Learning Structured Predictors

Xavier Carreras

TALP Research Center
Universitat Politècnica de Catalunya
Supervised (Structured) Prediction

- Learning to predict: given training data

\[ \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\} \]

learn a predictor \( x \rightarrow y \) that *works well* on unseen inputs \( x \)

- Non-Structured Prediction: outputs \( y \) are atomic
  - Binary prediction: \( y \in \{-1, +1\} \)
  - Multiclass prediction: \( y \in \{1, 2, \ldots, L\} \)

- Structured Prediction: outputs \( y \) are structured
  - Sequence prediction: \( y \) are sequences
  - Parsing: \( y \) are trees
  - \( \ldots \)
Named Entity Recognition

y  PER    -    QNT    -    -    ORG    ORG    -    TIME
x  Jim  bought  300  shares  of  Acme  Corp.  in  2006
Named Entity Recognition

y  PER   -   QNT   -   -   ORG   ORG   -   TIME
x  Jim bought 300 shares of Acme Corp. in 2006

y  PER   PER   -   -   LOC
x  Jack London went to Paris

y  PER   PER   -   -   LOC
x  Paris Hilton went to London
Part-of-speech Tagging

y NNP NNP VBZ NNP .

x Ms. Haag plays Elianti .
Unesco is now holding its biennial meetings in New York.

x are sentences
y are syntactic dependency trees
Machine Translation

x are sentences in Chinese
y are sentences in English aligned to x
Object Detection

\( x \) are images
\( y \) are grids labeled with object types

(Kumar and Hebert 2003)
Object Detection

(Kumar and Hebert 2003)

\[ x \text{ are images} \]
\[ y \text{ are grids labeled with object types} \]
Today’s Goals

▶ Introduce basic tools for structure prediction
  ▶ We will restrict to sequence prediction

▶ Understand what tools we can use from standard classification
  ▶ Learning paradigms and algorithms, in essence, work here too
  ▶ However, computations behind algorithms are prohibitive

▶ Understand what tools we can use from existing formalisms for structured data
  ▶ We will use inference algorithms for tractable computations
  ▶ E.g., algorithms for HMMs (Viterbi, forward-backward) will play a major role in today’s methods
Today’s Goals

- Introduce basic tools for structure prediction
  - We will restrict to sequence prediction

- Understand what tools we can use from standard classification
  - Learning paradigms and algorithms, in essence, work here too
  - However, computations behind algorithms are prohibitive

- Understand what tools we can use from existing formalisms for structured data
  - We will use inference algorithms for tractable computations
  - E.g., algorithms for HMMs (Viterbi, forward-backward) will play a major role in today’s methods
Sequence Prediction

\[
\begin{array}{cccccc}
  y & \text{PER} & \text{PER} & - & - & \text{LOC} \\
  x & \text{Jack} & \text{London} & \text{went} & \text{to} & \text{Paris}
\end{array}
\]
Sequence Prediction

- $x = x_1 x_2 \ldots x_n$ are input sequences, $x_i \in \mathcal{X}$
- $y = y_1 y_2 \ldots y_n$ are output sequences, $y_i \in \{1, \ldots, L\}$

**Goal:** given training data

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

learn a predictor $x \rightarrow y$ that **works well** on unseen inputs $x$

- What is the form of our prediction model?
Jack London went to Paris

Decompose the sequence into $n$ classification problems:

- A classifier predicts individual labels at each position

\[
\hat{y}_i = \arg\max_{l \in \{\text{loc, per, -}\}} w \cdot f(x, i, l)
\]

- $f(x, i, l)$ represents an assignment of label $l$ for $x_i$
- $w$ is a vector of parameters, has a weight for each feature of $f$
  - Use standard classification methods to learn $w$
- At test time, predict the best sequence by a simple concatenation of the best label for each position
Approach 1: Local Classifiers

Jack London went to Paris

Decompose the sequence into $n$ classification problems:

- A classifier predicts individual labels at each position

$$\hat{y}_i = \arg\max_{l \in \{\text{LOC}, \text{PER}, -\}} w \cdot f(x, i, l)$$

- $f(x, i, l)$ represents an assignment of label $l$ for $x_i$
- $w$ is a vector of parameters, has a weight for each feature of $f$
  - Use standard classification methods to learn $w$
- At test time, predict the best sequence by a simple concatenation of the best label for each position
Indicator Features

- \( f(x, i, l) \) is a vector of \( d \) features representing label \( l \) for \( x_i \)
  
  \[
  ( f_1(x, i, l), \ldots, f_j(x, i, l), \ldots, f_d(x, i, l) )
  \]

- What's in a feature \( f_j(x, i, l) \)?
  - Anything we can compute using \( x \) and \( i \) and \( l \)
  - Anything that indicates whether \( l \) is (not) a good label for \( x_i \)
  - Indicator features: binary-valued features looking at a single simple property

\[
    f_j(x, i, l) = \begin{cases} 
      1 & \text{if } x_i = \text{London and } l = \text{LOC} \\
      0 & \text{otherwise}
    \end{cases}
\]

\[
    f_k(x, i, l) = \begin{cases} 
      1 & \text{if } x_{i+1} = \text{went and } l = \text{LOC} \\
      0 & \text{otherwise}
    \end{cases}
\]
More Features for NE Recognition

\[ \textbf{PER} \]
Jack London went to Paris

In practice, construct \( f(x, i, l) \) by . . .

- Define a number of simple patterns of \( x \) and \( i \)
  - current word \( x_i \)
  - is \( x_i \) capitalized?
  - \( x_i \) has digits?
  - prefixes/suffixes of size 1, 2, 3, . . .
  - is \( x_i \) a known location?
  - is \( x_i \) a known person?
  - next word
  - previous word
  - current and next words together
  - other combinations

- Generate features by combining patterns with label identities \( l \)
More Features for NE Recognition

\[ \text{PER} \quad \text{PER} \quad - \]
Jack London went to Paris

In practice, construct \( f(x, i, l) \) by . . .

- Define a number of simple patterns of \( x \) and \( i \)
  - current word \( x_i \)
  - is \( x_i \) capitalized?
  - \( x_i \) has digits?
  - prefixes/suffixes of size 1, 2, 3, . . .
  - is \( x_i \) a known location?
  - is \( x_i \) a known person?
  - next word
  - previous word
  - current and next words together
  - other combinations

- Generate features by combining patterns with label identities \( l \)

Main limitation: features can’t capture interactions between labels!
Approach 2: HMM for Sequence Prediction

Define an HMM were each label is a state

Model parameters:
- $\pi_l$: probability of starting with label $l$
- $T_{l,l'}$: probability of transitioning from $l$ to $l'$
- $O_{l,x}$: probability of generating symbol $x$ given label $l$

Predictions:

$$p(x, y) = \pi_{y_1} O_{y_1, x_1} \prod_{i > 1} T_{y_{i-1}, y_i} O_{y_i, x_i}$$

Learning: relative counts + smoothing

Prediction: Viterbi algorithm
Approach 2: Representation in HMM

Label interactions are captured in the transition parameters
- But interactions between labels and input symbols are quite limited!
  - Only $O_{y_i,x_i} = p(x_i | y_i)$
  - Not clear how to exploit patterns such as:
    - Capitalization, digits
    - Prefixes and suffixes
    - Next word, previous word
    - Combinations of these with label transitions

Why? HMM independence assumptions:
  given label $y_i$, token $x_i$ is independent of anything else
Approach 2: Representation in HMM

- Label interactions are captured in the transition parameters
- But interactions between labels and input symbols are quite limited
  - Only $O_{y_i,x_i} = p(x_i | y_i)$
  - Not clear how to exploit patterns such as:
    - Capitalization, digits
    - Prefixes and suffixes
    - Next word, previous word
    - Combinations of these with label transitions

- Why? HMM independence assumptions:
  given label $y_i$, token $x_i$ is independent of anything else
Local Classifiers vs. HMM

**Local Classifiers**

- Form: \( w \cdot f(x, i, l) \)
- Learning: standard classifiers
- Prediction: independent for each \( x_i \)
- Advantage: feature-rich
- Drawback: no label interactions

**HMM**

- Form: \( \pi_{y_1} O_{y_1, x_1} \prod_{i>1} T_{y_{i-1}, y_i} O_{y_i, x_i} \)
- Learning: relative counts
- Prediction: Viterbi
- Advantage: label interactions
- Drawback: no fine-grained features
Approach 3: Global Sequence Predictors

\[
y: \text{PER PER - - LOC} \\
x: \text{Jack London went to Paris}
\]

Learn a single classifier from \( x \rightarrow y \)

\[
predict(x_{1:n}) = \operatorname{argmax}_{y \in \mathcal{Y}^n} \mathbf{w} \cdot f(x, y)
\]

Next questions: . . .

- How do we represent entire sequences in \( f(x, y) \)?
- There are exponentially-many sequences \( y \) for a given \( x \), how do we solve the \( \operatorname{argmax} \) problem?
Approach 3: Global Sequence Predictors

\[ y: \text{PER PER - - LOC} \]
\[ x: \text{Jack London went to Paris} \]

Learn a single classifier from \( x \rightarrow y \)

\[ \text{predict}(x_{1:n}) = \arg\max_{y \in Y^n} w \cdot f(x, y) \]

Next questions: . . .

- How do we represent entire sequences in \( f(x, y) \)?
- There are exponentially-many sequences \( y \) for a given \( x \), how do we solve the \( \arg\max \) problem?
Factored Representations

\[ \mathbf{y}: \text{PER PER } - - \text{ LOC} \]
\[ \mathbf{x}: \text{Jack London went to Paris} \]

- How do we represent entire sequences in \( f(\mathbf{x}, \mathbf{y}) \)?
  - Look at individual assignments \( y_i \) (standard classification)
  - Look at bigrams of outputs labels \( \langle y_{i-1}, y_i \rangle \)
  - Look at trigrams of outputs labels \( \langle y_{i-2}, y_{i-1}, y_i \rangle \)
  - Look at \( n \)-grams of outputs labels \( \langle y_{i-n+1}, \ldots, y_{i-1}, y_i \rangle \)
  - Look at the full label sequence \( \mathbf{y} \) (intractable)

- A factored representation will lead to a tractable model
Factored Representations

\[ y: \quad \text{PER} \quad \text{PER} \quad - \quad - \quad \text{LOC} \]
\[ x: \quad \text{Jack} \quad \text{London} \quad \text{went} \quad \text{to} \quad \text{Paris} \]

▶ How do we represent entire sequences in \( f(x, y) \)?

▶ Look at individual assignments \( y_i \) (standard classification)
▶ Look at bigrams of outputs labels \( \langle y_{i-1}, y_i \rangle \)
▶ Look at trigrams of outputs labels \( \langle y_{i-2}, y_{i-1}, y_i \rangle \)
▶ Look at \( n \)-grams of outputs labels \( \langle y_{i-n+1}, \ldots, y_{i-1}, y_i \rangle \)
▶ Look at the full label sequence \( y \) (intractable)

▶ A factored representation will lead to a tractable model
Factored Representations

\[
\begin{array}{ccccccc}
\text{y:} & \text{PER} & \text{PER} & - & - & \text{LOC} \\
\text{x:} & \text{Jack} & \text{London} & \text{went} & \text{to} & \text{Paris}
\end{array}
\]

- How do we represent entire sequences in \( f(x, y) \)?
  - Look at individual assignments \( y_i \) (standard classification)
  - Look at bigrams of outputs labels \( \langle y_{i-1}, y_i \rangle \)
  - Look at trigrams of outputs labels \( \langle y_{i-2}, y_{i-1}, y_i \rangle \)
  - Look at \( n \)-grams of outputs labels \( \langle y_{i-n+1}, \ldots, y_{i-1}, y_i \rangle \)
  - Look at the full label sequence \( y \) (intractable)

- A factored representation will lead to a tractable model
Factored Representations

<table>
<thead>
<tr>
<th>y:</th>
<th>PER</th>
<th>PER</th>
<th>-</th>
<th>-</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>x:</td>
<td>Jack London went to Paris</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- How do we represent entire sequences in $f(x, y)$?
  - Look at individual assignments $y_i$ (standard classification)
  - Look at bigrams of outputs labels $\langle y_{i-1}, y_i \rangle$
  - Look at trigrams of outputs labels $\langle y_{i-2}, y_{i-1}, y_i \rangle$
  - Look at $n$-grams of outputs labels $\langle y_{i-n+1}, \ldots, y_{i-1}, y_i \rangle$
  - Look at the full label sequence $y$ (intractable)

- A factored representation will lead to a tractable model
Bigram Indicator Features

**Indicator features:**

\[ f_j(x, i, y_{i-1}, y_i) = \begin{cases} 
1 & \text{if } x_i = "\text{London}" \text{ and } y_{i-1} = \text{PER} \text{ and } y_i = \text{PER} \\
0 & \text{otherwise} 
\end{cases} \]

E.g., \( f_j(x, 2, \text{PER}, \text{PER}) = 1 \), \( f_j(x, 3, \text{PER}, -) = 0 \)
### More Bigram Indicator Features

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Jack</td>
<td>London went</td>
<td>to</td>
<td></td>
<td>Paris</td>
</tr>
<tr>
<td>y</td>
<td>PER</td>
<td>PER</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>y’</td>
<td>PER</td>
<td>LOC</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>y”</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
<td>-</td>
</tr>
<tr>
<td>x’</td>
<td>My trip</td>
<td>to</td>
<td>London</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

\[ f_1(\ldots) = 1 \text{ iff } x_i = "London" \text{ and } y_{i-1} = \text{PER} \text{ and } y_i = \text{PER} \]

\[ f_2(\ldots) = 1 \text{ iff } x_i = "London" \text{ and } y_{i-1} = \text{PER} \text{ and } y_i = \text{LOC} \]

\[ f_3(\ldots) = 1 \text{ iff } x_{i-1} \sim (\text{in}|\text{to}|\text{at})/ \text{ and } x_i \sim /\text{[A-Z]}/ \text{ and } y_i = \text{LOC} \]

\[ f_4(\ldots) = 1 \text{ iff } y_i = \text{LOC} \text{ and } \text{WORLD-CITIES}(x_i) = 1 \]

\[ f_5(\ldots) = 1 \text{ iff } y_i = \text{PER} \text{ and } \text{FIRST-NAMES}(x_i) = 1 \]
### More Bigram Indicator Features

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>Jack</td>
<td>London</td>
<td>went</td>
<td>to</td>
<td>Paris</td>
</tr>
<tr>
<td>(y)</td>
<td>PER</td>
<td>PER</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>(y')</td>
<td>PER</td>
<td>LOC</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>(y'')</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
<td>-</td>
</tr>
<tr>
<td>(x')</td>
<td>My</td>
<td>trip</td>
<td>to</td>
<td>London</td>
<td>...</td>
</tr>
</tbody>
</table>

\[
f_1(\ldots) = 1 \text{ iff } x_i = \text{”London” and } y_{i-1} = \text{PER and } y_i = \text{PER}
\]

\[
f_2(\ldots) = 1 \text{ iff } x_i = \text{”London” and } y_{i-1} = \text{PER and } y_i = \text{LOC}
\]

\[
f_3(\ldots) = 1 \text{ iff } x_{i-1} \sim/(in|to|at)/ \text{ and } x_i \sim/[A-Z]/ \text{ and } y_i = \text{LOC}
\]

\[
f_4(\ldots) = 1 \text{ iff } y_i = \text{LOC and } \text{WORLD-CITIES}(x_i) = 1
\]

\[
f_5(\ldots) = 1 \text{ iff } y_i = \text{PER and } \text{FIRST-NAMES}(x_i) = 1
\]
## More Bigram Indicator Features

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Jack</td>
<td>London</td>
<td>went</td>
<td>to</td>
<td>Paris</td>
</tr>
<tr>
<td>y</td>
<td>PER</td>
<td>PER</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>y'</td>
<td>PER</td>
<td>LOC</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>y''</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>x'</td>
<td>My trip to London</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ f_1(\ldots) = 1 \text{ iff } x_i = "London" \text{ and } y_{i-1} = \text{PER} \text{ and } y_i = \text{PER} \]

\[ f_2(\ldots) = 1 \text{ iff } x_i = "London" \text{ and } y_{i-1} = \text{PER} \text{ and } y_i = \text{LOC} \]

\[ f_3(\ldots) = 1 \text{ iff } x_{i-1} \sim/(\text{in|to|at})/ \text{ and } x_i \sim/[^{A-Z}]/ \text{ and } y_i = \text{LOC} \]

\[ f_4(\ldots) = 1 \text{ iff } y_i = \text{LOC} \text{ and } \text{WORLD-CITIES}(x_i) = 1 \]

\[ f_5(\ldots) = 1 \text{ iff } y_i = \text{PER} \text{ and } \text{FIRST-NAMES}(x_i) = 1 \]
More Bigram Indicator Features

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Jack</td>
<td>London</td>
<td>went</td>
<td>to</td>
<td>Paris</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>PER</td>
<td>PER</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
<td></td>
</tr>
<tr>
<td>y'</td>
<td>PER</td>
<td>LOC</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
<td></td>
</tr>
<tr>
<td>y''</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>x'</td>
<td>My trip to London ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ f_1(\ldots) = 1 \quad \text{iff} \quad x_i = "London" \quad \text{and} \quad y_{i-1} = \text{PER} \quad \text{and} \quad y_i = \text{PER} \]

\[ f_2(\ldots) = 1 \quad \text{iff} \quad x_i = "London" \quad \text{and} \quad y_{i-1} = \text{PER} \quad \text{and} \quad y_i = \text{LOC} \]

\[ f_3(\ldots) = 1 \quad \text{iff} \quad x_{i-1} \sim/(\text{in}|\text{to}|\text{at})/ \quad \text{and} \quad x_i \sim/[\text{A-Z}]/ \quad \text{and} \quad y_i = \text{LOC} \]

\[ f_4(\ldots) = 1 \quad \text{iff} \quad y_i = \text{LOC} \quad \text{and} \quad \text{WORLD-CITIES}(x_i) = 1 \]

\[ f_5(\ldots) = 1 \quad \text{iff} \quad y_i = \text{PER} \quad \text{and} \quad \text{FIRST-NAMES}(x_i) = 1 \]
More Bigram Indicator Features

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Jack</td>
<td>London</td>
<td>went</td>
<td>to</td>
<td>Paris</td>
</tr>
<tr>
<td>y</td>
<td>PER</td>
<td>PER</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>y'</td>
<td>PER</td>
<td>LOC</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>y''</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
<td>-</td>
</tr>
<tr>
<td>x'</td>
<td>My</td>
<td>trip</td>
<td>to</td>
<td>London</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ f_1(\ldots) = 1 \text{ iff } x_i = "London" \text{ and } y_{i-1} = \text{PER} \text{ and } y_i = \text{PER} \]

\[ f_2(\ldots) = 1 \text{ iff } x_i = "London" \text{ and } y_{i-1} = \text{PER} \text{ and } y_i = \text{LOC} \]

\[ f_3(\ldots) = 1 \text{ iff } x_{i-1} \sim/(\text{in|to|at})/ \text{ and } x_i \sim/[\text{A-Z}]/ \text{ and } y_i = \text{LOC} \]

\[ f_4(\ldots) = 1 \text{ iff } y_i = \text{LOC} \text{ and } \text{WORLD-CITIES}(x_i) = 1 \]

\[ f_5(\ldots) = 1 \text{ iff } y_i = \text{PER} \text{ and } \text{FIRST-NAMES}(x_i) = 1 \]
Representations Factored at Bigrams

\[ y: \text{PER PER } - - \text{LOC} \]
\[ x: \text{Jack London went to Paris} \]

\[ f(x, i, y_{i-1}, y_i) \]
- A \( d \)-dimensional feature vector of a label bigram at \( i \)
- Each dimension is typically a boolean indicator (0 or 1)

\[ f(x, y) = \sum_{i=1}^{n} f(x, i, y_{i-1}, y_i) \]
- A \( d \)-dimensional feature vector of the entire \( y \)
- Aggregated representation by summing bigram feature vectors
- Each dimension is now a **count** of a feature pattern
Linear Sequence Prediction

\[
predict(x_{1:n}) = \arg\max_{y \in \mathcal{Y}^n} w \cdot f(x, y)
\]

where

\[
f(x, y) = \sum_{i=1}^{n} f(x, i, y_{i-1}, y_i)
\]

▶ Note the linearity of the expression:

\[
w \cdot f(x, y) = w \cdot \sum_{i=1}^{n} f(x, i, y_{i-1}, y_i)
\]

\[
= \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i)
\]

▶ Next questions:

▶ How do we solve the \( \arg\max \) problem?
▶ How do we learn \( w \)?
Linear Sequence Prediction

predict(x_{1:n}) = \arg\max_{y \in \mathcal{Y}^n} w \cdot f(x, y)

where

f(x, y) = \sum_{i=1}^{n} f(x, i, y_{i-1}, y_i)

▶ Note the linearity of the expression:

w \cdot f(x, y) = w \cdot \sum_{i=1}^{n} f(x, i, y_{i-1}, y_i)

= \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i)

▶ Next questions:
  ▶ How do we solve the argmax problem?
  ▶ How do we learn w?
Linear Sequence Prediction

\[ \text{predict}(x_{1:n}) = \arg\max_{y \in \mathcal{Y}^n} w \cdot f(x, y) \]

where

\[ f(x, y) = \sum_{i=1}^{n} f(x, i, y_{i-1}, y_i) \]

- Note the linearity of the expression:

\[ w \cdot f(x, y) = w \cdot \sum_{i=1}^{n} f(x, i, y_{i-1}, y_i) \]

\[ = \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i) \]

- Next questions:
  - How do we solve the \( \arg\max \) problem?
  - How do we learn \( w \)?
Consider a fixed $w$. Given $x_{1:n}$ find:

$$\arg\max_{y \in \mathcal{Y}^n} \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i)$$

We can use the Viterbi algorithm, takes $O(n|\mathcal{Y}|^2)$

Intuition: output sequences that share bigrams will share scores
Intuition for Viterbi

- Consider a fixed $x_{1:n}$
- Assume we have the best sub-sequences up to position $i - 1$

$\begin{align*}
1 & & \cdots & & i - 1 & & i \\
& & & & \text{best subsequence with } y_{i-1} = \text{PER} & & \\
& & & & \text{best subsequence with } y_{i-1} = \text{LOC} & & \\
& & & & \text{best subsequence with } y_{i-1} = - & & \\
\end{align*}$

- What is the best sequence up to position $i$ with $y_i = \text{LOC}$?
Intuition for Viterbi

- Consider a fixed \( x_{1:n} \)
- Assume we have the best sub-sequences up to position \( i - 1 \)

\[
\begin{array}{c}
\ \\
1 & \ldots & i - 1 & i \\
\text{best subsequence with } y_{i-1} = \text{PER} \\
\text{best subsequence with } y_{i-1} = \text{LOC} \\
\text{best subsequence with } y_{i-1} = - \\
\end{array}
\]

- What is the best sequence up to position \( i \) with \( y_i = \text{LOC} \)?
Viterbi for Linear Factored Predictors

\[ \hat{y} = \arg \max_{y \in \mathcal{Y}^n} \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i) \]

- **Definition:** score of optimal sequence for \( x_{1:i} \) ending with \( a \in \mathcal{Y} \)

\[ \delta_i(a) = \max_{y \in \mathcal{Y}^i: y_i = a} \sum_{j=1}^{i} w \cdot f(x, j, y_{j-1}, y_j) \]

- Use the following recursions, for all \( a \in \mathcal{Y} \):

\[ \delta_1(a) = w \cdot f(x, 1, y_0 = \text{NULL}, a) \]
\[ \delta_i(a) = \max_{b \in \mathcal{Y}} \delta_{i-1}(b) + w \cdot f(x, i, b, a) \]

- The optimal score for \( x \) is \( \max_{a \in \mathcal{Y}} \delta_n(a) \)
- The optimal sequence \( \hat{y} \) can be recovered through **pointers**
Linear Factored Sequence Prediction

\[
predict(x_{1:n}) = \arg\max_{y \in Y^n} w \cdot f(x, y)
\]

- Factored representation, e.g. based on bigrams
- Flexible, arbitrary features of full \( x \) and the factors
- Efficient prediction using Viterbi
- **Next**, learning \( w \):
  - Maximum-Entropy Markov Models (local)
  - Conditional Random Fields (global)
  - Structured Perceptron (global)
  - Structured SVM (global)
Log-linear Models
for Sequence Prediction

\[ y \quad \text{PER} \quad \text{PER} \quad - \quad - \quad \text{LOC} \]
\[ x \quad \text{Jack} \quad \text{London} \quad \text{went} \quad \text{to} \quad \text{Paris} \]
Log-linear Models for Sequence Prediction

- Model the conditional distribution:

\[
Pr(y \mid x; w) = \frac{\exp \{w \cdot f(x, y)\}}{Z(x; w)}
\]

where

- \(x = x_1 x_2 \ldots x_n \in X^*\)
- \(y = y_1 y_2 \ldots y_n \in Y^*\) and \(Y = \{1, \ldots, L\}\)
- \(f(x, y)\) represents \(x\) and \(y\) with \(d\) features
- \(w \in \mathbb{R}^d\) are the parameters of the model
- \(Z(x; w)\) is a normalizer called the *partition function*

\[
Z(x; w) = \sum_{z \in Y^*} \exp \{w \cdot f(x, z)\}
\]

- To predict the best sequence

\[
predict(x_{1:n}) = \arg\max_{y \in Y^n} Pr(y \mid x)
\]
Log-linear Models: Name

- Let’s take the log of the conditional probability:

\[
\log \Pr(y \mid x; w) = \log \frac{\exp\{w \cdot f(x, y)\}}{Z(x; w)}
\]

\[
= w \cdot f(x, y) - \log \sum_y \exp\{w \cdot f(x, y)\}
\]

\[
= w \cdot f(x, y) - \log Z(x; w)
\]

- Partition function: \(Z(x; w) = \sum_y \exp\{w \cdot f(x, y)\}\)

- \(\log Z(x; w)\) is a constant for a fixed \(x\)

- In the log space, computations are linear, i.e., we model log-probabilities using a linear predictor
For tractability, assume $f(x, y)$ decomposes into bigrams:

$$f(x_1:n, y_1:n) = \sum_{i=1}^{n} f(x, i, y_{i-1}, y_i)$$

Given $w$, given $x_{1:n}$, find:

$$\arg\max_{y_{1:n}} \Pr(y_{1:n} \mid x_{1:n}; w) = \max_y \frac{\exp \left\{ \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i) \right\}}{Z(x; w)}$$

$$= \max_y \exp \left\{ \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i) \right\}$$

$$= \max_y \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i)$$

We can use the Viterbi algorithm.
Making Predictions with Log-Linear Models

- For tractability, assume $f(x, y)$ decomposes into bigrams:

  $$f(x_1:n, y_1:n) = \sum_{i=1}^{n} f(x, i, y_{i-1}, y_i)$$

- Given $w$, given $x_1:n$, find:

  $$\arg\max_{y_1:n} \Pr(y_1:n | x_1:n; w) = \max_y \frac{\exp \left\{ \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i) \right\}}{Z(x; w)}$$

  $$= \max_y \exp \left\{ \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i) \right\}$$

  $$= \max_y \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i)$$

- We can use the Viterbi algorithm
Parameter Estimation in Log-Linear Models

\[ \Pr(y \mid x; w) = \frac{\exp \{ w \cdot f(x, y) \}}{Z(x; w)} \]

How to estimate \( w \) given training data?

Two approaches:

- **MEMMs**: assume that \( \Pr(y \mid x; w) \) decomposes
- **CRFs**: assume that \( f(x, y) \) decomposes
Parameter Estimation in Log-Linear Models

\[ \Pr(y \mid x; w) = \frac{\exp\{w \cdot f(x, y)\}}{Z(x; w)} \]

How to estimate \( w \) given training data?

Two approaches:
- MEMMs: assume that \( \Pr(y \mid x; w) \) decomposes
- CRFs: assume that \( f(x, y) \) decomposes
Maximum Entropy Markov Models (MEMMs)
(McCallum, Freitag, Pereira '00)

▶ Similarly to HMMs:

\[
\Pr(y_{1:n} \mid x_{1:n}) = \Pr(y_{1} \mid x_{1:n}) \times \Pr(y_{2:n} \mid x_{1:n}, y_{1})
\]

\[
= \Pr(y_{1} \mid x_{1:n}) \times \prod_{i=2}^{n} \Pr(y_{i} \mid x_{1:n}, y_{1:i-1})
\]

\[
= \Pr(y_{1} \mid x_{1:n}) \times \prod_{i=2}^{n} \Pr(y_{i} \mid x_{1:n}, y_{i-1})
\]

▶ Assumption under MEMMs:

\[
\Pr(y_{i} \mid x_{1:n}, y_{1:i-1}) = \Pr(y_{i} \mid x_{1:n}, y_{i-1})
\]
Parameter Estimation in MEMMs

- Decompose sequential problem:

\[
\Pr(y_{1:n} | x_{1:n}) = \Pr(y_1 | x_{1:n}) \times \prod_{i=2}^{n} \Pr(y_i | x_{1:n}, i, y_{i-1})
\]

- Learn local log-linear distributions (i.e. MaxEnt)

\[
\Pr(y | x, i, y') = \frac{\exp\{w \cdot f(x, i, y', y)\}}{Z(x, i, y')}
\]

where

- \(x\) is an input sequence
- \(y\) and \(y'\) are tags
- \(f(x, i, y', y)\) is a feature vector of \(x\), the position to be tagged, the previous tag and the current tag

- Sequence learning reduced to multi-class logistic regression
Conditional Random Fields
(Lafferty, McCallum, Pereira 2001)

- Log-linear model of the conditional distribution:

\[
\Pr(y|x; w) = \frac{\exp\{w \cdot f(x, y)\}}{Z(x)}
\]

where

- \(x = x_1 x_2 \ldots x_n \in X^*\)
- \(y = y_1 y_2 \ldots y_n \in Y^*\) and \(Y = \{1, \ldots, L\}\)
- \(f(x, y)\) is a feature vector of \(x\) and \(y\)
- \(w\) are model parameters

- To predict the best sequence

\[
\hat{y} = \arg\max_{y \in Y^*} \Pr(y|x)
\]

- Assumption in CRF (for tractability):

\(f(x, y)\) decomposes into factors
CRFs as Factored Log-Linear Models

- For tractability, \( f(x, y) \) needs to decompose. For bigram factorizations:

\[
f(x_{1:n}, y_{1:n}) = \sum_{i=1}^{n} f(x, i, y_{i-1}, y_i)
\]

- The model form is:

\[
Pr(y|x_{1:n}; w) = \frac{\exp \{w \cdot f(x, y)\}}{Z(x, w)}
\]

\[
= \frac{\exp \{\sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i)\}}{Z(x, w)}
\]

where

\[
Z(x_{1:n}, w) = \sum_{z \in \mathcal{Y}^n} \exp \left\{ \sum_{i=1}^{n} w \cdot f(x, i, z_{i-1}, z_i) \right\}
\]
Parameter Estimation in CRFs

- Given a training set

\[ \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\} \]

estimate \( w \)

- Define the conditional log-likelihood of the data:

\[ L(w) = \sum_{k=1}^{m} \log \Pr(y^{(k)}|x^{(k)}; w) \]

- \( L(w) \) measures how well \( w \) explains the data. A good value for \( w \) will give a high value for \( \Pr(y^{(k)}|x^{(k)}; w) \) for all \( k = 1 \ldots m \).

- We want \( w \) that maximizes \( L(w) \)
Learning the Parameters of a CRF

- We pose it as a convex optimization problem
- Find:

\[ w^* = \arg\max_{w \in \mathbb{R}^D} L(w) - \frac{\lambda}{2} ||w||^2 \]

where

- The first term is the log-likelihood of the data
- The second term is a regularization term, it penalizes solutions with large norm (similar to norm-minimization in SVM)
- \( \lambda \) is a parameter to control the trade-off between fitting the data and model complexity
Learning the Parameters of a CRF

Find

\[ w^* = \arg \max_{w \in \mathbb{R}^D} L(w) - \frac{\lambda}{2} \| w \|^2 \]

In general there is no analytical solution to this optimization. We use iterative techniques, i.e. gradient-based optimization:

1. Initialize \( w = 0 \)
2. Take derivatives of \( L(w) - \frac{\lambda}{2} \| w \|^2 \), compute gradient
3. Move \( w \) in steps proportional to the gradient
4. Repeat steps 2 and 3 until convergence

Fast and scalable algorithms exist.
Computing the Gradient in CRFs

\[
\frac{\partial L(w)}{\partial w_j} = \frac{1}{m} \sum_{k=1}^{m} f_j(x^{(k)}, y^{(k)}) - \sum_{k=1}^{m} \sum_{y \in \mathcal{Y}^*} \Pr(y|x^{(k)}; w) f_j(x^{(k)}, y)
\]

where

\[
f(x, y) = \sum_{i=1}^{n} f_j(x, i, y_{i-1}, y_i)
\]

- First term: observed mean feature value
- Second term: expected feature value under current \( w \)
- In the optimal, observed = expected
Computing the Gradient in CRFs

- The first term is easy to compute, by counting explicitly

\[
\frac{1}{m} \sum_{k=1}^{m} \sum_{i} f_j(x, i, y_{i-1}^{(k)}, y_i^{(k)})
\]

- The second term is more involved,

\[
\sum_{k=1}^{m} \sum_{y \in \mathcal{Y}^*} \Pr(y|x^{(k)}; w) \sum_{i} f_j(x^{(k)}, i, y_{i-1}, y_i)
\]

because it sums over all sequences \( y \in \mathcal{Y}^* \)
Computing the Gradient in CRFs

For an example \((x^{(k)}, y^{(k)})\):

\[
\sum_{y \in \mathcal{Y}^n} \Pr(y|x^{(k)}; w) \sum_{i=1}^{n} f_j(x^{(k)}, i, y_{i-1}, y_i) = \sum_{i=1}^{n} \sum_{a,b \in \mathcal{Y}} \mu_{i}^{k}(a, b) f_j(x^{(k)}, i, a, b)
\]

where

\[
\mu_{i}^{k}(a, b) = \sum_{y \in \mathcal{Y}^n : y_{i-1}=a, y_i=b} \Pr(y|x^{(k)}; w)
\]

The quantities \(\mu_{i}^{k}\) can be computed efficiently in \(O(nL^2)\) using the forward-backward algorithm.
Forward-Backward for CRFs

- Assume fixed $x$. Calculate in $O(n|\mathcal{Y}|^2)$

\[ \mu_i(a, b) = \sum_{y \in \mathcal{Y}^n : y_{i-1} = a, y_i = b} \Pr(y|x; w), \ 1 \leq i \leq n; \ a, b \in \mathcal{Y} \]

- **Definition:** forward and backward quantities

\[
\alpha_i(a) = \sum_{y_1:i \in \mathcal{Y}^i : y_i = a} \exp \left\{ \sum_{j=1}^{i} w \cdot f(x, j, y_{j-1}, y_j) \right\} \\
\beta_i(b) = \sum_{y_{i:n} \in \mathcal{Y}^{(n-i+1)} : y_i = b} \exp \left\{ \sum_{j=i+1}^{n} w \cdot f(x, j, y_{j-1}, y_j) \right\} 
\]

- $Z = \sum_a \alpha_n(a)$

- $\mu_i(a, b) = \{\alpha_{i-1}(a) \ast \exp\{w \cdot f(x, i, a, b)\} \ast \beta_i(b) \ast Z^{-1}\}$

- Similarly to Viterbi, $\alpha_i(a)$ and $\beta_i(b)$ can be computed efficiently in a recursive manner
Forward-Backward for CRFs

- Assume fixed $x$. Calculate in $O(n|\mathcal{Y}|^2)$

$$
\mu_i(a, b) = \sum_{y \in \mathcal{Y}^n: y_{i-1} = a, y_i = b} \Pr(y|x; \mathbf{w}) \quad 1 \leq i \leq n; \ a, b \in \mathcal{Y}
$$

- **Definition:** forward and backward quantities

$$
\alpha_i(a) = \sum_{y_1:i \in \mathcal{Y}^i: y_i = a} \exp \left\{ \sum_{j=1}^i \mathbf{w} \cdot \mathbf{f}(x, j, y_{j-1}, y_j) \right\}
$$

$$
\beta_i(b) = \sum_{y_{i:n} \in \mathcal{Y}^{n-i+1}: y_i = b} \exp \left\{ \sum_{j=i+1}^n \mathbf{w} \cdot \mathbf{f}(x, j, y_{j-1}, y_j) \right\}
$$

$$
Z = \sum_{a} \alpha_n(a)
$$

$$
\mu_i(a, b) = \{ \alpha_{i-1}(a) \ast \exp\{ \mathbf{w} \cdot \mathbf{f}(x, i, a, b) \} \ast \beta_i(b) \ast Z^{-1} \}
$$

- Similarly to Viterbi, $\alpha_i(a)$ and $\beta_i(b)$ can be computed efficiently in a recursive manner
CRFs: summary so far

- Log-linear models for sequence prediction, \( \Pr(y|x; w) \)
- Computations factorize on label bigrams
- Model form:
  \[
  \arg\max_{y \in Y^*} \sum_i w \cdot f(x, i, y_{i-1}, y_i)
  \]
- Prediction: uses Viterbi (from HMMs)
- Parameter estimation:
  - Gradient-based methods, in practice L-BFGS
  - Computation of gradient uses forward-backward (from HMMs)
CRFs: summary so far

- Log-linear models for sequence prediction, $\Pr(y|x; w)$
- Computations factorize on label bigrams
- Model form:
  $$\arg\max_{y \in \mathcal{Y}^*} \sum_{i} w \cdot f(x, i, y_{i-1}, y_i)$$
- Prediction: uses Viterbi (from HMMs)
- Parameter estimation:
  - Gradient-based methods, in practice L-BFGS
  - Computation of gradient uses forward-backward (from HMMs)

Next Question: MEMMs or CRFs? HMMs or CRFs?
MEMMs and CRFs

**MEMMs:** \[ \Pr(y | x) = \prod_{i=1}^{n} \frac{\exp \{ w \cdot f(x, i, y_{i-1}, y_i) \}}{Z(x, i, y_{i-1}; w)} \]

**CRFs:** \[ \Pr(y | x) = \exp \{ \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i) \} \frac{1}{Z(x)} \]

- MEMMs locally normalized; CRFs globally normalized
- MEMM assume that \( \Pr(y_i | x_{1:n}, y_{1:i-1}) = \Pr(y_i | x_{1:n}, y_{i-1}) \)
- Both exploit the same factorization, i.e. same features
- Same computations to compute \( \arg\max_y \Pr(y | x) \)
- MEMMs are cheaper to train
- CRFs are easier to extend to other structures (next lecture)
HMMs for sequence prediction

- \( \mathbf{x} \) are the observations, \( \mathbf{y} \) are the hidden states
- HMMs model the joint distribution \( \Pr(\mathbf{x}, \mathbf{y}) \)
- Parameters: (assume \( \mathcal{X} = \{1, \ldots, k\} \) and \( \mathcal{Y} = \{1, \ldots, l\} \))
  - \( \pi \in \mathbb{R}^l, \pi_a = \Pr(y_1 = a) \)
  - \( T \in \mathbb{R}^{l \times l}, T_{a,b} = \Pr(y_i = b | y_{i-1} = a) \)
  - \( O \in \mathbb{R}^{l \times k}, O_{a,c} = \Pr(x_i = c | y_i = a) \)
- Model form

\[
\Pr(\mathbf{x}, \mathbf{y}) = \pi_{y_1} O_{y_1, x_1} \prod_{i=2}^{n} T_{y_{i-1}, y_i} O_{y_i, x_i}
\]

- Parameter Estimation: maximum likelihood by counting events and normalizing
HMMs and CRFs

- In CRFs: \( \hat{y} = \max_y \sum_i w \cdot f(x, i, y_{i-1}, y_i) \)

- In HMMs:
  \[
  \hat{y} = \max_y \pi_{y_1} O_{y_1, x_1} \prod_{i=2}^n T_{y_{i-1}, y_i} O_{y_i, x_i} \\
  = \max_y \log(\pi_{y_1} O_{y_1, x_1}) + \sum_{i=2}^n \log(T_{y_{i-1}, y_i} O_{y_i, x_i})
  \]

- An HMM can be “ported” into a CRF by setting:
  \[
  f_j(x, i, y, y') = \begin{cases} 
  \log(\pi_a) & i = 1 \text{ & } y' = a \\
  \log(T_{a,b}) & i > 1 \text{ & } y = a \text{ & } y' = b \\
  \log(O_{a,b}) & y' = a \text{ & } x_i = c
  \end{cases}
  \]

- Hence, HMM parameters \( \subset \) CRF parameters
HMMs and CRFs: main differences

- **Representation:**
  - HMM “features” are tied to the generative process.
  - CRF features are **very** flexible. They can look at the whole input $x$ paired with a label bigram $(y, y')$.
  - In practice, for prediction tasks, “good” discriminative features can improve accuracy **a lot**.

- **Parameter estimation:**
  - HMMs focus on explaining the data, both $x$ and $y$.
  - CRFs focus on the mapping from $x$ to $y$.
  - A priori, it is hard to say which paradigm is better.
  - Same dilemma as Naive Bayes vs. Maximum Entropy.
Structured Prediction

Perceptron, SVMs, CRFs
Learning Structured Predictors

- Goal: given training data
  \[
  \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}
  \]
  learn a predictor \( x \rightarrow y \) with small error on unseen inputs

- In a CRF:
  \[
  \arg\max_{y \in Y^*} P(y|x; w) = \frac{\exp\left\{\sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i)\right\}}{Z(x; w)}
  \]
  \[
  = \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i)
  \]
  - To predict new values, \( Z(x; w) \) is not relevant
  - Parameter estimation: \( w \) is set to maximize likelihood

- Can we learn \( w \) more directly, focusing on errors?
Learning Structured Predictors

- Goal: given training data
  \[ \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\} \]
  learn a predictor \( x \rightarrow y \) with small error on unseen inputs

- In a CRF:
  \[
  \arg\max_{y \in Y^*} P(y|x; w) = \frac{\exp \left\{ \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i) \right\}}{Z(x; w)} \\
  = \sum_{i=1}^{n} w \cdot f(x, i, y_{i-1}, y_i)
  \]
  - To predict new values, \( Z(x; w) \) is not relevant
  - Parameter estimation: \( w \) is set to maximize likelihood

- Can we learn \( w \) more directly, focusing on errors?
The Structured Perceptron
(Collins, 2002)

- Set $w = 0$
- For $t = 1 \ldots T$
  - For each training example $(x, y)$
    1. Compute $z = \text{argmax}_z \sum_{i=1}^n f(x, i, z_{i-1}, z_i)$
    2. If $z \neq y$
      \[
      w \leftarrow w + \sum_i f(x, i, y_{i-1}, y_i) - \sum_i f(x, i, z_{i-1}, z_i)
      \]
- Return $w$
The Structured Perceptron + Averaging
(Freund and Schapire, 1998) (Collins 2002)

- Set \( w = 0, \ w^a = 0 \)
- For \( t = 1 \ldots T \)
  - For each training example \( (x, y) \)
    1. Compute \( z = \arg\max_z \sum_{i=1}^n f(x, i, z_{i-1}, z_i) \)
    2. If \( z \neq y \)
      \[
      w \leftarrow w + \sum_i f(x, i, y_{i-1}, y_i) - \sum_i f(x, i, z_{i-1}, z_i)
      \]
  3. \( w^a = w^a + w \)
- Return \( w^a / mT \), where \( m \) is the number of training examples
Perceptron Updates: Example

Let \( y \) be the correct output for \( x \).

Say we predict \( z \) instead, under our current \( w \).

The update is:

\[
\begin{align*}
g &= f(x, y) - f(x, z) \\
    &= \sum_i f(x, i, y_{i-1}, y_i) - \sum_i f(x, i, z_{i-1}, z_i) \\
    &= f(x, 2, \text{PER, PER}) - f(x, 2, \text{PER, LOC}) \\
    &\quad + f(x, 3, \text{PER, -}) - f(x, 3, \text{LOC, -})
\end{align*}
\]

Perceptron updates are typically very sparse.
Properties of the Perceptron

- Online algorithm. Often much more efficient than “batch” algorithms
- If the data is separable, it will converge to parameter values with 0 errors
- Number of errors before convergence is related to a definition of margin. Can also relate margin to generalization properties
- In practice:
  1. Averaging improves performance a lot
  2. Typically reaches a good solution after only a few (say 5) iterations over the training set
  3. Often performs nearly as well as CRFs, or SVMs
### Averaged Perceptron Convergence

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>90.79</td>
</tr>
<tr>
<td>2</td>
<td>91.20</td>
</tr>
<tr>
<td>3</td>
<td>91.32</td>
</tr>
<tr>
<td>4</td>
<td>91.47</td>
</tr>
<tr>
<td>5</td>
<td>91.58</td>
</tr>
<tr>
<td>6</td>
<td>91.78</td>
</tr>
<tr>
<td>7</td>
<td>91.76</td>
</tr>
<tr>
<td>8</td>
<td>91.82</td>
</tr>
<tr>
<td>9</td>
<td>91.88</td>
</tr>
<tr>
<td>10</td>
<td>91.91</td>
</tr>
<tr>
<td>11</td>
<td>91.92</td>
</tr>
<tr>
<td>12</td>
<td>91.96</td>
</tr>
</tbody>
</table>

... (results on validation set for a parsing task)
Margin-based Structured Prediction

- Let $f(x, y) = \sum_{i=1}^{n} f(x, i, y_{i-1}, y_{i})$

- Model: $\arg\max_{y \in \mathcal{Y}^*} w \cdot f(x, y)$

- Consider an example $(x^{(k)}, y^{(k)})$:
  $\exists y \neq y^{(k)} : w \cdot f(x^{(k)}, y^{(k)}) < w \cdot f(x^{(k)}, y) \implies \text{error}$

- Let $y' = \arg\max_{y \in \mathcal{Y}^* : y \neq y^{(k)}} w \cdot f(x^{(k)}, y)$
  Define $\gamma_k = w \cdot (f(x^{(k)}, y^{(k)}) - f(x^{(k)}, y'))$

- The quantity $\gamma_k$ is a notion of margin on example $k$:
  $\gamma_k > 0 \iff$ no mistakes in the example
  high $\gamma_k \iff$ high confidence
Let $f(x, y) = \sum_{i=1}^{n} f(x, i, y_{i-1}, y_{i})$

Model: $\arg\max_{y \in Y^*} w \cdot f(x, y)$

Consider an example $(x^{(k)}, y^{(k)})$:  
$\exists y \neq y^{(k)} : w \cdot f(x^{(k)}, y^{(k)}) < w \cdot f(x^{(k)}, y) \implies \text{error}$

Let $y' = \arg\max_{y \in Y^*: y \neq y^{(k)}} w \cdot f(x^{(k)}, y)$  
Define $\gamma_{k} = w \cdot (f(x^{(k)}, y^{(k)}) - f(x^{(k)}, y'))$

The quantity $\gamma_{k}$ is a notion of margin on example $k$:  
$\gamma_{k} > 0 \iff \text{no mistakes in the example}$  
high $\gamma_{k} \iff $ high confidence
Margin-based Structured Prediction

- Let $f(x, y) = \sum_{i=1}^{n} f(x, i, y_{i-1}, y_i)$

- Model: $\arg\max_{y \in Y^*} w \cdot f(x, y)$

- Consider an example $(x^{(k)}, y^{(k)})$:
  \[ \exists y \neq y^{(k)} : w \cdot f(x^{(k)}, y^{(k)}) < w \cdot f(x^{(k)}, y) \implies \text{error} \]

- Let $y' = \arg\max_{y \in Y^*: y \neq y^{(k)}} w \cdot f(x^{(k)}, y)$
  Define $\gamma_k = w \cdot (f(x^{(k)}, y^{(k)}) - f(x^{(k)}, y'))$

- The quantity $\gamma_k$ is a notion of margin on example $k$:
  $\gamma_k > 0 \iff$ no mistakes in the example
  high $\gamma_k \iff$ high confidence
Mistake-augmented Margins
(Taskar et al, 2004)

<table>
<thead>
<tr>
<th>$x^{(k)}$</th>
<th>Jack</th>
<th>London</th>
<th>went</th>
<th>to</th>
<th>Paris</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y^{(k)}$</td>
<td>PER</td>
<td>PER</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>$y'$</td>
<td>PER</td>
<td>LOC</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>$y''$</td>
<td>PER</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$y''''$</td>
<td>-</td>
<td>-</td>
<td>PER</td>
<td>PER</td>
<td>-</td>
</tr>
</tbody>
</table>

- **Def:** $e(y, y') = \sum_{i=1}^{n} [y_i \neq y'_i]$  
  e.g., $e(y^{(k)}, y^{(k)}) = 0$, $e(y^{(k)}, y') = 1$, $e(y^{(k)}, y''') = 5$

- **We want a $w$ such that**

  $\forall y \neq y^{(k)} : w \cdot f(x^{(k)}, y^{(k)}) > w \cdot f(x^{(k)}, y) + e(y^{(k)}, y)$

  (the higher the error of $y$, the larger the separation should be)
Structured Hinge Loss

1. Define a mistake-augmented margin

\[ \gamma_{k,y} = w \cdot f(x^{(k)}, y^{(k)}) - w \cdot f(x^{(k)}, y) - e(y^{(k)}, y) \]

\[ \gamma_k = \min_{y \neq y^{(k)}} \gamma_{k,y} \]

2. Define loss function on example \( k \) as:

\[ L(w, x^{(k)}, y^{(k)}) = \max_{y \in \mathcal{Y}^*} \left\{ e(y^{(k)}, y) - w \cdot f(x^{(k)}, y^{(k)}) - w \cdot f(x^{(k)}, y) \right\} \]

3. Leads to an SVM for structured prediction

4. Given a training set, find:

\[ \arg\min_{w \in \mathbb{R}^D} \sum_{k=1}^{m} L(w, x^{(k)}, y^{(k)}) + \frac{\lambda}{2} \|w\|^2 \]
Regularized Loss Minimization

- Given a training set \( \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\} \).
  Find:
  \[
  \arg\min_{w \in \mathbb{R}^D} \sum_{k=1}^{m} L(w, x^{(k)}, y^{(k)}) + \frac{\lambda}{2} \|w\|^2
  \]

- Two common loss functions \( L(w, x^{(k)}, y^{(k)}) \):
  - Log-likelihood loss (CRFs)
    \[
    - \log P(y^{(k)} | x^{(k)}; w)
    \]
  - Hinge loss (SVMs)
    \[
    \max_{y \in \mathcal{Y}^*} \left( e(y^{(k)}, y) - w \cdot (f(x^{(k)}, y^{(k)}) - f(x^{(k)}, y)) \right)
    \]
Learning Structure Predictors: summary so far

- Linear models for sequence prediction

\[
\text{argmax}_{y \in \mathcal{Y}^*} \sum_{i} w \cdot f(x, i, y_{i-1}, y_i)
\]

- Computations factorize on label bigrams
  - Decoding: using Viterbi
  - Marginals: using forward-backward

- Parameter estimation:
  - Perceptron, Log-likelihood, SVMs
  - Extensions from classification to the structured case
  - Optimization methods:
    - Stochastic (sub)gradient methods (LeCun et al 98) (Shalev-Shwartz et al. 07)
    - Exponentiated Gradient (Collins et al 08)
    - SVM Struct (Tsochantaridis et al. 04)
    - Structured MIRA (McDonald et al 05)
Generic Structure Prediction
It is easy to extend the scope of features to $k$-grams

\[ f(x, i, y_{i-k+1:i-1}, y_i) \]

In general, think of state $\sigma_i$ remembering relevant history

- $\sigma_i = y_{i-1}$ for bigrams
- $\sigma_i = y_{i-k+1:i-1}$ for $k$-grams
- $\sigma_i$ can be the state at time $i$ of a deterministic automaton generating $y$

The structured predictor is

\[
\text{argmax}_{y \in \mathcal{Y}^*} \sum_i w \cdot f(x, i, \sigma_i, y_i)
\]

Viterbi and forward-backward extend naturally, in $O(nL^k)$
Dependency Structures

* John saw a movie that he liked today

- Directed arcs represent dependencies between a head word and a modifier word.
- E.g.:
  - movie modifies saw,
  - John modifies saw,
  - today modifies saw
Dependency Parsing: arc-factored models

(McDonald et al. 2005)

- Parse trees decompose into single dependencies \( \langle h, m \rangle \)

\[
\begin{align*}
\argmax & \sum_{y \in \mathcal{Y}(x)} w \cdot f(x, h, m) \\
\end{align*}
\]

- Some features:
  \( f_1(x, h, m) = \text{["saw" \rightarrow "movie"]} \)
  \( f_2(x, h, m) = \text{[distance = +2]} \)

- Tractable inference algorithms exist (tomorrow’s lecture)
Linear Structured Prediction

- Sequence prediction (bigram factorization)
  \[
  \arg\max_{y \in \mathcal{Y}(x)} \sum_i w \cdot f(x, i, y_{i-1}, y_i)
  \]

- Dependency parsing (arc-factored)
  \[
  \arg\max_{y \in \mathcal{Y}(x)} \sum_{\langle h, m \rangle \in y} w \cdot f(x, h, m)
  \]

- In general, we can enumerate parts \( r \in y \)
  \[
  \arg\max_{y \in \mathcal{Y}(x)} \sum_{r \in y} w \cdot f(x, r)
  \]
Linear Structured Prediction Framework

- Abstract models of structures
  - Input domain $\mathcal{X}$, output domain $\mathcal{Y}$
  - A choice of factorization, $r \in \mathcal{Y}$
  - Features: $f(x, r) \to \mathbb{R}^d$

- The linear prediction model, with $w \in \mathbb{R}^d$
  \[
  \arg\max_{y \in \mathcal{Y}(x)} \sum_{r \in y} w \cdot f(x, r)
  \]

- Generic algorithms for Perceptron, CRF, SVM
  - Require tractable inference algorithms