Learning Structured Predictors

Xavier Carreras

https://dmetrics.com
Outline

Part I
  Introduction
  Four Approaches to Sequence Prediction
  Greedy Sequence Prediction

Part II
  Factored Sequence Prediction
  Algorithms for Factored Models
  Log-linear Factored Models

Part III
  Structured Perceptron
  Log-linear Models and CRFs
  Dependency Parsing
  Summary and Conclusion
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 classification: \( y \in \{-1, +1\} \)
  - Multiclass classification: \( y \in \{1, 2, \ldots, L\} \)

- 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

\[
\begin{align*}
y & \quad \text{PER} \quad - \quad \text{QNT} \quad - \quad - \quad \text{ORG} \quad \text{ORG} \quad - \quad \text{TIME} \\
x & \quad \text{Jim} \quad \text{bought} \quad 300 \quad \text{shares} \quad \text{of} \quad \text{Acme Corp.} \quad \text{in} \quad 2006
\end{align*}
\]

\[
\begin{align*}
y & \quad \text{PER} \quad \text{PER} \quad - \quad - \quad \text{LOC} \\
x & \quad \text{Jack} \quad \text{London} \quad \text{went} \quad \text{to} \quad \text{Paris}
\end{align*}
\]

\[
\begin{align*}
y & \quad \text{PER} \quad \text{PER} \quad - \quad - \quad \text{LOC} \\
x & \quad \text{Paris} \quad \text{Jackson} \quad \text{went} \quad \text{to} \quad \text{London}
\end{align*}
\]

\[
\begin{align*}
y & \quad \text{PER} \quad - \quad - \quad \text{LOC} \\
x & \quad \text{Jackie} \quad \text{went} \quad \text{to} \quad \text{Lisdon}
\end{align*}
\]
Part-of-speech Tagging

\[
y \quad \text{NOUN} \quad \text{NOUN} \quad \text{VERB} \quad \text{NOUN} \quad .
\]

\[
x \quad \text{Fruit} \quad \text{flies} \quad \text{like} \quad \text{bananas} \quad .
\]
Syntactic Dependency Parsing

x are sentences
y are syntactic dependency trees
Machine Translation

\[
\begin{align*}
x & \quad \text{are sentences in some source language (e.g. French)} \\
y & \quad \text{are sentence translations in a target language (e.g. English)}
\end{align*}
\]
Object Detection

\( x \) are images

\( y \) are grids labeled with object types

(Kumar and Hebert, 2003)
Object Detection

\( x \) are images

\( y \) are grids labeled with object types

(Kumar and Hebert, 2003)
Today’s Goals

▶ Introduce basic concepts for structured prediction
  ▶ We will focus on sequence prediction

▶ What can we can borrow from standard classification?
  ▶ Learning paradigms and algorithms, in essence, work here too
  ▶ However, computations behind algorithms are prohibitive

▶ Today’s main topics:
  ▶ Transition systems versus factored models
  ▶ Feature representations of structured input-output pairs
  ▶ Prediction algorithms
  ▶ Learning algorithms: Perceptron and CRF
  ▶ Local and global learning losses

▶ Topics not covered:
  ▶ NLP task overviews, evaluation, state-of-the-art systems
  ▶ Hidden (structured) representations
  ▶ Unsupervised learning (induction of labeled sequences and trees)
Outline

Part I
  Introduction
  Four Approaches to Sequence Prediction
  Greedy Sequence Prediction

Part II
  Factored Sequence Prediction
  Algorithms for Factored Models
  Log-linear Factored Models

Part III
  Structured Perceptron
  Log-linear Models and CRFs
  Dependency Parsing
  Summary and Conclusion
Sequence Prediction

y  PER    PER    -    -    LOC
x  Jack    London    went    to    Paris
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?
Exponentially-many Solutions

- Let $\mathcal{Y} = \{-, \text{PER}, \text{LOC}\}$

- The solution space (all output sequences):

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

- Each path is a possible solution

- For an input sequence of size $n$, there are $|\mathcal{Y}|^n$ possible outputs
Exponentially-many Solutions

- Let $\mathcal{Y} = \{-, \text{PER}, \text{LOC}\}$

- The solution space (all output sequences):

  Each path is a possible solution

- For an input sequence of size $n$, there are $|\mathcal{Y}|^n$ possible outputs
Approach 1: Label Classifiers

- Scoring of individual labels at each position
  \[
  \hat{y}_t = \arg\max_{l \in \{\text{LOC, PER, -}\}} \text{score}(x, t, l)
  \]

- For linear models, \(\text{score}(x, t, l) = w \cdot f(x, t, l)\)
  - \(f(x, t, l) \in \mathbb{R}^d\) represents an assignment of label \(l\) for \(x_t\)
  - \(w \in \mathbb{R}^d\) is a vector of parameters (learned), has a weight for each feature in \(f\)

- Can capture interactions between full input \(x\) and one output label \(l\)
  e.g.: current word, surrounding words, capitalization, prefix-suffix, gazetteer, ...

- Can not capture interactions between output labels!
Approach 2: Transition-based Sequence Prediction

▶ Score one label at a time, left-to-right, conditioning on previous predictions:

\[ \hat{y}_t = \arg\max_{l \in \{\text{LOC, PER, -}\}} \text{score}(x, t, l, \hat{y}_{1:t-1}) \]

▶ Captures interactions between full input \( x \) and prefixes of the output sequence
▶ Greedy predictions, prone to search errors even with beam search
▶ Why left-to-right and not right-to-left?
Approach 3: Factored Sequence Prediction

- Scoring of label bigrams (pairs of adjacent labels) at each position:

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

- Output sequence factored into label bigrams
- Captures interactions between full input $x$ and factors of output sequence
- Prediction is exact for many types of factorizations
Approach 4: Re-Ranking

\[ \hat{y} = \arg\max_{y \in A(\mathcal{Y}_n)} \text{score}(x, y) \]

- Scoring of full inputs and outputs: very expressive!
- Relies on an active set \( A(\mathcal{Y}_n) \) of full outputs, enumerated exhaustively
- A base model is used to select active set
  - The base model follows one of the previous approaches
## Sequence Prediction: Summary of Approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Input-Output Representation</th>
<th>Exact Prediction?</th>
</tr>
</thead>
<tbody>
<tr>
<td>label classifiers</td>
<td>only individual labels</td>
<td>yes</td>
</tr>
<tr>
<td>transition-based</td>
<td>full history of decisions</td>
<td>no (greedy, beam search)</td>
</tr>
<tr>
<td>factored</td>
<td>label factors</td>
<td>yes</td>
</tr>
<tr>
<td>re-ranking</td>
<td>full</td>
<td>limited to active set</td>
</tr>
</tbody>
</table>

**Take home message 1:** expressivity-tractability trade-off

**Take home message 2:** always pick the simplest approach that suits the task at hand
## Sequence Prediction: Summary of Approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Input-output representation</th>
<th>Exact Prediction?</th>
</tr>
</thead>
<tbody>
<tr>
<td>label classifiers</td>
<td>only individual labels</td>
<td>yes</td>
</tr>
<tr>
<td>transition-based</td>
<td>full history of decisions</td>
<td>no (greedy, beam search)</td>
</tr>
<tr>
<td>factored</td>
<td>label factors</td>
<td>yes</td>
</tr>
<tr>
<td>re-ranking</td>
<td>full</td>
<td>limited to active set</td>
</tr>
</tbody>
</table>

**Take home message 1:** expressivity-tractability trade-off
**Take home message 2:** always pick the simplest approach that suits the task at hand
Outline

Part I
  Introduction
  Four Approaches to Sequence Prediction
  Greedy Sequence Prediction

Part II
  Factored Sequence Prediction
  Algorithms for Factored Models
  Log-linear Factored Models

Part III
  Structured Perceptron
  Log-linear Models and CRFs
  Dependency Parsing
  Summary and Conclusion
Greedy Sequence Prediction

- Run a greedy classifier left-to-right:
  - For $t = 1 \ldots n$:
    $$\hat{y}_t = \arg\max_{l \in \{\text{loc, per, -}\}} \text{score}(x, t, l, \hat{y}_{1:t-1})$$

- What is the form of $\text{score}(x, t, l, \hat{y}_{1:t-1})$?
  - We focus on linear scoring functions: $\text{score}(x, t, l, \hat{y}_{1:t-1}) = w \cdot f(x, t, l, \hat{y}_{1:t-1})$
Representations in Greedy Sequence Prediction

- In linear greedy sequence prediction, at time $t$
  \[
  \text{score}(x, t, l, \hat{y}_{1:t-1}) = w \cdot f(x, t, l, \hat{y}_{1:t-1})
  \]

- $w \in \mathbb{R}^d$ is a parameter vector, to be learned
- $f(x, t, l, \hat{y}_{1:t-1}) \in \mathbb{R}^d$ is a feature vector
- Goal: guess the correct $l$ at position $t$
- How to construct $f(x, t, l, \hat{y}_{1:t-1})$?
  - New trend: representation learning
  - Old school: manually with feature templates
In linear greedy sequence prediction, at time $t$

$$\text{score}(x, t, l, \hat{y}_{1:t-1}) = w \cdot f(x, t, l, \hat{y}_{1:t-1})$$

- $w \in \mathbb{R}^d$ is a parameter vector, to be learned
- $f(x, t, l, \hat{y}_{1:t-1}) \in \mathbb{R}^d$ is a feature vector
- Goal: guess the correct $l$ at position $t$
- How to construct $f(x, t, l, \hat{y}_{1:t-1})$?
  - New trend: representation learning
  - Old school: manually with feature templates
Indicator Features for One Label Only

- \( f(x, t, l) \) is a vector of \( d \) features representing label \( l \) for \( x_t \)
- What’s in a feature \( f_j(x, t, l) \)?
  - Anything we can compute using \( x \) and \( t \) and \( l \)
  - Anything that indicates whether \( l \) is (not) a good label for \( x_t \)
- **Indicator features**: binary-valued features looking at:
  - a simple pattern of \( x \) and target position \( t \)
  - and the candidate label \( l \) for position \( t \)

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

\[
f_k(x, t, l) = \begin{cases} 
1 & \text{if } x_{t+1} = \text{went and } l = \text{LOC} \\
0 & \text{otherwise}
\end{cases}
\]

- Indicator features produce **sparse** feature vectors
Feature Templates

- Feature templates generate many indicator features.
- A feature template is identified by a type, and a number of values.
  - Example: template \( \text{WORD} \) indicates the current word
    \[
    f_{\langle \text{WORD},a,w \rangle}(x,t,l) = \begin{cases} 
    1 & \text{if } x_t = w \text{ and } l = a \\
    0 & \text{otherwise}
    \end{cases}
    \]
  - A feature of this type is identified by the tuple \( \langle \text{WORD}, a, w \rangle \).
  - Generates a feature for every label \( a \in Y \) and every word \( w \).
- Feature vectors and weight vectors are indexed by feature tuples.
Feature Templates

- Feature templates generate many indicator features
- A feature template is identified by a type, and a number of values
  - Example: template `WORD` indicates the current word
    \[
    f_{\langle \text{WORD},a,w \rangle}(x,t,l) = \begin{cases} 
    1 & \text{if } x_t = w \text{ and } l = a \\
    0 & \text{otherwise} 
    \end{cases}
    \]
  - A feature of this type is identified by the tuple \( \langle \text{WORD}, a, w \rangle \)
  - Generates a feature for every label \( a \in \mathcal{Y} \) and every word \( w \)
- Feature vectors and weight vectors are indexed by feature tuples
- In feature-based models:
  - Define feature templates manually
  - Instantiate the templates on every set of values in the training data
    \( \rightarrow \) generates a very high-dimensional feature space
  - Define parameter vector \( w \) indexed by such feature tuples
  - Let the learning algorithm choose the relevant features
More Features for NE Recognition

Jack London went to Paris

In practice, construct $f(x, t, l)$ by . . .

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

- Define feature templates by combining patterns with labels $l$
- Generate actual features by instantiating templates on training data
Feature Templates in Greedy Sequence Prediction

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

- \( f(x, t, l, \hat{y}_{1:t-1}) \) has access to all preceding labels
- Example: A template for word + current label + previous label:

\[
f_{(WB, a, b, w)}(x, t, l, \hat{y}_{1:t-1}) = \begin{cases} 
1 & \text{if } x_t = w \text{ and } \\ 
& \hat{y}_{t-1} = a \text{ and } l = b \\
0 & \text{otherwise}
\end{cases}
\]

- In practice:
  - Preceding labels next to \( t \)
  - Bag-of-labels in \( \hat{y}_{1:t-1} \)
  - Combinations with other features
- Neural networks automatically induce “good” features out of \( x \) and \( \hat{y}_{1:t-1} \)
Transition Systems (general form)

- Given an input $x$, a transition system defines:
  - A set of states $S(x)$
  - An initial state $s_0 \in S(x)$, and a set of final states $S_\infty \subseteq S(x)$
  - A set of allowed actions $A(s, x)$ for all $s \in S(x)$
  - A transition function $\text{transition} : s \times a \to s'$
  - A scoring function: $\text{score} : x \times s \times a \to \mathbb{R}$

- To predict output $y$ from input $x$:
  - $s = s_0$
  - while $s \not\in S_\infty$:
    - $a = \arg\max_{a \in A(s, x)} \text{score}(x, s, a)$
    - $s = \text{transition}(s, a)$
  - extract $y$ from $s$

- Simple, very fast and expressive! Very popular in NLP:
  - Greedy sequence prediction (one label at a time, left-to-right or right-to-left)
  - Shift-reduce parsing (more later)
  - Word segmentation, machine translation, ...
Greedy Predictions are not Optimal, even with Beam Search

- Greedy sequence predictions can not undo decisions at a later stage
- Sometimes the model is right at a global scope, but not at each greedy step!
- Solution: Beam Search
  - General *local search* method
  - Maintains several good hypotheses, instead of just the best one
  - Many strategies, sometimes specific to the task and transition system
  - Empirically, it often improves over greedy search
Greedy Predictions are not Optimal, even with Beam Search

- Greedy sequence predictions can not undo decisions at a later stage
- Sometimes the model is right at a global scope, but not at each greedy step!
- Solution: Beam Search
  - General *local search* method
  - Maintains several good hypotheses, instead of just the best one
  - Many strategies, sometimes specific to the task and transition system
  - Empirically, it often improves over greedy search
Outline

Part I
  Introduction
  Four Approaches to Sequence Prediction
  Greedy Sequence Prediction

Part II
  Factored Sequence Prediction
  Algorithms for Factored Models
  Log-linear Factored Models

Part III
  Structured Perceptron
  Log-linear Models and CRFs
  Dependency Parsing
  Summary and Conclusion
Factored Sequence Predictors

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

Next questions:

- What is the form of \( \text{score}(x, i, a, b) \)?
  We will use linear scoring functions: \( \text{score}(x, i, a, b) = w \cdot f(x, i, a, b) \)

- There are exponentially-many sequences \( y \) for a given \( x \), how do we solve the \( \arg \max \) problem?
Representations Factored at Bigrams

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

1. \( \text{score}(\textbf{x}, i, a, b) = \mathbf{w} \cdot \mathbf{f}(\textbf{x}, i, a, b) \)

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

3. \( \mathbf{f}(\textbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \mathbf{f}(\textbf{x}, i, y_{i-1}, y_i) \)
   - A \( d \)-dimensional feature vector of the entire \( \mathbf{y} \)
   - Aggregated representation by summing bigram feature vectors
   - Each dimension is now a count of a feature pattern
Representations Factored at Bigrams

\[
\begin{align*}
&\mathbf{y}: \text{PER} \quad \text{PER} \quad - \quad - \quad \text{LOC} \\
&\mathbf{x}: \text{Jack} \quad \text{London} \quad \text{went} \quad \text{to} \quad \text{Paris}
\end{align*}
\]

- \(\text{score}(\mathbf{x}, i, a, b) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, a, b)\)
- \(\mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)\)
  - A \(d\)-dimensional feature vector of a label bigram at \(i\)
  - Each dimension is typically a boolean indicator (0 or 1)

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

argmax \quad \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) \quad \text{where} \quad \mathbf{f}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_{i})

- Note the linearity of the expression:

\[
\text{score}(\mathbf{x}, \mathbf{y}) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) \\
= \mathbf{w} \cdot \sum_{i=1}^{n} \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_{i}) \\
= \sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_{i}) \\
= \sum_{i=1}^{n} \text{score}(\mathbf{x}, i, y_{i-1}, y_{i})
\]
Assume we have a score function \(\text{score}(x, i, a, b)\)

Given \(x_{1:n}\) find:

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

Use the Viterbi algorithm, takes \(O(n|\mathcal{Y}|^2)\)

Notational change: since \(x_{1:n}\) is fixed we will use

\[s(i, a, b) = \text{score}(x, i, a, b)\]
Outline

Part I
  Introduction
  Four Approaches to Sequence Prediction
  Greedy Sequence Prediction

Part II
  Factored Sequence Prediction
  Algorithms for Factored Models
  Log-linear Factored Models

Part III
  Structured Perceptron
  Log-linear Models and CRFs
  Dependency Parsing
  Summary and Conclusion
Viterbi for Factored Sequence Models

- Given scores $s(i, a, b)$ for each position $i$ and output bigram $a, b$, find:

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

- Intuition: consider this example $x$ and two alternative solutions $y$ and $y'$:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>Jack</td>
<td>London</td>
<td>went</td>
<td>to</td>
<td>Paris</td>
<td>before</td>
<td>visiting</td>
<td>Lisbon</td>
</tr>
<tr>
<td>$y$</td>
<td>PER</td>
<td>LOC</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>$y'$</td>
<td>PER</td>
<td>PER</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
</tbody>
</table>

- What is the score of $y'$ relative to the score of $y$?

$$s(x, y') = s(x, y) + \quad -$$

$$+ \quad -$$
Viterbi for Factored Sequence Models

- Given scores $s(i, a, b)$ for each position $i$ and output bigram $a, b$, find:

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

- Intuition: consider this example $x$ and two alternative solutions $y$ and $y'$:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>Jack</td>
<td>London</td>
<td>went</td>
<td>to</td>
<td>Paris</td>
<td>before</td>
<td>visiting</td>
<td>Lisbon</td>
</tr>
<tr>
<td>$y$</td>
<td>PER</td>
<td>LOC</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
<tr>
<td>$y'$</td>
<td>PER</td>
<td>PER</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
<td>-</td>
<td>-</td>
<td>LOC</td>
</tr>
</tbody>
</table>

- What is the score of $y'$ relative to the score of $y$?

$$s(x, y') = s(x, y) + s(2, \text{PER, PER}) - s(2, \text{PER, LOC}) + s(3, \text{PER, -}) - s(3, \text{LOC, -})$$

output sequences that share bigrams also share their scores
Viterbi recurrence

- Viterbi is a dynamic programming algorithm that uses the following recurrence

- Assume that, for a certain position $i$ and each label $l \in \mathcal{Y}$, we have the best sub-sequence from positions 1 to $i$ ending with label $l$:

  $1 \quad \cdots \quad i \quad i + 1$

  
  best subsequence with $y_i = \text{PER}$

  best subsequence with $y_i = \text{LOC}$

  best subsequence with $y_i = -$ 

- What is the best sequence up to position $i + 1$ with $y_{i+1} = \text{LOC}$?
Viterbi recurrence

- Viterbi is a dynamic programming algorithm that uses the following recurrence
- Assume that, for a certain position $i$ and each label $l \in \mathcal{Y}$, we have the best sub-sequence from positions $1$ to $i$ ending with label $l$:

$$
\begin{array}{cccc}
1 & \cdots & i & i+1 \\
\hline \\
\text{best subsequence with } y_i = \text{PER} \\
\text{best subsequence with } y_i = \text{LOC} \\
\text{best subsequence with } y_i = - \\
\end{array}
$$

- What is the best sequence up to position $i + 1$ with $y_{i+1} = \text{LOC}$?
Viterbi recurrence

- Viterbi is a dynamic programming algorithm that uses the following recurrence
- Assume that, for a certain position $i$ and each label $l \in \mathcal{Y}$, we have the best sub-sequence from positions 1 to $i$ ending with label $l$:

$$
\begin{align*}
1 & \cdots & i & i+1 \\
\text{best subsequence with } y_i = \text{PER} & & s(i+1, \text{PER, LOC}) \\
\text{best subsequence with } y_i = \text{LOC} & & s(i+1, \text{LOC, LOC}) \\
\text{best subsequence with } y_i = - & & s(i+1, -, \text{LOC})
\end{align*}
$$

- What is the best sequence up to position $i+1$ with $y_{i+1} = \text{LOC}$?
Viterbi for Factored Sequence Models

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

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

\[ \delta(i, a) = \max_{y \in \mathcal{Y}^i : y_i = a} \sum_{j=1}^i s(j, y_{j-1}, y_j) \]

▶ Use the following recursions, for all \(a \in \mathcal{Y}\), for \(i = 2 \ldots n\):

\[ \delta(1, a) = s(1, y_0 = \text{NULL}, a) \]
\[ \delta(i, a) = \max_{b \in \mathcal{Y}} \delta(i - 1, b) + s(i, b, a) \]

▶ The optimal score for \(x\) is \(\max_{a \in \mathcal{Y}} \delta(n, a)\)

▶ The optimal sequence \(\hat{y}\) can be recovered through *back-pointers*

▶ Cost: \(O(n |\mathcal{Y}|^2)\)
### Viterbi for Factored Sequence Models

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

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

\[
\delta(i, a) = \max_{y \in \mathcal{Y}^{i-1}: y_i = a} \sum_{j=1}^{i} s(j, y_{j-1}, y_j)
\]

**Use the following recursions, for all \( a \in \mathcal{Y} \), for \( i = 2 \ldots n \):**

\[
\begin{align*}
\delta(1, a) &= s(1, y_0 = \text{NULL}, a) \\
\delta(i, a) &= \max_{b \in \mathcal{Y}} \delta(i - 1, b) + s(i, b, a)
\end{align*}
\]

**The optimal score for \( x \) is** \( \max_{a \in \mathcal{Y}} \delta(n, a) \)

**The optimal sequence \( \hat{y} \) can be recovered through back-pointers**

**Homework:** rewrite the Viterbi equations such that the algorithm proceeds right-to-left. Observe that the factored model remains the same (i.e. it is not a directional model)
Variations of Viterbi

- **Sparse Viterbi**
  - Only a few labels in $\mathcal{Y}$ apply to a position
  - Only a few label bigrams are possible
  - A sparse implementation cuts the $O(|\mathcal{Y}|^2)$ factor

- **Higher-order Viterbi**: factorize at trigrams instead of bigrams
  - Cost $O(n|\mathcal{Y}|^3)$
  - Very common in POS tagging (using sparse Viterbi to cut the $O(|\mathcal{Y}|^3)$ cost factor)

- **$k$-best Viterbi**: return the best $k$ sequences (not just the single best)
  - Used in re-ranking approaches and some loss functions

- **Forward-Backward**: Viterbi for sum-product computations (instead of max-sum)
The Viterbi algorithm solves a max-sum recurrence

\[
\max_{\mathbf{y} \in Y^n} \sum_{i=1}^{n} s(i, y_{i-1}, y_i)
\]

The sum-product recurrence is also very useful (more later)

\[
\sum_{\mathbf{y} \in Y^n} \prod_{i=1}^{n} s(i, y_{i-1}, y_i)
\]

The same style of dynamic programming works
Forward Algorithm

\[ \sum_{y \in \mathcal{Y}^n} \prod_{i=1}^{n} s(i, y_{i-1}, y_i) \]

**Definition:** forward quantities

\[ \alpha(i, a) = \sum_{y_1: \in \mathcal{Y}^i: y_i = a} \prod_{j=1}^{i} s(j, y_{j-1}, y_j) \]

**Use the following recursions, for all** \( a \in \mathcal{Y}, \) for \( i = 2 \ldots n:\)

\[ \alpha(i, a) = s(1, y_0 = \text{NULL}, a) \]

\[ \alpha(i, a) = \sum_{b \in \mathcal{Y}} \alpha(i - 1, b) \ast s(i, b, a) \]

**The total sum-product is** \( \sum_a \alpha(n, a) \)

**Like Viterbi, the forward algorithm runs in** \( O(n|\mathcal{Y}|^2) \)
Backward Algorithm

\[ \sum_{\mathbf{y} \in \mathcal{Y}^n} \prod_{i=1}^{n} s(i, y_{i-1}, y_i) \]

- **Definition:** backward quantities

\[ \beta(i, a) = \sum_{\mathbf{y}_{i:n} \in \mathcal{Y}^{(n-i+1)}: y_i = a} \prod_{j=i+1}^{n} s(j, y_{j-1}, y_j) \]

- Now the recursions run **backwards**! For all \( a \in \mathcal{Y} \), for \( i = n - 1 \ldots 1 \):

\[
\begin{align*}
\beta(n, a) & = 1 \\
\beta(i, a) & = \sum_{b \in \mathcal{Y}} s(i, a, b) * \beta(i + 1, b)
\end{align*}
\]

- The total sum-product is \( \sum_{a} s(1, y_0 = \text{NULL}, a) * \beta(1, a) \)

- Like Viterbi and forward algorithms, the backward algorithm runs in \( O(n|\mathcal{Y}|^2) \)
Outline

Part I
  Introduction
  Four Approaches to Sequence Prediction
  Greedy Sequence Prediction

Part II
  Factored Sequence Prediction
  Algorithms for Factored Models
  Log-linear Factored Models

Part III
  Structured Perceptron
  Log-linear Models and CRFs
  Dependency Parsing
  Summary and Conclusion
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

- \( f(x, y) \) 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*

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

- To predict the best sequence

\[ \arg\max_{y \in \mathcal{Y}^n} \Pr(y \mid x) \]

\[ 42/83 \]
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: \( Z(x; w) = \sum_z \exp\{w \cdot f(x, z)\} \)

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

In the log space, computations are linear, i.e., we model log-probabilities using a linear predictor
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:

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

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

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

- 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:

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

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

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

- We can use the Viterbi algorithm
Probability of an Output Sequence given an Input Sequence

- Given \( x \) and \( y \), compute \( \Pr(y \mid x; w) = \frac{\exp\{w \cdot f(x, y)\}}{Z(x; w)} \)

- To compute \( Z(x; w) \) we need to sum over \( Y^n \!

- But with some algebraic massaging: (let \( s(i, y_{i-1}, y_i) = w \cdot f(x, i, y_{i-1}, y_i) \))

\[
Z(x; w) = \sum_y \exp\{w \cdot f(x, y)\}
= \sum_y \exp \left\{ \sum_{i=1}^n s(i, y_{i-1}, y_i) \right\}
= \sum_y \prod_{i=1}^n \exp \{s(i, y_{i-1}, y_i)\}
\]

- \( Z(x; w) \) is a sum-product computation: forward algorithm (with exponentiated scores)!
  - \( Z(x; w) = \sum_a \alpha(n, a) \)
Marginal Probability of a Single Label

What’s the probability that token $i$ has label $a$?

We need to compute the marginal distribution of $y_i$:

$$
\mu_i(a) = \Pr(y_i = a | x; w) = \sum_{y \in \mathcal{Y}^n : y_i = a} \Pr(y | x; w)
$$

(algebraic massaging)

$$
= \frac{\alpha(i, a) \ast \beta(i, a)}{Z(x; w)}
$$

Use forward-backward (using exponentiated scores)

Recall that $Z(x; w) = \sum_l \alpha(n, l)$
Marginal Probability of a Single Label

What's the probability that token $i$ has label $a$?

We need to compute the marginal distribution of $y_i$:

$$
\mu_i(a) = \Pr(y_i = a|x; w) = \sum_{y \in Y^n: y_i = a} \Pr(y|x; w)
$$

= (algebraic massaging)

= $\alpha(i, a) \cdot \beta(i, a)$

= $\frac{\alpha(i, a) \cdot \beta(i, a)}{Z(x; w)}$

Use forward-backward (using exponentiated scores)

Recall that $Z(x; w) = \sum_l \alpha(n, l)$
Marginal Probability of a Label Bigram

What’s the probability that token $i - 1$ has label $a$ and token $i$ has label $b$?

We need to compute the marginal distribution of label bigrams at position $i$:

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

(algebraic massaging)

$$
= \alpha(i - 1, a) \exp \{ w \cdot f(x, i, a, b) \} \beta(i, b) / Z(x; w)
$$

Again forward-backward (using exponentiated scores)

Recall that $Z(x; w) = \sum_l \alpha(n, l)$
Marginal Probability of a Label Bigram

What’s the probability that token \(i - 1\) has label \(a\) and token \(i\) has label \(b\)?

We need to compute the marginal distribution of label bigrams at position \(i\):

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

\[
= \alpha(i - 1, a) * \exp\{w \cdot f(x, i, a, b)\} * \beta(i, b) / Z(x; w)
\]

Again forward-backward (using exponentiated scores)

Recall that \(Z(x; w) = \sum_l \alpha(n, l)\)
Linear Factored Sequence Prediction

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

- Factored representation, e.g. based on bigrams

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

- Flexible, arbitrary features of full $x$ and the factors
- Efficient prediction using Viterbi
- In probabilistic models, efficient computation of marginals using Forward-Backward
- Next, learning $w$:
  - The Structured Perceptron
  - Probabilistic log-linear models:
    - Local learning, a.k.a. Maximum-Entropy Markov Models
    - Global learning, a.k.a. Conditional Random Fields
Outline

Part I
  Introduction
  Four Approaches to Sequence Prediction
  Greedy Sequence Prediction

Part II
  Factored Sequence Prediction
  Algorithms for Factored Models
  Log-linear Factored Models

Part III
  Structured Perceptron
  Log-linear Models and CRFs
  Dependency Parsing
  Summary and Conclusion
The Structured Perceptron
Collins (2002)

- Set $w = 0$
- For $t = 1 \ldots T$
  - For each training example $(x, y)$
    1. Compute $z = \text{argmax}_z \ w \cdot f(x, z)$
    2. If $z \neq y$
       
       $$w \leftarrow w + f(x, y) - f(x, z)$$

- Return $w$
The Structured Perceptron + Averaging
Freund and Schapire (1999); Collins (2002)

- Set \( w = 0, \ w^a = 0 \)
- For \( t = 1 \ldots T \)
  - For each training example \((x, y)\)
    1. Compute \( z = \text{argmax}_z w \cdot f(x, z) \)
    2. If \( z \neq y \)
       \[ w \leftarrow w + f(x, y) - f(x, z) \]
    3. \( w^a = w^a + w \)
- Return \( w^a \)
Perceptron Updates: Example

Let $y$ be the correct output for $x$.

Say we predict $z$ instead, under our current $w$.

The update is:

$$g = f(x, y) - f(x, z)$$

$$= \sum_i f(x, i, y_{i-1}, y_i) - \sum_i f(x, i, z_{i-1}, z_i)$$

$$= f(x, 2, \text{PER, PER}) - f(x, 2, \text{PER, LOC})$$

$$+ f(x, 3, \text{PER, -}) - f(x, 3, \text{LOC, -})$$

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.
- Structured Perceptron and Beam Search:
  - Transition systems cannot recover the argmax solution.
  - Structured Perceptron can use beam search instead (i.e., an approximation to argmax).
  - See Collins and Roark (2004); Zhang and Clark (2011); Huang et al. (2012).
### 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 with perceptron with beam search (Zhang and Clark, 2011)
Outline

Part I
   Introduction
   Four Approaches to Sequence Prediction
   Greedy Sequence Prediction

Part II
   Factored Sequence Prediction
   Algorithms for Factored Models
   Log-linear Factored Models

Part III
   Structured Perceptron
   Log-linear Models and CRFs
   Dependency Parsing
   Summary and Conclusion
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

- \( f(x, y) \) 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*

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

- To predict the best sequence

\[
\arg\max_{y \in \mathcal{Y}^n} Pr(y \mid x)
\]
Parameter Estimation in Log-Linear Models

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

- Given training data

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

- How to estimate \( w \)?
  - Define the conditional log-likelihood (or cross-entropy) of the data:
    \[
    L(w) = \sum_{k=1}^{m} \log Pr(y^{(k)} \mid x^{(k)}; w)
    \]

- \( L(w) \) measures how well \( w \) explains the data. A good value for \( w \) will give a high value for \( Pr(y^{(k)} \mid x^{(k)}; w) \) for all \( k = 1 \ldots m \).
  - We want \( w \) that maximizes \( L(w) \).
Parameter Estimation in Log-Linear Models

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

- Given training data
  \[ \left\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\right\}, \]

- How to estimate \( w \)?
  - Define the conditional log-likelihood (or cross-entropy) of the data:
  \[ L(w) = \sum_{k=1}^{m} \log Pr(y^{(k)} \mid x^{(k)}; w) \]

- \( L(w) \) measures how well \( w \) explains the data. A good value for \( w \) will give a high value for \( Pr(y^{(k)} \mid x^{(k)}; w) \) for all \( k = 1 \ldots m \).
  - We want \( w \) that maximizes \( L(w) \)
Learning Log-Linear Models: Loss + Regularization

Solve:

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

where

- The first term is the negative conditional log-likelihood
- The second term is a regularization term, it penalizes solutions with large norm
- \( \lambda \in \mathbb{R} \) controls the trade-off between loss and regularization

Convex optimization problem \( \rightarrow \) gradient descent

Two common losses based on log-likelihood that make learning tractable:

- Local Loss (MEMM): assume that \( \Pr(y | x; w) \) decomposes
- Global Loss (CRF): assume that \( f(x, y) \) decomposes
Learning Log-Linear Models: Loss + Regularization

- Solve:

\[
\mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathbb{R}^d} -L(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2
\]

where

- The first term is the negative conditional log-likelihood
- The second term is a regularization term, it penalizes solutions with large norm
- \( \lambda \in \mathbb{R} \) controls the trade-off between loss and regularization

- Convex optimization problem \( \rightarrow \) gradient descent

- Two common losses based on log-likelihood that make learning tractable:
  - Local Loss (MEMM): assume that \( \Pr(y \mid x; \mathbf{w}) \) decomposes
  - Global Loss (CRF): assume that \( f(x, y) \) decomposes
Local Log-linear Loss (a.k.a. Maximum Entropy Markov Models)
McCallum, Freitag, and Pereira (2000)

If we apply the chain rule:

\[
\text{Pr}(y_{1:n} \mid x_{1:n}) = \text{Pr}(y_1 \mid x_{1:n}) \times \text{Pr}(y_{2:n} \mid x_{1:n}, y_1)
\]

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

Markov assumption (the model becomes factored):

\[
\text{Pr}(y_i \mid x_{1:n}, y_{1:i-1}) = \text{Pr}(y_i \mid x_{1:n}, y_{i-1})
\]

Now we can write

\[
\text{Pr}(y_{1:n} \mid x_{1:n}) = \text{Pr}(y_1 \mid x_{1:n}) \times \prod_{i=2}^{n} \text{Pr}(y_i \mid x_{1:n}, y_{i-1})
\]
Parameter Estimation with Local Log-Linear Markov Models

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

- The log-linear model is normalized \textit{locally} (i.e. at each position):

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

- The log-likelihood is also \textit{local}:

\[
L(w) = \sum_{k=1}^{m} \sum_{i=1}^{n^{(k)}} \log \Pr(y_i^{(k)} \mid x_i^{(k)}, i, y_i^{(k)} - 1)
\]

\[
\frac{\partial L(w)}{\partial w_j} = \frac{1}{m} \sum_{k=1}^{m} \sum_{i=1}^{n^{(k)}} \left[ \text{observed} \begin{array}{c}
\mathbf{f}_j(x_i^{(k)}, i, y_i^{(k)}, y_i^{(k)})
\end{array} - \text{expected} \begin{array}{c}
\sum_{y \in Y} \Pr(y \mid x_i^{(k)}, i, y_i^{(k)} - 1, y) \mathbf{f}_j(x_i^{(k)}, i, y_i^{(k)}, y)
\end{array} \right]
\]
Conditional Random Fields
Lafferty, McCallum, and Pereira (2001)

- Log-linear model of the conditional distribution:

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

where

- \(x\) and \(y\) are input and output sequences
- \(f(x, y)\) is a feature vector of \(x\) and \(y\) that decomposes into factors
- \(w\) are model parameters

- To predict the best sequence

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

- Log-Likelihood at the global (sequence) level:

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

Consider a parameter $w_j$ and its associated feature $f_j$:

$$\frac{\partial L(w)}{\partial w_j} = \frac{1}{m} \sum_{k=1}^{m} \left[ \begin{array}{c} \text{observed} \\ f_j(x^{(k)}, y^{(k)}) \\ \text{expected} \\ \sum_{y \in \mathcal{Y}^*} \Pr(y|x^{(k)}; w) f_j(x^{(k)}, y) \end{array} \right]$$

where

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

- First term: observed value of $f_j$ in training examples
- Second term: expected value of $f_j$ under current $w$
  they require summing over all sequences $y \in \mathcal{Y}^n$
Computing the Gradient in CRFs

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

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

\(\mu^k_i(a,b)\) is the marginal probability of having labels \((a,b)\) at position \(i\):

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

The quantities \(\mu^k_i\) can be computed efficiently in \(O(nL^2)\) using the forward-backward algorithm.
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
- Parameter estimation:
  - Gradient-based methods, in practice L-BFGS or SGD
  - Computation of gradient uses forward-backward
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
- Parameter estimation:
  - Gradient-based methods, in practice L-BFGS or SGD
  - Computation of gradient uses forward-backward
- Next Question: Local or Global loss?
Local vs. Global Log-linear Losses

Local Loss:  \( \text{Pr}(y \mid x) = \prod_{i=1}^{n} \frac{\exp \{ w \cdot f(x, i, y_{i-1}, y_i) \}}{Z(x, i, y_{i-1}; w)} \)

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

- Both exploit the same factorization, i.e. same features
- Same computations to compute \( \text{argmax}_y \text{Pr}(y \mid x) \)
- Local loss is locally normalized; CRFs globally normalized
  - Local loss assumes that \( \text{Pr}(y_i \mid x_{1:n}, y_{1:i-1}) = \text{Pr}(y_i \mid x_{1:n}, y_{i-1}) \)
  - Leads to “Label Bias Problem” (Lafferty et al., 2001; Andor et al., 2016)
- Local loss is cheaper to train (reduces to multiclass MaxEnt learning)
- CRFs are easier to extend to other structures
Learning Structure Predictors: summary so far

▶ Linear models for sequence prediction

\[
\arg\max_{y \in Y^*} \sum_i w \cdot f(x, i, y_{i-1}, y_i)
\]

▶ 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., 1998; Shalev-Shwartz et al., 2011)
    ▶ Exponentiated Gradient (Collins et al., 2008)
    ▶ SVM Struct (Tsochantaridis et al., 2005)
    ▶ Structured MIRA (Crammer et al., 2005)
Outline

Part I
  Introduction
  Four Approaches to Sequence Prediction
  Greedy Sequence Prediction

Part II
  Factored Sequence Prediction
  Algorithms for Factored Models
  Log-linear Factored Models

Part III
  Structured Perceptron
  Log-linear Models and CRFs
  Dependency Parsing
  Summary and Conclusion
They solved the problem with statistics.
They solved the problem with statistics.
Theories of Syntactic Structure

Dependency Trees

- Main element: dependency
- Focus on relations between words

Constituent Trees

- Main element: constituents (or phrases, or bracketings)
- Constituents = abstract linguistic units
- Results in nested trees
**Dependency Parsing: Arc-factored models**

McDonald, Pereira, Ribarov, and Hajič (2005)

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

$$\text{argmax}_{y \in \mathcal{Y}(x)} \sum_{\langle h, m \rangle \in y} w \cdot f(x, h, m)$$

- Each arc or dependency $(h, m)$ is scored independently of each other

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

- Tractable inference algorithms exist
MST Parsing for Arc-factored models
McDonald, Pereira, Ribarov, and Hajič (2005)

- Parsing problem, given a sentence $x$:

$$\arg\max_{y \in Y(x)} \sum_{\langle h,m \rangle \in y} \text{score}(x, h, m)$$

- Can be formulated as a directed Maximum Spanning Tree (MST) problem:

- The Chu-Liu-Edmonds algorithm finds the optimal tree in $O(n^2)$
The Eisner (1996) algorithm is a variant of CKY specific to non-crossing dep trees

- Finds optimal tree in $O(n^3)$

Extension to higher-order parsing:

- First-order $O(n^3)$
- Second-order:
  - Horizontal $O(n^3)$ (McDonald and Pereira, 2006)
  - Vertical $O(n^4)$ (Carreras, 2007)
- Third-order $O(n^4)$ (Koo and Collins, 2010)
Transition-based Parsing: Nivre’s Arc-Standard System
Nivre (2008)

- **State:**
  - Buffer: list of upcoming words to be parsed
  - Stack: stack of subtrees that are already parsed

- **Parsing actions:**
  - Shift: shift next word in the buffer to the task
  - Left-arc ($l$): add a left arc between the two top subtrees of the stack, with label $l$
  - Right-arc ($l$): add a right arc between the two top subtrees of the stack, with label $l$

- Parsing is linear in the sentence length, very fast! But prone to greedy mistakes!
- Parsing model: score a candidate action in the context of a state
  - Has access to the full sentence and the full history of actions
Arc-Standard Parsing: Example

(illustration by Miguel Ballesteros)

Mark Watney visited Mars

<table>
<thead>
<tr>
<th>transition</th>
<th>Stack</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ ]</td>
<td>[Mark, Watney, visited, Mars]</td>
</tr>
</tbody>
</table>

Mark Watney visited Mars
Arc-Standard Parsing: Example
(illustration by Miguel Ballesteros)

```
transition | Stack   | Buffer                      
------------|---------|----------------------------
             | [ ]     | [Mark, Watney, visited, Mars]
SHIFT       | [Mark]  | [Watney, visited, Mars]
```

Mark Watney visited Mars
Arc-Standard Parsing: Example
(illustration by Miguel Ballesteros)

Mark Watney visited Mars

<table>
<thead>
<tr>
<th>transition</th>
<th>Stack</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ ]</td>
<td>[Mark, Watney, visited, Mars]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[Mark]</td>
<td>[Watney, visited, Mars]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[Mark, Watney]</td>
<td>[visited, Mars]</td>
</tr>
</tbody>
</table>

Mark Watney visited Mars
Arc-Standard Parsing: Example

(illustration by Miguel Ballesteros)

Transition	Stack	Buffer
---
SHIFT	[Mark]	[Mark, Watney, visited, Mars]
SHIFT	[Mark, Watney]	[Watney, visited, Mars]
LA(NAME)	[Watney]	[visited, Mars]

NAME
Mark Watney visited Mars
### Arc-Standard Parsing: Example
(illustration by Miguel Ballesteros)

Mark Watney visited Mars

<table>
<thead>
<tr>
<th>transition</th>
<th>Stack</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ ]</td>
<td>[Mark, Watney, visited, Mars]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[Mark]</td>
<td>[Watney, visited, Mars]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[Mark, Watney]</td>
<td>[visited, Mars]</td>
</tr>
<tr>
<td>LA(NAME)</td>
<td>[Watney]</td>
<td>[visited, Mars]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[Watney, visited]</td>
<td>[Mars]</td>
</tr>
</tbody>
</table>

NAME  
Mark Watney visited Mars
Arc-Standard Parsing: Example
(illustration by Miguel Ballesteros)

transition | Stack             | Buffer                                 |
-----------|-------------------|----------------------------------------|
SHIFT      | [Mark]            | [Mark, Watney, visited, Mars]          |
SHIFT      | [Mark, Watney]    | [Watney, visited, Mars]                |
LA(NAME)   | [Watney]          | [visited, Mars]                        |
SHIFT      | [Watney, visited] | [Mars]                                 |
LA(SUBJ)   | [visited]         | [Mars]                                 |

Mark Watney visited Mars
Arc-Standard Parsing: Example
(illustration by Miguel Ballesteros)

Mark Watney visited Mars

<table>
<thead>
<tr>
<th>transition</th>
<th>Stack</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ ]</td>
<td>[Mark, Watney, visited, Mars]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[Mark]</td>
<td>[Watney, visited, Mars]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[Mark, Watney]</td>
<td>[visited, Mars]</td>
</tr>
<tr>
<td>LA(NAME)</td>
<td>[Watney]</td>
<td>[visited, Mars]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[Watney, visited]</td>
<td>[Mars]</td>
</tr>
<tr>
<td>LA(SUBJ)</td>
<td>[visited]</td>
<td>[Mars]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[visited, Mars]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>
# Arc-Standard Parsing: Example

(illustration by Miguel Ballesteros)

<table>
<thead>
<tr>
<th>transition</th>
<th>Stack</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFT</td>
<td>[Mark]</td>
<td>[Watney, visited, Mars]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[Mark, Watney]</td>
<td>[Watney, visited, Mars]</td>
</tr>
<tr>
<td>LA(NAME)</td>
<td>[Watney]</td>
<td>[visited, Mars]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[Watney, visited]</td>
<td>[visited, Mars]</td>
</tr>
<tr>
<td>LA(SUBJ)</td>
<td>[visited]</td>
<td>[Mars]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[visited, Mars]</td>
<td>[ ]</td>
</tr>
<tr>
<td>RA(OBJ)</td>
<td>[visited]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

Mark Watney visited Mars

[Diagram showing the parsing process with arrows labeled NAME, SBJ, and OBJ, and the sentence Mark Watney visited Mars highlighted.]
Outline

Part I
  Introduction
  Four Approaches to Sequence Prediction
  Greedy Sequence Prediction

Part II
  Factored Sequence Prediction
  Algorithms for Factored Models
  Log-linear Factored Models

Part III
  Structured Perceptron
  Log-linear Models and CRFs
  Dependency Parsing
  Summary and Conclusion
Linear (Structured) Prediction

▶ Multiclass classification

\[
\arg\max_{y \in \{1, \ldots, L\}} w \cdot f(x, y)
\]

▶ Sequence prediction (bigram factorization)

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

▶ Dependency parsing (arc-factored)

\[
\arg\max_{y \in \mathcal{Y}(x)} w \cdot f(x, y) = \arg\max_{y \in \mathcal{Y}(x)} \sum_{\langle h, m, l \rangle \in y} w \cdot f(x, h, m, l)
\]

▶ Factored models: Applicable to other tasks and factorizations

▶ Alternative: transition systems (very fast and expressive, but prone to search errors)
Factored Sequence Prediction: from Linear to Non-linear

\[
\text{score}(x, y) = \sum_i s(x, i, y_{i-1}, y_i)
\]

- **Linear:**
  \[
s(x, i, y_{i-1}, y_i) = w \cdot f(x, i, y_{i-1}, y_i)
\]

- **Non-linear, using a feed-forward neural network:**
  \[
s(x, i, y_{i-1}, y_i) = w \cdot [e_{y_{i-1}, y_i} \otimes h(f(x, i))]\]
  where:
  \[
h(f(x, i)) = \sigma(W^2 \sigma(W^1 \sigma(W^0 f(x, i))))
\]

- **Remarks:**
  - The non-linear model computes a hidden representation of the input
  - Still factored: Viterbi and Forward-Backward work
  - Parameter estimation becomes non-convex, use backpropagation
Induction of hidden vectors (i.e. embeddings) that keep track of previous observations and predictions

Making predictions is not tractable
  ▶ In practice: greedy predictions or beam search
  ▶ Making predictions was not tractable for transition systems either!

Learning is non-convex, so what?

Popular methods: RNN, LSTM, Spectral Models, ...
Neural Architectures for Named Entity Recognition

Guillaume Lample*  Miguel Ballesteros**
Sandeep Subramanian*  Kazuya Kawakami*  Chris Dyer*

<table>
<thead>
<tr>
<th>Model</th>
<th>F_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collobert et al. (2011)*</td>
<td>89.59</td>
</tr>
<tr>
<td>Lin and Wu (2009)</td>
<td>83.78</td>
</tr>
<tr>
<td>Lin and Wu (2009)*</td>
<td>90.90</td>
</tr>
<tr>
<td>Huang et al. (2015)*</td>
<td>90.10</td>
</tr>
<tr>
<td>Passos et al. (2014)</td>
<td>90.05</td>
</tr>
<tr>
<td>Passos et al. (2014)*</td>
<td>90.90</td>
</tr>
<tr>
<td>Luo et al. (2015)* + gaz</td>
<td>89.9</td>
</tr>
<tr>
<td>Luo et al. (2015)* + gaz + linking</td>
<td>91.2</td>
</tr>
<tr>
<td>Chiu and Nichols (2015)</td>
<td>90.69</td>
</tr>
<tr>
<td>Chiu and Nichols (2015)*</td>
<td>90.77</td>
</tr>
<tr>
<td>LSTM-CRF (no char)</td>
<td>90.20</td>
</tr>
<tr>
<td>LSTM-CRF</td>
<td><strong>90.94</strong></td>
</tr>
<tr>
<td>S-LSTM (no char)</td>
<td>87.96</td>
</tr>
<tr>
<td>S-LSTM</td>
<td>90.33</td>
</tr>
</tbody>
</table>

Table 1: English NER results (CoNLL-2003 test set).
End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF

Xuezhe Ma and Eduard Hovy

<table>
<thead>
<tr>
<th>Model</th>
<th>POS</th>
<th>NER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dev</td>
<td>Test</td>
</tr>
<tr>
<td></td>
<td>Acc</td>
<td>Recall</td>
</tr>
<tr>
<td></td>
<td>Prec</td>
<td></td>
</tr>
<tr>
<td>BRNN</td>
<td>96.56</td>
<td>92.04</td>
</tr>
<tr>
<td>BLSTM</td>
<td>96.88</td>
<td>92.31</td>
</tr>
<tr>
<td>BLSTM-CNN</td>
<td>97.34</td>
<td>92.52</td>
</tr>
<tr>
<td>BRNN-CNN-CRF</td>
<td>97.46</td>
<td>94.85</td>
</tr>
</tbody>
</table>

Table 3: Performance of our model on both the development and test sets of the two tasks, together with three baseline systems.

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Giménez and Márquez (2004)</td>
<td>97.16</td>
</tr>
<tr>
<td>Toutanova et al. (2003)</td>
<td>97.27</td>
</tr>
<tr>
<td>Manning (2011)</td>
<td>97.28</td>
</tr>
<tr>
<td>Collobert et al. (2011)†</td>
<td>97.29</td>
</tr>
<tr>
<td>Santos and Zadrozny (2014)†</td>
<td>97.32</td>
</tr>
<tr>
<td>Shen et al. (2007)</td>
<td>97.33</td>
</tr>
<tr>
<td>Sun (2014)</td>
<td>97.36</td>
</tr>
<tr>
<td>Søgaard (2011)</td>
<td>97.50</td>
</tr>
<tr>
<td>This paper</td>
<td>97.55</td>
</tr>
</tbody>
</table>

Table 4: POS tagging accuracy of our model on test data from WSJ proportion of PTB, together with prior work.

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chieu and Ng (2002)</td>
<td>88.31</td>
</tr>
<tr>
<td>Florian et al. (2003)</td>
<td>88.76</td>
</tr>
<tr>
<td>Ando and Zhang (2005)</td>
<td>89.31</td>
</tr>
<tr>
<td>Collobert et al. (2011)†</td>
<td>89.59</td>
</tr>
<tr>
<td>Huang et al. (2015)†</td>
<td>90.10</td>
</tr>
<tr>
<td>Chiu and Nichols (2015)†</td>
<td>90.77</td>
</tr>
<tr>
<td>Ratinov and Roth (2009)</td>
<td>90.80</td>
</tr>
<tr>
<td>Lin and Wu (2009)</td>
<td>90.90</td>
</tr>
<tr>
<td>Passos et al. (2014)</td>
<td>90.90</td>
</tr>
<tr>
<td>Lample et al. (2016)†</td>
<td>90.94</td>
</tr>
<tr>
<td>Luo et al. (2015)</td>
<td>91.20</td>
</tr>
<tr>
<td>This paper</td>
<td>91.21</td>
</tr>
</tbody>
</table>

Table 5: NER F1 score of our model on test data set from CoNLL-2003. For the purpose of com-
Thanks!
References


