Predicting Sequences: Structured Perceptron

CS 6355: Structured Prediction

THE UNIVERSITY OF UTAH

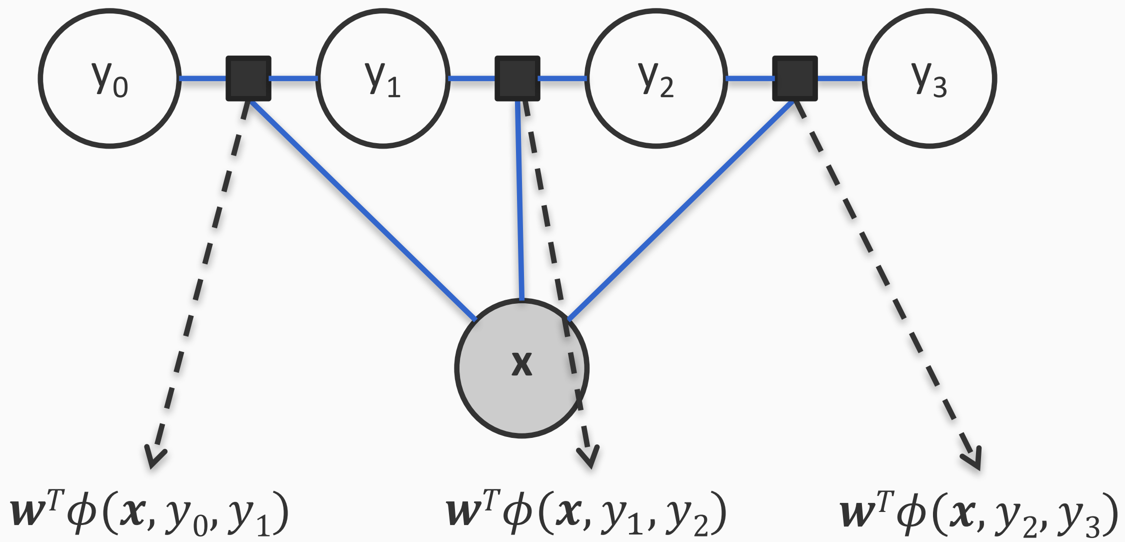
Conditional Random Fields summary

- 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
- Connections to other models
  - Effectively a linear classifier
  - A generalization of logistic regression to structures
  - An conditional variant of a Markov Random Field
    - We will see this soon

Global features

The feature function decomposes over the sequence

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



Outline

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

HMM is also a linear classifier

Consider the HMM:

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

HMM is also a linear classifier

Consider the HMM:

$$P(\mathbf{x}, \mathbf{y}) = \prod_i \underbrace{P(y_i \mid y_{i-1})}_{\text{Transitions}} \underbrace{P(x_i \mid y_i)}_{\text{Emissions}}$$

HMM is also a linear classifier

Consider the HMM:

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

Or equivalently

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

Log joint probability = transition scores + emission scores

HMM is also a linear classifier

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

Log joint probability = transition scores + emission scores

Let us examine this expression using a carefully defined set of *indicator functions*

HMM is also a linear classifier

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

Log joint probability = transition scores + emission scores

Let us examine this expression using a carefully defined set of *indicator functions*

$$I_{[z]} = \begin{cases} 1, & z \text{ is true,} \\ 0, & z \text{ is false.} \end{cases}$$

Indicators are functions that map Booleans to 0 or 1

HMM is also a linear classifier

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

Log joint probability = transition scores + emission scores

Let us examine this expression using a carefully defined set of *indicator functions*

Equivalent to

$$\boxed{\sum_s \sum_{s'} \log P(s \mid s') \cdot I_{[y_i = s]} \cdot I_{[y_{i-1} = s']}}$$

The indicators ensure that only one of the elements of the double summation is non-zero

HMM is also a linear classifier

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

Log joint probability = transition scores + emission scores

Let us examine this expression using a carefully defined set of *indicator functions*

Equivalent to

$$\boxed{\sum_s \log P(x_i \mid s) \cdot I_{[y_i = s]}}$$

The indicators ensure that only one of the elements of the summation is non-zero

HMM is also a linear classifier

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

Let us examine this expression using a carefully defined set of *indicator functions*

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \sum_s \sum_{s'} \log P(s \mid s') \cdot I_{[y_i = s]} \cdot I_{[y_{i-1} = s']}$$

$$+ \sum_i \sum_s \log P(x_i \mid s) \cdot I_{[y_i = s]}$$

HMM is also a linear classifier

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

Let us examine this expression using a carefully defined set of *indicator functions*

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \sum_s \sum_{s'} \log P(s \mid s') \cdot I_{[y_i = s]} \cdot I_{[y_{i-1} = s']} + \sum_i \sum_s \log P(x_i \mid s) \cdot I_{[y_i = s]}$$

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_s \sum_{s'} \log P(s \mid s') \boxed{\sum_i I_{[y_i = s]} I_{[y_{i-1} = s']}} + \sum_s \log P(x_i \mid s) \boxed{\sum_i I_{[y_i = s]}}$$

HMM is also a linear classifier

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

Let us examine this expression using a carefully defined set of *indicator functions*

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_s \sum_{s'} \log P(s \mid s') \sum_i I_{[y_i = s]} I_{[y_{i-1} = s']}$$
$$+ \sum_s \log P(x_i \mid s) \sum_i I_{[y_i = s]}$$

Number of times there is a transition in the sequence from state s' to state s

$\mathrm{Count}(s' \to s)$

HMM is also a linear classifier

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

Let us examine this expression using a carefully defined set of *indicator functions*

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_s \sum_{s'} \log P(s \mid s') \cdot \mathrm{Count}(s' \to s)$$
$$+ \sum_s \log P(x_i \mid s) \boxed{\sum_i I_{[y_i = s]}}$$

HMM is also a linear classifier

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

Let us examine this expression using a carefully defined set of *indicator functions*

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_s \sum_{s'} \log P(s \mid s') \cdot \text{Count}(s' \to s)$$

$$+ \sum_s \log P(x_i \mid s) \boxed{\sum_i I_{[y_i = s]}}$$

Number of times state s occurs in the sequence: $\text{Count}(s)$

HMM is also a linear classifier

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

Let us examine this expression using a carefully defined set of *indicator functions*

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_s \sum_{s'} \log P(s \mid s') \cdot \mathrm{Count}(s' \to s)$$
$$+ \sum_s \log P(x_i \mid s) \cdot \mathrm{Count}(s)$$

HMM is also a linear classifier

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

Let us examine this expression using a carefully defined set of *indicator functions*

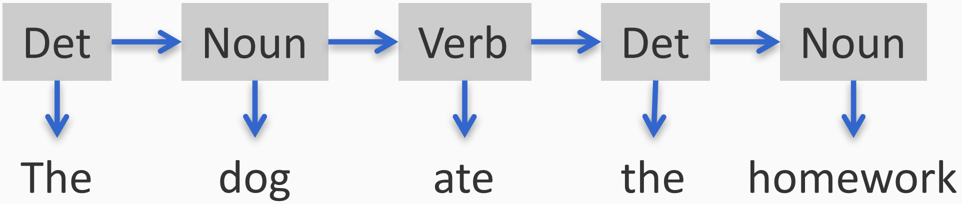
$$\log P(\mathbf{x},\mathbf{y}) = \sum_s \sum_{s'} \log P(s \mid s') \cdot \mathrm{Count}(s' \to s) + \sum_s \log P(x_i \mid s) \cdot \mathrm{Count}(s)$$

This is a linear function

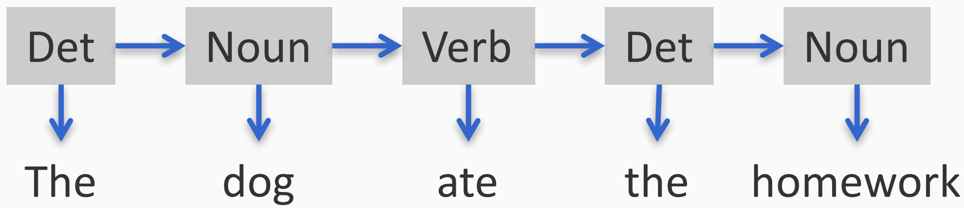
log P terms are the weights; counts via indicators are features

Can be written as $\mathbf{w}^T\phi(\mathbf{x}, \mathbf{y})$ and add more features

HMM is a linear classifier: An example

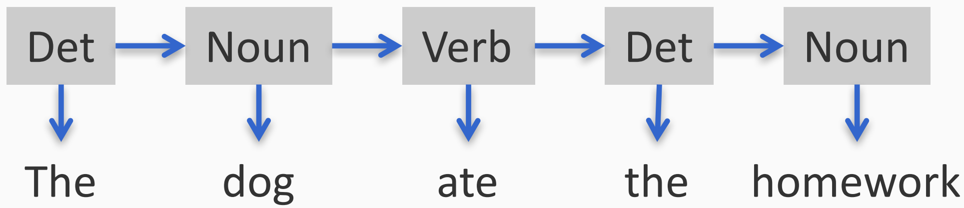


HMM is a linear classifier: An example



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

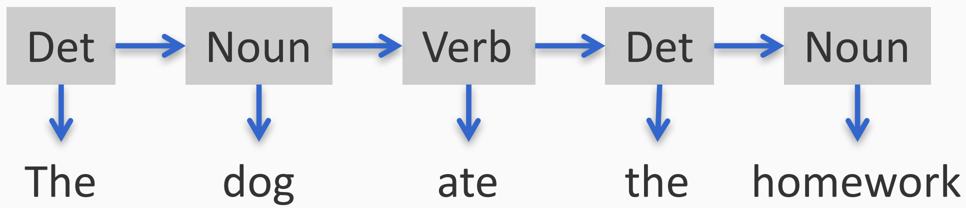
HMM is a linear classifier: An example



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

Transition scores + Emission scores

HMM is a linear classifier: An example



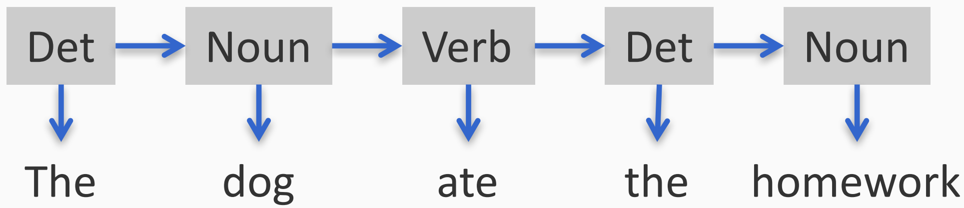
Consider $\log P(\mathbf{y}, \mathbf{x}) = \sum_i \log P(y_i | y_{i-1}) + \log P(x_i | y_i)$

log P(Det $\rightarrow$ Noun) $\times$ 2

+ log P(Noun $\rightarrow$ Verb) $\times$ 1 + Emission scores

+ log P(Verb $\rightarrow$ Det) $\times$ 1

HMM is a linear classifier: An example



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

log P(Det $\rightarrow$ Noun) $\times$ 2

+ log P(Noun $\rightarrow$ Verb) $\times$ 1

+ log P(Verb $\rightarrow$ Det) $\times$ 1

+

log P(The | Det) $\times$ 1

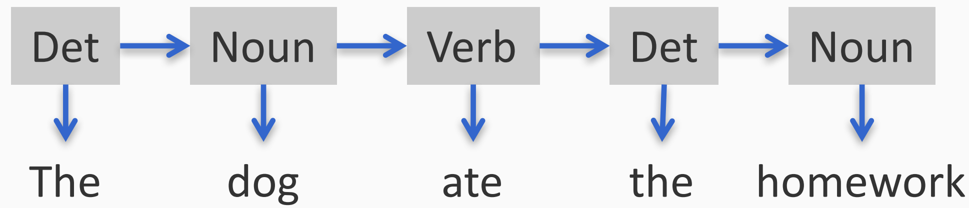
+ log P(dog| Noun) $\times$ 1

+ log P(ate| Verb) $\times$ 1

+ log P(the| Det) $\times$ 1

+ log P(homework| Noun) $\times$ 1

HMM is a linear classifier: An example



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

$\boxed{\text{log P(Det} \rightarrow \text{Noun)}} \times 2$

$+ \boxed{\text{log P(Noun} \rightarrow \text{Verb)}} \times 1$

$+ \boxed{\text{log P(Verb} \rightarrow \text{Det)}} \times 1$

$+$

$\boxed{\text{log P(The | Det)}} \times 1$

$+ \boxed{\text{log P(dog| Noun)}} \times 1$

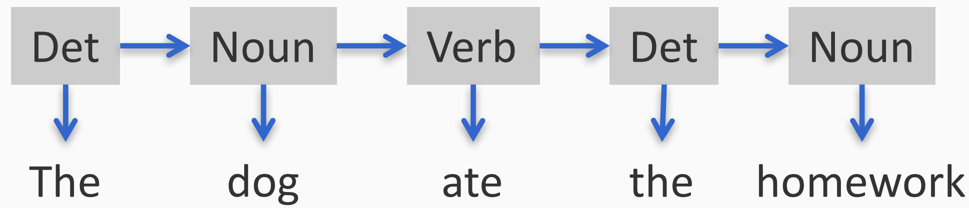
$+ \boxed{\text{log P(ate| Verb)}} \times 1$

$+ \boxed{\text{log P(the| Det)}} \times 1$

$+ \boxed{\text{log P(homework| Noun)}} \times 1$

w: Parameters of the model

HMM is a linear classifier: An example



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

$\log \mathrm{P(The} \mid \mathrm{Det)} \times \boxed{1}$

$\log \mathrm{P(Det} \rightarrow \mathrm{Noun)} \times \boxed{2}$

$+ \log \mathrm{P(dog} \mid \mathrm{Noun)} \times \boxed{1}$

$+ \log \mathrm{P(Noun} \rightarrow \mathrm{Verb)} \times \boxed{1}$ +

$+ \log \mathrm{P(ate} \mid \mathrm{Verb)} \times \boxed{1}$

$+ \log \mathrm{P(Verb} \rightarrow \mathrm{Det)} \times \boxed{1}$

$+ \log \mathrm{P(the} \mid \mathrm{Det)} \times \boxed{1}$

$+ \log \mathrm{P(homework} \mid \mathrm{Noun)} \times \boxed{1}$

$\phi(\mathbf{x}, \mathbf{y})$: Properties of this output and the input

HMM is a linear classifier: An example

Det → Noun → Verb → Det → Noun

The dog ate the homework

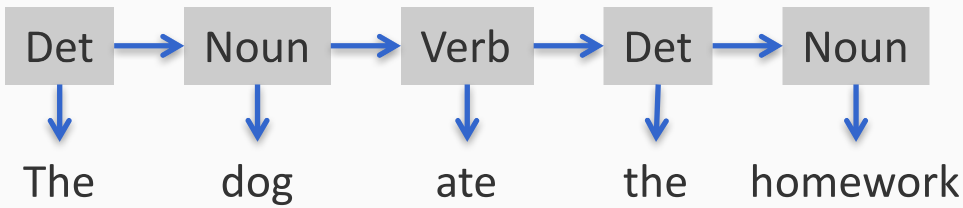
Consider $\log P(\mathbf{y}, \mathbf{x}) = \sum_i \log P(y_i | y_{i-1}) + \log P(x_i | y_i)$

$$
\begin{bmatrix}
\log P(Det \rightarrow Noun) \\
\log P(Noun \rightarrow Verb) \\
\log P(Verb \rightarrow Det) \\
\log P(The \mid Det) \\
\log P(dog \mid Noun) \\
\log P(ate \mid Verb) \\
\log P(the \mid Det \\
\log P(homework \mid Noun)
\end{bmatrix}
\cdot
\begin{bmatrix}
2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1
\end{bmatrix}
$$

w: Parameters of the model

ϕ(**x**, **y**): Properties of this output and the input

HMM is a linear classifier: An example



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

$$\begin{bmatrix} \log P(Det \rightarrow Noun) \\ \log P(Noun \rightarrow Verb) \\ \log P(Verb \rightarrow Det) \\ \log P(The \mid Det) \\ \log P(dog \mid Noun) \\ \log P(ate \mid Verb) \\ \log P(the \mid Det \\ \log P(homework \mid Noun) \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

log P(**x**, **y**) = A linear scoring function = $\mathbf{w}^T\phi(\mathbf{x},\mathbf{y})$

w: Parameters of the model

ϕ(**x**, **y**): Properties of this output and the input

Towards structured Perceptron

1. HMM is a linear classifier
   - Can we treat it as any linear classifier for training?
   - If so, we could add additional features that are global properties
     - As long as the output can be decomposed for easy inference

Towards structured Perceptron

1. HMM is a linear classifier
   - Can we treat it as any linear classifier for training?
   - If so, we could add additional features that are global properties
     - As long as the output can be decomposed for easy inference

2. The Viterbi algorithm calculates max $w^T\phi(\mathbf{x}, \mathbf{y})$

   Viterbi only cares about scores to structures (not necessarily normalized)

Towards structured Perceptron

1. HMM is a linear classifier
   - Can we treat it as any linear classifier for training?
   - If so, we could add additional features that are global properties
     - As long as the output can be decomposed for easy inference

2. The Viterbi algorithm calculates max $w^T\phi(\mathbf{x}, \mathbf{y})$

   Viterbi only cares about scores to structures (not necessarily normalized)

3. We could push the learning algorithm to train for un-normalized scores
   - If we need normalization, we could always normalize by exponentiating and dividing by Z (the partition function)
   - That is, the learning algorithm can effectively just focus on the score of **y** for a particular **x**
   - *Train a discriminative model instead of the generative one!*

Structured Perceptron algorithm

Given a training set D = {($\mathbf{x}$,$\mathbf{y}$)}

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
    1. For each training example ($\mathbf{x}$, $\mathbf{y}$) $\in$ D:

       Structured Perceptron update

3. Return $\mathbf{w}$

Prediction: $\mathrm{argmax}_{\mathbf{y}}\, \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y})$

Structured Perceptron algorithm

Given a training set D = {($\mathbf{x}$,$\mathbf{y}$)}

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
    1. For each training example ($\mathbf{x}$, $\mathbf{y}$) $\in$ D:
        1. Predict $\mathbf{y'} = \text{argmax}_{\mathbf{y'}}\ \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y'})$
        2. If $\mathbf{y} \neq \mathbf{y'}$, update $\mathbf{w} \leftarrow \mathbf{w} + r\ (\phi(\mathbf{x}, \mathbf{y}) - \phi(\mathbf{x}, \mathbf{y'}))$
3. Return $\mathbf{w}$

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

Structured Perceptron algorithm

Given a training set D = {(**x**,**y**)}

1. Initialize **w** = 0 $\in \Re^n$
2. For epoch = 1 … T:
    1. For each training example (**x**, **y**) $\in$ D:
        1. Predict **y'** = $\text{argmax}_{\mathbf{y'}}\ \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y'})$
        2. If **y** ≠ **y'**, update $\mathbf{w} \leftarrow \mathbf{w} + r\,(\phi(\mathbf{x}, \mathbf{y}) - \phi(\mathbf{x}, \mathbf{y'}))$
3. Return **w**

Update only on an error. Perceptron is an mistake-driven algorithm. If there is a mistake, promote **y** and demote **y'**

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

Structured Perceptron algorithm

Given a training set D = $\{(\mathbf{x},\mathbf{y})\}$

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. For each training example $(\mathbf{x}, \mathbf{y}) \in D$:
      1. Predict $\mathbf{y'} = \text{argmax}_{\mathbf{y'}}\, \mathbf{w}^T \phi(\mathbf{x}, \mathbf{y'})$
      2. If $\mathbf{y} \neq \mathbf{y'}$, update $\mathbf{w} \leftarrow \mathbf{w} + r\,(\phi(\mathbf{x}, \mathbf{y}) - \phi(\mathbf{x}, \mathbf{y'}))$
3. Return $\mathbf{w}$

T is a hyperparameter to the algorithm

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

Structured Perceptron algorithm

In practice, good to shuffle D before the inner loop

Given a training set $D = \{(\mathbf{x},\mathbf{y})\}$

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
    1. For each training example $(\mathbf{x}, \mathbf{y}) \in D$:
        1. Predict $\mathbf{y'} = \text{argmax}_{\mathbf{y'}}\, \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y'})$
        2. If $\mathbf{y} \neq \mathbf{y'}$, update $\mathbf{w} \leftarrow \mathbf{w} + r\,(\phi(\mathbf{x}, \mathbf{y}) - \phi(\mathbf{x}, \mathbf{y'}))$
3. Return $\mathbf{w}$

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

Structured Perceptron algorithm

Given a training set D = {($\mathbf{x}$,$\mathbf{y}$)}

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
    1. For each training example ($\mathbf{x}$, $\mathbf{y}$) $\in$ D:
        1. Predict $\mathbf{y}' = \text{argmax}_{\mathbf{y}'}\ \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y}')$
        2. If $\mathbf{y} \neq \mathbf{y}'$, update $\mathbf{w} \leftarrow \mathbf{w} + r\,(\phi(\mathbf{x}, \mathbf{y}) - \phi(\mathbf{x}, \mathbf{y}'))$
3. Return $\mathbf{w}$

Inference in training loop!

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

Notes on structured perceptron

- Mistake bound for separable data, just like perceptron

- In practice, use averaging for better generalization
  - Initialize **a** = 0
  - After each step, whether there is an update or not, **a** $\leftarrow$ **w** + **a**
    - Note, we still check for mistake using **w** not **a**
  - Return **a** at the end instead of **w**

    Exercise: Optimize this for performance – modify **a** *only* on errors

- Global update
  - One weight vector for entire sequence
    - Not for each position
  - Same algorithm can be derived via constraint classification
    - Create a binary classification data set and run perceptron

Structured Perceptron with averaging

Given a training set D = {($\mathbf{x}$,$\mathbf{y}$)}

1. Initialize $\mathbf{w} = 0 \in \Re^n$, $\mathbf{a} = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. For each training example ($\mathbf{x}$, $\mathbf{y}$) $\in$ D:
      1. Predict $\mathbf{y'} = \text{argmax}_{\mathbf{y'}}\ \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y'})$
      2. If $\mathbf{y} \neq \mathbf{y'}$, update $\mathbf{w} \leftarrow \mathbf{w} + r\,(\phi(\mathbf{x}, \mathbf{y}) - \phi(\mathbf{x}, \mathbf{y'}))$
      3. Set $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{w}$
3. Return $\mathbf{a}$

CRF vs. structured perceptron

Stochastic gradient descent update for CRF

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

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

Expectation vs max

Structured perceptron

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

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

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

Caveat: Adding regularization will change the CRF update, averaging changes the perceptron update

The lay of the land

HMM: A generative model, assigns probabilities to sequences

The lay of the land

HMM: A generative model, assigns probabilities to sequences

Two roads diverge

The lay of the land

HMM: A generative model, assigns probabilities to sequences

Two roads diverge

- Hidden Markov Models are actually just linear classifiers

- Don't really care whether we are predicting probabilities. We are assigning scores to a full output for a given input (like multiclass)

- Generalize algorithms for linear classifiers. Sophisticated models that can use arbitrary features

- Structured Perceptron
  Structured SVM

The lay of the land

HMM: A generative model, assigns probabilities to sequences

Two roads diverge

- Hidden Markov Models are actually just linear classifiers
- Don't really care whether we are predicting probabilities. We are assigning scores to a full output for a given input (like multiclass)
- Generalize algorithms for linear classifiers. Sophisticated models that can use arbitrary features
- Structured Perceptron
  Structured SVM

- Model probabilities via exponential functions. Gives us the log-linear representation
- Log-probabilities for sequences for a given input
- Learn by maximizing likelihood. Sophisticated models that can use arbitrary features
- Conditional Random field

The lay of the land

HMM: A generative model, assigns probabilities to sequences

Two roads diverge

- Hidden Markov Models are actually just linear classifiers
- Don't really care whether we are predicting probabilities. We are assigning scores to a full output for a given input (like multiclass)
- Generalize algorithms for linear classifiers. Sophisticated models that can use arbitrary features
- Structured Perceptron
  Structured SVM

Discriminative/Conditional models

- Model probabilities via exponential functions. Gives us the log-linear representation
- Log-probabilities for sequences for a given input
- Learn by maximizing likelihood. Sophisticated models that can use arbitrary features
- Conditional Random field

The lay of the land

HMM: A generative model, assigns probabilities to sequences

Two roads diverge

- Hidden Markov Models are actually just linear classifiers
- Don't really care whether we are predicting probabilities. We are assigning scores to a full output for a given input (like multiclass)
- Generalize algorithms for linear classifiers. Sophisticated models that can use arbitrary features
- Structured Perceptron
  Structured SVM

- Model probabilities via exponential functions. Gives us the log-linear representation
- Log-probabilities for sequences for a given input
- Learn by maximizing likelihood. Sophisticated models that can use arbitrary features
- Conditional Random field

Discriminative/Conditional models

Applicable beyond sequences

Eventually, similar objective minimized with different loss functions

The lay of the land

HMM: A generative model, assigns probabilities to sequences

Two roads diverge

- Hidden Markov Models are actually just linear classifiers
- Don't really care whether we are predicting probabilities. We are assigning scores to a full output for a given input (like multiclass)
- Generalize algorithms for linear classifiers. Sophisticated models that can use arbitrary features
- Structured Perceptron
  Structured SVM

- Model probabilities via exponential functions. Gives us the log-linear representation
- Log-probabilities for sequences for a given input
- Learn by maximizing likelihood. Sophisticated models that can use arbitrary features
- Conditional Random field

Discriminative/Conditional models

Applicable beyond sequences
Eventually, similar objective minimized with different loss functions

Coming soon…

Sequence models: Summary

- Goal: Predict an output sequence given input sequence
- Hidden Markov Model
- Inference
  - Predict via Viterbi algorithm
- Conditional models/discriminative models
  - Local approaches (no inference during training)
    - MEMM, conditional Markov model
  - Global approaches (inference during training)
    - CRF, structured perceptron
- To think
  - What are the *parts* in a sequence model?
  - How is each model scoring these parts?

Prediction is not always tractable for general structures

Same dichotomy for more general structures