Loss-augmented Structured Prediction

CMSC 723 / LING 723 / INST 725

Marine Carpuat

Figures, algorithms & equations
from CIML chap 17

POS tagging
Sequence labeling with the perceptron

Sequence labeling problem

- Input:
  - sequence of tokens $x = [x_1 \dots x_L]$
  - Variable length L
- Output (aka label):
  - sequence of tags $y = [y_1 \dots y_L]$
  - # tags = K
  - Size of output space?

Structured Perceptron

- Perceptron algorithm can be used for sequence labeling
- But there are challenges
  - How to compute argmax efficiently?
  - What are appropriate features?
- Approach: leverage structure of output space

Solving the argmax problem for sequences with dynamic programming

$x =$ " monsters eat tasty bunnies "

$y =$ noun verb adj noun

- Efficient algorithms possible if **the feature function decomposes over the input**

- This holds for unary and markov features used for POS tagging

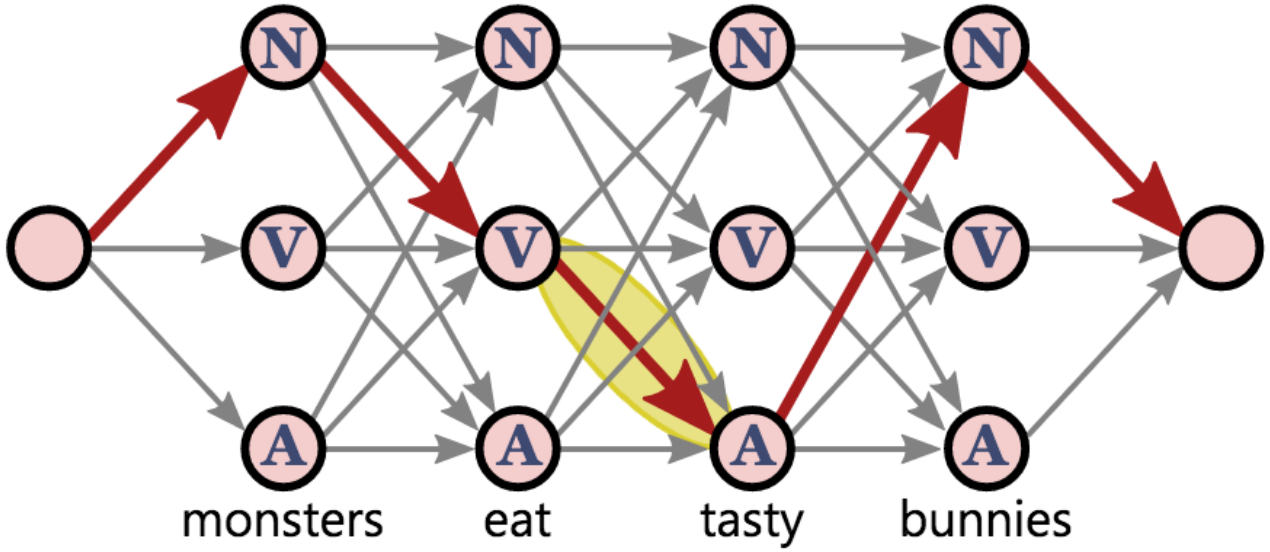
Feature functions for sequence labeling

$\boldsymbol{x}$ = " monsters eat tasty bunnies "

$\boldsymbol{y}$ = noun verb adj noun

- Standard features of POS tagging
  - **Unary features:** # times word w has been labeled with tag l for all words w and all tags l
  - **Markov features:** # times tag l is adjacent to tag l' in output for all tags l and l'
- Size of feature representation is constant wrt input length

Solving the argmax problem for sequences



- Trellis sequence labeling
  - Any path represents a labeling of input sentence
  - Gold standard path in red
  - Each edge receives a weight such that adding weights along the path corresponds to score for input/ouput configuration
- Any max-weight max-weight path algorithm can find the argmax
  - e.g. Viterbi algorithm $O(LK^2)$

Defining weights of edge in treillis

Unary features at position l together with Markov features that end at position l

$$\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{w} \cdot \sum_{l=1}^{L} \boldsymbol{\phi}_l(\boldsymbol{x}, \boldsymbol{y}) \qquad \text{decomposition of structure} \qquad (17.35)$$

$$= \sum_{l=1}^{L} \boldsymbol{w} \cdot \boldsymbol{\phi}_l(\boldsymbol{x}, \boldsymbol{y}) \qquad \text{associative law} \qquad (17.36)$$

- Weight of edge that goes from time l-1 to time l, and transitions from y to y'

$$\boldsymbol{w} \cdot \boldsymbol{\phi}_l(\boldsymbol{x}, \cdots \circ y \circ y')$$

Dynamic program

- Define: the score of best possible output prefix up to and including position l that labels the l-th word with label k

$$\alpha_{l,k} = \max_{\hat{\boldsymbol{y}}_{1:l-1}} \boldsymbol{w} \cdot \boldsymbol{\phi}_{1:l}(\boldsymbol{x}, \hat{\boldsymbol{y}} \circ k)$$

- With decomposable features, alphas can be computed recursively

$$\alpha_{l+1,k} = \max_{k'} \left[\alpha_{l,k'} + \boldsymbol{w} \cdot \boldsymbol{\phi}_{l+1}(\boldsymbol{x}, \langle \ldots, k', k \rangle) \right]$$

$$\alpha_{0,k} = 0 \quad \forall k \tag{17.41}$$

$$\zeta_{0,k} = \emptyset \quad \forall k \tag{17.42}$$

the score for any empty sequence is zero

$$\alpha_{l+1,k} = \max_{\hat{\boldsymbol{y}}_{1:l}} \boldsymbol{w} \cdot \boldsymbol{\phi}_{1:l+1}(\boldsymbol{x}, \hat{\boldsymbol{y}} \circ k) \tag{17.43}$$

separate score of prefix from score of position l+1

$$= \max_{\hat{\boldsymbol{y}}_{1:l}} \boldsymbol{w} \cdot \left(\boldsymbol{\phi}_{1:l}(\boldsymbol{x}, \hat{\boldsymbol{y}}) + \boldsymbol{\phi}_{l+1}(\boldsymbol{x}, \hat{\boldsymbol{y}} \circ k) \right) \tag{17.44}$$

distributive law over dot products

$$= \max_{\hat{\boldsymbol{y}}_{1:l}} \left[\boldsymbol{w} \cdot \boldsymbol{\phi}_{1:l}(\boldsymbol{x}, \hat{\boldsymbol{y}}) + \boldsymbol{w} \cdot \boldsymbol{\phi}_{l+1}(\boldsymbol{x}, \hat{\boldsymbol{y}} \circ k) \right] \tag{17.45}$$

separate out final label from prefix, call it k'

$$= \max_{\hat{\boldsymbol{y}}_{1:l-1}} \max_{k'} \left[\boldsymbol{w} \cdot \boldsymbol{\phi}_{1:l}(\boldsymbol{x}, \hat{\boldsymbol{y}} \circ k') + \boldsymbol{w} \cdot \boldsymbol{\phi}_{l+1}(\boldsymbol{x}, \hat{\boldsymbol{y}} \circ k' \circ k) \right] \tag{17.46}$$

swap order of maxes, and last term doesn't depend on prefix

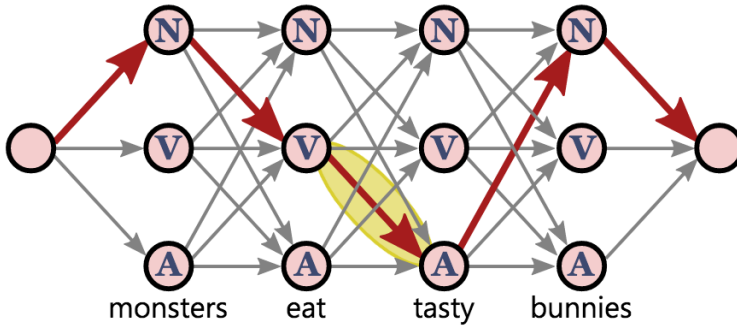
$$= \max_{k'} \left[\left[\max_{\hat{\boldsymbol{y}}_{1:l-1}} \boldsymbol{w} \cdot \boldsymbol{\phi}_{1:l}(\boldsymbol{x}, \hat{\boldsymbol{y}} \circ k') \right] + \boldsymbol{w} \cdot \boldsymbol{\phi}_{l+1}(\boldsymbol{x}, \langle \ldots, k', k \rangle) \right] \tag{17.47}$$

apply recursive definition

$$= \max_{k'} \left[\alpha_{l,k'} + \boldsymbol{w} \cdot \boldsymbol{\phi}_{l+1}(\boldsymbol{x}, \langle \ldots, k', k \rangle) \right] \tag{17.48}$$

Algorithm 42 ARGMAXFORSEQUENCES(x, w)

1: $L \leftarrow \text{LEN}(x)$
2: $\alpha_{l,k} \leftarrow o, \quad \zeta_{k,l} \leftarrow o, \quad \forall\, k = 1 \ldots K, \quad \forall l = 0 \ldots L$ // initialize variables
3: **for** $l = o \ldots L\text{-}1$ **do**
4: **for** $k = 1 \ldots K$ **do**
5: $\alpha_{l+1,k} \leftarrow \max_{k'} \left[\alpha_{l,k'} + w \cdot \phi_{l+1}(x, \langle \ldots, k', k \rangle)\right]$ // recursion:
// here, $\phi_{l+1}(\ldots k', k \ldots)$ is the set of features associated with
// output position $l+1$ and two adjacent labels k' and k at that position
6: $\zeta_{l+1,k} \leftarrow$ the k' that achieves the maximum above // store backpointer
7: **end for**
8: **end for**
9: $y \leftarrow \langle o, o, \ldots, o \rangle$ // initialize predicted output to L-many zeros
10: $y_L \leftarrow \text{argmax}_k\, \alpha_{L,k}$ // extract highest scoring final label
11: **for** $l = L\text{-}1 \ldots 1$ **do**
12: $y_l \leftarrow \zeta_{l,y_{l+1}}$ // traceback ζ based on y_{l+1}
13: **end for**
14: **return** y // return predicted output



A more general approach for argmax Integer Linear Programming

- ILP: optimization problem of the form, for a fixed vector a

$$\max_{z} \quad a \cdot z \quad \text{subj. to} \quad \text{linear constraints on } z$$

  - With integer constraints

- Pro: can leverage well-engineered solvers (e.g., Gurobi)
- Con: not always most efficient

POS tagging as ILP

- Markov features as binary indicator variables

$$z_{l,k',k} = \mathbf{1}[\text{label } l \text{ is } k \text{ and label } l-1 \text{ is } k']$$

  - Output sequence: y(z) obtained by reading off variables z
- Define a such that a.z is equal to score

$$a_{l,k',k} = \boldsymbol{w} \cdot \boldsymbol{\phi}_l(\boldsymbol{x}, \langle \ldots, k', k \rangle)$$

- Enforcing constraints for well formed solutions

1. That all the zs are binary. That's easy: just say $z_{l,k',k} \in \{0,1\}$, for all l, k', k.
2. That for a given position l, there is exactly one active z. We can do this with an equality constraint: $\sum_k \sum_{k'} z_{l,k',k} = 1$ for all l.
3. That the zs are internally consistent: if the label at position 5 is supposed to be "noun" then both $z_{5,.,.}$ *and* $z_{6,.,.}$ need to agree on this. We can do this as: $\sum_{k'} z_{l,k',k} = \sum_{k''} z_{l+1,k,k''}$ for all l, k. Effectively what this is saying is that $z_{5,?,\text{verb}} = z_{6,\text{verb},?}$ where the "?" means "sum over all possibilities."

Sequence labeling

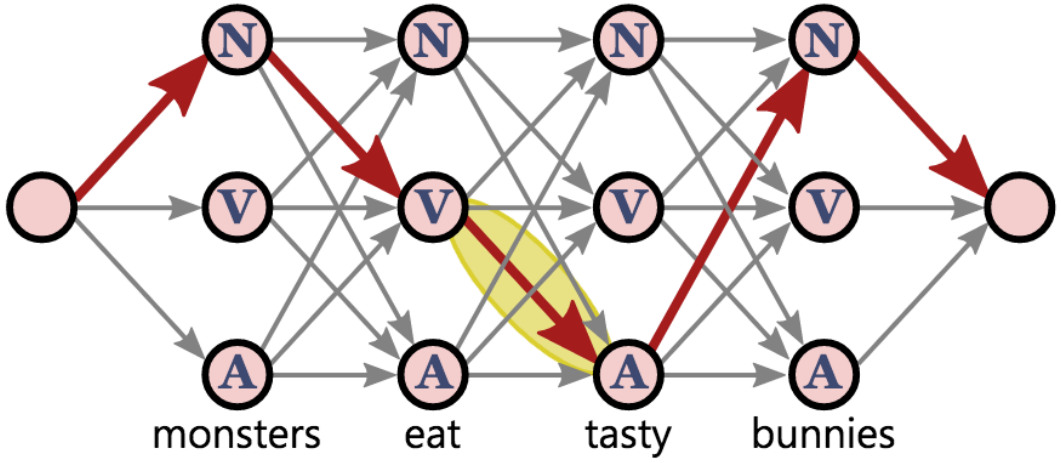
- Structured perceptron
  - A general algorithm for structured prediction problems such as sequence labeling
- The Argmax problem
  - Efficient argmax for sequences with Viterbi algorithm, given some assumptions on feature structure
  - A more general solution: Integer Linear Programming
- Loss-augmented structured prediction
  - Training algorithm
  - Loss-augmented argmax

In structured perceptron, all errors are equally bad

Algorithm 40 StructuredPerceptronTrain(D, *MaxIter*)

```
1  w ← 0                                                    // initialize weights
2  for iter = 1 ... MaxIter do
3      for all (x,y) ∈ D do
4          ŷ ← argmax_{ŷ∈Y(x)} w · φ(x, ŷ)                  // compute prediction
5          if ŷ ≠ y then
6              w ← w + φ(x, y) − φ(x, ŷ)                    // update weights
7          end if
8      end for
9  end for
10 return w                                                 // return learned weights
```

All bad output sequences are not equally bad



- Consider
- $\widehat{y_1} = [A, A, A, A]$
- $\widehat{y_2} = [N, V, N, N]$

- Hamming Loss
  - Gives a more nuanced evaluation of output than 0–1 loss

$$\ell^{(\text{Ham})}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{l=1}^{L} \mathbf{1}[\boldsymbol{y}_l \neq \hat{\boldsymbol{y}}_l]$$

Loss functions for structured prediction

- Recall learning as optimization for classification
  - e.g., $\min_{w} \quad \frac{1}{2}||w||^2 + C\sum_n \ell^{(\text{hin})}(y_n, w \cdot x_n + b)$
- Let's define a structure-aware optimization objective
  - e.g., $\min_{w} \quad \frac{1}{2}||w||^2 + C\sum_n \ell^{(\text{s-h})}(y_n, x_n, w)$

$$\ell^{(\text{s-h})}(y_n, x_n, w) = \max\left\{0, \max_{\hat{y}\in\mathcal{Y}(x_n)}\left[s_w(x_n, \hat{y}) + \ell^{(\text{Ham})}(y_n, \hat{y})\right] - s_w(x_n, y_n)\right\}$$

Structured hinge loss

- 0 if true output beats score of every imposter output
- Otherwise: scales linearly as function of score diff between most confusing imposter and true output

Optimization: stochastic *sub*gradient descent

- Subgradients of structured hinge loss?

$$= \max \left\{ 0, \max_{\hat{y} \in \mathcal{Y}(x_n)} \left[s_w(x_n, \hat{y}) + \ell^{(\text{Ham})}(y_n, \hat{y}) \right] - s_w(x_n, y_n) \right\}$$

$$\nabla_w \ell^{(\text{s-h})}(y, x, w) \quad \textit{if the loss is} > 0 \tag{17.25}$$

expand definition using arbitrary structured loss ℓ

$$= \nabla_w \left\{ \max_{\hat{y} \in \mathcal{Y}(x_n)} \left[w \cdot \phi(x_n, \hat{y}) + \ell(y_n, \hat{y}) \right] - w \cdot \phi(x_n, y_n) \right\} \tag{17.26}$$

define $\hat{y}_n$ to be the output that attains the maximum above, rearrange

$$= \nabla_w \left\{ w \cdot \phi(x_n, \hat{y}) - w \cdot \phi(x_n, y_n) + \ell(y_n, \hat{y}) \right\} \tag{17.27}$$

take gradient

$$= \phi(x_n, \hat{y}) - \phi(x_n, y_n) \tag{17.28}$$

Optimization: stochastic subgradient descent

- subgradients of structured hinge loss

$$\nabla_{\boldsymbol{w}} \ell^{(\text{s-h})}(\boldsymbol{y}_n, \boldsymbol{x}_n, \boldsymbol{w}) = \begin{cases} \mathbf{0} & \text{if } \ell^{(\text{s-h})}(\boldsymbol{y}_n, \boldsymbol{x}_n, \boldsymbol{w}) = 0 \\ \boldsymbol{\phi}(\boldsymbol{x}_n, \hat{\boldsymbol{y}}_n) - \boldsymbol{\phi}(\boldsymbol{x}_n, \boldsymbol{y}_n) & \text{otherwise} \end{cases}$$

$$\text{where } \hat{\boldsymbol{y}}_n = \underset{\hat{\boldsymbol{y}}_n \in \mathcal{Y}(\boldsymbol{x}_n)}{\text{argmax}} \left[\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_n, \hat{\boldsymbol{y}}_n) + \ell(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n) \right] \quad (17.29)$$

Optimization: stochastic subgradient descent Resulting training algorithm

Algorithm 41 StochSubGradStructSVM(**D**, *MaxIter*, λ, ℓ)

1: $w \leftarrow 0$ // initialize weights
2: **for** $iter = 1 \ldots MaxIter$ **do**
3: **for all** $(x,y) \in \mathbf{D}$ **do**
4: $\hat{y} \leftarrow \text{argmax}_{\hat{y} \in \mathcal{Y}(x)} \; w \cdot \phi(x, \hat{y}) + \ell(y, \hat{y})$ // loss-augmented prediction
5: **if** $\hat{y} \neq y$ **then**
6: $w \leftarrow w + \phi(x, y) - \phi(x, \hat{y})$ // update weights
7: **end if**
8: $w \leftarrow w - \frac{\lambda}{N} w$ // shrink weights due to regularizer
9: **end for**
10: **end for**
11: **return** w // return learned weights

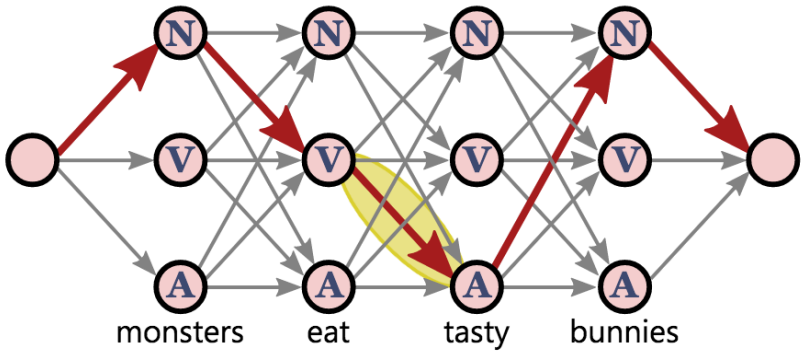
Only 2 differences compared to structured perceptron!

Loss-augmented inference/search

Recall dynamic programming solution without Hamming loss

$$\tilde{\alpha}_{l+1,k} = \max_{\hat{\boldsymbol{y}}_{1:l}} \boldsymbol{w} \cdot \boldsymbol{\phi}_{1:l+1}(\boldsymbol{x}, \hat{\boldsymbol{y}} \circ k)$$

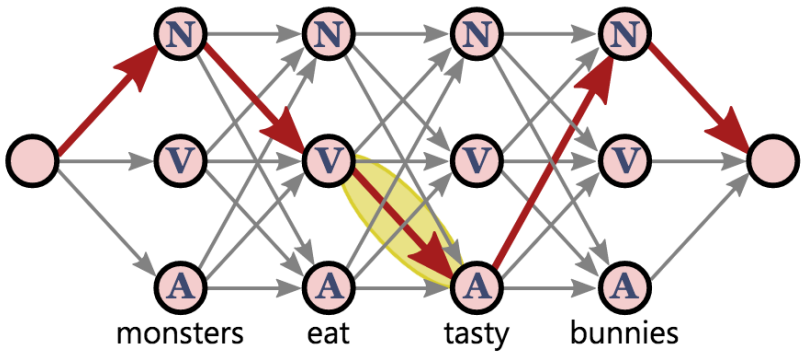
$$= \max_{k'} \left[\tilde{\alpha}_{l,k'} + \boldsymbol{w} \cdot \boldsymbol{\phi}_{l+1}(\boldsymbol{x}, \langle \ldots, k', k \rangle) \right]$$



Loss-augmented inference/search
Dynamic programming with Hamming loss

$$\tilde{\alpha}_{l+1,k} = \max_{\hat{\boldsymbol{y}}_{1:l}} \boldsymbol{w} \cdot \boldsymbol{\phi}_{1:l+1}(\boldsymbol{x}, \hat{\boldsymbol{y}} \circ k) + \ell^{(\text{Ham})}_{1:l+1}(\boldsymbol{y}, \hat{\boldsymbol{y}} \circ k)$$

$$= \max_{k'} \left[\tilde{\alpha}_{l,k'} + \boldsymbol{w} \cdot \boldsymbol{\phi}_{l+1}(\boldsymbol{x}, \langle \ldots, k', k \rangle) \right] + \mathbf{1}[k \neq \boldsymbol{y}_{l+1}]$$



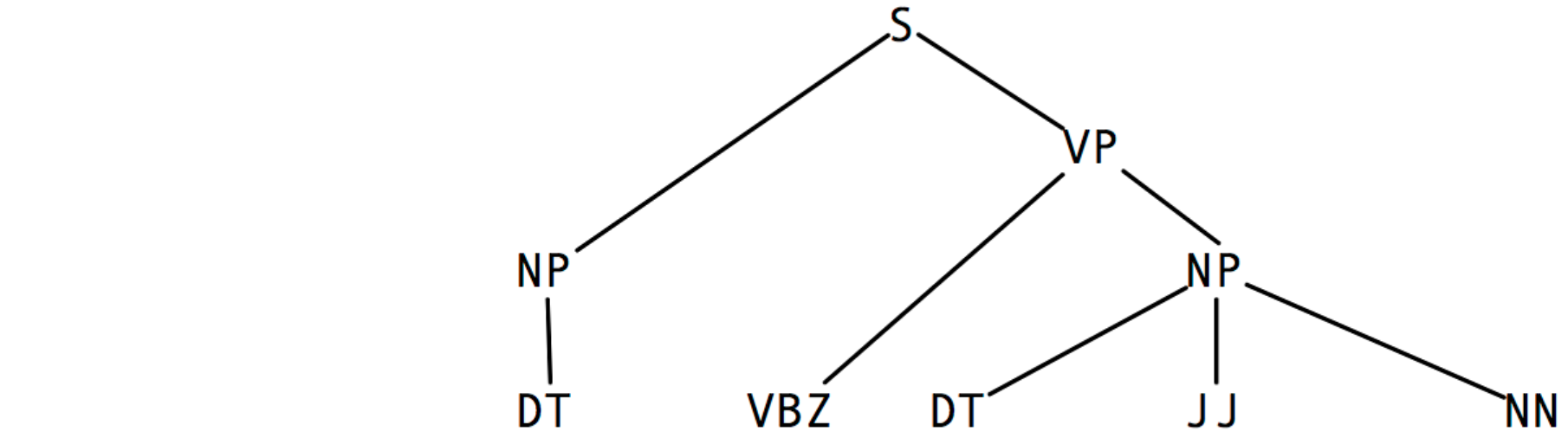
We can use Viterbi algorithm as before **as long as the loss function decomposes over the input consistently w features!**

Sequence labeling

- Structured perceptron
  - A general algorithm for structured prediction problems such as sequence labeling
- The Argmax problem
  - Efficient argmax for sequences with Viterbi algorithm, given some assumptions on feature structure
  - A more general solution: Integer Linear Programming
- Loss-augmented structured prediction
  - Training algorithm
  - Loss-augmented argmax

Syntax & Grammars

From Sequences to Trees



SYNTAX

PART OF SPEECH

This is a simple sentence

WORDS

MORPHOLOGY

be
3sg
present

SIMPLE1
having
few parts

SENTENCE1
string of words
satisfying the
grammatical rules
of a language

SEMANTICS

CONTRAST

But it is an instructive one.

DISCOURSE

Syntax & Grammar

- Syntax
  - From Greek syntaxis, meaning “setting out together”
  - refers to the way words are arranged together.

- Grammar
  - Set of structural rules governing composition of clauses, phrases, and words in any given natural language
  - Descriptive, not prescriptive
  - Panini’s grammar of Sanskrit ~2000 years ago

Syntax and Grammar

- Goal of syntactic theory
  - “explain how people combine words to form sentences and how children attain knowledge of sentence structure”

- Grammar
  - implicit knowledge of a native speaker
  - acquired without explicit instruction
  - minimally able to generate all and only the possible sentences of the language

[Philips, 2003]

Syntax in NLP

- Syntactic analysis often a key component in applications
  - Grammar checkers
  - Dialogue systems
  - Question answering
  - Information extraction
  - Machine translation
  - …

Two views of syntactic structure

- Constituency (phrase structure)
  - Phrase structure organizes words in nested constituents

- Dependency structure
  - Shows which words depend on (modify or are arguments of) which on other words

Constituency

- Basic idea: groups of words act as a single unit

- Constituents form coherent classes that behave similarly
  - With respect to their internal structure: e.g., at the core of a noun phrase is a noun
  - With respect to other constituents: e.g., noun phrases generally occur before verbs

Constituency: Example

- The following are all noun phrases in English...

Harry the Horse	a high-class spot such as Mindy's
the Broadway coppers	the reason he comes into the Hot Box
they	three parties from Brooklyn

- Why?
  - They can all precede verbs
  - They can all be preposed/postposed
  - …

Grammars and Constituency

- For a particular language:
  - What are the “right” set of constituents?
  - What rules govern how they combine?

- Answer: not obvious and difficult
  - That’s why there are many different theories of grammar and competing analyses of the same data!

- Our approach
  - Focus primarily on the “machinery”

Context-Free Grammars

- Context-free grammars (CFGs)
  - Aka phrase structure grammars
  - Aka Backus-Naur form (BNF)
- Consist of
  - Rules
  - Terminals
  - Non-terminals

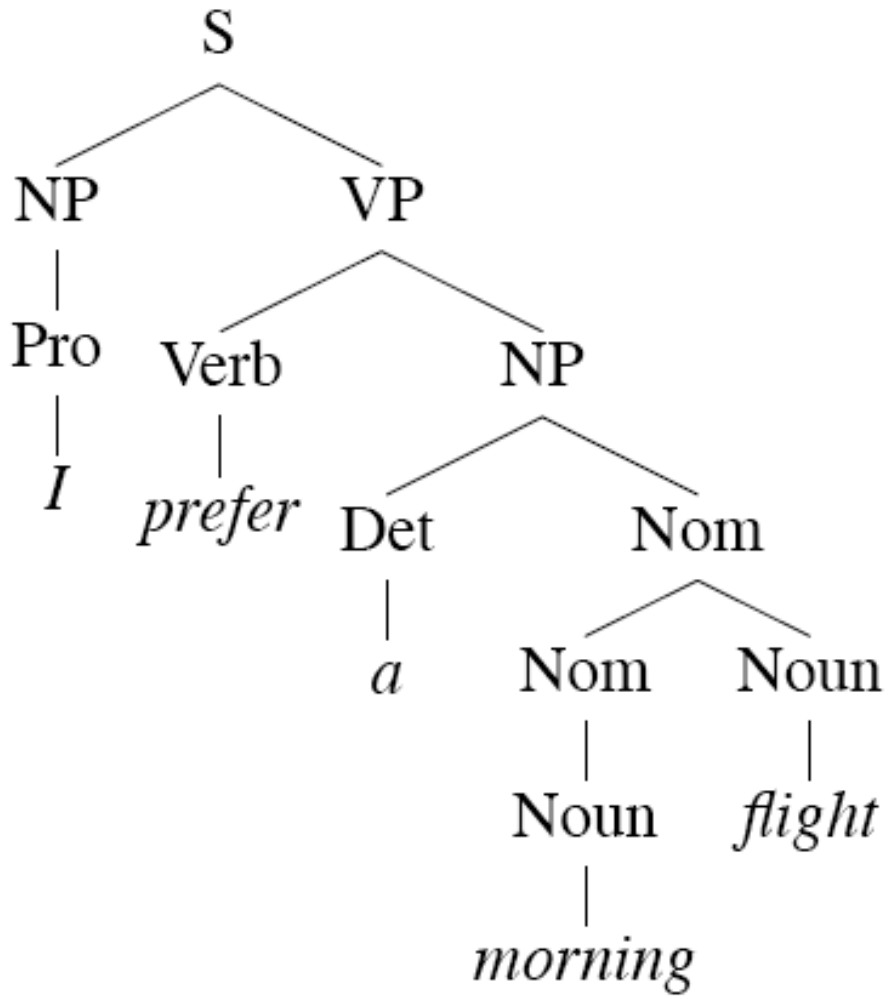
Context-Free Grammars

- Terminals
  - We'll take these to be words
- Non-Terminals
  - The constituents in a language (e.g., noun phrase)
- Rules
  - Consist of a single non-terminal on the left and any number of terminals and non-terminals on the right

An Example Grammar

Grammar Rules			Examples
S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

Parse Tree: Example



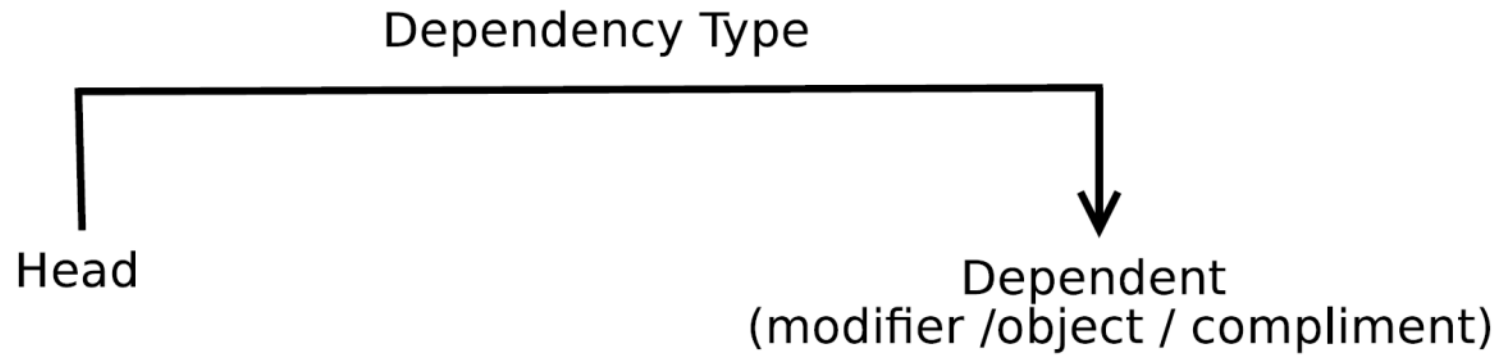
Dependency Grammars

- CFGs focus on constituents
  - Non-terminals don’t actually appear in the sentence

- In dependency grammar, a parse is a graph (usually a tree) where:
  - Nodes represent words
  - Edges represent dependency relations between words
    (typed or untyped, directed or undirected)

Dependency Grammars

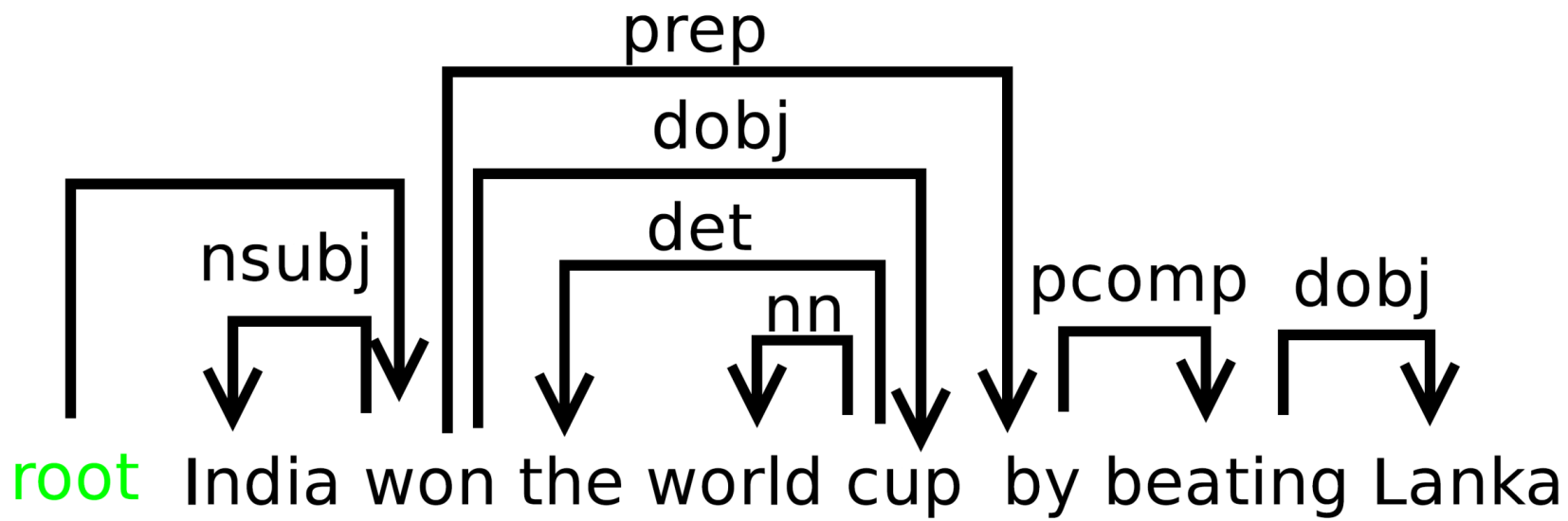
- Syntactic structure = lexical items linked by binary asymmetrical relations called dependencies



Dependency Relations

Argument Dependencies	Description
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

Example Dependency Parse



Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
DOBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

Figure 14.3 Examples of core Universal Dependency relations.

Universal Dependencies project

- Set of dependency relations that are
  - Linguistically motivated
  - Computationally useful
  - Cross-linguistically applicable
  - [Nivre et al. 2016]

- Universaldependencies.org

Summary

- Syntax & Grammar

- Two views of syntactic structures
  - Context-Free Grammars
  - Dependency grammars
  - Can be used to capture various facts about the structure of language (but not all!)

- Treebanks as an important resource for NLP