# Learning Structured Predictors

Xavier Carreras

DMETRICS

https://dmetrics.com

# Outline

# Supervised (Structured) Prediction

- Learning to predict: given training data

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

  learn a predictor $\mathbf{x} \to \mathbf{y}$ that *works well* on unseen inputs $\mathbf{x}$

- Non-Structured Prediction: outputs $\mathbf{y}$ are atomic
  - Binary classification: $\mathbf{y} \in \{-1, +1\}$
  - Multiclass classification: $\mathbf{y} \in \{1, 2, \ldots, L\}$

- Structured Prediction: outputs $\mathbf{y}$ are structured
  - Sequence prediction: $\mathbf{y}$ are sequences
  - Parsing: $\mathbf{y}$ are trees
  - ...

# Named Entity Recognition

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{y}$ | PER | - | QNT | - | - | ORG | ORG | - | TIME |
| $\mathbf{x}$ | Jim | bought | 300 | shares | of | Acme | Corp. | in | 2006 |

# Named Entity Recognition

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{y}$ | PER | - | QNT | - | - | ORG | ORG | - | TIME |
| $\mathbf{x}$ | Jim | bought | 300 | shares | of | Acme | Corp. | in | 2006 |

| | | | | | |
|---|---|---|---|---|---|
| $\mathbf{y}$ | PER | PER | - | - | LOC |
| $\mathbf{x}$ | Jack | London | went | to | Paris |

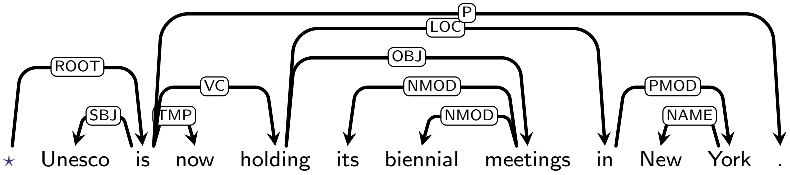
| | | | | | |
|---|---|---|---|---|---|
| $\mathbf{y}$ | PER | PER | - | - | LOC |
| $\mathbf{x}$ | Paris | Jackson | went | to | London |

| | | | | |
|---|---|---|---|---|
| $\mathbf{y}$ | PER | - | - | LOC |
| $\mathbf{x}$ | Jackie | went | to | Lisdon |

# Part-of-speech Tagging

| | | | | | |
|---|---|---|---|---|---|
| $\mathbf{y}$ | NOUN | NOUN | VERB | NOUN | . |
| $\mathbf{x}$ | Fruit | flies | like | bananas | . |

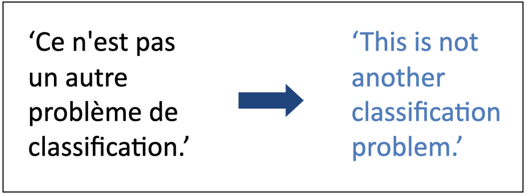
# Syntactic Dependency Parsing




$\mathbf{x}$ are sentences

$\mathbf{y}$ are syntactic dependency trees

# Machine Translation




(illustration by Ben Taskar)

$\mathbf{x}$ are sentences in some source language (e.g. French)

$\mathbf{y}$ are sentence translations in a target language (e.g. English)

# Object Detection

(Kumar and Hebert, 2003)

$\mathbf{x}$ are images

$\mathbf{y}$ are grids labeled with object types

# Object Detection

(Kumar and Hebert, 2003)

$\mathbf{x}$ are images

$\mathbf{y}$ are grids labeled with object types

# 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

# Sequence Prediction

| | | | | | |
|---|---|---|---|---|---|
| $\mathbf{y}$ | PER | PER | - | - | LOC |
| $\mathbf{x}$ | Jack | London | went | to | Paris |

# Sequence Prediction

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

- **Goal:** given training data

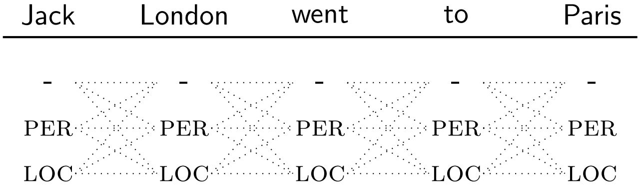
$$\left\{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \dots, (\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\right\}$$

  learn a predictor $\mathbf{x} \to \mathbf{y}$ that works well on unseen inputs $\mathbf{x}$

- What is the form of our prediction model?

# Exponentially-many Solutions

- Let $\mathcal{Y} = \{\text{-}, \textsc{per}, \textsc{loc}\}$
- The solution space (all output sequences):

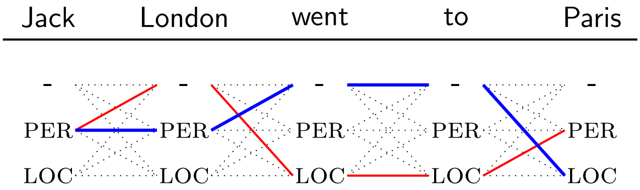



  - Each path is a possible solution

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

# Exponentially-many Solutions

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

| PER | PER | - | - | LOC |
|---|---|---|---|---|
| Jack | London | went | to | Paris |

- Scoring of individual labels at each position

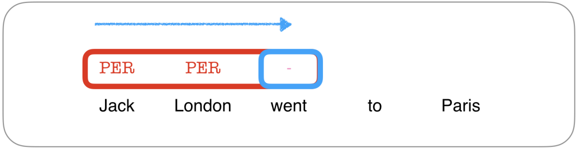
对单个位置t

$$\hat{y}_t = \underset{l \,\in\, \{\text{LOC},\ \text{PER},\ \text{-}\}}{\text{argmax}} \ \text{score}(\mathbf{x}, t, l)$$

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



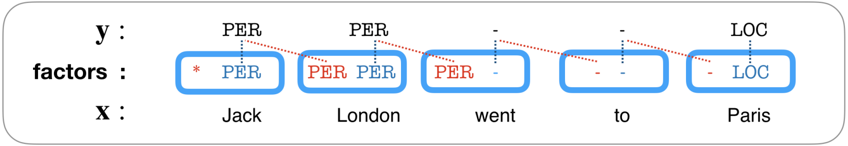

- Score one label at a time, left-to-right, conditioning on previous predictions:

对单个位置t

$$\hat{y}_t = \underset{l \in \{\text{LOC, PER, -}\}}{\operatorname{argmax}} \operatorname{score}(\mathbf{x}, t, l, \hat{\mathbf{y}}_{1:t-1})$$

- Captures interactions between full input $\mathbf{x}$ 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:

对整个序列y

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}^n}{\operatorname{argmax}} \operatorname{score}(\mathbf{x}, \mathbf{y}) = \underset{\mathbf{y} \in \mathcal{Y}^n}{\operatorname{argmax}} \sum_{i=1}^{n} \operatorname{score}(\mathbf{x}, i, y_{i-1}, y_i)$$

- Output sequence factored into label bigrams
- Captures interactions between full input $\mathbf{x}$ and factors of output sequence
- Prediction is exact for many types of factorizations

# Approach 4: Re-Ranking

| | | | | |
|---|---|---|---|---|
| PER | PER | - | - | LOC |
| PER | LOC | - | - | LOC |
| LOC | LOC | - | - | LOC |
| PER | PER | - | - | PER |
| PER | PER | PER | - | LOC |
| ... | ... | ... | ... | ... |
| Jack | London | went | to | Paris |

对整个序列y

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{A}(\mathcal{Y}^n)}{\operatorname{argmax}} \ \operatorname{score}(\mathbf{x}, \mathbf{y})$$

- Scoring of full inputs and outputs: very expressive!
- Relies on an active set $\mathcal{A}(\mathcal{Y}^n)$ 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

| | input-output representation | exact prediction? |
|---|---|---|
| label classifiers | only individual labels | yes |
| transition-based | full history of decisons | no (greedy, beam search) |
| factored | label factors | yes |
| re-ranking | full | limited to active set |

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

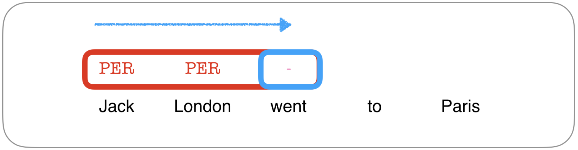
| | input-output representation | exact prediction? |
|---|---|---|
| label classifiers | only individual labels | yes |
| transition-based | full history of decisons | no<br>(greedy, beam search) |
| factored | label factors | yes |
| re-ranking | full | limited to active set |

take home message 1: expressivity-tractability trade-off

take home message 2: always pick the simplest approach that suits the task at hand

# Outline

# Greedy Sequence Prediction




- Run a greedy classifier left-to-right:
  - For $t = 1 \ldots n$:

$$\hat{y}_t = \underset{l \,\in\, \{\textsc{loc},\ \textsc{per},\ \text{-}\}}{\operatorname{argmax}} \ \operatorname{score}(\mathbf{x}, t, l, \hat{\mathbf{y}}_{1:t-1})$$

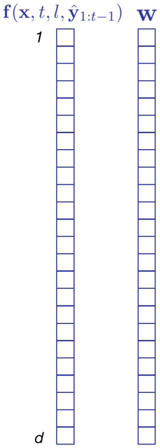
- What is the form of $\operatorname{score}(\mathbf{x}, t, l, \hat{y}_{1:t-1})$?
  - We focus on linear scoring functions: $\operatorname{score}(\mathbf{x}, t, l, \hat{y}_{1:t-1}) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, t, l, \hat{y}_{1:t-1})$

# Representations in Greedy Sequence Prediction

- In linear greedy sequence prediction, at time $t$

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

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

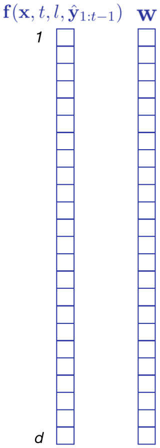


# Representations in Greedy Sequence Prediction

- In linear greedy sequence prediction, at time $t$

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

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

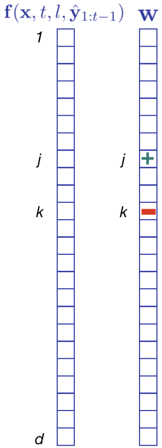


# Indicator Features for One Label Only

- $\mathbf{f}(\mathbf{x}, t, l)$ is a vector of $d$ features representing label $l$ for $x_t$
- What's in a feature $\mathbf{f}_j(\mathbf{x}, t, l)$?
  - Anything we can compute using $\mathbf{x}$ 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 $\mathbf{x}$ and target position $t$
  - and the candidate label $l$ for position $t$

$$\mathbf{f}_j(\mathbf{x}, t, l) = \begin{cases} 1 & \text{if } x_t = \text{London and } l = \text{LOC} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{f}_k(\mathbf{x}, t, l) = \begin{cases} 1 & \text{if } x_{t+1} = \text{went and } l = \text{LOC} \\ 0 & \text{otherwise} \end{cases}$$



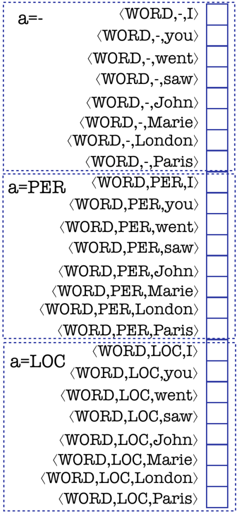

- Indicator features produce sparse feature vectors

# Feature Templates

- Feature templates generate many indicator features
- A feature template is identified by a type, and a number of values
  - Example: template WORD indicates the current word

$$\mathbf{f}_{\langle \textsc{word},a,w\rangle}(\mathbf{x},t,l) = \begin{cases} 1 & \text{if } x_t = w \text{ and } l = a \\ 0 & \text{otherwise} \end{cases}$$

  - A feature of this type is identified by the tuple $\langle \textsc{word}, a, w \rangle$
  - Generates a feature for every label $a \in \mathcal{Y}$ and every word $w$
- Feature vectors and weight vectors are indexed by feature tuples

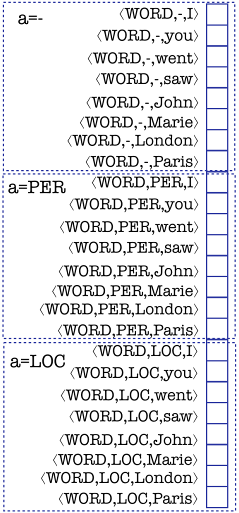


# Feature Templates

- 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

$$\mathbf{f}_{\langle \textsc{word}, a, w \rangle}(\mathbf{x}, t, l) = \begin{cases} 1 & \text{if } x_t = w \text{ and } l = a \\ 0 & \text{otherwise} \end{cases}$$

  - A feature of this type is identified by the tuple $\langle \textsc{word}, a, w \rangle$
  - 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 $\mathbf{w}$ indexed by such feature tuples
  - Let the learning algorithm choose the relevant features



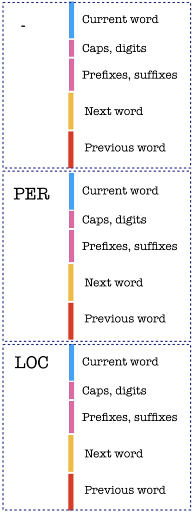
# More Features for NE Recognition

PER

Jack London went to Paris

In practice, construct $\mathbf{f}(\mathbf{x}, t, l)$ by ...

- Define a number of simple patterns of $\mathbf{x}$ 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

| | | | | | |
|---|---|---|---|---|---|
| $\mathbf{y}$ | PER | PER | - | | |
| $\mathbf{x}$ | Jack | London | went | to | Paris |

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

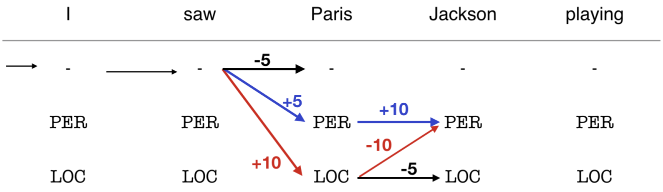
$$\mathbf{f}_{\langle \text{WB}, a, b, w\rangle}(\mathbf{x}, t, l, \hat{\mathbf{y}}_{1:t-1}) = \begin{cases} 1 & \text{if } x_t = w \text{ and} \\ & \quad \hat{\mathbf{y}}_{t-1} = a \text{ and } l = b \\ 0 & \text{otherwise} \end{cases}$$

- In practice:
  - Preceeding labels next to $t$
  - Bag-of-labels in $\hat{\mathbf{y}}_{1:t-1}$
  - Combinations with other features
- Neural networks automatically induce “good” features out of $\mathbf{x}$ and $\hat{\mathbf{y}}_{1:t-1}$

# Transition Systems (general form)

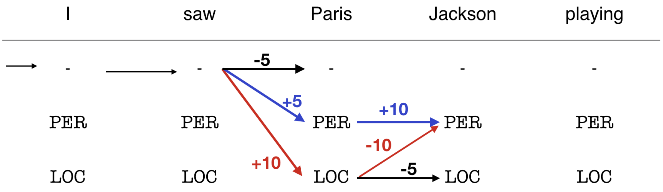
- Given an input $\mathbf{x}$, a transition system defines:
  - A set of states $\mathcal{S}(\mathbf{x})$
  - An initial state $s_0 \in \mathcal{S}(\mathbf{x})$, and a set of final states $S_\infty \subseteq \mathcal{S}(\mathbf{x})$
  - A set of allowed actions $\mathcal{A}(s, \mathbf{x})$ for all $s \in \mathcal{S}(\mathbf{x})$
  - A transition function $\text{transition} : s \times a \to s'$
  - A scoring function: $\text{score} : \mathbf{x} \times s \times a \to \mathbb{R}$
- To predict output $\mathbf{y}$ from input $\mathbf{x}$:
  - $s = s_0$
  - while $s \notin S_\infty$:
    - $a = \text{argmax}_{a \in \mathcal{A}(s,\mathbf{x})} \text{score}(\mathbf{x}, s, a)$
    - $s = \text{transition}(s, a)$
  - extract $\mathbf{y}$ 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

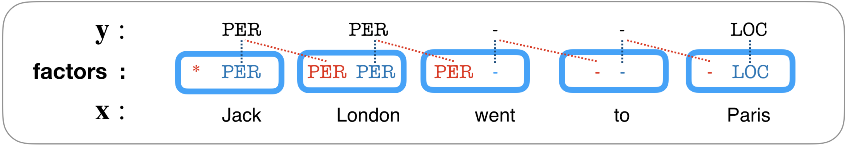
# Greedy Predictions are not Optimal, even with Beam Search




- 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

# Factored Sequence Predictors




$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}^n}{\text{argmax}} \sum_{i=1}^{n} \text{score}(\mathbf{x}, i, y_{i-1}, y_i)$$

Next questions:

- What is the form of $\text{score}(\mathbf{x}, i, a, b)$?
  We will use linear scoring functions: $\text{score}(\mathbf{x}, i, a, b) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, a, b)$
- There are exponentially-many sequences $\mathbf{y}$ for a given $\mathbf{x}$,
  how do we solve the argmax problem?

# Representations Factored at Bigrams

| | | | | | |
|---|---|---|---|---|---|
| $\mathbf{y}$: | PER | PER | - | - | LOC |
| $\mathbf{x}$: | Jack | London | went | to | Paris |

- $\text{score}(\mathbf{x}, i, a, b) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, a, b)$
- $\mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$
  - 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

# Representations Factored at Bigrams

| | | | | | |
|---|---|---|---|---|---|
| $\mathbf{y}$: | PER | PER | - | - | LOC |
| $\mathbf{x}$: | Jack | London | went | to | Paris |

- $\text{score}(\mathbf{x}, i, a, b) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, a, b)$
- $\mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$
  - 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

$$\underset{\mathbf{y}\in\mathcal{Y}^n}{\operatorname{argmax}}\, \mathbf{w}\cdot\mathbf{f}(\mathbf{x},\mathbf{y}) \qquad \text{where} \qquad \mathbf{f}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{n} \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$$

- Note the linearity of the expression:

$$\begin{aligned}
\text{score}(\mathbf{x},\mathbf{y}) &= \mathbf{w}\cdot\mathbf{f}(\mathbf{x},\mathbf{y}) \\
&= \mathbf{w}\cdot\sum_{i=1}^{n}\mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i) \\
&= \sum_{i=1}^{n}\mathbf{w}\cdot\mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i) \\
&= \sum_{i=1}^{n}\text{score}(\mathbf{x}, i, y_{i-1}, y_i)
\end{aligned}$$

# Predicting with Factored Sequence Models

- Assume we have a score function $\mathrm{score}(\mathbf{x}, i, a, b)$
- Given $\mathbf{x}_{1:n}$ find:

$$\operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}^n} \sum_{i=1}^{n} \mathrm{score}(\mathbf{x}, i, y_{i-1}, y_i)$$

- Use the Viterbi algorithm, takes $O(n|\mathcal{Y}|^2)$
- Notational change: since $\mathbf{x}_{1:n}$ is fixed we will use

$$s(i, a, b) = \mathrm{score}(\mathbf{x}, i, a, b)$$

# Outline

# Viterbi for Factored Sequence Models

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

$$\operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}^n} \sum_{i=1}^{n} s(i, y_{i-1}, y_i)$$

- Intuition: consider this example $\mathbf{x}$ and two alternative solutions $\mathbf{y}$ and $\mathbf{y}'$:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $\mathbf{x}$ | Jack | London | went | to | Paris | before | visiting | Lisbon |
| $\mathbf{y}$ | PER | LOC | - | - | LOC | - | - | LOC |
| $\mathbf{y}'$ | PER | PER | - | - | LOC | - | - | LOC |

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

$$s(\mathbf{x}, \mathbf{y}') = s(\mathbf{x}, \mathbf{y}) + \qquad - $$
$$+ \qquad - $$

# Viterbi for Factored Sequence Models

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

$$\underset{\mathbf{y} \in \mathcal{Y}^n}{\operatorname{argmax}} \sum_{i=1}^{n} s(i, y_{i-1}, y_i)$$

- Intuition: consider this example $\mathbf{x}$ and two alternative solutions $\mathbf{y}$ and $\mathbf{y}'$:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $\mathbf{x}$ | Jack | London | went | to | Paris | before | visiting | Lisbon |
| $\mathbf{y}$ | PER | LOC | - | - | LOC | - | - | LOC |
| $\mathbf{y}'$ | PER | PER | - | - | LOC | - | - | LOC |

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

$$\begin{aligned} s(\mathbf{x}, \mathbf{y}') &= s(\mathbf{x}, \mathbf{y}) \quad + s(2, \text{PER}, \text{ PER}) - s(2, \text{PER}, \text{ LOC}) \\ & \qquad\qquad\quad + s(3, \text{PER}, \text{ -}) - s(3, \text{LOC}, \text{ -}) \end{aligned}$$

output sequences that share bigrams also share their scores

# Viterbi recurrence

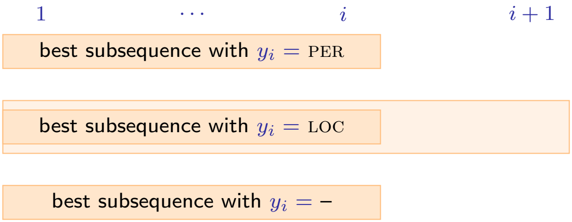
- Viterbi is a dynamic programming algorithm that uses the following recurrence
- Assume that, for a certain position $i$ and each label $l \in \mathcal{Y}$, we have the best sub-sequence from positions $1$ to $i$ ending with label $l$:

| $1$ | $\cdots$ | $i$ | $i+1$ |
|---|---|---|---|
| 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

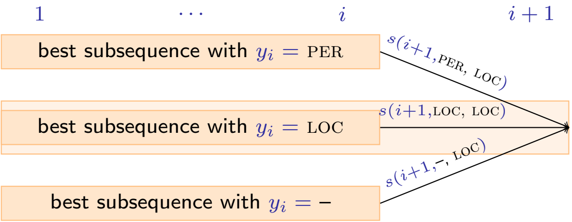
- Viterbi is a dynamic programming algorithm that uses the following recurrence
- Assume that, for a certain position $i$ and each label $l \in \mathcal{Y}$, we have the best sub-sequence from positions $1$ to $i$ ending with label $l$:




- 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 $l$:




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

# Viterbi for Factored Sequence Models

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}^n} \sum_{i=1}^{n} s(i, y_{i-1}, y_i)$$

- **Definition:** score of optimal sequence for $\mathbf{x}_{1:i}$ ending with $a \in \mathcal{Y}$

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

- Use the following recursions, for all $a \in \mathcal{Y}$, for $i = 2 \dots n$:

**max-sum**

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

- The optimal score for $\mathbf{x}$ is $\max_{a \in \mathcal{Y}} \delta(n, a)$
- The optimal sequence $\hat{\mathbf{y}}$ can be recovered through *back-pointers*
- Cost: $O(n|\mathcal{Y}|^2)$

# Viterbi for Factored Sequence Models

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}^n} \sum_{i=1}^{n} s(i, y_{i-1}, y_i)$$

- **Definition:** score of optimal sequence for $\mathbf{x}_{1:i}$ ending with $a \in \mathcal{Y}$

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

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

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

- The optimal score for $\mathbf{x}$ is $\max_{a \in \mathcal{Y}} \delta(n, a)$
- The optimal sequence $\hat{\mathbf{y}}$ 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)

# Forward-Backward Max-Sum Computations

- The Viterbi algorithm solves a max-sum recurrence

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

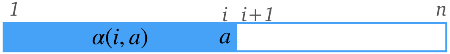
- The sum-product recurrence is also very useful (more later)

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

- The same style of dynamic programming works

# Forward Algorithm

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




- **Definition:** forward quantities

$$\alpha(i,a) \quad = \quad \sum_{\mathbf{y}_{1:i} \in \mathcal{Y}^i : y_i = a} \prod_{j=1}^{i} s(j, y_{j-1}, y_j)$$

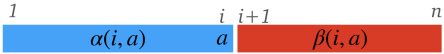
- Use the following recursions, for all $a \in \mathcal{Y}$, for $i = 2 \ldots n$:

$$\begin{aligned} \alpha(i,a) \quad &= \quad s(1, y_0 = \text{NULL}, a) \\ \alpha(i,a) \quad &= \quad \sum_{b \in \mathcal{Y}} \alpha(i-1, b) * s(i, b, a) \end{aligned}$$

- 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

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

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

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

- 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

# Log-linear Models for Sequence Prediction

- Model the conditional distribution:

$$\Pr(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) = \frac{\exp\left\{\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})\right\}}{Z(\mathbf{x}; \mathbf{w})}$$

  where
  - $\mathbf{f}(\mathbf{x}, \mathbf{y})$ represents $\mathbf{x}$ and $\mathbf{y}$ with $d$ features
  - $\mathbf{w} \in \mathbb{R}^d$ are the parameters of the model
  - $Z(\mathbf{x}; \mathbf{w})$ is a normalizer called the *partition function*

$$Z(\mathbf{x}; \mathbf{w}) = \sum_{\mathbf{z} \in \mathcal{Y}^*} \exp\left\{\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{z})\right\}$$

- To predict the best sequence

$$\operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}^n} \Pr(\mathbf{y} | \mathbf{x})$$

# Log-linear Models: Name

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

$$
\begin{aligned}
\log \Pr(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) &= \log \frac{\exp\{\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})\}}{Z(\mathbf{x}; \mathbf{w})} \\
&= \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) - \log \sum_{y} \exp\{\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})\} \\
&= \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) - \log Z(\mathbf{x}; \mathbf{w})
\end{aligned}
$$

- Partition function: $Z(\mathbf{x}; \mathbf{w}) = \sum_{\mathbf{z}} \exp\{\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{z})\}$
- $\log Z(\mathbf{x}; \mathbf{w})$ is a constant for a fixed $\mathbf{x}$
- 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 $\mathbf{f}(\mathbf{x}, \mathbf{y})$ decomposes into bigrams:

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

- Given $\mathbf{w}$, given $\mathbf{x}_{1:n}$, find:

$$\begin{aligned}
\operatorname*{argmax}_{\mathbf{y}_{1:n}} \Pr(\mathbf{y}_{1:n} | \mathbf{x}_{1:n}; \mathbf{w}) &= \operatorname*{amax}_{\mathbf{y}} \frac{\exp\left\{\sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)\right\}}{Z(\mathbf{x}; \mathbf{w})} \\
&= \operatorname*{amax}_{\mathbf{y}} \exp\left\{\sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)\right\} \\
&= \operatorname*{amax}_{\mathbf{y}} \sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)
\end{aligned}$$

- We can use the Viterbi algorithm

# Making Predictions with Log-Linear Models

- For tractability, assume $\mathbf{f}(\mathbf{x}, \mathbf{y})$ decomposes into bigrams:

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

- Given $\mathbf{w}$, given $\mathbf{x}_{1:n}$, find:

$$\begin{aligned}
\operatorname*{argmax}_{\mathbf{y}_{1:n}} \Pr(\mathbf{y}_{1:n} | \mathbf{x}_{1:n}; \mathbf{w}) &= \operatorname*{amax}_{\mathbf{y}} \frac{\exp\left\{\sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)\right\}}{Z(\mathbf{x}; \mathbf{w})} \\
&= \operatorname*{amax}_{\mathbf{y}} \exp\left\{\sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)\right\} \\
&= \operatorname*{amax}_{\mathbf{y}} \sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)
\end{aligned}$$

- We can use the Viterbi algorithm

# Probability of an Output Sequence given an Input Sequence

- Given $\mathbf{x}$ and $\mathbf{y}$, compute $\Pr(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) = \frac{\exp\{\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})\}}{Z(\mathbf{x}; \mathbf{w})}$
- To compute $Z(\mathbf{x}; \mathbf{w})$ we need to sum over $\mathcal{Y}^n$!
- But with some algebraic massaging: (let $s(i, y_{i-1}, y_i) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$)

$$\begin{aligned} Z(\mathbf{x}; \mathbf{w}) &= \sum_{\mathbf{y}} \exp\{\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})\} \\ &= \sum_{\mathbf{y}} \exp\left\{\sum_{i=1}^{n} s(i, y_{i-1}, y_i)\right\} \\ &= \sum_{\mathbf{y}} \prod_{i=1}^{n} \exp\{s(i, y_{i-1}, y_i)\} \end{aligned}$$

- $Z(\mathbf{x}; \mathbf{w})$ is a sum-product computation: forward algorithm (with exponentiated scores)!
  - $Z(\mathbf{x}; \mathbf{w}) = \sum_a \alpha(n, a)$

# Marginal Probability of a Single Label

PER

I saw Paris Jackson playing

$i$

- What's the probability that token $i$ has label $a$?
- We need to compute the marginal distribution of $y_i$:

$$
\begin{aligned}
\mu_i(a) = \Pr(y_i = a | \mathbf{x}; \mathbf{w}) &= \sum_{\mathbf{y} \in \mathcal{Y}^n : y_i = a} \Pr(\mathbf{y} | \mathbf{x}; \mathbf{w}) \\
&= \text{(algebraic massaging)} \\
&= \frac{\alpha(i, a) * \beta(i, a)}{Z(\mathbf{x}; \mathbf{w})}
\end{aligned}
$$

- Use forward-backward (using exponentiated scores)
  - Recall that $Z(\mathbf{x}; \mathbf{w}) = \sum_l \alpha(n, l)$

# Marginal Probability of a Single Label


- What's the probability that token $i$ has label $a$?
- We need to compute the marginal distribution of $y_i$:

$$\begin{aligned}
\mu_i(a) = \Pr(y_i = a | \mathbf{x}; \mathbf{w}) &= \sum_{\mathbf{y} \in \mathcal{Y}^n : y_i = a} \Pr(\mathbf{y} | \mathbf{x}; \mathbf{w}) \\
&= \text{(algebraic massaging)} \\
&= \frac{\alpha(i, a) * \beta(i, a)}{Z(\mathbf{x}; \mathbf{w})}
\end{aligned}$$

- Use forward-backward (using exponentiated scores)
  - Recall that $Z(\mathbf{x}; \mathbf{w}) = \sum_l \alpha(n, l)$

# Marginal Probability of a Label Bigram

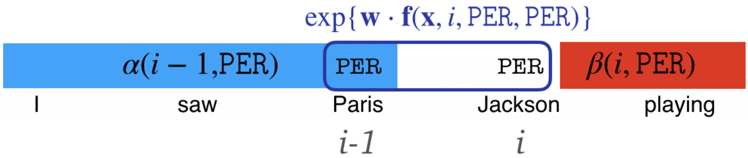
| | | | | |
|---|---|---|---|---|
| | | PER | PER | |
| I | saw | Paris | Jackson | playing |
| | | *i-1* | *i* | |

- 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 $i$:

$$\begin{aligned}
\mu_i(a,b) = \Pr(y_{i-1}=a, y_i=b|\mathbf{x};\mathbf{w}) &= \sum_{\mathbf{y}\in\mathcal{Y}^n : y_{i-1}=a, y_i=b} \Pr(\mathbf{y}|\mathbf{x};\mathbf{w}) \\
&= \text{(algebraic massaging)} \\
&= \frac{\alpha(i-1,a) * \exp\{\mathbf{w}\cdot\mathbf{f}(\mathbf{x},i,a,b)\} * \beta(i,b)}{Z(\mathbf{x};\mathbf{w})}
\end{aligned}$$

- Again forward-backward (using exponentiated scores)
  - Recall that $Z(\mathbf{x};\mathbf{w}) = \sum_l \alpha(n,l)$

# 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 $i$:

$$\begin{aligned}\mu_i(a,b) = \Pr(y_{i-1}=a, y_i=b|\mathbf{x};\mathbf{w}) &= \sum_{\mathbf{y}\in\mathcal{Y}^n : y_{i-1}=a, y_i=b} \Pr(\mathbf{y}|\mathbf{x};\mathbf{w}) \\ &= \text{(algebraic massaging)} \\ &= \frac{\alpha(i-1,a) * \exp\{\mathbf{w}\cdot\mathbf{f}(\mathbf{x},i,a,b)\} * \beta(i,b)}{Z(\mathbf{x};\mathbf{w})}\end{aligned}$$

- Again forward-backward (using exponentiated scores)
  - Recall that $Z(\mathbf{x};\mathbf{w}) = \sum_l \alpha(n,l)$

# Linear Factored Sequence Prediction

$$\underset{\mathbf{y} \in \mathcal{Y}^n}{\operatorname{argmax}} \, \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})$$

- Factored representation, e.g. based on bigrams

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

- Flexible, arbitrary features of full $\mathbf{x}$ and the factors
- Efficient prediction using Viterbi
- In probabilistic models, efficient computation of marginals using Forward-Backward
- Next, learning $\mathbf{w}$:
  - 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

# The Structured Perceptron

## Collins (2002)

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

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(\mathbf{x}, \mathbf{y}) - \mathbf{f}(\mathbf{x}, \mathbf{z})$$

增加 已知正确标签y 的 w

减少 错误预测标签 z 的 w

注意 f 是 指示函数，取值0/1

- Return $\mathbf{w}$

# The Structured Perceptron + Averaging

## Freund and Schapire (1999); Collins (2002)

- Set $\mathbf{w} = \mathbf{0}$, $\mathbf{w}^a = \mathbf{0}$
- For $t = 1 \ldots T$
  - For each training example $(\mathbf{x}, \mathbf{y})$
    1. Compute $\mathbf{z} = \mathrm{argmax}_{\mathbf{z}}\, \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{z})$
    2. If $\mathbf{z} \neq \mathbf{y}$

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(\mathbf{x}, \mathbf{y}) - \mathbf{f}(\mathbf{x}, \mathbf{z})$$

    3. $\mathbf{w^a} = \mathbf{w^a} + \mathbf{w}$
- Return $\mathbf{w^a}$

# Perceptron Updates: Example

| | | | | | |
|---|---|---|---|---|---|
| $\mathbf{y}$ | PER | PER | - | - | LOC |
| $\mathbf{z}$ | PER | LOC | - | - | LOC |
| $\mathbf{x}$ | Jack | London | went | to | Paris |

- Let $\mathbf{y}$ be the correct output for $\mathbf{x}$.
- Say we predict $\mathbf{z}$ instead, under our current $\mathbf{w}$
- The update is:

$$
\begin{aligned}
\mathbf{g} &= \mathbf{f}(\mathbf{x}, \mathbf{y}) - \mathbf{f}(\mathbf{x}, \mathbf{z}) \\
&= \sum_i \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i) - \sum_i \mathbf{f}(\mathbf{x}, i, z_{i-1}, z_i) \\
&= \mathbf{f}(\mathbf{x}, 2, \text{PER}, \text{PER}) - \mathbf{f}(\mathbf{x}, 2, \text{PER}, \text{LOC}) \\
&+ \mathbf{f}(\mathbf{x}, 3, \text{PER}, \text{-}) - \mathbf{f}(\mathbf{x}, 3, \text{LOC}, \text{-})
\end{aligned}
$$

- Perceptron updates are typically very sparse

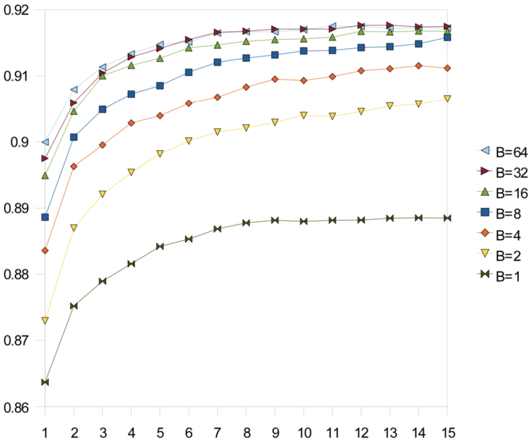
# Properties of the Perceptron

- 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 can not recover the $\mathrm{argmax}$ solution
  - Structured Perceptron can use beam search instead (i.e. an approximation to $\mathrm{argmax}$)
  - See Collins and Roark (2004); Zhang and Clark (2011); Huang et al. (2012)

# Averaged Perceptron Convergence

| Iteration | Accuracy |
|---|---|
| 1 | 90.79 |
| 2 | 91.20 |
| 3 | 91.32 |
| 4 | 91.47 |
| 5 | 91.58 |
| 6 | 91.78 |
| 7 | 91.76 |
| 8 | 91.82 |
| 9 | 91.88 |
| 10 | 91.91 |
| 11 | 91.92 |
| 12 | 91.96 |
| ... | |

results on validation set for a parsing task




perceptron with beam search
(Zhang and Clark, 2011)

# Outline

# Log-linear Models for Sequence Prediction

- Model the conditional distribution:

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

  where

  - $\mathbf{f}(\mathbf{x}, \mathbf{y})$ represents $\mathbf{x}$ and $\mathbf{y}$ with $d$ features
  - $\mathbf{w} \in \mathbb{R}^d$ are the parameters of the model
  - $Z(\mathbf{x}; \mathbf{w})$ is a normalizer called the *partition function*

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

- To predict the best sequence

$$\operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}^n} \Pr(\mathbf{y}|\mathbf{x})$$

# Parameter Estimation in Log-Linear Models

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

- Given training data

$$\left\{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \ldots, (\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\right\} \quad ,$$

- How to estimate $\mathbf{w}$?
- Define the conditional log-likelihood (or cross-entropy) of the data:

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

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

# Parameter Estimation in Log-Linear Models

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

- Given training data

$$\left\{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \ldots, (\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\right\} \quad ,$$

- How to estimate $\mathbf{w}$?
- Define the conditional log-likelihood (or cross-entropy) of the data:

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

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

# Learning Log-Linear Models: Loss + Regularization

- Solve:

$$\mathbf{w}^* = \underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmin}} \overbrace{-L(\mathbf{w})}^{\text{Loss}} + \overbrace{\frac{\lambda}{2}||\mathbf{w}||^2}^{\text{Regularization}}$$

  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 $\Pr(\mathbf{y} \mid \mathbf{x}; \mathbf{w})$ decomposes
  - Global Loss (CRF): assume that $\mathbf{f}(\mathbf{x}, \mathbf{y})$ decomposes

# Learning Log-Linear Models: Loss + Regularization

- Solve:

$$\mathbf{w}^* = \underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmin}} \overbrace{-L(\mathbf{w})}^{\text{Loss}} + \overbrace{\frac{\lambda}{2}||\mathbf{w}||^2}^{\text{Regularization}}$$

  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 $\Pr(\mathbf{y} \mid \mathbf{x}; \mathbf{w})$ decomposes
  - Global Loss (CRF): assume that $\mathbf{f}(\mathbf{x}, \mathbf{y})$ decomposes

# Local Log-linear Loss (a.k.a. Maximum Entropy Markov Models)

**McCallum, Freitag, and Pereira (2000)**

- If we apply the chain rule:

$$\begin{aligned}\Pr(\mathbf{y}_{1:n} \mid \mathbf{x}_{1:n}) &= \Pr(y_1 \mid \mathbf{x}_{1:n}) \times \Pr(\mathbf{y}_{2:n} \mid \mathbf{x}_{1:n}, y_1) \\ &= \Pr(y_1 \mid \mathbf{x}_{1:n}) \times \prod_{i=2}^{n} \Pr(y_i | \mathbf{x}_{1:n}, \mathbf{y}_{1:i-1})\end{aligned}$$

- Markov assumption (the model becomes factored):

$$\Pr(y_i | \mathbf{x}_{1:n}, \mathbf{y}_{1:i-1}) = \Pr(y_i | \mathbf{x}_{1:n}, y_{i-1})$$

- Now we can write

$$\Pr(\mathbf{y}_{1:n} \mid \mathbf{x}_{1:n}) = \Pr(y_1 | \mathbf{x}_{1:n}) \times \prod_{i=2}^{n} \Pr(y_i | \mathbf{x}_{1:n}, \mathbf{y}_{i-1})$$

# Parameter Estimation with Local Log-Linear Markov Models

$$\Pr(y_{1:n} \mid \mathbf{x}_{1:n}) \quad = \quad \Pr(y_1 \mid \mathbf{x}_{1:n}) \times \prod_{i=2}^{n} \Pr(y_i | \mathbf{x}_{1:n}, i, y_{i-1})$$

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

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

- The log-likelihood is also local :

$$L(\mathbf{w}) = \sum_{k=1}^{m} \sum_{i=1}^{n^{(k)}} \log \Pr(\mathbf{y}_i^{(k)} | \mathbf{x}^{(k)}, i, \mathbf{y}_{i-1}^{(k)})$$

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}_j} = \frac{1}{m} \sum_{k=1}^{m} \sum_{i=1}^{n^{(k)}} \left[ \overbrace{\mathbf{f}_j(\mathbf{x}^{(k)}, i, \mathbf{y}_{i-1}^{(k)}, \mathbf{y}_i^{(k)})}^{\text{observed}} - \overbrace{\sum_{y \in \mathcal{Y}} \Pr(\mathbf{y} | \mathbf{x}^{(k)}, i, \mathbf{y}_{i-1}^{(k)}, y) \; \mathbf{f}_j(\mathbf{x}^{(k)}, i, \mathbf{y}_{i-1}^{(k)}, y)}^{\text{expected}} \right]$$

# Conditional Random Fields

**Lafferty, McCallum, and Pereira (2001)**

- Log-linear model of the conditional distribution:

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

  where
  - $\mathbf{x}$ and $\mathbf{y}$ are input and output sequences
  - $\mathbf{f}(\mathbf{x},\mathbf{y})$ is a feature vector of $\mathbf{x}$ and $\mathbf{y}$ that decomposes into factors
  - $\mathbf{w}$ are model parameters
- To predict the best sequence

$$\hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}^*} \Pr(\mathbf{y}|\mathbf{x})$$

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

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

# Computing the Gradient in CRFs

Consider a parameter $\mathbf{w}_j$ and its associated feature $\mathbf{f}_j$:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}_j} = \frac{1}{m} \sum_{k=1}^{m} \left[ \overbrace{\mathbf{f}_j(\mathbf{x}^{(k)}, \mathbf{y}^{(k)})}^{\text{observed}} - \overbrace{\sum_{\mathbf{y} \in \mathcal{Y}^*} \Pr(\mathbf{y}|\mathbf{x}^{(k)}; \mathbf{w})\ \mathbf{f}_j(\mathbf{x}^{(k)}, \mathbf{y})}^{\text{expected}} \right]$$

where

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

- First term: observed value of $\mathbf{f}_j$ in training examples
- Second term: expected value of $\mathbf{f}_j$ under current $\mathbf{w}$
  they require summing over all sequences $\mathbf{y} \in \mathcal{Y}^n$

# Computing the Gradient in CRFs

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

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

- $\mu_i^k(a,b)$ is the marginal probability of having labels $(a,b)$ at position $i$:

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

- The quantities $\mu_i^k$ can be computed efficiently in $O(nL^2)$ using the forward-backward algorithm

# CRFs: summary so far

- Log-linear models for sequence prediction, $\Pr(\mathbf{y}|\mathbf{x};\mathbf{w})$
- Computations factorize on label bigrams
- Model form:

$$\operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}^*} \sum_i \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$$

- 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, $\Pr(\mathbf{y}|\mathbf{x};\mathbf{w})$
- Computations factorize on label bigrams
- Model form:

$$\operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}^*} \sum_i \mathbf{w}\cdot\mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$$

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

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

- Both exploit the same factorization, i.e. same features
- Same computations to compute $\operatorname{argmax}_{\mathbf{y}} \Pr(\mathbf{y} \mid \mathbf{x})$
- Local loss is locally normalized; CRFs globally normalized
  - Local loss assumes that $\Pr(y_i \mid x_{1:n}, y_{1:i-1}) = \Pr(y_i \mid x_{1:n}, y_{i-1})$
  - 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

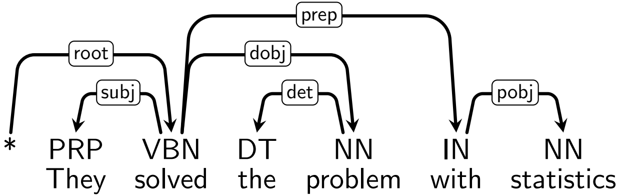
- Linear models for sequence prediction

$$\operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}^*} \sum_i \mathbf{w}\cdot\mathbf{f}(\mathbf{x}, i, y_{i-1}, y_i)$$

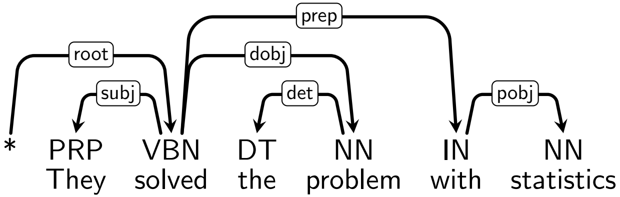
- Computations factorize on label bigrams
  - Decoding: using Viterbi
  - Marginals: using forward-backward
- Parameter estimation:
  - Perceptron, Log-likelihood, SVMs
  - Extensions from classification to the structured case
  - Optimization methods:
    - Stochastic (sub)gradient methods (LeCun et 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)

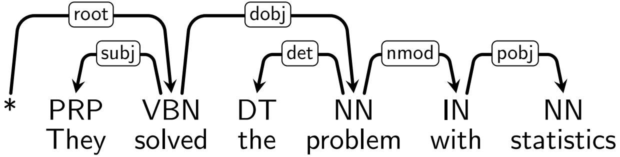
# Outline

# Dependency Parsing



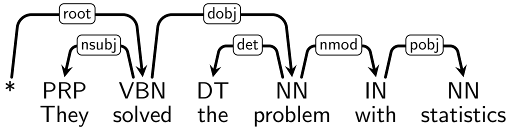
# Dependency Parsing



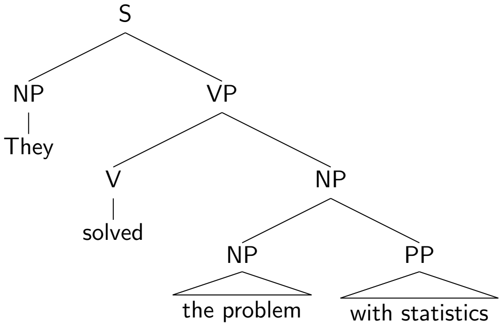
# Theories of Syntactic Structure

## Dependency Trees




- Main element: dependency
- Focus on relations between words

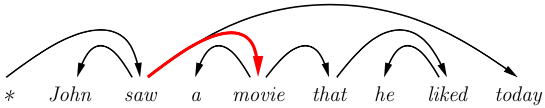
## Constituent Trees




- 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 $\langle h, m \rangle$

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

- Each arc or dependency $(h, m)$ is scored independently of each other
- Some features: $\mathbf{f}_1(\mathbf{x}, h, m) = [\ \text{"saw"} \ \rightarrow \text{"movie"}\ ]$
  $\mathbf{f}_2(\mathbf{x}, h, m) = [\ \text{distance} = +2\ ]$
- Tractable inference algorithms exist

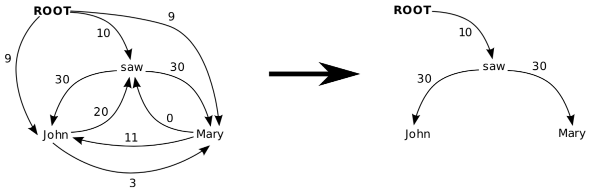
# MST Parsing for Arc-factored models

**McDonald, Pereira, Ribarov, and Hajič (2005)**

- Parsing problem, given a sentenc $\mathbf{x}$:

$$\underset{\mathbf{y} \in \mathcal{Y}(\mathbf{x})}{\operatorname{argmax}} \sum_{\langle h,m \rangle \in \mathbf{y}} \operatorname{score}(\mathbf{x}, h, m)$$

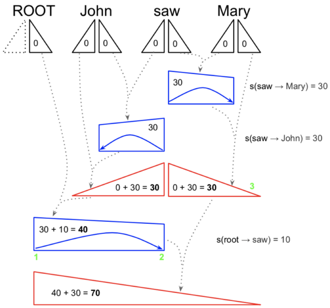
- Can be formulated as a directed Maximum Spanning Tree (MST) problem:




- The Chu-Liu-Edmonds algorithm finds the optimal tree in $O(n^2)$

# The Eisner Algorithm for Arc-factored models

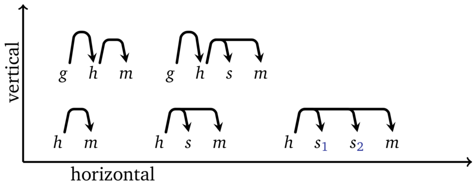
## Eisner (1996); McDonald and Pereira (2006); Carreras (2007); Koo and Collins (2010)




(illustration by Joakim Nivre)

- 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)


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

Mark Watney visited Mars

# Arc-Standard Parsing: Example
## (illustration by Miguel Ballesteros)


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

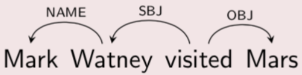
Mark Watney visited Mars

# Arc-Standard Parsing: Example
## (illustration by Miguel Ballesteros)

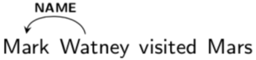

| **transition** | **Stack** | **Buffer** |
|---|---|---|
| | [ ] | [Mark, Watney, visited, Mars] |
| SHIFT | [Mark] | [Watney, visited, Mars] |
| SHIFT | [Mark, Watney] | [visited, Mars] |

Mark Watney visited Mars

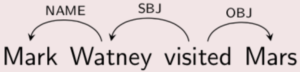
# Arc-Standard Parsing: Example
## (illustration by Miguel Ballesteros)



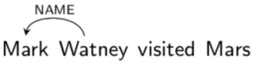

| **transition** | **Stack** | **Buffer** |
|---|---|---|
| | [ ] | [Mark, Watney, visited, Mars] |
| SHIFT | [Mark] | [Watney, visited, Mars] |
| SHIFT | [Mark, Watney] | [visited, Mars] |
| LA(NAME) | [Watney] | [visited, Mars] |



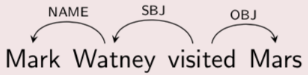
# Arc-Standard Parsing: Example
## (illustration by Miguel Ballesteros)



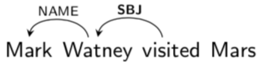

| transition | Stack | Buffer |
|---|---|---|
| | [ ] | [Mark, Watney, visited, Mars] |
| SHIFT | [Mark] | [Watney, visited, Mars] |
| SHIFT | [Mark, Watney] | [visited, Mars] |
| LA(NAME) | [Watney] | [visited, Mars] |
| SHIFT | [Watney, visited] | [Mars] |



# Arc-Standard Parsing: Example
## (illustration by Miguel Ballesteros)




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

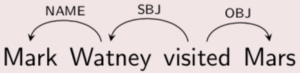


# Arc-Standard Parsing: Example

## (illustration by Miguel Ballesteros)


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

# Arc-Standard Parsing: Example
## (illustration by Miguel Ballesteros)




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

# Outline

# Linear (Structured) Prediction

- Multiclass classification

$$\operatorname*{argmax}_{\mathbf{y} \in \{1,\dots,L\}} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})$$

- Sequence prediction (bigram factorization)

$$\operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \sum_i \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, \mathbf{y}_{i-1}, \mathbf{y}_i)$$

- Dependency parsing (arc-factored)

$$\operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \sum_{\langle h,m,l \rangle \in y} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, h, m, l)$$

- 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

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

- Linear:

$$\text{s}(\mathbf{x}, i, y_{i-1}, y_i) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, \mathbf{y}_{i-1}, \mathbf{y}_i)$$

- Non-linear, using a feed-forward neural network:

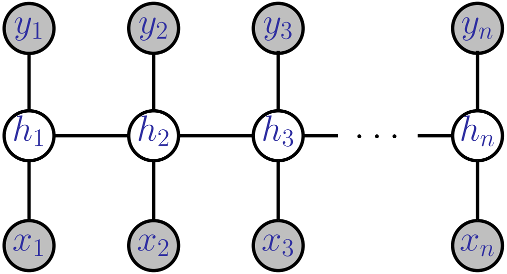
$$\text{s}(\mathbf{x}, i, y_{i-1}, y_i) = \mathbf{w} \cdot [e_{y_{i-1}, y_i} \otimes h(\mathbf{f}(\mathbf{x}, i))]$$

where:

$$h(\mathbf{f}(\mathbf{x}, i)) = \sigma(W^2 \sigma(W^1 \sigma(W^0 \mathbf{f}(\mathbf{x}, i))))$$

- 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

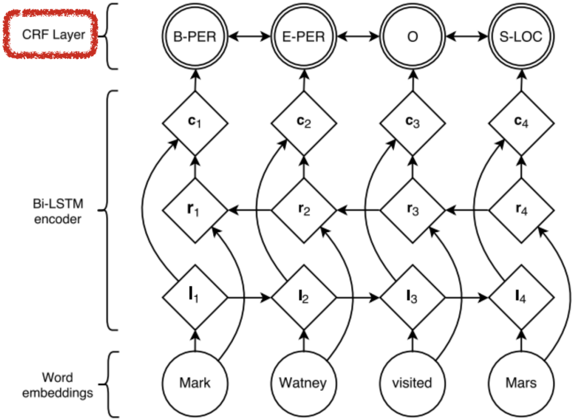
# Recurrent Sequence Prediction




- 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♠

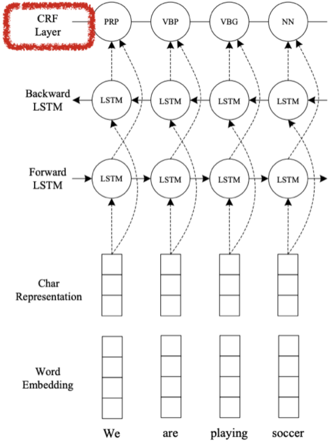



| Model | $F_1$ |
|---|---|
| Collobert et al. (2011)* | 89.59 |
| Lin and Wu (2009) | 83.78 |
| Lin and Wu (2009)* | 90.90 |
| Huang et al. (2015)* | 90.10 |
| Passos et al. (2014) | 90.05 |
| Passos et al. (2014)* | 90.90 |
| Luo et al. (2015)* + gaz | 89.9 |
| Luo et al. (2015)* + gaz + linking | **91.2** |
| Chiu and Nichols (2015) | 90.69 |
| Chiu and Nichols (2015)* | 90.77 |
| LSTM-CRF (no char) | 90.20 |
| LSTM-CRF | **90.94** |
| S-LSTM (no char) | 87.96 |
| S-LSTM | 90.33 |

**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**




| Model | POS Dev Acc. | POS Test Acc. | NER Dev Prec. | NER Dev Recall | NER Dev F1 | NER Test Prec. | NER Test Recall | NER Test F1 |
|---|---|---|---|---|---|---|---|---|
| BRNN | 96.56 | 96.76 | 92.04 | 89.13 | 90.56 | 87.05 | 83.88 | 85.44 |
| BLSTM | 96.88 | 96.93 | 92.31 | 90.85 | 91.57 | 87.77 | 86.23 | 87.00 |
| BLSTM-CNN | 97.34 | 97.33 | 92.52 | 93.64 | 93.07 | 88.53 | 90.21 | 89.36 |
| BRNN-CNN-CRF | 97.46 | 97.55 | 94.85 | 94.63 | 94.74 | 91.35 | 91.06 | 91.21 |

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

| Model | Acc. |
|---|---|
| Giménez and Màrquez (2004) | 97.16 |
| Toutanova et al. (2003) | 97.27 |
| Manning (2011) | 97.28 |
| Collobert et al. (2011)‡ | 97.29 |
| Santos and Zadrozny (2014)‡ | 97.32 |
| Shen et al. (2007) | 97.33 |
| Sun (2014) | 97.36 |
| Søgaard (2011) | 97.50 |
| **This paper** | **97.55** |

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

| Model | F1 |
|---|---|
| Chieu and Ng (2002) | 88.31 |
| Florian et al. (2003) | 88.76 |
| Ando and Zhang (2005) | 89.31 |
| Collobert et al. (2011)‡ | 89.59 |
| Huang et al. (2015)‡ | 90.10 |
| Chiu and Nichols (2015)‡ | 90.77 |
| Ratinov and Roth (2009) | 90.80 |
| Lin and Wu (2009) | 90.90 |
| Passos et al. (2014) | 90.90 |
| Lample et al. (2016)‡ | 90.94 |
| Luo et al. (2015) | 91.20 |
| **This paper** | **91.21** |

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

Thanks!

# References I


Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1231.

Xavier Carreras. Experiments with a higher-order projective dependency parser. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002.

Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics, 2004.

Michael Collins, Amir Globerson, Terry Koo, Xavier Carreras, and Peter L Bartlett. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. *The Journal of Machine Learning Research*, 9:1775–1822, 2008.

Koby Crammer, Ryan McDonald, and Fernando Pereira. Scalable large-margin online learning for structured classification. In *NIPS Workshop on Learning With Structured Outputs*, 2005.

Jason M Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics, 1996.

Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Mach. Learn.*, 37(3):277–296, December 1999. ISSN 0885-6125. doi: 10.1023/A:1007662407062. URL http://dx.doi.org/10.1023/A:1007662407062.

Liang Huang, Suphan Fayong, and Yang Guo. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151. Association for Computational Linguistics, 2012.

Terry Koo and Michael Collins. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11. Association for Computational Linguistics, 2010.

Terry Koo, Xavier Carreras, and Michael Collins. Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, pages 595–603, 2008.

# References II


Sanjiv Kumar and Martial Hebert. Man-made structure detection in natural images using a causal multiscale random field. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003), 16-22 June 2003, Madison, WI, USA*, pages 119–126, 2003. doi: 10.1109/CVPR.2003.1211345. URL https://doi.org/10.1109/CVPR.2003.1211345.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL http://dl.acm.org/citation.cfm?id=645530.655813.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

Andrew McCallum, Dayne Freitag, and Fernando CN Pereira. Maximum entropy markov models for information extraction and segmentation. In *Icml*, volume 17, pages 591–598, 2000.

Ryan McDonald and Fernando Pereira. Online learning of approximate dependency parsing algorithms. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics, 2005.

Joakim Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553, December 2008. ISSN 0891-2017.

Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of machine learning research*, 6(Sep):1453–1484, 2005.

Yue Zhang and Stephen Clark. Syntactic processing using the generalized perceptron and beam search. *Computational linguistics*, 37(1):105–151, 2011.