Machine Learning 2

DS 4420 - Spring 2020

Structured prediction, I

Byron C Wallace

Today

- So far: focus on *unsupervised* methods, where we have typically been interested in associated instances with single components (or "labels")

- We'll switch gears a bit today and consider *structured* spaces where an instance is associated with multiple labels

Today

- So far: focus on *unsupervised* methods, where we have typically been interested in associated instances with single components (or "labels")
- We'll switch gears a bit today and consider *structured* spaces where an instance is associated with multiple labels
- Material today based (mostly) on

Daumé, CIML reading

Structured output spaces

So far have assumed it is reasonable to assume:

$$(x_i, y_i)$$

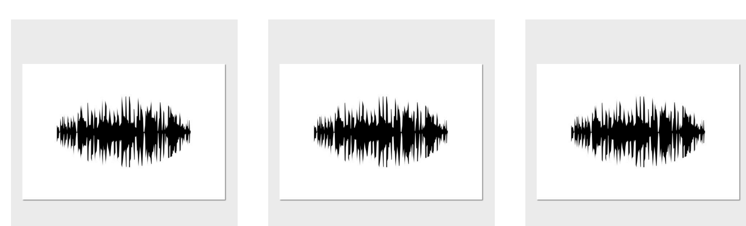
Structured output spaces

So far have assumed it is reasonable to assume:

$$(x_i, y_i)$$

Consider *speech transcription*

$$y_1 \quad y_2 \quad y_3$$



$$x_1 \quad x_2 \quad x_3$$

Structured output spaces

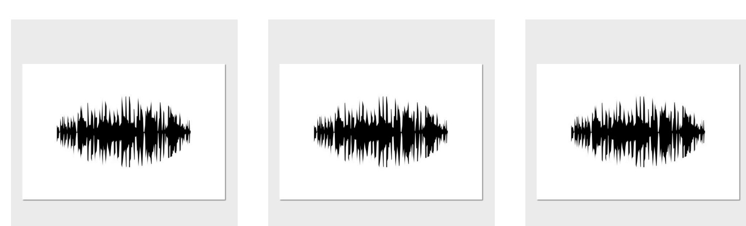
So far have assumed it is reasonable to assume:

$$(x_i, y_i)$$

Consider *speech transcription*

y_1 y_2 y_3

“*Play*



x_1 x_2 x_3

Structured output spaces

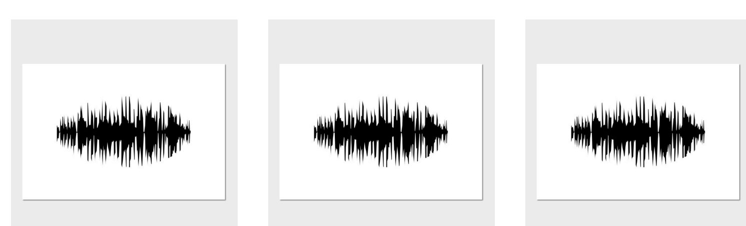
So far have assumed it is reasonable to assume:

$$(x_i, y_i)$$

Consider *speech transcription*

y_1 y_2 y_3

"*Play Kanye*



x_1 x_2 x_3

Structured output spaces

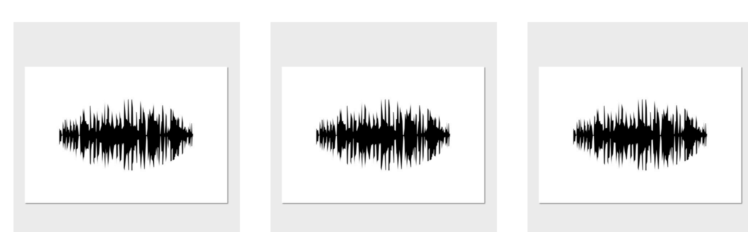
So far have assumed it is reasonable to assume:

$$(x_i, y_i)$$

Consider *speech transcription*

$y_1 \quad y_2 \quad y_3$

"*Play Kanye West*"



$x_1 \quad x_2 \quad x_3$

Structured output spaces

Source: http://cocodataset.org/

Structured output spaces

John	lives	in	New	York	and	works	for	the	European	Union
B-PER	O	O	B-LOC	I-LOC	O	O	O	O	B-ORG	I-ORG

Source: https://guillaumegenthial.github.io/sequence-tagging-with-tensorflow.html

Structured output spaces

y is now a vector (or *tensor)*

Structured output spaces

y is now a vector (or *tensor)*

We will generally be interested in scoring pairs

$$(x, \hat{y})$$

Structured output spaces

y is now a vector (or *tensor)*

We will generally be interested in scoring pairs

$$(x, \hat{y})$$

Often called **sequence labeling**

Space of problems

Given	**Predict**	**Type?**
An image	Contains a cat?	

Space of problems

Given	Predict	Type?
An image	Contains a cat?	**Classification**

Space of problems

Given	Predict	Type?
An image	Contains a cat?	**Classification**
An image	Coordinates that outline all cats	

Space of problems

Given	**Predict**	**Type?**
An image	Contains a cat?	**Classification**
An image	Coordinates that outline all cats	**Structured prediction**

Space of problems

Given	**Predict**	**Type?**
An image	Contains a cat?	**Classification**
An image	Coordinates that outline all cats	**Structured prediction**
A tweet	Names in the tweet	

Space of problems

Given	Predict	Type?
An image	Contains a cat?	**Classification**
An image	Coordinates that outline all cats	**Structured prediction**
A tweet	Names in the tweet	**Structured prediction**

Space of problems

Given	Predict	Type?
An image	Contains a cat?	**Classification**
An image	Coordinates that outline all cats	**Structured prediction**
A tweet	Names in the tweet	**Structured prediction**
A tweet	Sentiment in tweet	

Space of problems

Given	Predict	Type?
An image	Contains a cat?	Classification
An image	Coordinates that outline all cats	Structured prediction
A tweet	Names in the tweet	Structured prediction
A tweet	Sentiment in tweet	Classification

Models

- Structured perceptron
- Hidden Markov Models (HMMs)
- Conditional Random Fields (CRFs)

The *structured* perceptron

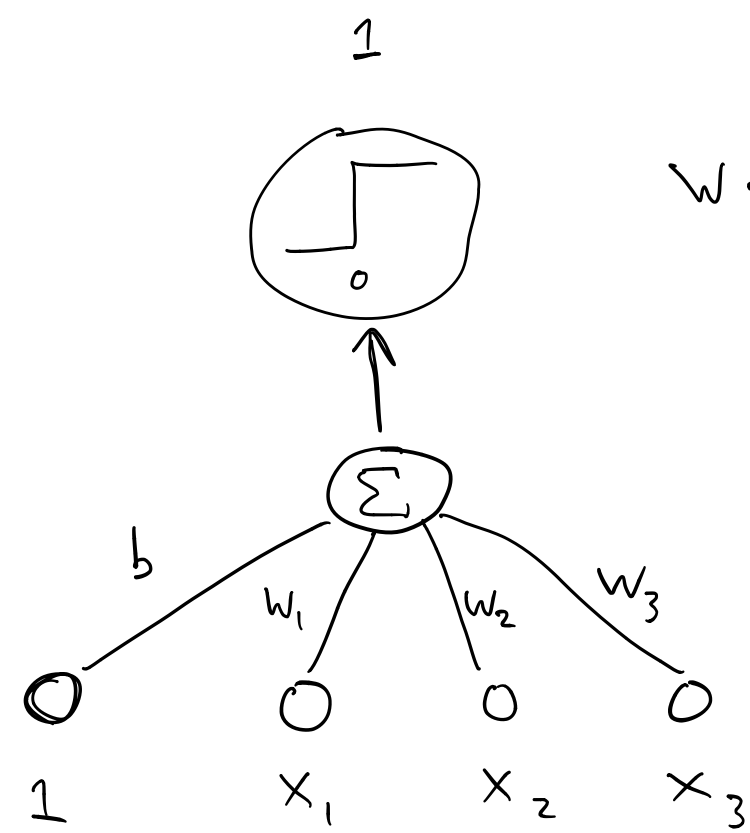
The *structured* perceptron

Will build up to this by first introducing the *multi-class* perceptron.

The Perceptron

Perceptron

$$\hat{y} = \begin{cases} 1 & \text{if } w \cdot x > 0 \\ -1 & \text{otherwise} \end{cases}$$



$$W \cdot X = 1 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

Multiclass perceptron

Assume $y \in \{1, 2, ..., K\}$

Joint feature vector $\phi(x, y)$

We're after a *scoring function*

$$s(x, y) = w \cdot \phi(x, y)$$

Multiclass perceptron

Assume $y \in \{1, 2, ..., K\}$

Joint feature vector $\phi(x, y)$

We're after a *scoring function*

$$s(x, y) = w \cdot \phi(x, y)$$

should be high for correct y

Multiclass perceptron

How should we design $\phi(x, y)$?

One option:

$$\phi(\boldsymbol{x}, k) = \Big\langle \underbrace{0, 0, \ldots, 0}_{D(k-1)\text{ zeros}}, \underbrace{\boldsymbol{x}}_{\in \mathbb{R}^D}, \underbrace{0, 0, \ldots, 0}_{D(K-k)\text{ zeros}} \Big\rangle \in \mathbb{R}^{DK}$$

Learning

Consider making a prediction, given $\boldsymbol{w}$

$$\hat{y} = \operatorname*{argmax}_{\hat{y} \in [1, K]} s(\boldsymbol{x}, \hat{y})$$

Learning

Consider making a prediction, given $\boldsymbol{w}$

$$\hat{y} = \operatorname*{argmax}_{\hat{y} \in [1,K]} s(\boldsymbol{x}, \hat{y})$$

$$= \operatorname*{argmax}_{\hat{y} \in [1,K]} \boldsymbol{w} \cdot \phi(\boldsymbol{x}, \hat{y})$$

Learning

If $\hat{y} = y$ we do nothing

If we are wrong ($\hat{y} \neq y$) then update

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \phi(\boldsymbol{x}, y) - \phi(\boldsymbol{x}, \hat{y})$$

Learning

If $\hat{y} = y$ we do nothing

If we are wrong ($\hat{y} \neq y$) then update

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \phi(\boldsymbol{x}, y) - \phi(\boldsymbol{x}, \hat{y})$$

features for true label

features for wrong label

Learning

Is this doing what we want?

$$\boldsymbol{w}^{\text{(new)}} \leftarrow \boldsymbol{w} + \phi(\boldsymbol{x}, y) - \phi(\boldsymbol{x}, \hat{y})$$

Learning

Is this doing what we want?

$$\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} + \phi(\boldsymbol{x}, y) - \phi(\boldsymbol{x}, \hat{y})$$

Consider updated prediction for true label

$$\boldsymbol{w}^{(\text{new})} \cdot \phi(\boldsymbol{x}, y)$$

Learning

Is this doing what we want?

$$\boldsymbol{w}^{\text{(new)}} \leftarrow \boldsymbol{w} + \phi(\boldsymbol{x}, y) - \phi(\boldsymbol{x}, \hat{y})$$

Consider updated prediction for true label

$$\begin{aligned}
&\boldsymbol{w}^{\text{(new)}} \cdot \phi(\boldsymbol{x}, y) \\
&= \left(\boldsymbol{w}^{\text{(old)}} + \phi(\boldsymbol{x}, y) - \phi(\boldsymbol{x}, \hat{y})\right) \cdot \phi(\boldsymbol{x}, y)
\end{aligned}$$

Learning

Is this doing what we want?

$$\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} + \phi(\boldsymbol{x}, y) - \phi(\boldsymbol{x}, \hat{y})$$

Consider updated prediction for true label

$$\begin{aligned}
&\boldsymbol{w}^{(\text{new})} \cdot \phi(\boldsymbol{x}, y) \\
&= \left(\boldsymbol{w}^{(\text{old})} + \phi(\boldsymbol{x}, y) - \phi(\boldsymbol{x}, \hat{y})\right) \cdot \phi(\boldsymbol{x}, y) \\
&= \underbrace{\boldsymbol{w}^{(\text{old})} \cdot \phi(\boldsymbol{x}, y)}_{\text{old prediction}} + \underbrace{\phi(\boldsymbol{x}, y) \cdot \phi(\boldsymbol{x}, y)}_{\geq 0} - \underbrace{\phi(\boldsymbol{x}, \hat{y}) \cdot \phi(\boldsymbol{x}, y)}_{=0}
\end{aligned}$$

(by construction)

Sharing features

Suppose there are three classes: *music*, *movies,* and *oncology*.

We think the first two are probably more similar.

We can **encode** this in the feature space.

Sharing features

Suppose there are three classes: *music*, *movies*, and *oncology*.

We think the first two are probably more similar.

We can **encode** this in the feature space.

$$\phi(\boldsymbol{x}, \text{music}) = \langle \boldsymbol{x}, \mathbf{0}, \mathbf{0}, \boldsymbol{x} \rangle$$

$$\phi(\boldsymbol{x}, \text{movies}) = \langle \mathbf{0}, \boldsymbol{x}, \mathbf{0}, \boldsymbol{x} \rangle$$

$$\phi(\boldsymbol{x}, \text{oncology}) = \langle \mathbf{0}, \mathbf{0}, \boldsymbol{x}, \mathbf{0} \rangle$$

Sharing features

Suppose there are three classes: *music*, *movies*, and *oncology*.

We think the first two are probably more similar.

We can **encode** this in the feature space.

$$\phi(\boldsymbol{x}, \text{music}) = \langle \boldsymbol{x}, \mathbf{0}, \mathbf{0}, \boldsymbol{x} \rangle$$

$$\phi(\boldsymbol{x}, \text{movies}) = \langle \mathbf{0}, \boldsymbol{x}, \mathbf{0}, \boldsymbol{x} \rangle$$

$$\phi(\boldsymbol{x}, \text{oncology}) = \langle \mathbf{0}, \mathbf{0}, \boldsymbol{x}, \mathbf{0} \rangle$$

"extra" copy; allows ***w*** to capture shared aspects of movies/music

Structured perceptron

Let's come back to our motivation of *structured* outputs

To be concrete, we will focus on *sequence labeling*

$$\boldsymbol{x} = \text{“ monsters eat tasty bunnies “}$$

$$\boldsymbol{y} = \quad \text{noun} \;\; \text{verb adj} \quad \text{noun}$$

Structured perceptron

Let's come back to our motivation of *structured* outputs

To be concrete, we will focus on *sequence labeling*

$$\boldsymbol{x} = \text{“ monsters eat tasty bunnies “}$$

$$\boldsymbol{y} = \quad \text{noun} \quad \text{verb} \quad \text{adj} \quad \text{noun}$$

So *y* is now a sequence. But goal is the same; Score true *y* (whole sequence) higher than other potential *y*'s.

How big is our output space?

Assume x has length L, and that there are K possible labels at each position

How big is our output space?

Assume *x* has length *L*, and that there are *K* possible labels at each position

$$\mathcal{Y} = K^L$$

Designing features

$\boldsymbol{x} =$ " monsters eat tasty bunnies "

$\boldsymbol{y} =$ noun verb adj noun

Want to design $\phi(x, y)$

Designing features

$$\boldsymbol{x} = \text{“ monsters eat tasty bunnies “}$$

$$\boldsymbol{y} = \quad \text{noun} \quad \text{verb} \quad \text{adj} \quad \text{noun}$$

Want to design $\phi(x, y)$

Some possibilities

- # of times *w* gets label *l* (for all *w*, *l*)
- # of times *l* is adjacent to *l'* (for all *l* and *l'*)

Designing features

$\boldsymbol{x} =$ " monsters eat tasty bunnies "

$\boldsymbol{y} =$ noun verb adj noun

Want to design $\phi(x, y)$

Some possibilities

- # of times *w* gets label *l* (for all *w*, *l*) **Unary**
- # of times *l* is adjacent to *l'* (for all *l* and *l'*)

Designing features

$$\boldsymbol{x} = \text{“ monsters eat tasty bunnies “}$$

$$\boldsymbol{y} = \quad \text{noun} \;\; \text{verb} \;\; \text{adj} \quad \text{noun}$$

Want to design $\phi(x, y)$

Some possibilities

- # of times *w* gets label *l* (for all *w*, *l*) **Unary**
- # of times *l* is adjacent to *l'* (for all *l* and *l'*) **Markov**

Algorithm 40 StructuredPerceptronTrain($\mathbf{D}$, *MaxIter*)

```
w ← 0                                                    // initialize weights
for iter = 1 ... MaxIter do
    for all (x,y) ∈ D do
        ŷ ← argmax_{ŷ∈𝒴(x)} w · φ(x, ŷ)                  // compute prediction
        if ŷ ≠ y then
            w ← w + φ(x, y) − φ(x, ŷ)                    // update weights
        end if
    end for
end for
return w                                                 // return learned weights
```

Algorithm 40 StructuredPerceptronTrain($\mathbf{D}$, *MaxIter*)

$w \leftarrow \mathbf{0}$ // initialize weights
for *iter* = 1 … *MaxIter* **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $\boxed{\hat{y} \leftarrow \operatorname{argmax}_{\hat{y} \in \mathcal{Y}(x)} w \cdot \phi(x, \hat{y})}$ // compute prediction
    if $\hat{y} \neq y$ **then**
      $w \leftarrow w + \phi(x, y) - \phi(x, \hat{y})$ // update weights
    end if
  end for
end for
return w // return learned weights

argmax problem

$$\hat{\boldsymbol{y}} \leftarrow \operatorname*{argmax}_{\hat{\boldsymbol{y}} \in \mathcal{Y}(\boldsymbol{x})} \boldsymbol{w} \cdot \phi(\boldsymbol{x}, \hat{\boldsymbol{y}})$$

Why is this hard?

Decomposing structure

Problem We want to compute an argmax over ridiculously large set of elements

Key idea If we restrict ourselves to "local" features, we can decompose ϕ over the input

Decomposing structure

$$\boldsymbol{w} \cdot \phi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{w} \cdot \sum_{l=1}^{L} \phi_l(\boldsymbol{x}, \boldsymbol{y}) \qquad \text{decomposition of structure}$$

$$= \sum_{l=1}^{L} \boldsymbol{w} \cdot \phi_l(\boldsymbol{x}, \boldsymbol{y}) \qquad \text{associative law}$$

$\phi_l(x, y)$ encodes only features about position *l*

Decomposing structure

$$\boldsymbol{w} \cdot \phi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{w} \cdot \sum_{l=1}^{L} \phi_l(\boldsymbol{x}, \boldsymbol{y})$$

decomposition of structure

$$= \sum_{l=1}^{L} \boldsymbol{w} \cdot \phi_l(\boldsymbol{x}, \boldsymbol{y})$$

associative law

$\phi_l(x, y)$ encodes only features about position *l*

Markov

The Viterbi algorithm
(on board)

Face extraction

- Suppose you are given a dataset of 28 x 28 images containing within them faces

- In the train data, these have been labeled at the *pixel* level

Face extraction

- Suppose you are given a dataset of 28 x 28 images containing within them faces
- In the train data, these have been labeled at the *pixel* level

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]]
```

X → Y

Face extraction

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.]]
```

X $\longrightarrow$ Y

How would you design $\phi_l(x, y)$?

How would you construct your lattice?

A probabilistic view on structured learning

Some content that follows derived from
Michael Collins' materials

A probabilistic view on structured learning

- **Perceptrons** lack any probabilistic semantics
- We can introduce these for structured problems of course!

Hidden Markov Models (HMMs)

- A fully generative model that explicitly bakes in Markov assumption

Hidden Markov Models (HMMs)

- A fully generative model that explicitly bakes in Markov assumption

- Recall in Naive Bayes we had:

$$p(y|x) = \frac{p(y)p(x|y)}{p(x)}$$

Hidden Markov Models (HMMs)

- A fully generative model that explicitly bakes in Markov assumption

- Recall in Naive Bayes we had:

$$p(y|x) = \frac{p(y)p(x|y)}{p(x)}$$

Modeling Sequences

Want:

$$P(X_1 = x_1 \dots X_n = x_n, Y_1 = y_1 \dots Y_n = y_n)$$

Modeling Sequences

$$P(X_1 = x_1 \ldots X_n = x_n, Y_1 = y_1 \ldots Y_n = y_n)$$
$$= \prod_{i=1}^{n+1} P(y_i | y_{i-1}) \prod_{i=1}^{n} P(x_i | y_i)$$

Modeling Sequences

$$P(X_1 = x_1 \ldots X_n = x_n, Y_1 = y_1 \ldots Y_n = y_n)$$
$$= \prod_{i=1}^{n+1} P(y_i | y_{i-1}) \prod_{i=1}^{n} P(x_i | y_i)$$

Transition probability

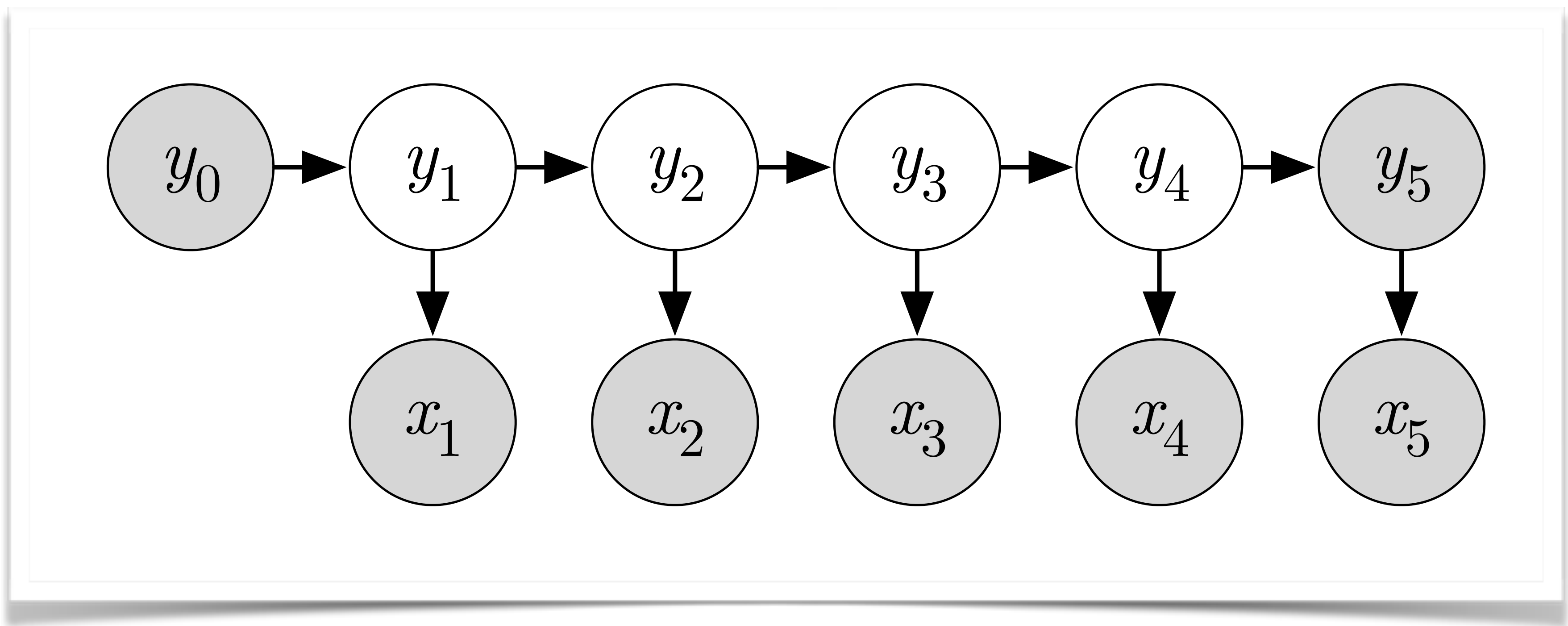
Modeling Sequences

$$P(X_1 = x_1 \ldots X_n = x_n, Y_1 = y_1 \ldots Y_n = y_n)$$
$$= \prod_{i=1}^{n+1} P(y_i | y_{i-1}) \prod_{i=1}^{n} P(x_i | y_i)$$

Transition probability

Emission probability

Graphical Model (HMMs)



Consider a “Tri-gram” variant

$$P(Y_i = y_i | Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) = q(y_i | y_{i-2}, y_{i-1})$$

Transition probability

$$P(X_i = x_i | Y_i = y_i) = e(x_i | y_i)$$

Emission probability

How should we estimate our parameters?

Maximum Likelihood Estimates

Maximum Likelihood Estimates

$$q(s|u,v) = \frac{c(u,v,s)}{c(u,v)}$$

Maximum Likelihood Estimates

$$q(s|u,v) = \frac{c(u,v,s)}{c(u,v)}$$

$$e(x|s) = \frac{c(s \rightsquigarrow x)}{c(s)}$$

Maximum Likelihood Estimates

$$q(s|u,v) = \frac{c(u,v,s)}{c(u,v)}$$

$$e(x|s) = \frac{c(s \rightsquigarrow x)}{c(s)}$$

$$\boxed{\text{i.e., counts!}}$$

What about *decoding*?

$$\arg\max_{y_1 \ldots y_{n+1}} p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$$

What about *decoding*?

$$\arg\max_{y_1 \cdots y_{n+1}} p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$$

Viterbi!

Consider: How might we do *unsupervised* or *semi-supervised* learning in HMMs?

Summary

Structured problems are those in which *y*'s are correlated

- Image segmentation
- Language modeling
- Credit fraud detection
- Any time we have *sequences*

Learning to recognize a structured problem when you see them is important!