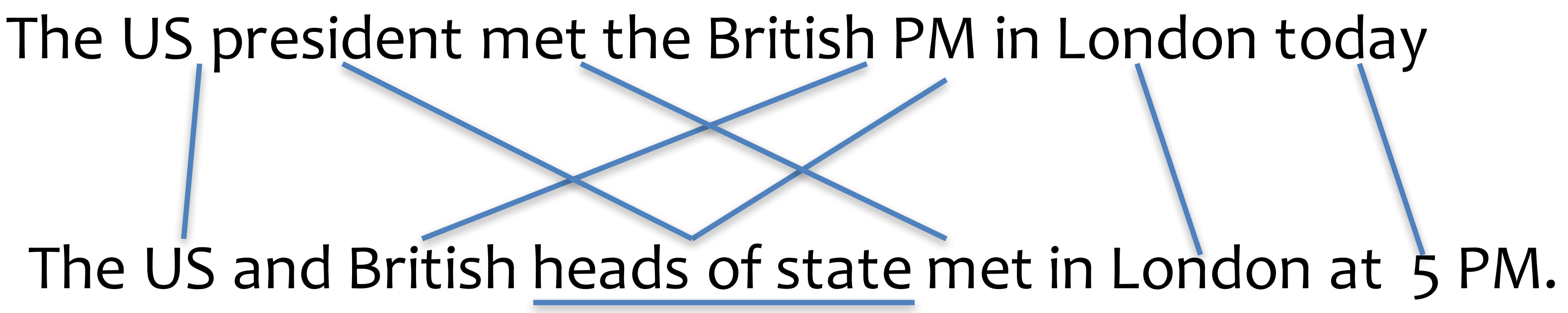
The British PM met the president of the London club today.

Can you design a classifier for this task?

Paraphrase detection

It’s clear that a better representation of the training instances is needed

Intermediate task: *explain* why two sentences form a paraphrase pair.

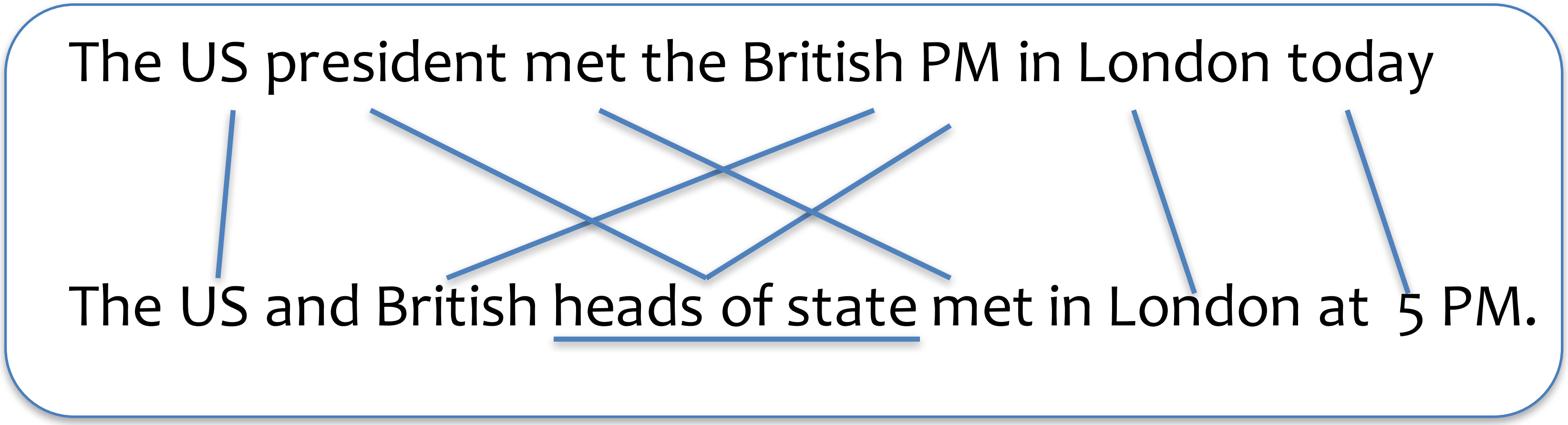


The US president met the British PM in London today

The US and British heads of state met in London at 5 PM.

However – the training data does not provide it!

Learning with Latent Variables


The US president met the British PM in London today
The US and British heads of state met in London at 5 PM.

Basic idea: *define a structured prediction problem as explanation*

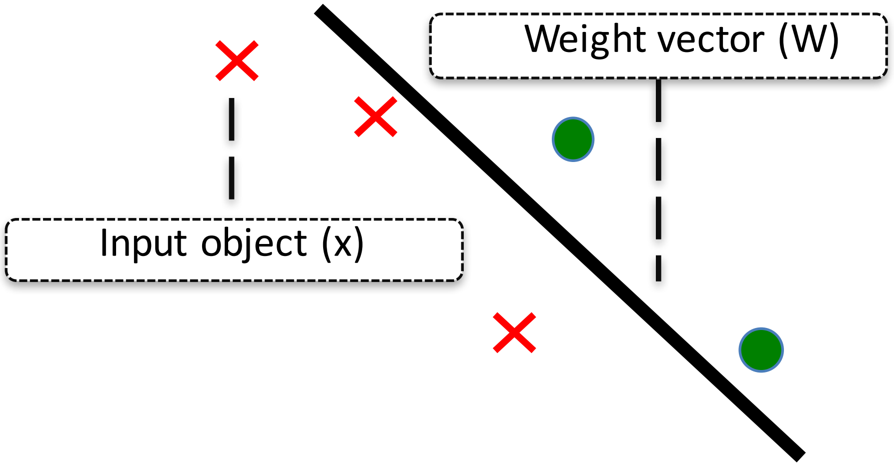
$$h^* = \arg\max_{h \in H(x)} \mathbf{w}^T \Phi(\mathbf{x}, h)$$

$$\Phi(x,h) = <\phi(x,h_0), \phi(x,h_0,h_1), \ldots, \phi(x,h_{d-1},h_d)>$$

.. And then use it to make predictions:

$$f_{\mathbf{w}}(\mathbf{x}) = \max_{\mathbf{h}} \sum_s h_s \mathbf{w}^T \phi_s(\mathbf{x})$$

Classification 101

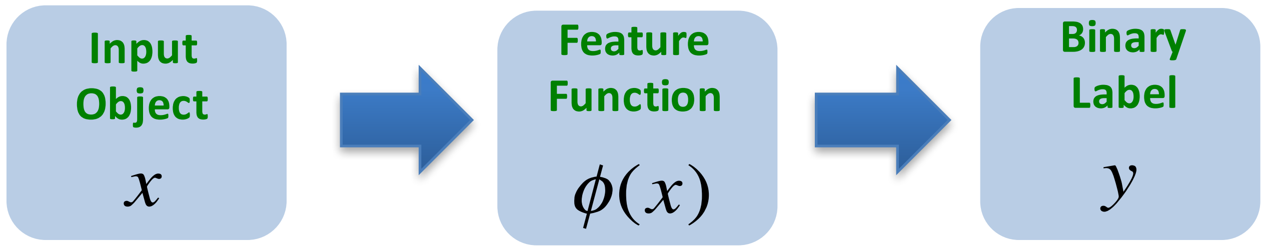


- **Input Representation**: feature functions $\phi(x)$
  - E.g. *Bag-of-words, N-grams features*
- **Prediction** : $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$, $f_{\mathbf{w}}(\mathbf{x}) > 0$
- **Learning**: given training data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_i \ell\left(-y_i f_{\mathbf{w}}(\mathbf{x}_i)\right)$$

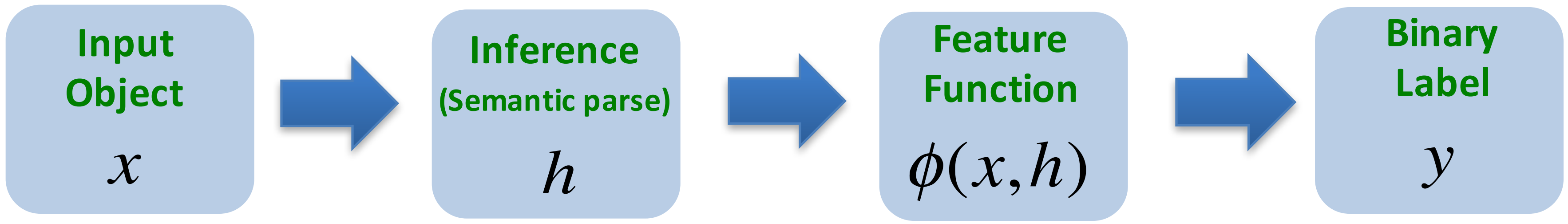
Prediction over Latent Structures

The typical classification settings -



Prediction over Latent Structures

But our settings are different



- **Different Prediction function!** $f_{\mathbf{w}}(\mathbf{x}) > 0$

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) \longrightarrow f_{\mathbf{w}}(\mathbf{x}) = \max_{\mathbf{h}} \sum_s h_s \mathbf{w}^T\phi_s(\mathbf{x})$$

- Feature representation is no longer deterministic
  - Depends on inference (**structured prediction**)

Structured Prediction Definitions

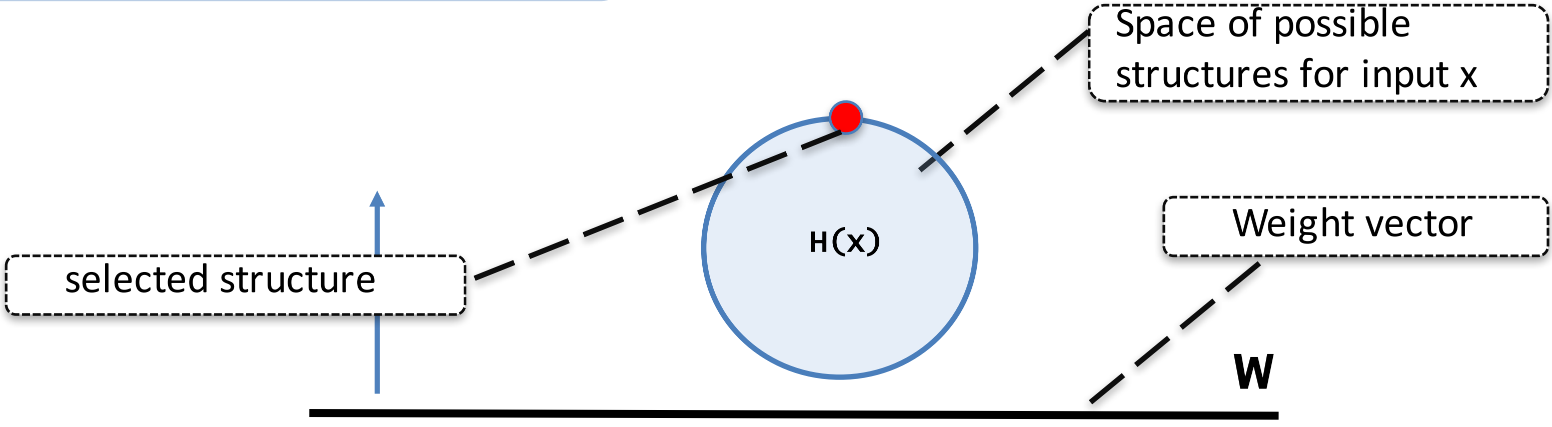
Finding the optimal structure

Structure represented as features

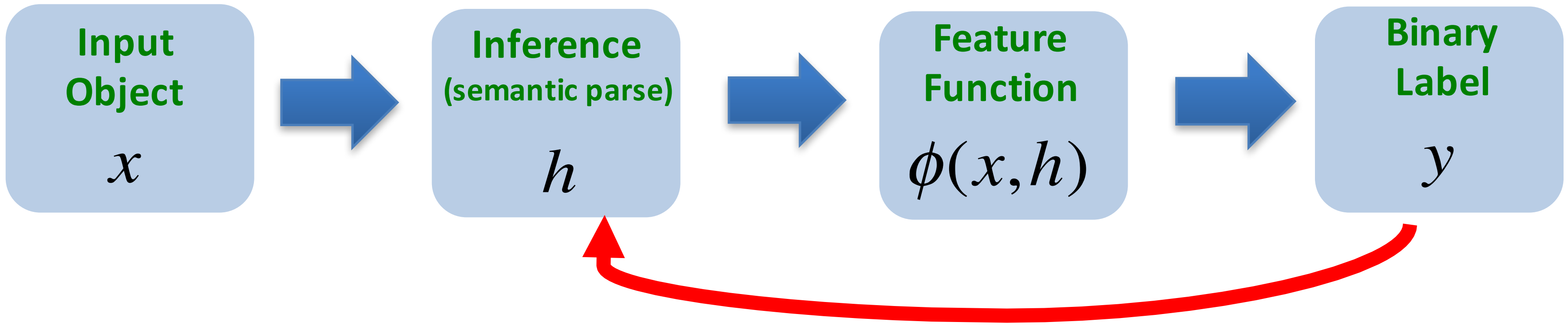
$$\Phi(x,h) = <\phi(x,h_0),\phi(x,h_0,h_1),\ldots,\phi(x,h_{d-1},h_d)>$$

Given a weight vector *W,* use a linear model

$$h^* = \arg\max_{h \in H(x)} \mathbf{w}^T \Phi(\mathbf{x}, h)$$



Learning over Latent Structures



*Learn both problems **jointly** using Binary supervision*

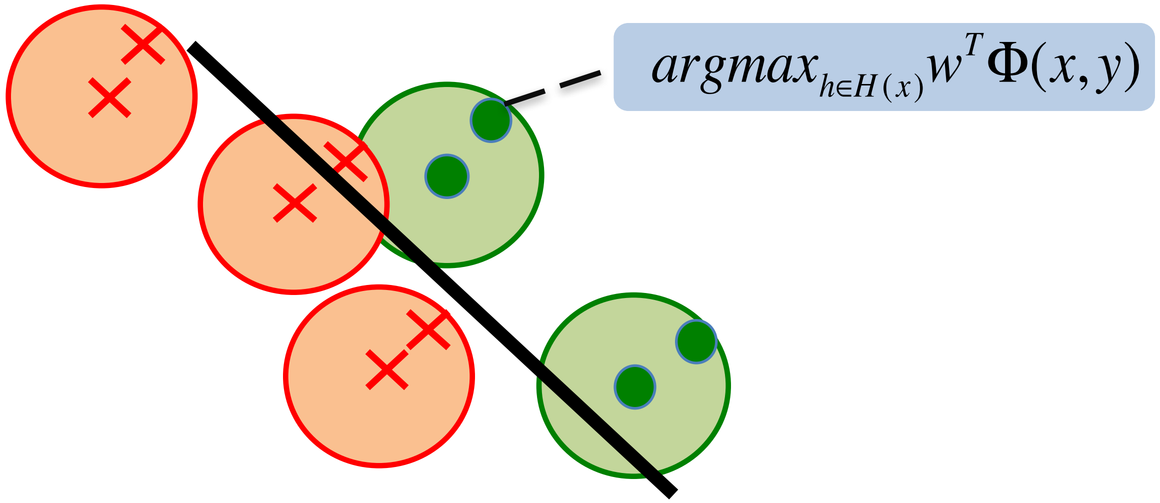
- **Fixed Supervision**: Feedback is associated with x

  All representations have the same label

  Binary Classification with latent structure

Learning over Latent Representations

Each input defines a set of possible representations

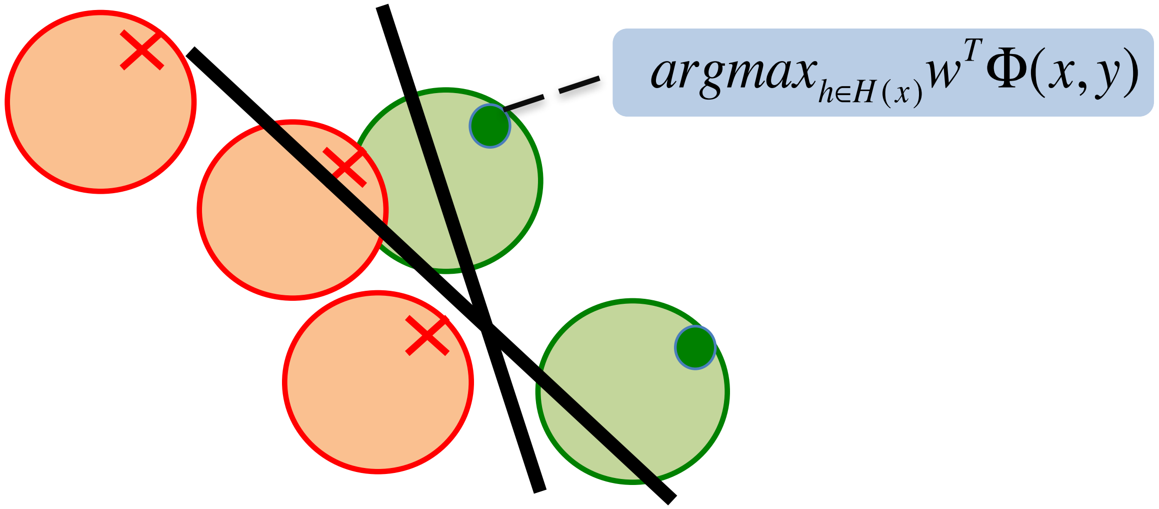


w should separate **single** representation

Best negative structure <0 ; Best positive structure >0

Fix Representation.

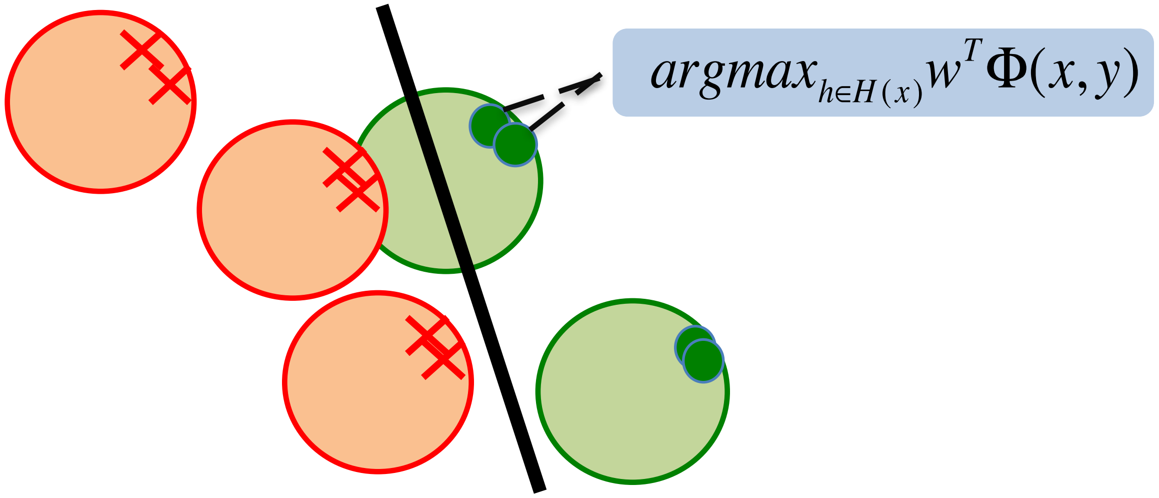
Learning over Latent Representations



- **Each data point has a single feature representation**
  - *Learning is now straight-forward*

Fix Representation.

Learning over Latent Representations



The new weight vector defines a new representation

Fix Representation. Update Weights.

Learning over Latent Representations

NAACL'10

Prediction is similar to SVM:

$$f_{\mathbf{w}}(\mathbf{x}) \geq 0$$

One difference:

Learning: Given a s $f_{\mathbf{w}}(\mathbf{x}) = \max_{\mathbf{h}} \sum_{s} h_s \mathbf{w}^T \phi_s(\mathbf{x})$ $(\mathrm{x_k}, \mathrm{y_k})\}$

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_i \ell\left(-y_i f_{\mathbf{w}}(\mathbf{x}_i)\right)$$

$$\Downarrow$$

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_i \ell\left(-y_i \max_{\mathbf{h} \in \mathcal{C}} \mathbf{w}^T \sum_{s \in \Gamma(\mathbf{x})} h_s \phi_s(\mathbf{x}_i)\right)$$

Non Convex Objective!
- Local minimum
- Exploit semi-convexity
  - Fix positive representation, solve for negatives

Structure Learning with Latent Variables

We can easily extend this framework for the structured output case.

Let define the prediction problem:

$$\arg\max_{\mathbf{h},\mathbf{y}} \mathrm{w}^T \Phi(\mathbf{x},\mathbf{h},\mathbf{y})$$

E.g., assume a latent variable associated with each state in a sequence.

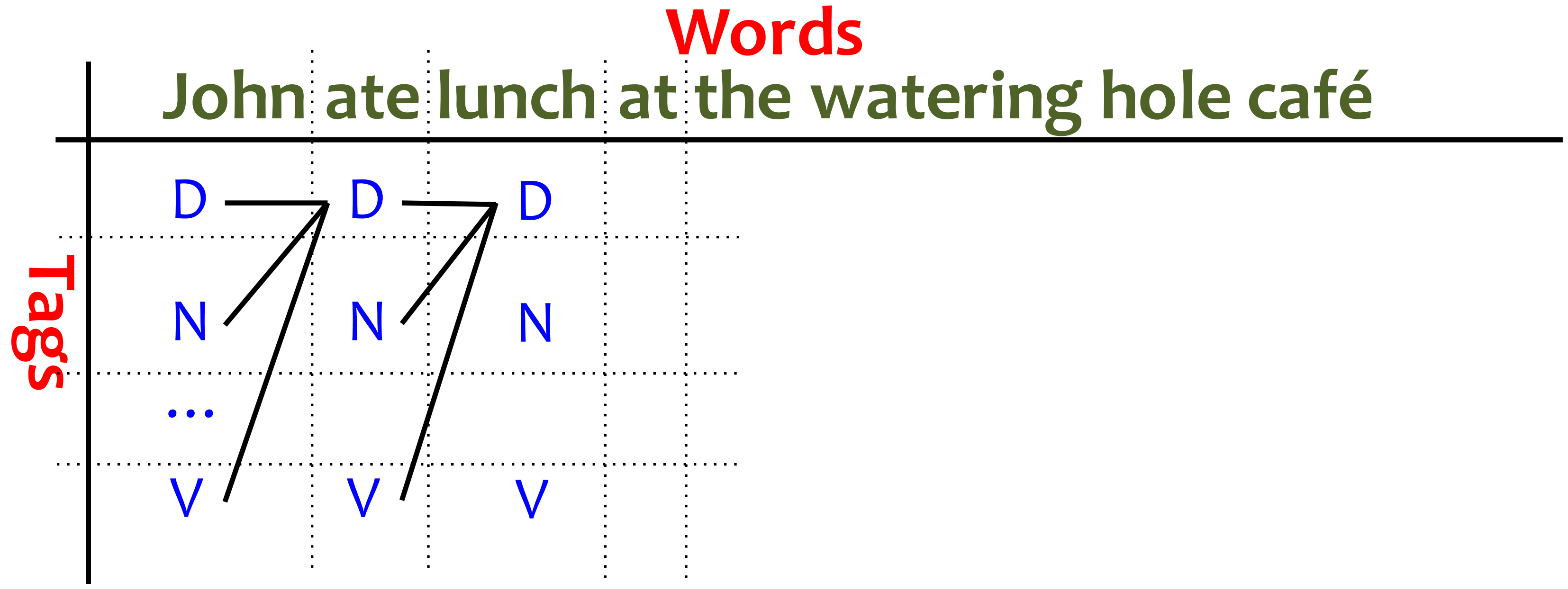
How would you modify traditional Viterbi for this case?

Viterbi: DP Table

$$\pi(k,v) = max_{(y_1,\ldots,y_k; y_k = v)} r(y_1,\ldots,y_k)$$

max probability tag sequence of size k ending with v

$$\pi(k,v) = \max_{u \in S_k - 1} (\pi(k-1,u) \times p(v|u) \times p(x_k|v))$$



$$\pi(3,D) = max_{(y_1,\ldots,y_3; y_3 = D)} r(y_1,\ldots,y_3)$$

Example

Consider a dialog analysis system, that classifies users utterances into *dialog-acts*

Hello, good morning!

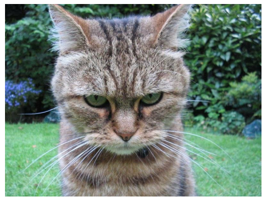
Can you please tell me where to go? Q

I'm sorry I did not get that

Where should I go? Q

I'm sorry I did not get that

Me. Where. Go. S

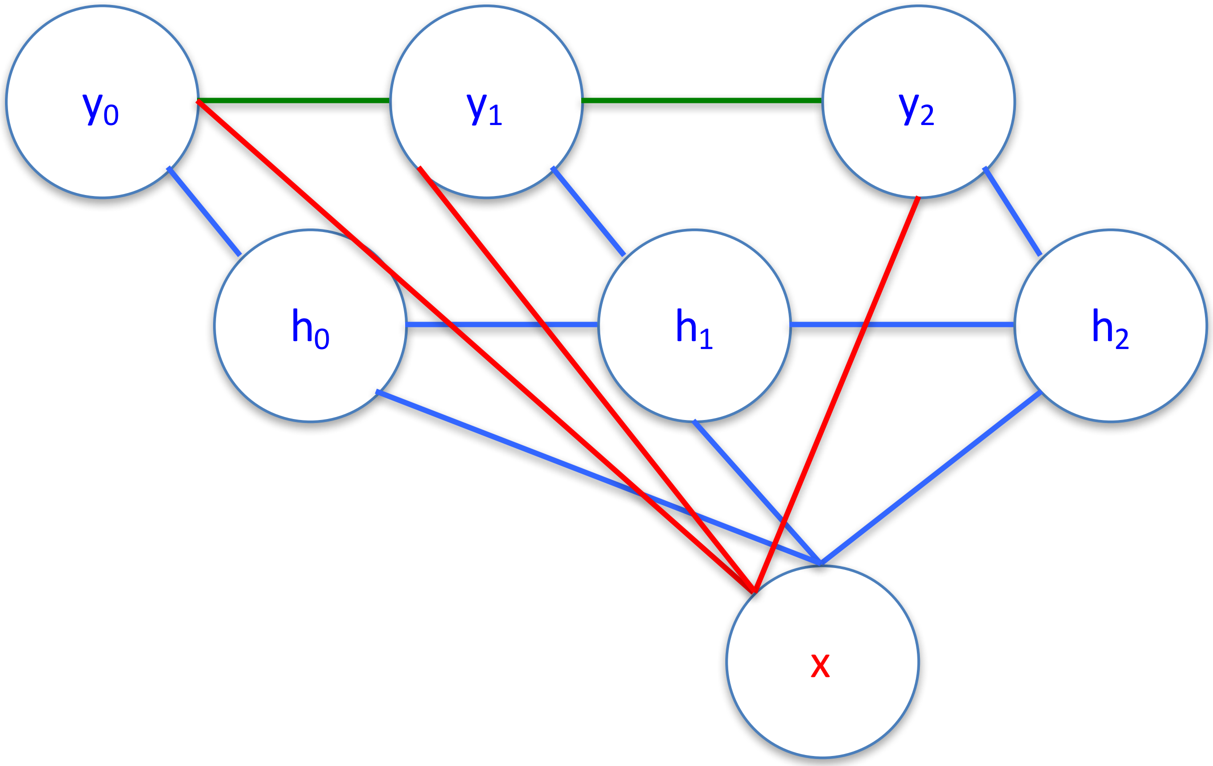


I'm sorry I did not get that

Why don't you go to @#$# ? Q

Example

Consider a dialog analysis system, that classifies users utterances into *dialog-acts*



Realistic Scenarios

natural language processing is fun!

traitement du langage naturel est amusant!

自然语言处理是乐趣！

обработки естественного языка это весело!

procesamiento del lenguaje natural es muy divertido!

प्राकृतिक भाषा संसाधन मजेदार है!

Training data consists of full sentences, not word pairs!

Realistic Scenarios

"Hi Siri, please make a reminder of the 23rd of the month to call mom"

↓

Make_reminder(Date(Feb, 23,2106), r1)
ReminderAction(r1, FindEntry(Mom), call)

Training data consists of English + Logical, not word and predicates!

Structured Perceptron algorithm

```
Given a training set D = {($\mathbf{x},\mathbf{y}^*$)}
1.  Initialize $\mathbf{w} \in \mathbf{R}^n$
2.  For Iteration= 1 … T:
    1.  For each training example ($\mathbf{x}, \mathbf{y}$) $\in$ D:
        1.  Predict $\mathbf{h}', \mathbf{y}'$ = $\text{argmax}_{\mathbf{y}'}\, \mathbf{w}^T\Phi(\mathbf{x}, h, \mathbf{y}')$
        2.  Predict $\mathbf{h}^*$ = $\text{argmax}_{\mathbf{h}'}\, \mathbf{w}^T\Phi(\mathbf{x}, h, \mathbf{y}^*)$
        3.  If $\mathbf{y} \neq \mathbf{y}'$, update $\mathbf{w} \leftarrow \mathbf{w}$ + learningRate $(\Phi(\mathbf{x},\mathbf{h}^*,\mathbf{y}^*) - \Phi(\mathbf{x}, \mathbf{h}', \mathbf{y}'))$
3.  Return $\mathbf{w}$
```

Prediction: $\text{argmax}_{y}\, \mathbf{w}^T\Phi(\mathbf{x}, \mathbf{h}, \mathbf{y})$

Latent Structure SVM

$$L_D(\mathbf{w}) = \min_{w} \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{n}\sum_{i=1}^{n} \xi_i$$

$$\xi_i = \max_{y,\mathbf{h}} f(\mathbf{x}, \mathbf{h}, y, \mathbf{w}) + cost(y, y_i)$$

$$- \max_{\mathbf{h}} f(\mathbf{x}, \mathbf{h}, y_i, \mathbf{w})$$

Latent Structure SVM

The Gradient update is similar to S-SVM:

$$\phi(\mathbf{x}_i, \mathbf{h}^*, y_i) - \phi(\mathbf{x}_i, \mathbf{h}^*, y^*))$$

(.. And also shrink W at each update step)

Beyond Sequence Models

Graphical models

- **A language to represent probability distributions over multiple random variables**
  - Representation: Directed or undirected graphs
  - Encodes conditional independence assumptions
- **General machinery for:**
  - Representing, estimating and computing marginal + conditional probabilities
- In general – support **inferences about a domain**

Interestingly, you have already seen two graphical models!
Which ones?

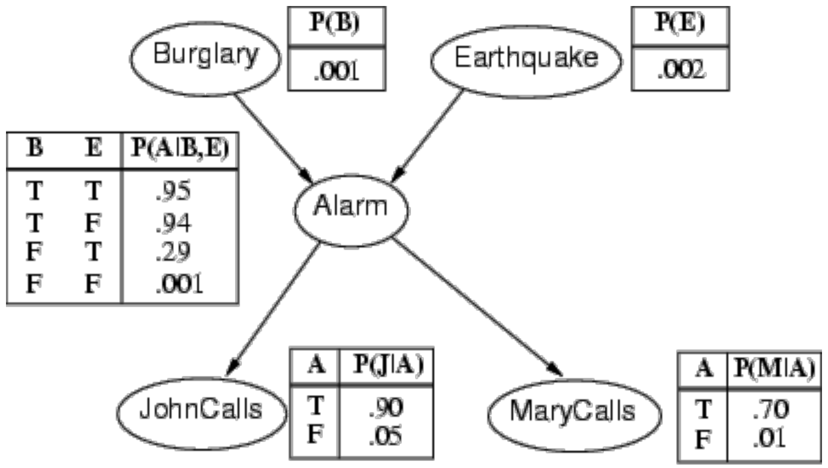
Bayesian Network

Data structure for representing **joint probability**

Encodes independence assumptions

Decompose joint probability via a directed acyclic graph

- **Nodes** represent random variables
- **Edges** represent conditional dependencies
- Each node is associated with a conditional probability table (CPT)



P(B)
.001

P(E)
.002

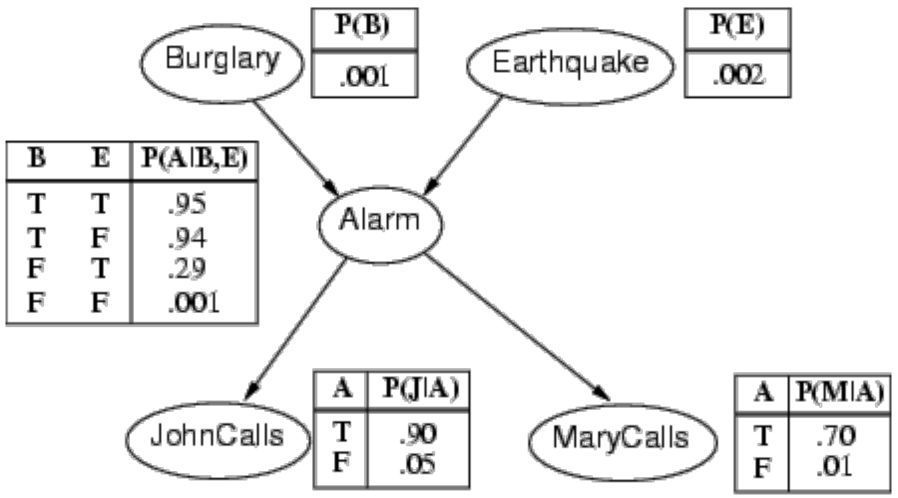
B	E	P(A|B,E)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J|A)
T	.90
F	.05

A	P(M|A)
T	.70
F	.01

$$P(z_1, z_2, \cdots, z_n) = \prod_i P(z_i | \text{Parents}(z_i))$$

Bayesian Network



$$P(z_1, z_2, \cdots, z_n) = \prod_i P(z_i|\text{Parents}(z_i))$$

Compact representation of the joint probability distribution

P(B, E, A, J, M) = P(B) P(E) P(A| B, E) P (J | A) P) M |A)

Question: Where do the independence assumptions come from?

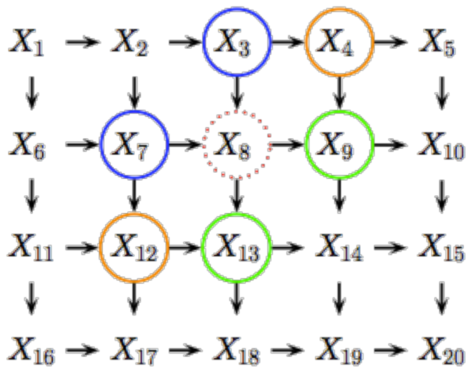
Domain knowledge, ***can they be learned?***

Example from Russell and Norvig

Bayesian Network

- **Example**: Hidden Markov Model
  - Naïve bayes classifier is a simple Bayes net

- **Problem**: *Bayesian nets can create unnatural conditional independence*
  - Eg: Segmenting an image by assigning a label to each pixel
    - Say, we want adjacent labels to influence each other



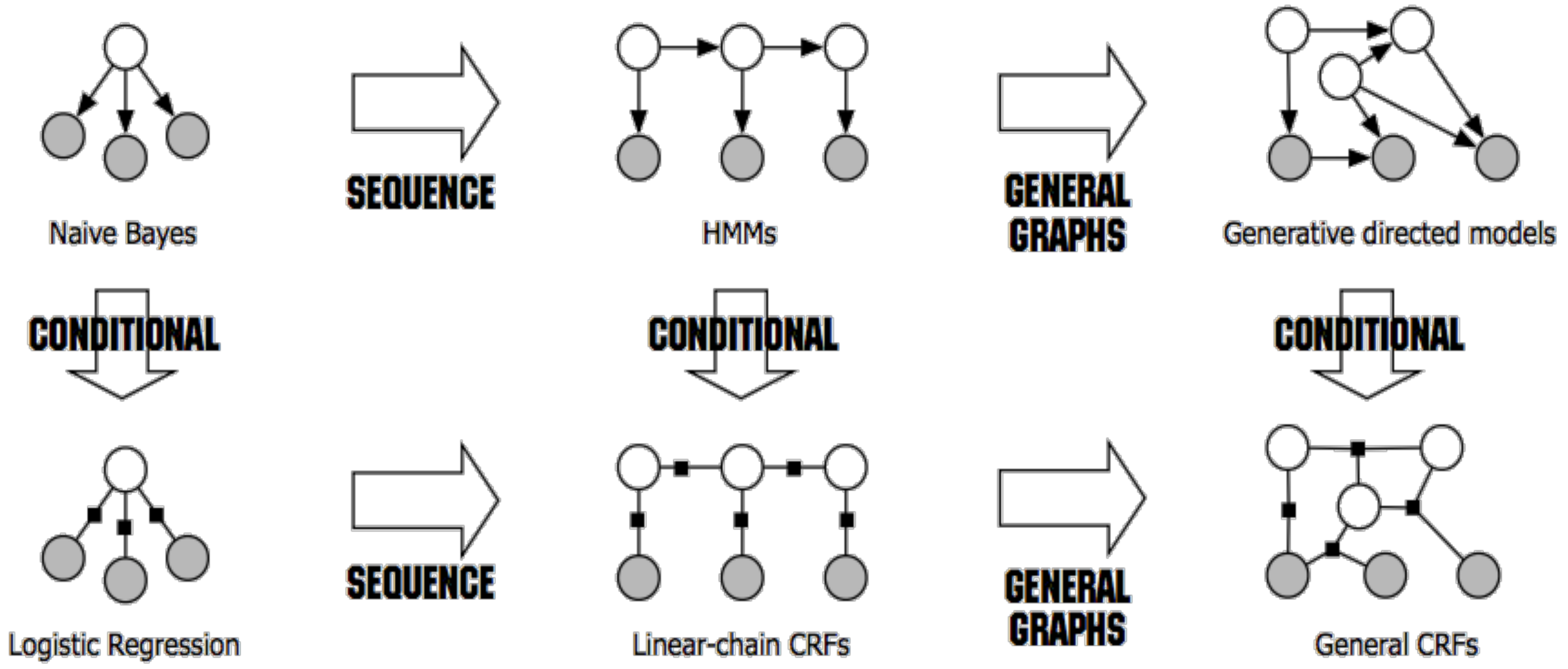
Two problems:

1. **What is the right direction of arrows?**

2. For any choice of the arrows, strange dependencies show up. X_8 is independent of everything given its Markov blanket (other circled nodes here)

Example from Kevin Murphy

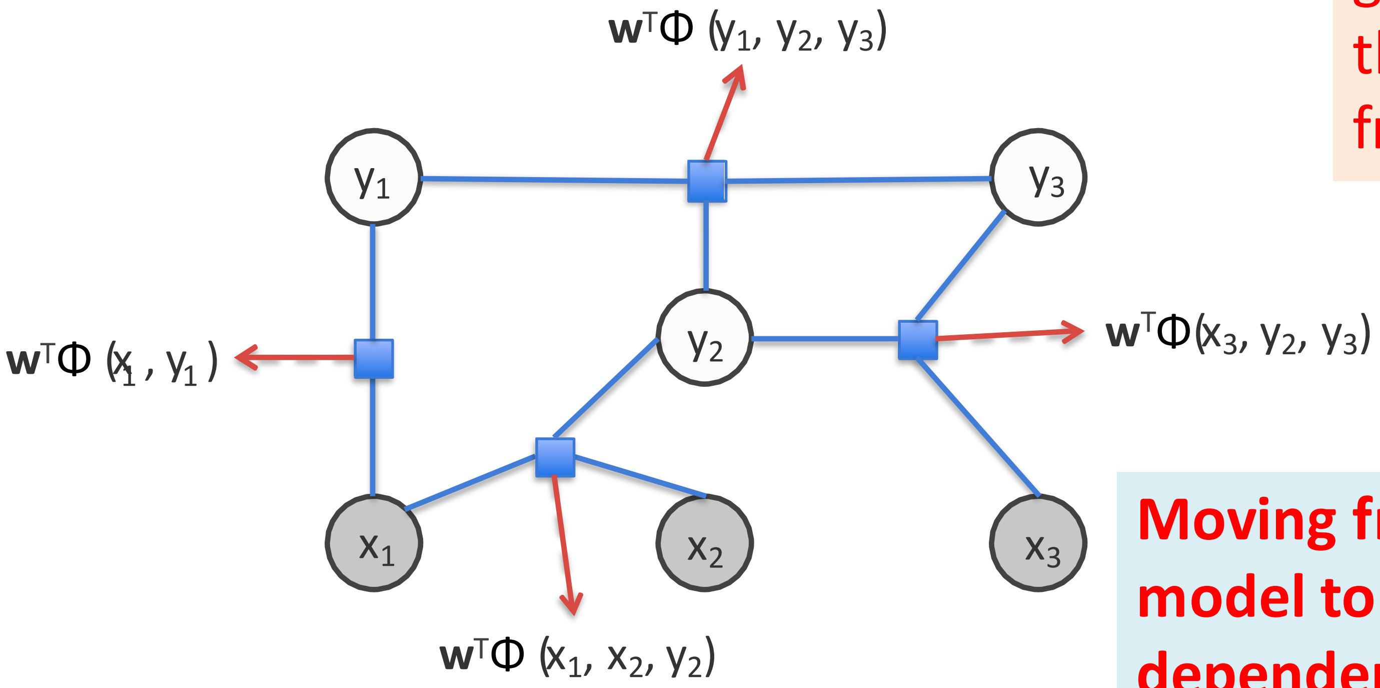
From generative models to CRF



[Figure from Sutton and McCallum, ‘05]

General CRFs

We can talk about NB, sequence models and general graphical models in the same framework


$\mathbf{w}^T\Phi(y_1, y_2, y_3)$
y_1
y_3
y_2
$\mathbf{w}^T\Phi(x_1, y_1)$
$\mathbf{w}^T\Phi(x_3, y_2, y_3)$
x_1
x_2
x_3
$\mathbf{w}^T\Phi(x_1, x_2, y_2)$

Moving from a sequence model to complex dependency is straight forward

- Higher order Features
- Partition function

$$P(\mathbf{y}|\mathbf{x}) = \frac{\exp\left(\mathbf{w}^T\phi(\mathbf{x},\mathbf{y})\right)}{\sum_{\hat{\mathbf{y}}}\exp\left(\mathbf{w}^T\phi(\mathbf{x},\hat{\mathbf{y}})\right)}$$

$$\Phi(\mathbf{x},\mathbf{y}) = \Phi(x_1, y_1) + \Phi(y_1, y_2, y_3) + \Phi(x_3, y_2, y_3) + \Phi(x_1, x_2, y_2)$$

Structured Perceptron with averaging

```
Given a training set D = {(x,y)}
1. Initialize  $\mathbf{w} \in \mathbf{R}^n$,  $\mathbf{a} \in \mathbf{R}^n$
2. For epoch = 1 … T:
    1. For each training example $(\mathbf{x}, \mathbf{y}) \in$ D:
        1. Predict $\mathbf{y}' = \text{argmax}_{y'}\, \mathbf{w}^T\Phi(\mathbf{x}, \mathbf{y}')$
        2. If $\mathbf{y} \neq \mathbf{y}'$, update $\mathbf{w} \leftarrow \mathbf{w} + \text{learningRate}\,(\Phi(\mathbf{x},\mathbf{y}) - \Phi(\mathbf{x},\mathbf{y}'))$
        3. Set $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{w}$
3. Return a/ (nT)
```

The same algorithm can still be used, we only need to **adapt our inference procedure** (*argmax*) and use **different feature functions**

→ *Introduces a computational issue!*

Inference: take 2

- So far we have only seen the Viterbi algorithm
  - Solve a combinatorial optimization problem, by making a strong assumption – sequence models.
- What happens if our problem requires more complex inference:
  - Parse trees?
  - Segmentation?

Dynamic Programming

- **You actually already know the answer (hint: CS 580)**
- General strategy, used by many combinatorial optimization algorithms
  - Viterbi : sequences
  - CYK: Parse trees
  - Max-Spanning tree: Dependency parse
  - Min-cut/max-flow: segmentation
  - Edit distance: string alignment

Inference as Search

It's easy to think about inference as a search problem

- Similar to most of AI..

A search problem is defined by –

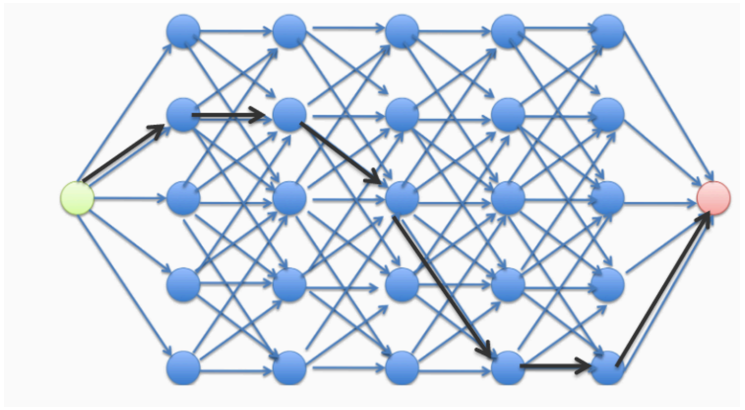
- State (partial assignment of the structure)
- State transitions
- State state/End state
- Scoring function over states

The search defines the highest scoring path from start to end ➔ solves the argmax problem!

this is where learning comes in!

Inference as Search

Viterbi can be though of as a search problem



This is a version of **exact search**

Instead, we can also run cheaper greedy search

- At each step, take the highest scoring transition
- **Is this a problem?**

Greedy algorithms are sometimes optimal!

- Sub-modular functions

Inference as Search

Beam search: mid-point between exact and greedy

Keep a priority queue of size k, *known as* ***the beam***
At each level only explore k next states

If k = infinity: this is just BFS
Otherwise: greedy search over the top k states

Very popular!

Integer Linear Programming

An **expensive**, **declarative** alternative to search algorithms.
Explicitly states the objective of the search.
General form:

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0. \end{aligned}$$

Solution (assignment to x) has to be an integer!

We will look at a subset, 0-1 ILP

Integer Linear Programming

The CEO problem:

We have 8 short wood pieces, and 6 long wood pieces
Table requires 2 long pieces, 2 short pieces
Chair requires 1 long pieces, 2 short pieces
We can sell tables for $20, chairs for $15

We want to maximize our profits!

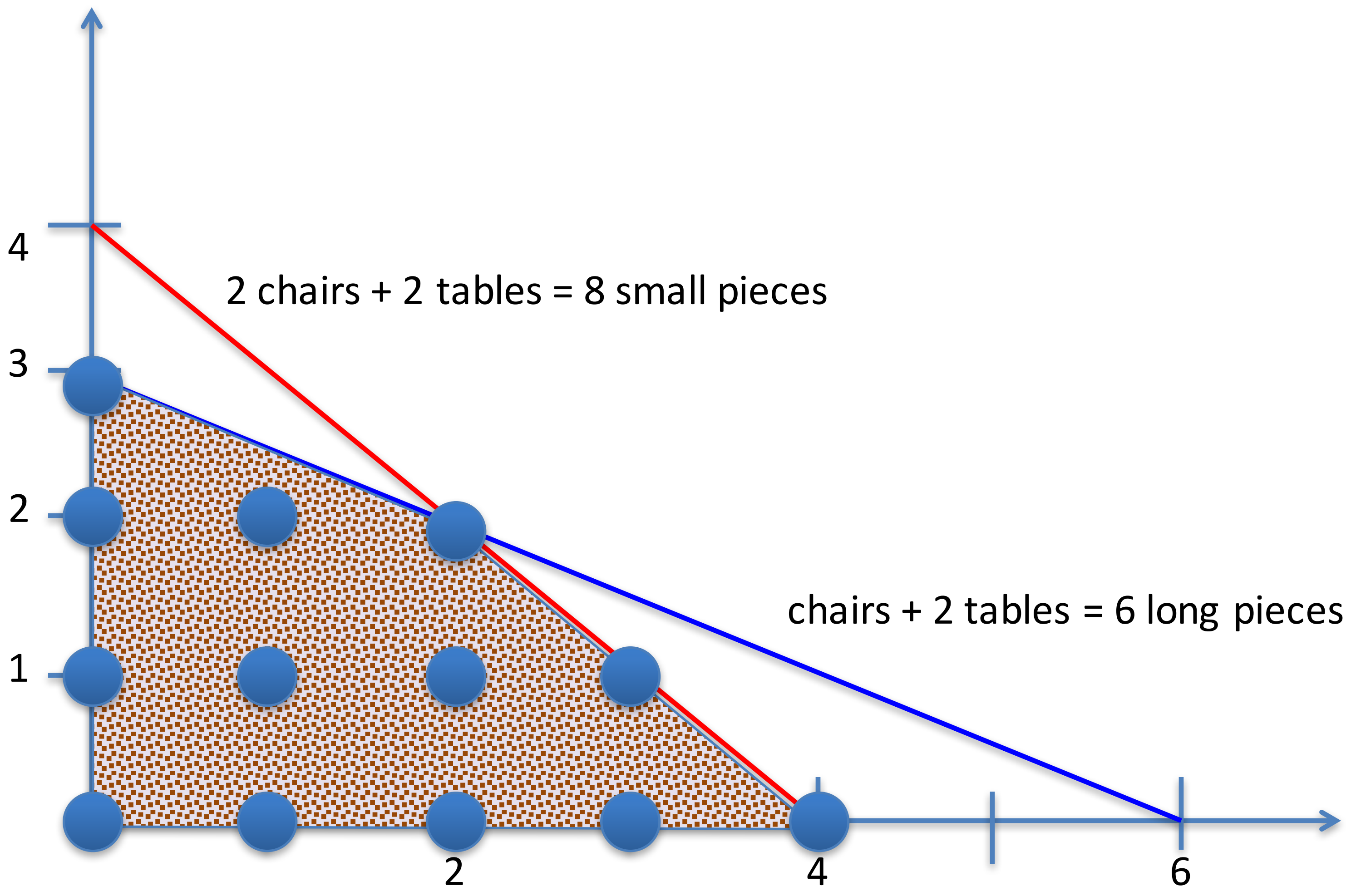
```
Max 15 * chairs + 20 * tables

Subject to

Long pieces:  chairs + 2 tables ≤ 6
Short pieces:  2 chairs + 2 tables ≤ 8

(chairs≥0, tables≥0)
```

Integer Linear Programming


4
3
2
1
2 chairs + 2 tables = 8 small pieces
chairs + 2 tables = 6 long pieces
2
4
6

How can we use ILP for inference?

ILP is a very convenient way to express many combinatorial optimization problems!

Let's start with an easy example: **multi-class classification**

- z_A = 1 if output = A, 0 otherwise
- z_B = 1 if output = B, 0 otherwise
- z_C = 1 if output = C, 0 otherwise

$$\max_z z_A \cdot c(A) + z_B \cdot c(B) + z_C \cdot c(C) \qquad (maximize\ the\ score)$$

$$s.t.$$

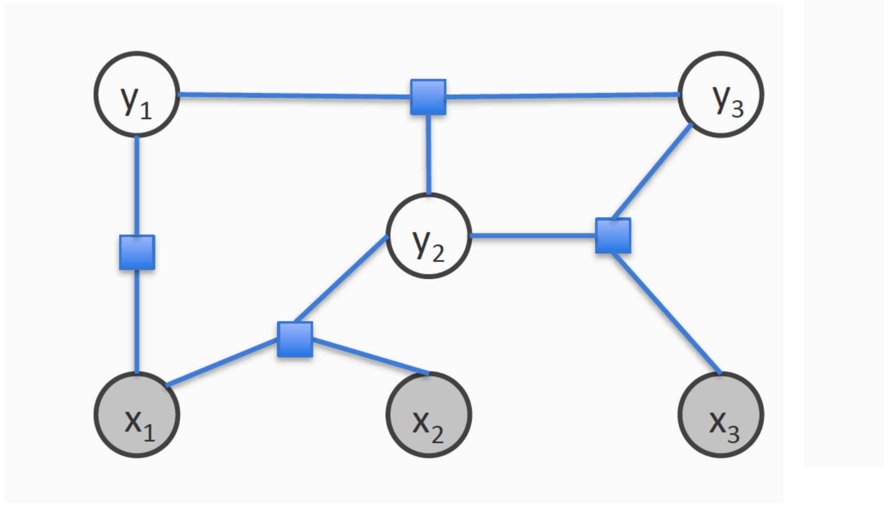
$$z_A, z_B, z_C \in \{0, 1\}$$

$$z_A + z_B + z_C = 1 \qquad (only\ a\ single\ label\ can\ be\ active)$$

How can we use ILP for inference?

we assign a decision variable for each factor

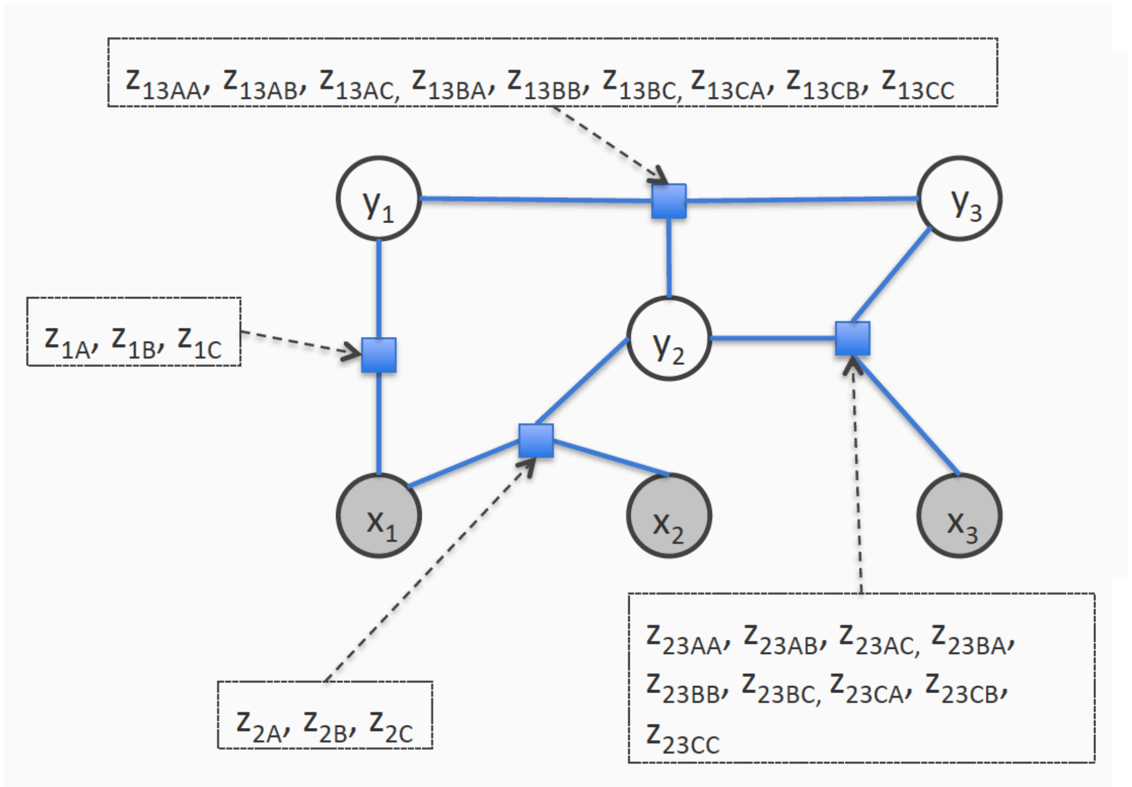
$$\max_y \mathrm{w}^T\phi(x_1, y_1) + \mathrm{w}^T\phi(y_1, y_2, y_3) + \mathrm{w}^T\phi(x_3, y_2, y_3) + \mathrm{w}^T\phi(x_1, x_2, y_2)$$



How can we use ILP for inference?

we assign a decision variable for each factor

$$\max_y \mathrm{w}^T\phi(x_1, y_1) + \mathrm{w}^T\phi(y_1, y_2, y_3) + \mathrm{w}^T\phi(x_3, y_2, y_3) + \mathrm{w}^T\phi(x_1, x_2, y_2)$$


$z_{13AA}, z_{13AB}, z_{13AC}, z_{13BA}, z_{13BB}, z_{13BC}, z_{13CA}, z_{13CB}, z_{13CC}$
y_1
y_3
z_{1A}, z_{1B}, z_{1C}
y_2
x_1
x_2
x_3
z_{2A}, z_{2B}, z_{2C}
$z_{23AA}, z_{23AB}, z_{23AC}, z_{23BA}, z_{23BB}, z_{23BC}, z_{23CA}, z_{23CB}, z_{23CC}$

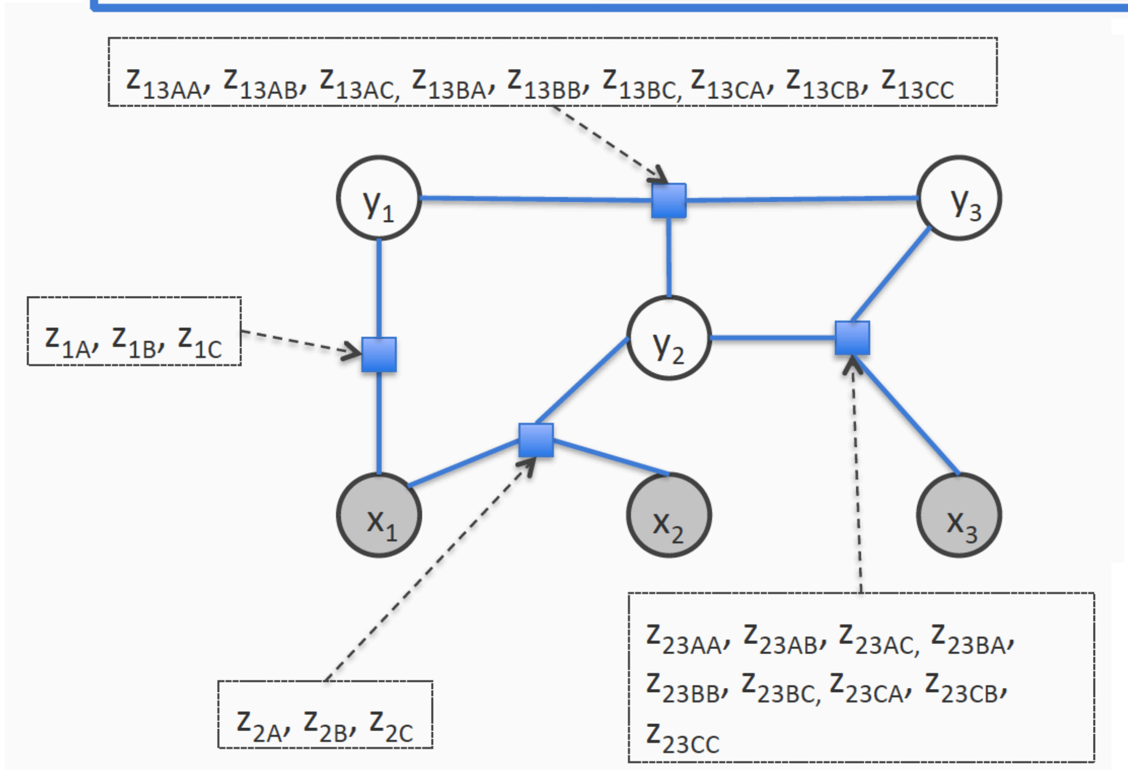
Slides adapted from V. Srikumar slides

How can we use ILP for inference?

we assign a decision variable for each factor

$$\max_{\mathbf{z}} \quad \textstyle\sum_l z_{1l}s_{1l} + \sum_l z_{2l}s_{2l} + \sum_{l,l'} z_{13ll'}s_{13ll'} \sum_{l,l'} z_{23ll'}s_{23ll'}$$

$$\text{s.t} \quad \boxed{\text{Only valid output allowed}}$$



$z_{13AA}, z_{13AB}, z_{13AC}, z_{13BA}, z_{13BB}, z_{13BC}, z_{13CA}, z_{13CB}, z_{13CC}$
y_1
y_3
z_{1A}, z_{1B}, z_{1C}
y_2
x_1
x_2
x_3
$z_{23AA}, z_{23AB}, z_{23AC}, z_{23BA}, z_{23BB}, z_{23BC}, z_{23CA}, z_{23CB}, z_{23CC}$
z_{2A}, z_{2B}, z_{2C}

z_{13AB} implies $z_{1A} \wedge z_{3B}$

Slides adapted from V. Srikumar slides

Adding Constraints

Boolean formulas can be converted into linear constraints

One out of $z_1,..z_k$

$z_1+..+z_k = 1$

At least m out $z_1,..z_k$

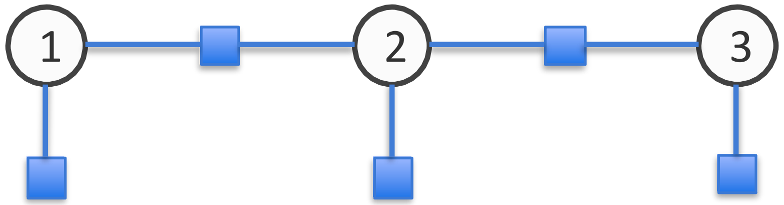
$z_1+..+z_k \geq m$

Implication: $z_1 \rightarrow z_k$

$z_k \geq z_1$

Let’s practice!

How would you write a sequence labeling problem as ILP instance?



We presented a nice "map" for structured prediction:

Generative vs Discriminative

Local vs global

Binary < multiclass < sequence < **graph**

Now, given a new problem you have to decide:

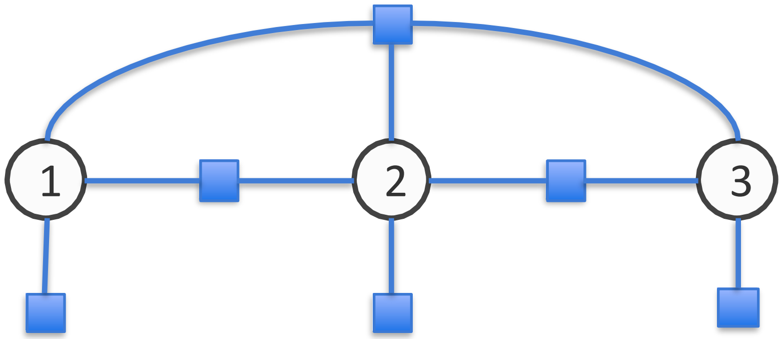
output decomposition, training regime and learning algorithm

Is the distinction really that clear cut?

Consistency of outputs

*Or: **How to introduce knowledge into prediction***

Suppose we have a sequence labeling problem where the outputs can be one of A or B We want to add a condition: **There should be no more than one B in the output**



Here is a simple way to capture this dependency

y1	y2	y3	f
A	A	A	0
A	A	B	0
A	B	A	0
A	B	B	-1
B	A	A	0
B	A	B	-1
B	B	A	-1
B	B	B	-1

But the standard CRF learning does not allow for potential functions to be set manually

Should we learn what we can write down easily?

Especially for large, computationally cumbersome factors

Another look at learning and inference

Learning a global model encoding our knowledge about the problem

- Complex model (many more features)
- Computationally intractable
- Recall: Inference-based training is costly!

Alternatively, **Local learning**:

- *Learn local decisions independently*
- *piece them together* ***legally***

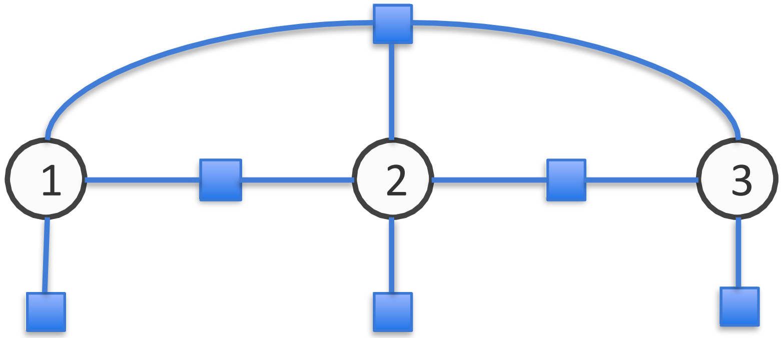
Training: local models

Prediction: inference can still be global

Combining local classifiers

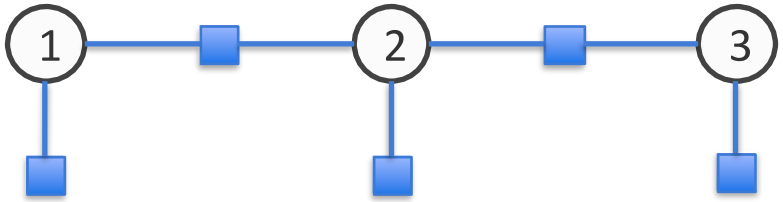
Knowledge: *no more than one 'B' in the output*

Option 1: learn a global model, penalizing such assignments



$$\arg\max_{\mathbf{y}} \quad \text{score}(\mathbf{x}, \mathbf{y})$$

- **Option 2: Inference can "glue" together local decisions**
  - And enforce *global* coherence (*constrainted* optimization)



$$\arg\max_{\mathbf{y}} \quad \text{score}(\mathbf{x}, \mathbf{y})$$

s.t. There is no more than one B in the output

Constrained Conditional Model

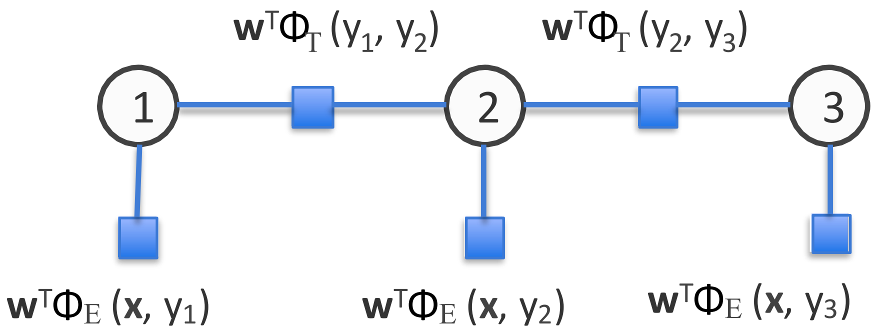
Inference consists of two components

1. **Local classifiers** (may be a collection of structures themselves)
   - These are trained models

2. A set of **constraints** that restrict the space of joint assignments of the local classifiers

Where do constraints come from?

Prediction in a CCM



$\mathbf{w}^T\Phi_T(y_1, y_2)$
$\mathbf{w}^T\Phi_T(y_2, y_3)$
1
2
3
$\mathbf{w}^T\Phi_E(\mathbf{x}, y_1)$
$\mathbf{w}^T\Phi_E(\mathbf{x}, y_2)$
$\mathbf{w}^T\Phi_E(\mathbf{x}, y_3)$

Emission scores

Transition scores

$$\arg\max_y \left(\sum_i \sum_{l \in \{A,B,C\}} I_{y_i = l} \cdot c(x, l) + \sum_i \sum_{l_1, l_2 \in \{A,B,C\}} I_{y_i = l_1 \wedge y_i = l_2} \cdot c(x, l_1, l_2) \right)$$

Indicator functions

I_z = 1 if z is true, 0 otherwise

One indicator per factor

One score per indicator

Prediction

Global Prediction Objective:

$$\arg\max_{y} \left(\sum_{i} \sum_{l \in \{A,B,C\}} I_{y_i = l} \cdot c(x, l) + \sum_{i} \sum_{l_1, l_2 \in \{A,B,C\}} I_{y_i = l_1 \wedge y_i = l_2} \cdot c(x, l_1, l_2) \right)$$

Constraint according to -

- Each y_i can either be a A,B,C label
- **At most one 'B' in the output**

$$I_{y_1=B} + I_{y_2=B} + I_{y_3=B} \leq 1$$

Inference with hard constraints

$$\arg\max_{y} \sum_{p \in Parts} \sum_{l \in Labels} \underset{\uparrow\ \text{Indicator variables}}{I_{y_{p=l}}} \cdot c(x, l)$$

Constraints can be written over the indicator variables

The result is an ILP instance, which can be solved using an ILP solver,
If our problem is too big (ILP is NP-Complete), we can use approximate methods (LP, beam search)

Question: how would you change beam search to avoid illegal states?

CCM with *soft constraints*

Can we replace the hard constraints with soft?

Why would we want to do it?
What will need to change?

Option 1: set a fixed high cost for breaking the constraint

Option 2: optimize the constraint violation costs

CCM with *soft constraints*

Question:

Assuming we take option-2, and try to optimize the cost of constraint violation.

What is the difference between option-2 and the global model with long range dependencies we started with?

Summary

Global joint inference

Output structures decomposes into factors/parts

Parts can be scored independently ***BUT***

Their assignment is interdependent! ***Encode using constraints***

Constraints encode your knowledge about the domain

“knowledge injection”

Intuition: *no need to learn what you already know*

Question: Can constraints replace data?