Learning for Structured Prediction

Linear Methods For Sequence Labeling: Hidden Markov Models vs Structured Perceptron

Ivan Titov

$\boldsymbol{x}$ is an input (e.g., sentence), **$\boldsymbol{y}$** is an output (syntactic tree)

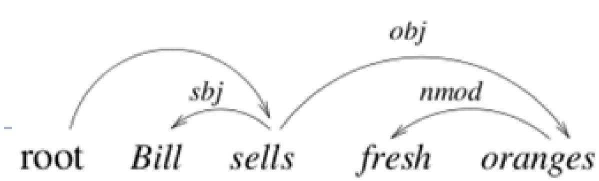
Last Time: Structured Prediction

1. Selecting feature representation $\varphi(\boldsymbol{x}, \boldsymbol{y})$
   - It should be sufficient to discriminate correct structure from incorrect ones
   - It should be possible to decode with it (see (3))
2. Learning
   - Which error function to optimize on the training set, for example

     $$w \cdot \varphi(\boldsymbol{x}, \boldsymbol{y}^\star) - \max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x}), \boldsymbol{y} \neq \boldsymbol{y}^\star} w \cdot \varphi(\boldsymbol{x}, \boldsymbol{y}') > \gamma$$

   - How to make it efficient (see (3))
3. Decoding: $\boldsymbol{y} = \mathrm{argmax}_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} w \cdot \varphi(\boldsymbol{x}, \boldsymbol{y}')$
   - Dynamic programming for simpler representations φ ?
   - Approximate search for more powerful ones?

We illustrated all these challenges on the example of dependency parsing



Outline

- Sequence labeling / segmentation problems: settings and example problems:
  - Part-of-speech tagging, named entity recognition, gesture recognition
- Hidden Markov Model
  - Standard definition + maximum likelihood estimation
  - General views: as a representative of linear models
- Perceptron and Structured Perceptron
  - algorithms / motivations
- Decoding with the Linear Model
- Discussion: Discriminative vs. Generative

Sequence Labeling Problems

- Definition:
  - Input: sequences of variable length $\boldsymbol{x} = (x_1, x_2, \ldots, x_{|\boldsymbol{x}|}),\ x_i \in X$
  - Output: every position is labeled $\boldsymbol{y} = (y_1, y_2, \ldots, y_{|\boldsymbol{x}|}),\ y_i \in Y$
- Examples:
  - Part-of-speech tagging

$\boldsymbol{x} =$	**John**	**carried**	**a**	**tin**	**can**	**.**
$\boldsymbol{y} =$	NP	VBD	DT	NN	NN	.

  - Named-entity recognition, shallow parsing ("chunking"), gesture recognition from video-streams, …

Part-of-speech tagging

$x =$	**John**	**carried**	**a**	**tin**	**can**	**.**
$y =$	NNP	VBD	DT	NN	NN or MD?	.

In fact, even knowing that the previous word is a noun is not enough

he most
word you
ke here

- Labels:
  - NNP – proper singular noun;
  - VBD - verb, past tense
  - DT - determin
  - NN – singular noun
  - MD - modal

One need to model interactions between labels to successfully resolve ambiguities, so this should be tackled as a structured prediction problem

- Consider

$x =$	**Tin**	**can**	**cause**	**poisoning**	**…**
$y =$	NN	MD	VB	NN	…

Named Entity Recognition

[ORG Chelsea], despite their name, are not based in [LOC Chelsea], but in neighbouring [LOC Fulham] .

- Not as trivial as it may seem, consider:
  - [PERS Bill Clinton] will not embarrass [PERS Chelsea] at her wedding
  - Tiger failed to make a birdie in the South Course …
- Encoding example (BIO-encoding)

Chelsea can be a person too!

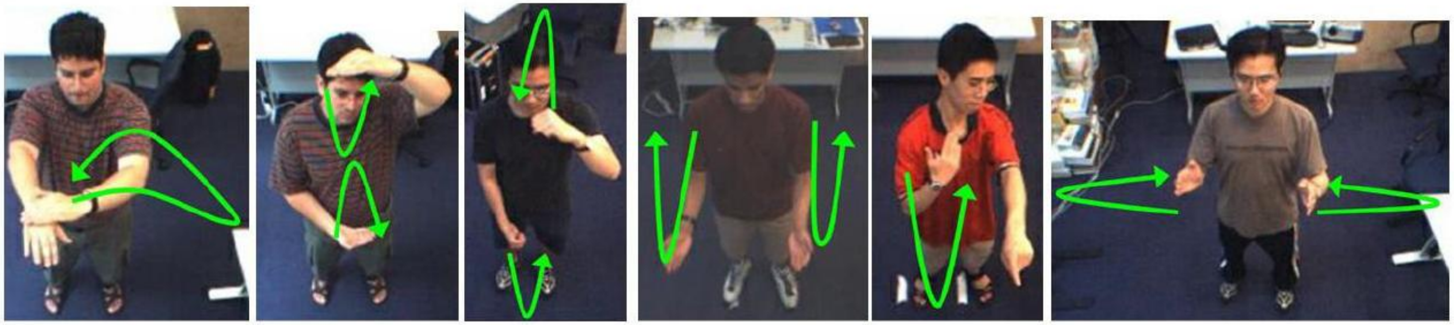
Is it an animal or a person?

$x =$	**Bill**	**Clinton**	**embarrassed**	**Chelsea**	**at**	**her**	**wedding**	**at**	**Astor**	**Courts**
$y =$	B-PERS	I-PERS	O	B-PERS	O	O	O	O	B-LOC	I-LOC

Vision: Gesture Recognition

- Given a sequence of frames in a video annotate each frame with a gesture type:

- Types of gestures:



Flip back | Shrink vertically | Expand vertically | Double back | Point and back | Expand horizontally

- It is hard to predict gestures from each frame in isolation, you need to **exploit relations** between frames and gesture types

Outline

- Sequence labeling / segmentation problems: settings and example problems:
  - Part-of-speech tagging, named entity recognition, gesture recognition
- Hidden Markov Model
  - Standard definition + maximum likelihood estimation
  - General views: as a representative of linear models
- Perceptron and Structured Perceptron
  - algorithms / motivations
- Decoding with the Linear Model
- Discussion: Discriminative vs. Generative

Hidden Markov Models

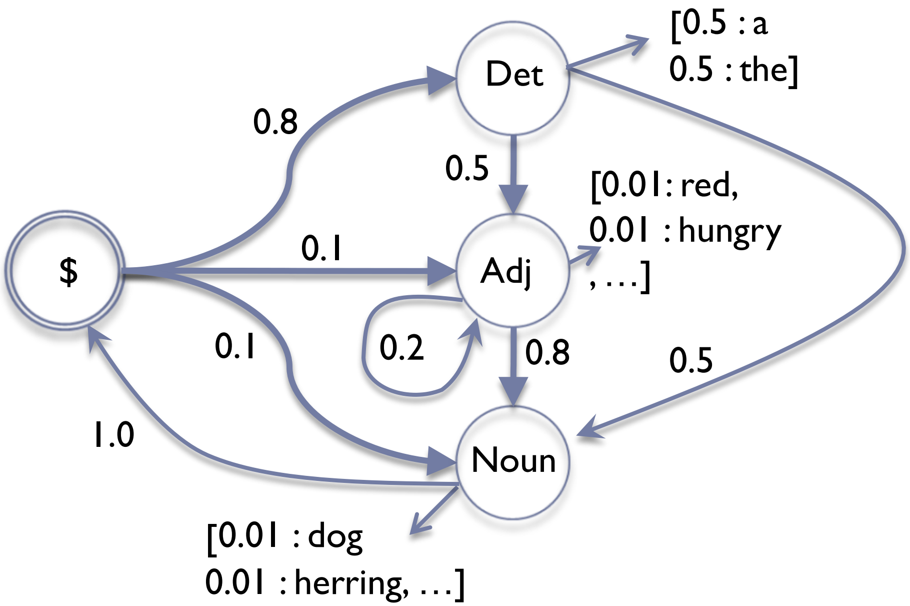
- We will consider the part-of-speech (POS) tagging example

John	**carried**	**a**	**tin**	**can**	**.**
NP	VBD	DT	NN	NN	.

- A "generative" model, i.e.:
  - Model: Introduce a parameterized model of how both words and tags are generated $P(\boldsymbol{x}, \boldsymbol{y} | \theta)$
  - Learning: use a labeled training set to estimate the most likely parameters of the model $\hat{\theta}$
  - Decoding: $\boldsymbol{y} = \text{argmax}_{\boldsymbol{y}'} P(\boldsymbol{x}, \boldsymbol{y}' | \hat{\theta})$

Hidden Markov Models

A simplistic state diagram for noun phrases: N – tags, M – vocabulary size



Example:

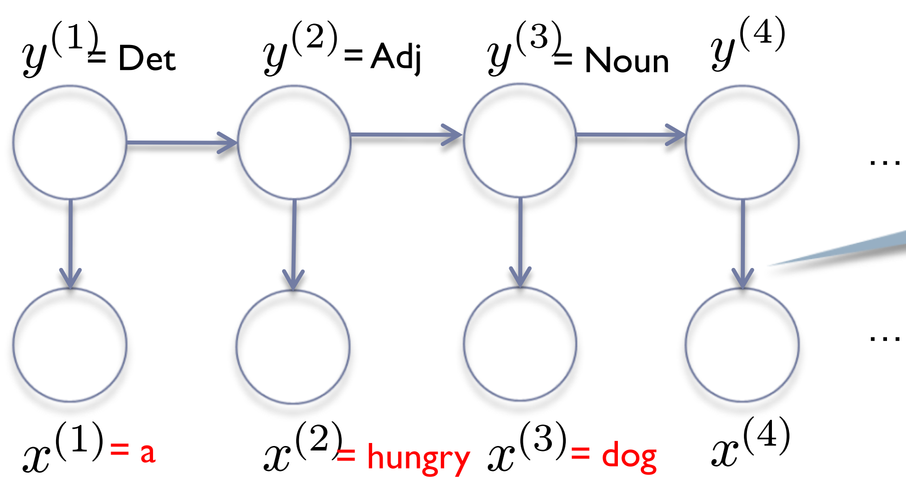
a hungry dog

- States correspond to POS tags,
- Words are emitted independently from each POS tag
- Parameters (to be estimated from the training set):
  - Transition probabilities $P(y^{(t)}|y^{(t-1)})$: [N x N] matrix
  - Emission probabilities $P(x^{(t)}|y^{(t)})$: [N x M] matrix

Stationarity assumption: this probability does not depend on the position in the sequence *t*

Hidden Markov Models

Representation as an instantiation of a *graphical model*: N – tags, M – vocabulary size



A arrow means that in the generative story $x^{(4)}$ is generated from some $P(x^{(4)} \mid y^{(4)})$

- States correspond to POS tags,
- Words are emitted independently from each POS tag
- Parameters (to be estimated from the training set):
  - Transition probabilities $P(y^{(t)}|y^{(t-1)})$: [N x N] matrix
  - Emission probabilities $P(x^{(t)}|y^{(t)})$: [N x M] matrix

Stationarity assumption: this probability does not depend on the position in the sequence *t*

Hidden Markov Models: Estimation

- N – the number tags, M – vocabulary size
- Parameters (to be estimated from the training set):
  - Transition probabilities $a_{ji} = P(y^{(t)} = i | y^{(t-1)} = j)$, A - [N x N] matrix
  - Emission probabilities $b_{ik} = P(x^{(t)} = k | y^{(t)} = i)$, B - [N x M] matrix
- Training corpus:
  - $\boldsymbol{x}^{(1)}$= (In, an, Oct., 19, review, of, ….), $\boldsymbol{y}^{(1)}$= (IN, DT, NNP, CD, NN, IN, ….)
  - $\boldsymbol{x}^{(2)}$= (Ms., Haag, plays, Elianti,.), $\boldsymbol{y}^{(2)}$= (NNP, NNP, VBZ, NNP, .)
  - …
  - $\boldsymbol{x}^{(L)}$= (The, company, said,…), $\boldsymbol{y}^{(L)}$= (DT, NN, VBD, NNP, .)
- How to estimate the parameters using maximum likelihood estimation?
  - You probably can guess what these estimation should be?

Hidden Markov Models: Estimation

- Parameters (to be estimated from the training set):
  - Transition probabilities $a_{ji} = P(y^{(t)} = i | y^{(t-1)} = j)$, A - [N x N] matrix
  - Emission probabilities $b_{ik} = P(x^{(t)} = k | y^{(t)} = i)$, B - [N x M] matrix
- Training corpus: $(\mathbf{x}^{(l)}, \mathbf{y}^{(l)})$, l = 1, … L
- Write down the probability of the corpus according to the HMM:

$$P(\{\boldsymbol{x}^{(l)}, \boldsymbol{y}^{(l)}\}_{l=1}^{L}) = \prod_{l=1}^{L} P(\boldsymbol{x}^{(l)}, \boldsymbol{y}^{(l)}) =$$

$$= \prod_{l=1}^{L} \underbrace{a_{\$, y_1^{(l)}}}_{\text{Select tag for the first word}} \left(\prod_{t=1}^{|\boldsymbol{x}^l|-1} \underbrace{b_{y_t^{(l)}, x_t^{(l)}}}_{\text{Draw a word from this state}} \underbrace{a_{y_t^{(l)}, y_{t+1}^{(l)}}}_{\text{Select the next state}} \right) \underbrace{b_{y_{|\mathrm{x}^l|}^{(l)}, x_{|\mathrm{x}^l|}^{(l)}}}_{\text{Draw last word}} \underbrace{a_{y_{|\mathrm{x}^l|}^{(l)}, \$}}_{\text{Transit into the \$ state}} =$$

$$= \prod_{i,j=1}^{N} a_{i,j}^{C_T(i,j)} \prod_{i=1}^{N} \prod_{k=1}^{M} b_{i,k}^{C_E(i,k)}$$

$C_T(i,j)$ is #times tag i is followed by tag j. Here we assume that \$ is a special tag which precedes and succeeds every sentence

$C_E(i,k)$ is #times word k is emitted by tag i

Hidden Markov Models: Estimation

- Maximize: $P(\{\boldsymbol{x}^{(l)}, \boldsymbol{y}^{(l)}\}_{l=1}^{L}) =$

$$= \prod_{i,j=1}^{N} a_{i,j}^{C_T(i,j)} \prod_{i=1}^{N} \prod_{k=1}^{M} b_{i,k}^{C_E(i,k)}$$

C_T(i,j) is #times tag i is followed by tag j.

C_E(i,k) is #times word k is emitted by tag i

- Equivalently maximize the logarithm of this: $\log(P(\{\boldsymbol{x}^{(l)}, \boldsymbol{y}^{(l)}\}_{l=1}^{L})) =$

$$= \sum_{i=1}^{N} \left(\sum_{j=1}^{N} C_T(i,j) \log a_{i,j} + \sum_{k=1}^{M} C_E(i,k) \log b_{i,k} \right)$$

subject to probabilistic constraints: $\sum_{j=1}^{N} a_{i,j} = 1, \quad \sum_{i=1}^{N} b_{i,k} = 1, \quad i = 1, \ldots, N$

- Or, we can decompose it into 2N optimization tasks:

For transitions

$i = 1, \ldots, N$:

$$\max_{a_{i,1},\ldots,a_{i,N}} \sum_{j=1}^{N} C_T(i,j) \log a_{i,j}$$

$$\text{s.t. } \sum_{j=1}^{N} a_{i,j} = 1$$

For emissions

$i = 1, \ldots, N$:

$$\max_{b_{i,1},\ldots,b_{i,M}} C_E(i,k) \log b_{i,k}$$

$$\text{s.t. } \sum_{i=1}^{N} b_{i,k} = 1$$

Hidden Markov Models: Estimation

- For transitions (some *i*)

$$\max_{a_{i,1},\ldots,a_{i,N}} \sum_{j=1}^{N} C_T(i,j) \log a_{i,j}$$

$$\text{s.t. } 1 - \sum_{j=1}^{N} a_{i,j} = 0$$

- Constrained optimization task, Lagrangian:

$$L(a_{i,1},\ldots,a_{i,N},\lambda) = \sum_{j=1}^{N} C_T(i,j) \log a_{i,j} + \lambda \times (1 - \sum_{j=1}^{N} a_{i,j})$$

- Find critical points of Lagrangian by solving the system of equations:

$$\frac{\partial L}{\partial \lambda} = 1 - \sum_{j=1}^{N} a_{i,j} = 0$$

$$\frac{\partial L}{\partial a_{ij}} = \frac{C_T(i,j)}{a_{ij}} - \lambda = 0 \implies a_{ij} = \frac{C_T(i,j)}{\lambda}$$

$$P(y^t = j | y^{t-1} = i) = a_{i,j} = \frac{C_T(i,j)}{\sum_{j'} C_T(i,j')}$$

- Similarly, for emissions:

$$P(x^t = k | y^t = i) = b_{i,k} = \frac{C_E(i,k)}{\sum_{k'} C_E(i,k')}$$

The maximum likelihood solution is just normalized counts of events. Always like this for generative models if all the labels are visible in training

I ignore "smoothing" to process rare or unseen word tag combinations… Outside score of the seminar

HMMs as linear models

John	carried	a	tin	can	.
?	?	?	?	?	.

- Decoding: $\boldsymbol{y} = \text{argmax}_{\boldsymbol{y}'} P(\boldsymbol{x}, \boldsymbol{y}' | A, B) = \text{argmax}_{\boldsymbol{y}'} \log P(\boldsymbol{x}, \boldsymbol{y}' | A, B)$
- We will talk about the decoding algorithm slightly later, let us generalize Hidden Markov Model:

$$\log P(\boldsymbol{x}, \boldsymbol{y}' | A, B) = \sum_{l=1}^{|x|+1} \log b_{y'_i, x_i} + \log a_{y'_i, y'_{i+1}}$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} C_T(\boldsymbol{y}', i, j) \times \log a_{i,j} + \sum_{i=1}^{N} \sum_{k=1}^{M} C_E(\boldsymbol{x}, \boldsymbol{y}', i, k) \times \log b_{i,k}$$

The number of times tag i is followed by tag j in the candidate *y'*

The number of times tag i corresponds to word k in (*x*, *y'*)

- But this is just a linear model!!

Scoring: example

$$(\boldsymbol{x}, \boldsymbol{y}') = \begin{array}{llllll} \textbf{John} & \textbf{carried} & \textbf{a} & \textbf{tin} & \textbf{can} & \textbf{.} \\ \text{NP} & \text{VBD} & \text{DT} & \text{NN} & \text{NN} & . \end{array}$$

$$\varphi(\boldsymbol{x}, \boldsymbol{y}') = \big(\begin{array}{cccccccc} 1 & 0 & \ldots & 1 & 1 & 0 & \ldots \\ \text{NP: John} & \text{NP:Mary} & \ldots & \text{NP-VBD} & \text{NN-.} & \text{MD-.} & \ldots \end{array} \big)$$

Unary features $C_E(\boldsymbol{x}, \boldsymbol{y}', i, k)$

Edge features $C_T(\boldsymbol{y}', i, j)$

$$w_{ML} = \big(\log b_{NP,John} \quad \log b_{NP,Mary} \quad \ldots \quad \log a_{NN,VBD} \quad \log a_{NN,.} \quad \log a_{MD,.} \quad \ldots \big)$$

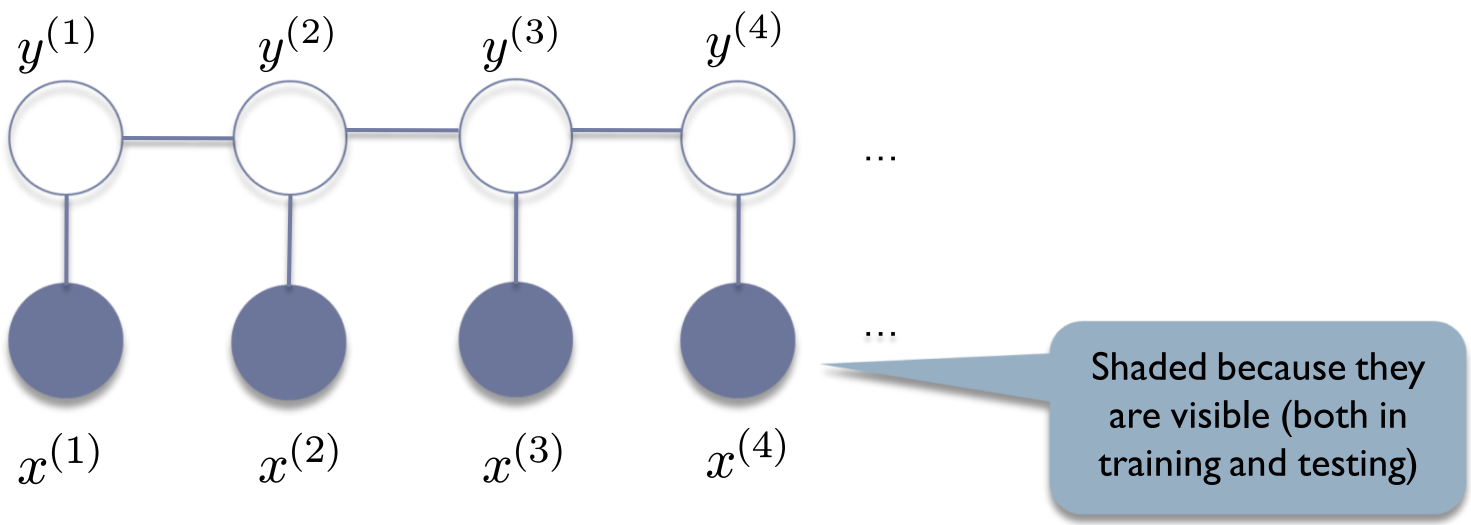
- Their inner product is exactly $\log P(\boldsymbol{x}, \boldsymbol{y}' | A, B)$

$$w_{ML} \cdot \varphi(\boldsymbol{x}, \boldsymbol{y}') = \sum_{i=1}^{N} \sum_{j=1}^{N} C_T(\boldsymbol{y}', i, j) \times \log a_{i,j} + \sum_{i=1}^{N} \sum_{k=1}^{M} C_E(\boldsymbol{x}, \boldsymbol{y}', i, k) \times \log b_{i,k}$$

- But may be there other (and better?) ways to estimate w, especially when we know that HMM is not a faithful model of reality?
- It is not only a theoretical question! (we'll talk about that in a moment)

Feature view

Basically, we define features which correspond to edges in the graph:

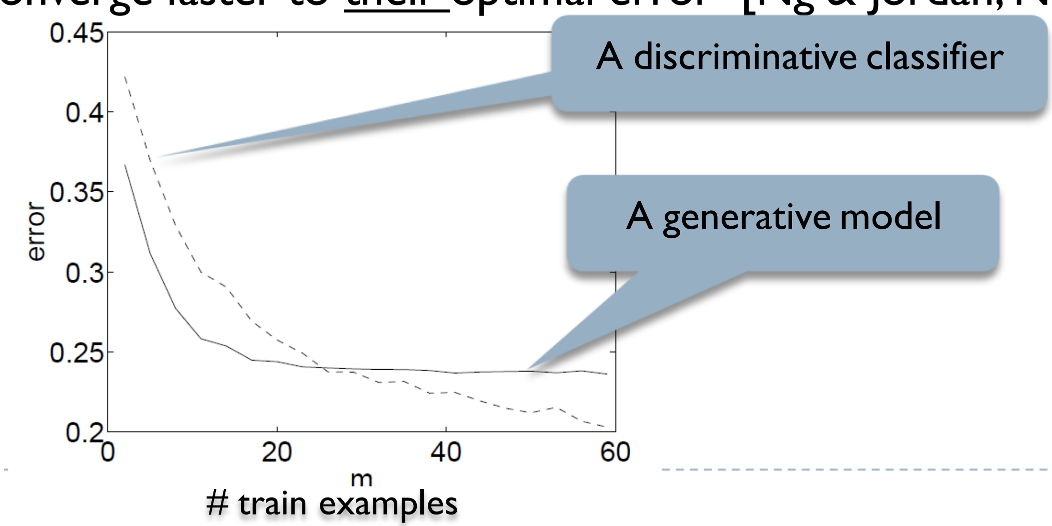


Generative modeling

Real case: HMM is a coarse approximation of reality

- For a very large dataset (asymptotic analysis):
  - If data is generated from some "true" HMM, then (if the training is sufficiently large), we are guaranteed to have an optimal tagger
  - Otherwise, (generally) HMM will not correspond to an optimal linear classifier
  - Discriminative methods which minimize the error more directly **are guaranteed** (under some fairly general conditions) to converge to an optimal linear classifier
- For smaller training sets
  - Generative classifiers converge faster to <u>their</u> optimal error [Ng & Jordan, NIPS 01]

Errors on a regression dataset (predict housing prices in Boston area):



Outline

- Sequence labeling / segmentation problems: settings and example problems:
  - Part-of-speech tagging, named entity recognition, gesture recognition
- Hidden Markov Model
  - Standard definition + maximum likelihood estimation
  - General views: as a representative of linear models
- Perceptron and Structured Perceptron
  - algorithms / motivations
- Decoding with the Linear Model
- Discussion: Discriminative vs. Generative

Perceptron

- Let us start with a binary classification problem $y \in \{+1, -1\}$
  - For binary classification the prediction rule is: $y = \text{sign}\left(\boldsymbol{w} \cdot \varphi(x)\right)$

break ties (0) in some deterministic way

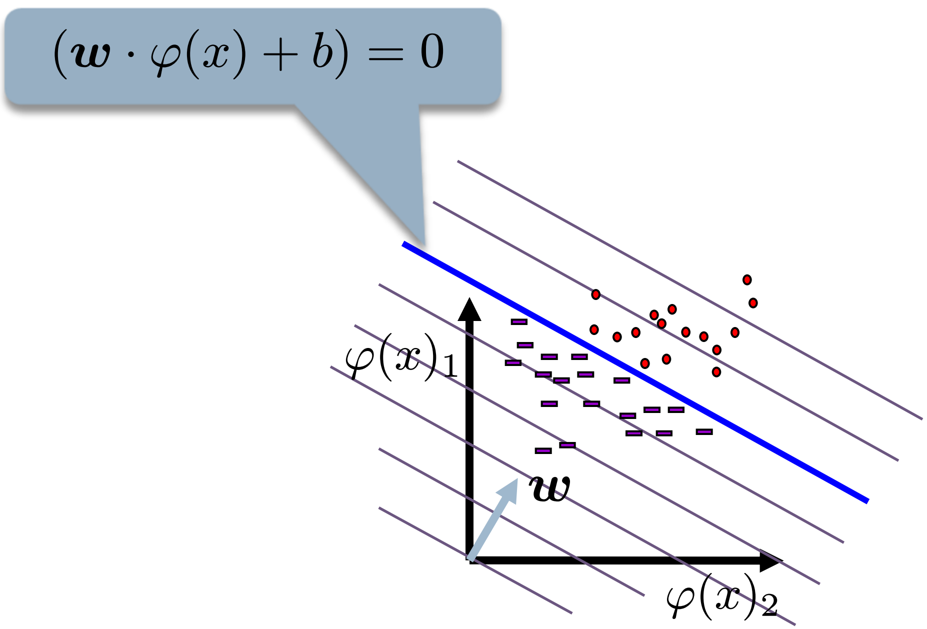
- Perceptron algorithm, given a training set $\{x^{(l)}, y^{(l)}\}_{l=1}^{L}$

```
w = 0  // initialize
do
    err = 0
    for l = 1 .. L      // over the training examples
        if ( y^(l) (w · φ(x^(l))) < 0)      // if mistake
            w += η y^(l) φ(x^(l))      // update, η > 0
            err ++   // # errors
        endif
    endfor
while ( err > 0 )   // repeat until no errors
return w
```

Figure adapted from Dan Roth's class at UIUC

Linear classification

- Linear separable case, "a perfect" classifier:



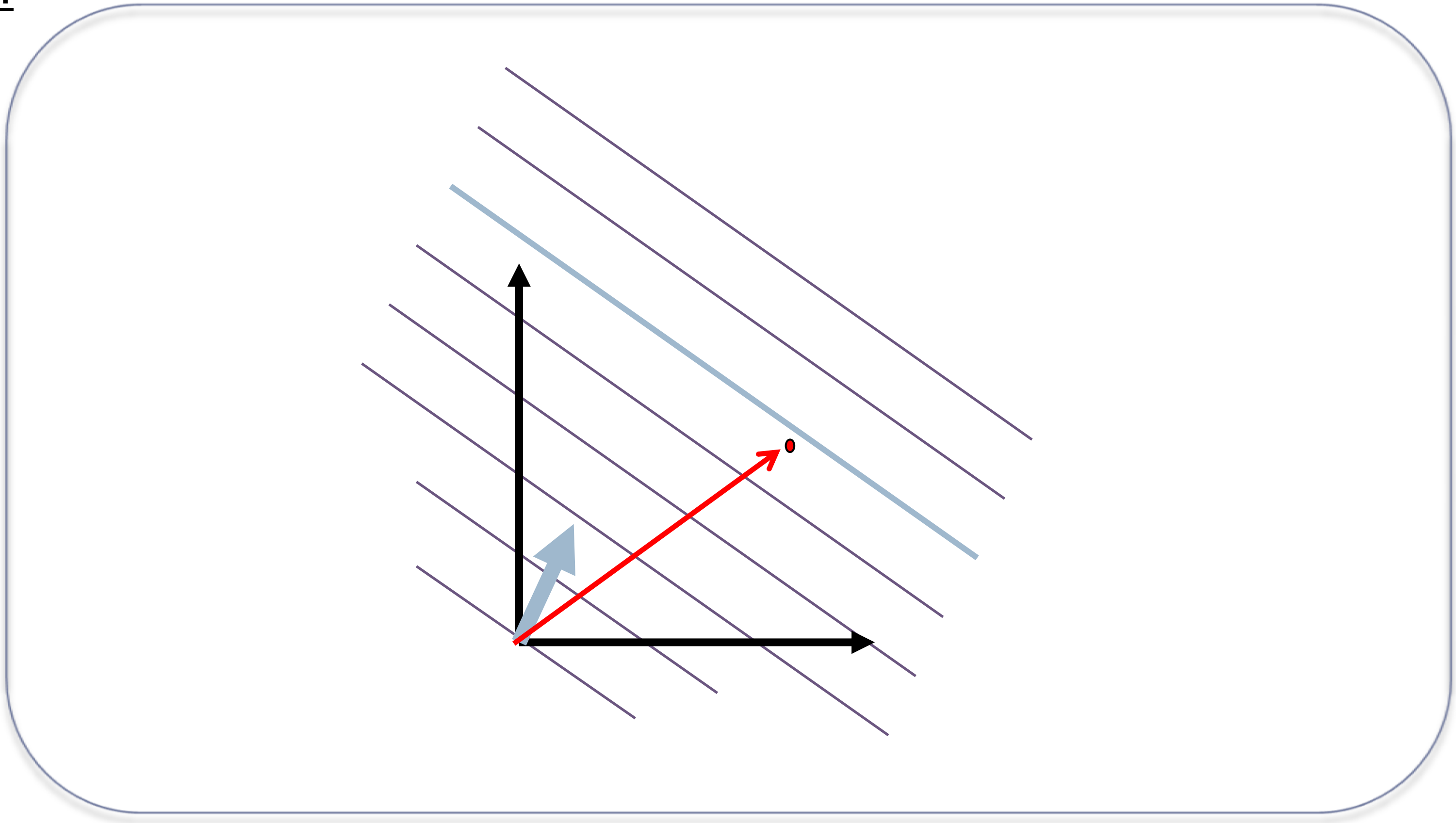
- Linear functions are often written as: $y = \text{sign}\,(\boldsymbol{w} \cdot \varphi(x) + b)$, but we can assume that $\varphi(x)_0 = 1$ for any x

Perceptron: geometric interpretation

if ($y^{(l)}\left(\boldsymbol{w}\cdot\varphi(x^{(l)})\right)$ < 0) // if mistake

$\boldsymbol{w}$ += $\eta y^{(l)}\varphi(x^{(l)})$ // update

endif

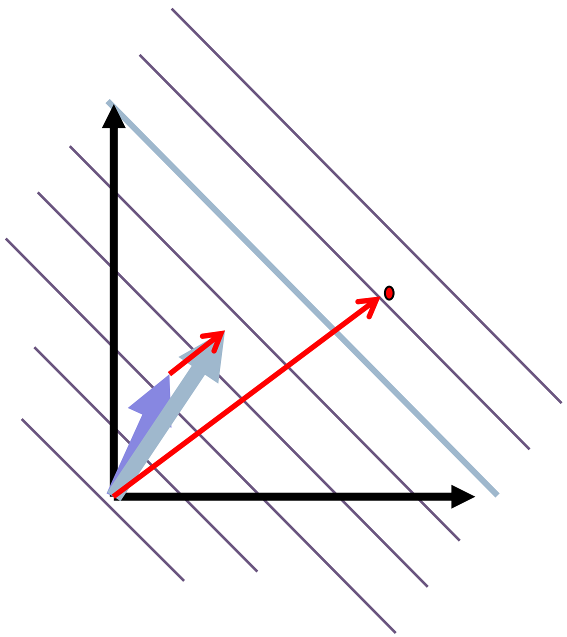


Perceptron: geometric interpretation

<u>if</u> ($y^{(l)}\left(\boldsymbol{w} \cdot \varphi(x^{(l)})\right) < 0$) // if mistake

$\boldsymbol{w}$ += $\eta y^{(l)} \varphi(x^{(l)})$ // update

<u>endif</u>

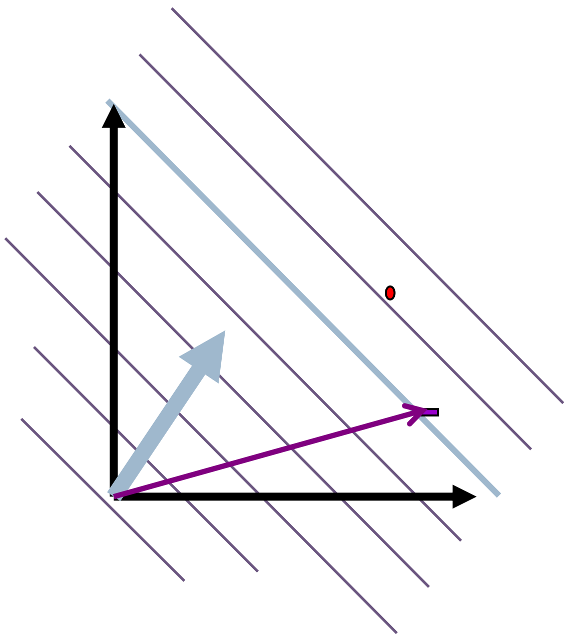


Perceptron: geometric interpretation

<u>if</u> ($y^{(l)}\left(\boldsymbol{w}\cdot\varphi(x^{(l)})\right)$ < 0) // if mistake

$\boldsymbol{w}$ += $\eta y^{(l)}\varphi(x^{(l)})$ // update

<u>endif</u>

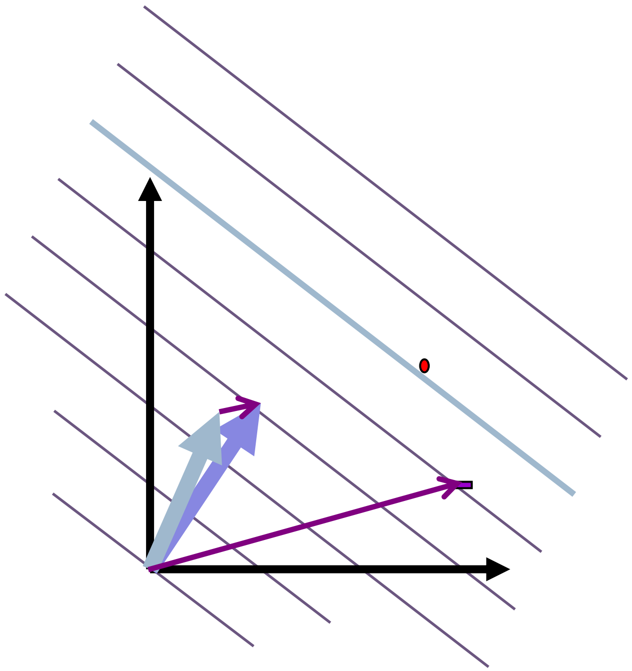


Perceptron: geometric interpretation

<u>if</u> ($y^{(l)}\left(\boldsymbol{w} \cdot \varphi(x^{(l)})\right)$ < 0) // if mistake

$\boldsymbol{w}$ += $\eta y^{(l)} \varphi(x^{(l)})$ // update

<u>endif</u>



Perceptron: algebraic interpretation

<u>if</u> ($y^{(l)}\left(\boldsymbol{w}\cdot\varphi(x^{(l)})\right)$ < 0)   // if mistake
    $\boldsymbol{w}$ += $\eta y^{(l)}\varphi(x^{(l)})$ // update
<u>endif</u>

- We want after the update to <u>increase</u> $y^{(l)}\left(w\cdot\varphi(x^{(l)})\right)$
  - If the increase is large enough than there will be no misclassification
- Let's see that's what happens after the update

$$y^{(l)}\left((\boldsymbol{w}+\eta y^{(l)}\varphi(x^{(l)}))\cdot\varphi(x^{(l)})\right)$$

$$= y^{(l)}\left(\boldsymbol{w}\cdot\varphi(x^{(l)})\right) + \eta(y^{(l)})^2\left(\varphi(x^{(l)})\cdot\varphi(x^{(l)})\right)$$

$(y^{(l)})^2 = 1$

squared norm > 0

- So, the perceptron update moves the decision hyperplane towards misclassified $\varphi(x^{(l)})$

Perceptron

- The perceptron algorithm, obviously, can only converge if the training set is linearly separable
- It is guaranteed to converge in a finite number of iterations, dependent on how well two classes are separated (Novikoff, 1962)

Averaged Perceptron

- A small modification

$\boldsymbol{w} = \mathbf{0}$, $\boldsymbol{w}_{\sum} = \mathbf{0}$ // initialize
<u>for</u> k = 1 .. K // for a number of iterations
    <u>for</u> l = 1 .. L // over the training examples
        <u>if</u> ($y^{(l)}\left(\boldsymbol{w} \cdot \varphi(x^{(l)})\right)$ < 0) // if mistake
            $\boldsymbol{w}$ += $\eta y^{(l)} \varphi(x^{(l)})$ // update, $\eta > 0$
        <u>endif</u>
        $\boldsymbol{w}_{\sum}$ += $\boldsymbol{w}$ // sum of $\boldsymbol{w}$ over the course of training
    <u>endfor</u>
<u>endfor</u>
<u>return</u> $\frac{1}{KL}\boldsymbol{w}_{\sum}$

Do not run until convergence

Note: it is after <u>endif</u>

More stable in training: a vector $\boldsymbol{w}$ which survived more iterations without updates is more similar to the resulting vector $\frac{1}{KL}\boldsymbol{w}_{\sum}$, as it was added a larger number of times

Structured Perceptron

- Let us start with <u>structured problem</u>: $\boldsymbol{y} = \text{argmax}_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \boldsymbol{w} \cdot \varphi(\boldsymbol{x}, \boldsymbol{y}')$
- Perceptron algorithm, given a training set $\{\boldsymbol{x}^{(l)}, \boldsymbol{y}^{(l)}\}_{l=1}^{L}$

```
w = 0  // initialize
do
    err = 0
    for l = 1 .. L      // over the training examples
        ŷ = argmax_{y' ∈ Y(x^(l))} w · φ(x^(l), y')   // model prediction
        if ( w · φ(x^(l), ŷ) > w · φ(x^(l), y^(l)))     // if mistake
            w += η (φ(x^(l), y^(l)) − φ(x^(l), ŷ))      // update
            err ++   // # errors
        endif
    endfor
while ( err > 0 )   // repeat until no errors
return w
```

Pushes the correct sequence up and the incorrectly predicted one down

Str. perceptron: algebraic interpretation

<u>if</u> $(\boldsymbol{w} \cdot \varphi(\boldsymbol{x}^{(l)}, \hat{\boldsymbol{y}}) > \boldsymbol{w} \cdot \varphi(\boldsymbol{x}^{(l)}, \boldsymbol{y}^{(l)}))$ // if mistake

$\boldsymbol{w}$ += $\eta \left(\varphi(\boldsymbol{x}^{(l)}, \boldsymbol{y}^{(l)}) - \varphi(\boldsymbol{x}^{(l)}, \hat{\boldsymbol{y}})\right)$ // update

<u>endif</u>

- We want after the update to <u>increase</u> $\boldsymbol{w} \cdot (\varphi(\boldsymbol{x}^{(l)}, \boldsymbol{y}^{(l)}) - \varphi(\boldsymbol{x}^{(l)}, \hat{\boldsymbol{y}}))$
  - If the increase is large enough then $\boldsymbol{y}^{(l)}$ will be scored above $\hat{\boldsymbol{y}}$
- Clearly, that this is achieved as this product will be increased by

$$\eta ||\varphi(\boldsymbol{x}^{(l)}, \boldsymbol{y}^{(l)}) - \varphi(\boldsymbol{x}^{(l)}, \hat{\boldsymbol{y}})||^2$$

There might be other $\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x}^{(l)})$ but we will deal with them on the next iterations

Structured Perceptron

- Positive:
  - Very easy to implement
  - Often, achieves respectable results
  - As other discriminative techniques, does not make assumptions about the generative process
  - Additional features can be easily integrated, as long as decoding is tractable
- Drawbacks
  - "Good" discriminative algorithms should optimize some measure which is closely related to the expected testing results: what perceptron is doing on non-linearly separable data seems not clear
  - However, for the averaged (voted) version a generalization bound which generalization properties of Perceptron (Freund & Shapire 98)

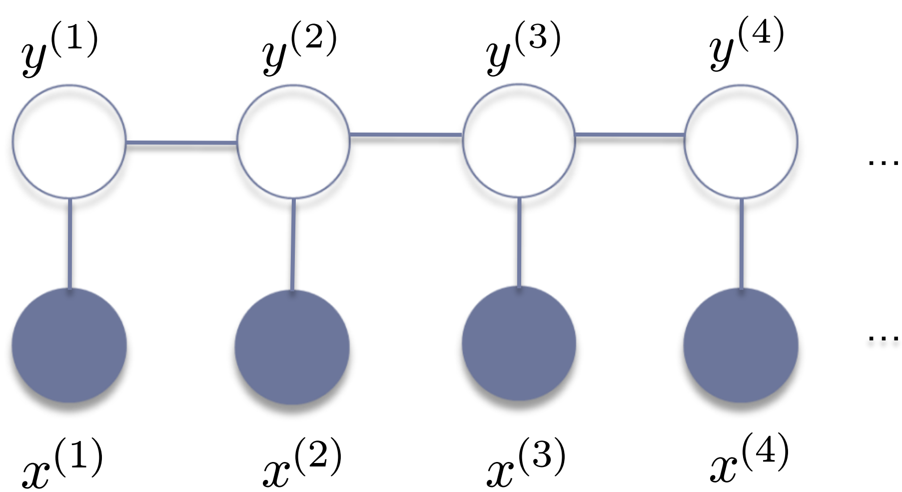
- Later, we will consider more advance learning algorithms

Outline

- Sequence labeling / segmentation problems: settings and example problems:
  - Part-of-speech tagging, named entity recognition, gesture recognition
- Hidden Markov Model
  - Standard definition + maximum likelihood estimation
  - General views: as a representative of linear models
- Perceptron and Structured Perceptron
  - algorithms / motivations
- Decoding with the Linear Model
- Discussion: Discriminative vs. Generative

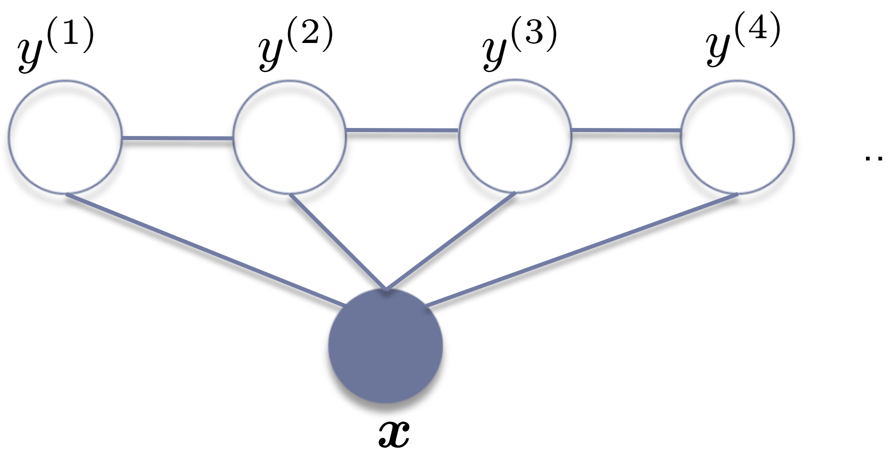
Decoding with the Linear model

- Decoding: $\boldsymbol{y} = \operatorname{argmax}_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} w \cdot \varphi(\boldsymbol{x}, \boldsymbol{y}')$
- Again a linear model with the following edge features (a generalization of a HMM)
- In fact, the algorithm does not depend on the feature of input (they do not need to be *local*)



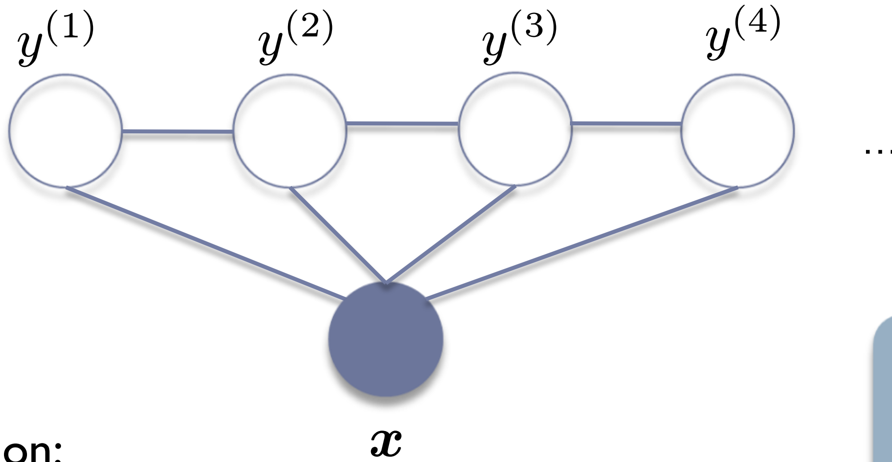
Decoding with the Linear model

- Decoding: $\boldsymbol{y} = \text{argmax}_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} w \cdot \varphi(\boldsymbol{x}, \boldsymbol{y}')$
- Again a linear model with the following edge features (a generalization of a HMM)
- In fact, the algorithm does not depend on the feature of input (they do not need to be *local*)



Decoding with the Linear model

- Decoding: $\boldsymbol{y} = \mathrm{argmax}_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \, w \cdot \varphi(\boldsymbol{x}, \boldsymbol{y}')$



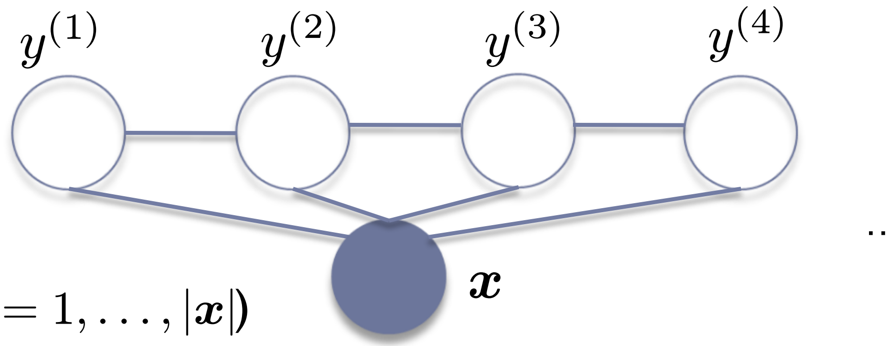
- Let's change notation:

Start/Stop symbol information ($) can be encoded with them too.

  - Edge scores $f_t(y_{t-1}, y_t, \boldsymbol{x})$: roughly corresponds to $\log a_{y_{t-1}, y_t} + \log b_{y_t, x_t}$
  - Defined for t = 0 too ("start" feature: $y_0 = \$$)
  - Decode: $\boldsymbol{y} = \mathrm{argmax}_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \sum_{t=1}^{|\boldsymbol{x}|} f_t(y'_{t-1}, y'_t, \boldsymbol{x})$
- Decoding: a dynamic programming algorithm - Viterbi algorithm

Viterbi algorithm

- Decoding: $\boldsymbol{y} = \mathrm{argmax}_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \sum_{t=1}^{|\boldsymbol{x}|} f_t(y'_{t-1}, y'_t, \boldsymbol{x})$



- Loop invariant: $(t = 1, \ldots, |\boldsymbol{x}|)$
  - $\mathrm{score}_t[y]$ - score of the highest scoring sequence up to position t with
  - $\mathrm{prev}_t[y]$ - previous tag on this sequence
- Init: $\mathrm{score}_0[\$] = 0$, $\mathrm{score}_0[y] = -\infty$ for other y
- Recomputation $(t = 1, \ldots, |\boldsymbol{x}|)$

$$\mathrm{prev}_t[y] = \mathrm{argmax}_{y'}\ score_t[y'] + f_t(y', y, \boldsymbol{x})$$

$$\mathrm{score}_t[y] = \mathrm{score}_{t-1}[prev_t[y]] + f_t(\mathrm{prev}_t[y], y, \boldsymbol{x})$$

Time complexity ?

$O(N^2|x|)$

- Return: retrace prev pointers starting from $\mathrm{argmax}_y\ \mathrm{score}_{|x|}[y]$

Outline

- Sequence labeling / segmentation problems: settings and example problems:
  - Part-of-speech tagging, named entity recognition, gesture recognition
- Hidden Markov Model
  - Standard definition + maximum likelihood estimation
  - General views: as a representative of linear models
- Perceptron and Structured Perceptron
  - algorithms / motivations
- Decoding with the Linear Model
- Discussion: Discriminative vs. Generative

Recap: Sequence Labeling

- Hidden Markov Models:
  - How to estimate
- Discriminative models
  - How to learn with structured perceptron
- Both learning algorithms result in a linear model
  - How to label with the linear models

Discriminative vs Generative

Not necessary the case for generative models with latent variables

- Generative models:
  - Cheap to estimate: simply normalized counts
  - Hard to integrate complex features: need to come up with a generative story and this story may be wrong
  - Does not result in an optimal classifier when model assumptions are wrong (i.e., always)
- Discriminative models
  - More expensive to learn: need to run decoding (here, Viterbi) during training and usually multiple times per an example
  - Easy to integrate features: though some feature may make decoding intractable
  - Usually less accurate on small datasets

Reminders

- Speakers: slides about a week before the talk, meetings with me before/after this point will normally be needed
- Reviewers: reviews are accepted only before the day we consider the topic
- Everyone: References to the papers to read at GoogleDocs,

- These slides (and previous ones) will be online today
  - speakers: send me the last version of your slides too

- Next time: Lea about Models of Parsing, PCFGs vs general WCFGs (Michael Collins' book chapter)