

# TTIC 31190: Natural Language Processing

Kevin Gimpel  
Winter 2016

Lecture 13:  
Dependency Syntax/Parsing  
& Review for Midterm

# Announcement

- project proposal due today
- email me to set up a 15-minute meeting next week to discuss your project proposal
- times posted on course webpage
- let me know if none of those work for you

# Announcement

- midterm is Thursday, room #530
- closed-book, but you can bring an 8.5x11 sheet (though I don't think you'll need to)
- we will start at 10:35 am, finish at 11:50 am

# Roadmap

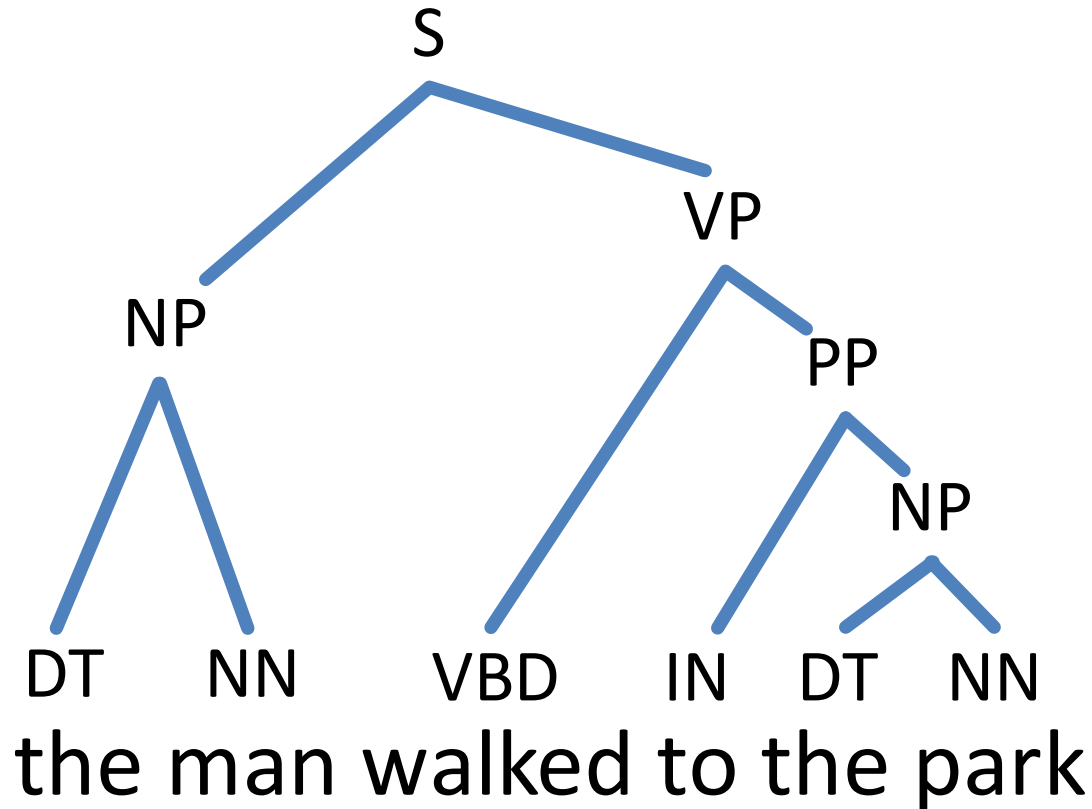
- classification
- words
- lexical semantics
- language modeling
- sequence labeling
- neural network methods in NLP
- **syntax and syntactic parsing**
- semantic compositionality
- semantic parsing
- unsupervised learning
- machine translation and other applications

# What is Syntax?

- rules, principles, processes that govern sentence structure of a language
- can differ widely among languages
- but every language has systematic structural principles

# Constituent Parse (Bracketing/Tree)

(S (NP the man) (VP walked (PP to (NP the park))))



Key:

S = sentence

NP = noun phrase

VP = verb phrase

PP = prepositional phrase

DT = determiner

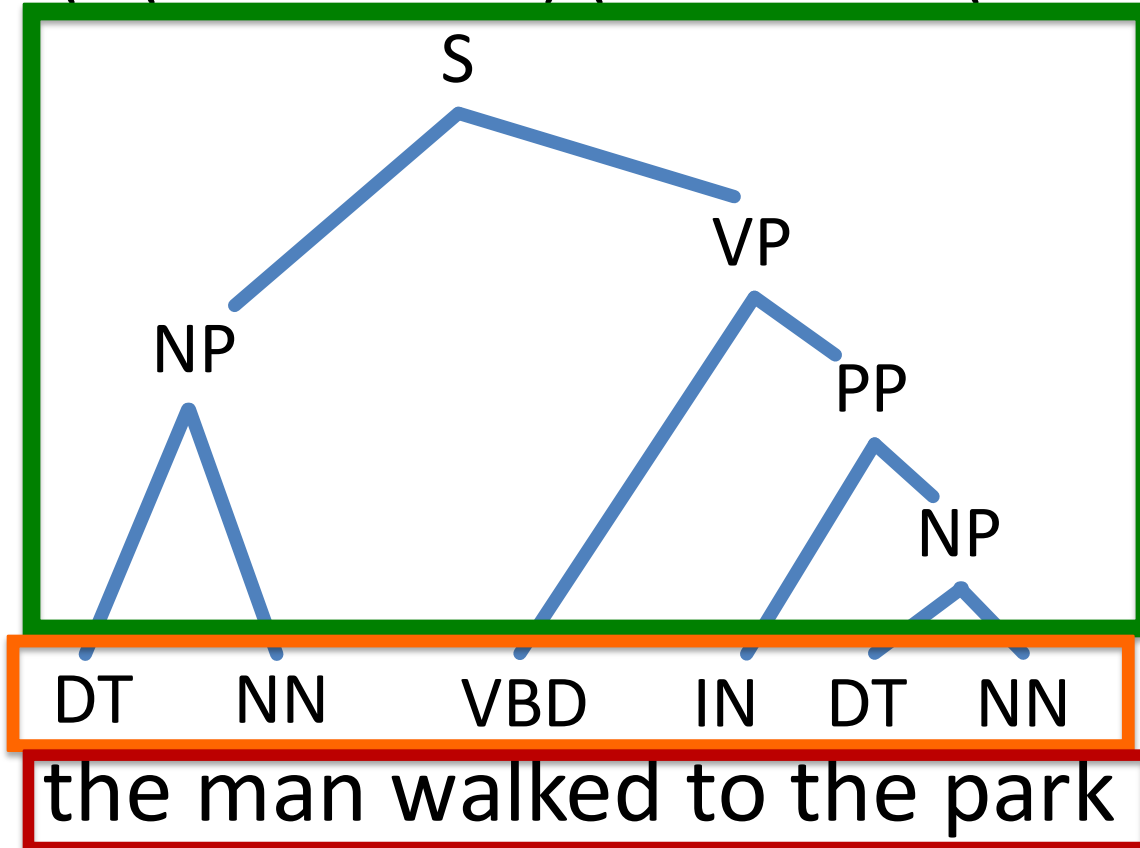
NN = noun

VBD = verb (past tense)

IN = preposition

# Constituent Parse (Bracketing/Tree)

(S (NP the man) (VP walked (PP to (NP the park))))



**nonterminals**

**preterminals**

**terminals**

# Penn Treebank Nonterminals

S	Sentence or clause.	PP	Prepositional Phrase.
SBAR	Clause introduced by a (possibly empty) subordinating conjunction.	PRN	Parenthetical.
SBARQ	Direct question introduced by a <i>wh</i> -word or <i>wh</i> -phrase.	PRT	Particle.
SINV	Inverted declarative sentence.	QP	Quantity Phrase (i.e., complex measure/amount) within NP.
SQ	Inverted yes/no question, or main clause of a <i>wh</i> -question.	RRC	Reduced Relative Clause.
ADJP	Adjective Phrase.	UCP	Unlike Coordinated Phrase.
ADVP	Adverb Phrase.	VP	Verb Phrase.
CONJP	Conjunction Phrase.	WHADJP	<i>Wh</i> -adjective Phrase, as in <i>how hot</i> .
FRAG	Fragment.	WHADVP	<i>Wh</i> -adverb Phrase.
INTJ	Interjection.	WHNP	<i>Wh</i> -noun Phrase, e.g. <i>who</i> , <i>which book</i> , <i>whose daughter</i> , <i>none of which</i> , or <i>how many leopards</i> .
LST	List marker. Includes surrounding punctuation.	WHPP	<i>Wh</i> -prepositional Phrase, e.g., <i>of which</i> or <i>by whose authority</i> .
NAC	Not A Constituent; used within an NP.	X	Unknown, uncertain, or unbracketable.
NP	Noun Phrase.		
NX	Used within certain complex NPs to mark the head.		



# Probabilistic Context-Free Grammar (PCFG)

- assign probabilities to rewrite rules:

NP → DT NN 0.5

NP → NNS 0.3

NP → NP PP 0.2

given a treebank, estimate these probabilities using MLE (“count and normalize”)

NN → man 0.01

NN → park 0.0004

NN → walk 0.002

NN → ....

# How well does a PCFG work?

- PCFG learned from the Penn Treebank with MLE gets about 73% F1 score
- state-of-the-art parsers are around 92%
- simple modifications can improve PCFGs:
  - smoothing
  - tree transformations (selective flattening)
  - parent annotation

# Parent Annotation

$VP \rightarrow V \ NP \ PP$



$VP^S \rightarrow V \ NP^{VP} \ PP^{VP}$

adds more information, but also fragments counts, making parameter estimates noisier (since we're just using MLE)

# How well does a PCFG work?

- PCFG learned from the Penn Treebank with MLE gets about 73% F1 score
- state-of-the-art parsers are around 92%
- simple modifications can improve PCFGs:
  - smoothing
  - tree transformations (selective flattening)
  - parent annotation
  - **lexicalization**

# Collins (1997)

## Three Generative, Lexicalised Models for Statistical Parsing

Michael Collins\*

Dept. of Computer and Information Science  
University of Pennsylvania  
Philadelphia, PA, 19104, U.S.A.  
mcollins@gradient.cis.upenn.edu

### Abstract

In this paper we first propose a new statistical parsing model, which is a generative model of lexicalised context-free grammar. We then extend the model to include a probabilistic treatment of both subcategorisation and wh-movement. Results on Wall Street Journal text show that the parser performs at 88.1/87.5% constituent precision/recall, an average improvement of 2.3% over (Collins 96).

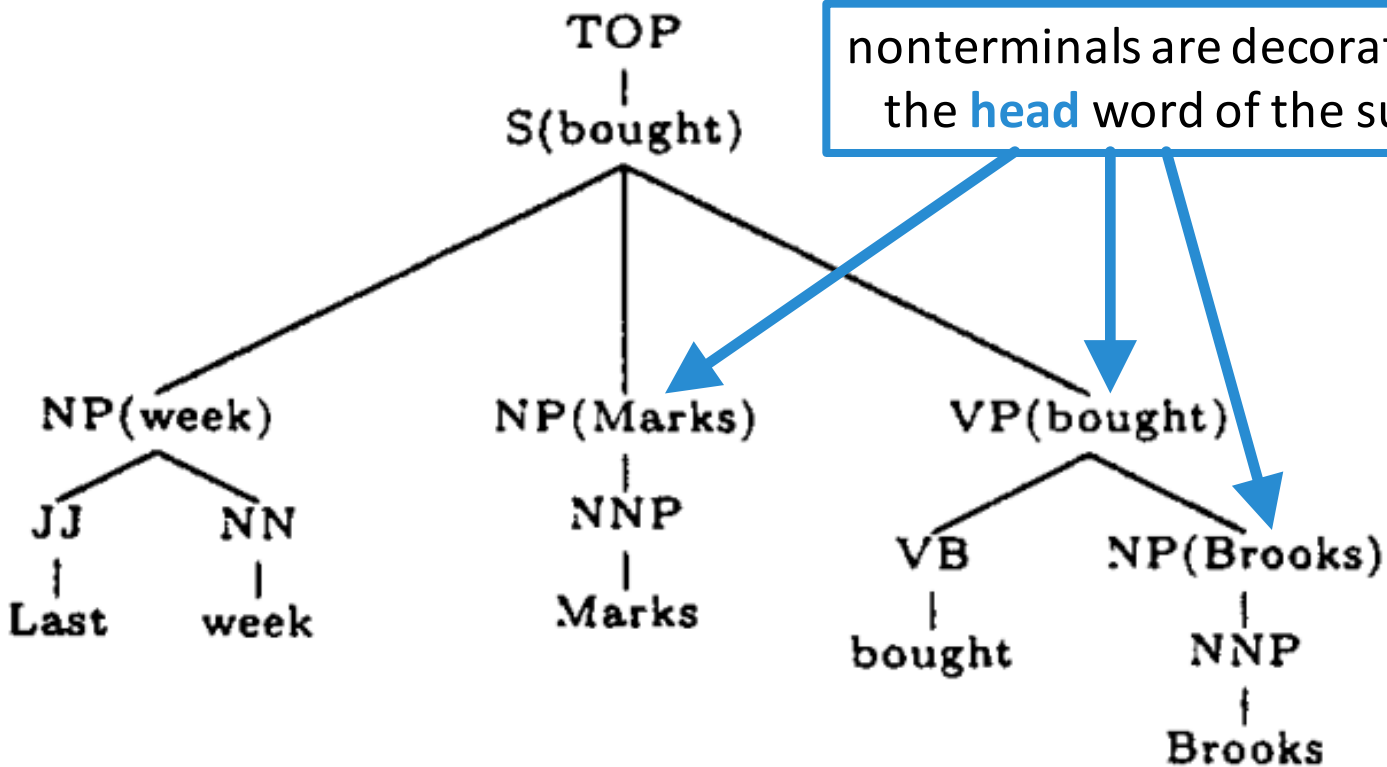
### 1 Introduction

Generative models of syntax have been central in linguistics since they were introduced in (Chom-

is derived from the analysis given in Generalized Phrase Structure Grammar (Gazdar et al. 95). The work makes two advances over previous models: First, Model 1 performs significantly better than (Collins 96), and Models 2 and 3 give further improvements — our final results are 88.1/87.5% constituent precision/recall, an average improvement of 2.3% over (Collins 96). Second, the parsers in (Collins 96) and (Magerman 95; Jelinek et al. 94) produce trees without information about wh-movement or subcategorisation. Most NLP applications will need this information to extract predicate-argument structure from parse trees.

In the remainder of this paper we describe the 3 models in section 2, discuss practical issues in section 3, give results in section 4, and give conclusions in section 5.

# Lexicalized PCFGs



nonterminals are decorated with the **head** word of the subtree

TOP	->	S(bought)		
S(bought)	->	NP(week)	NP(Marks)	VP(bought)
NP(week)	->	JJ>Last	NN(week)	
NP(Marks)	->	NNP(Marks)		
VP(bought)	->	VB(bought)	NP(Brooks)	
NP(Brooks)	->	NNP(Brooks)		

# Lexicalization

- this adds a lot more rules!
- many more parameters to estimate →  
smoothing becomes much more important
  - e.g., right-hand side of rule might be factored into several steps
- but it's worth it because head words are really useful for constituent parsing

# Results (Collins, 1997)

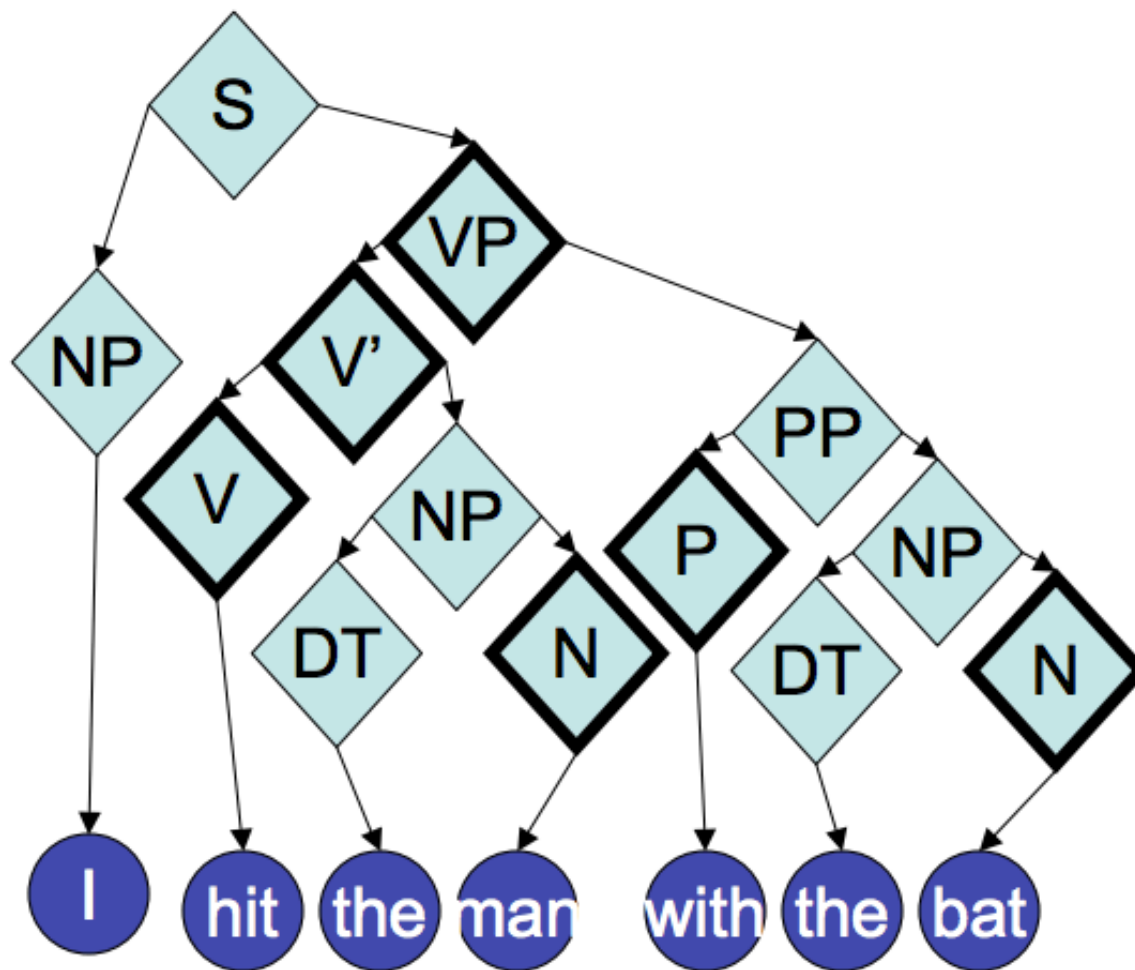
MODEL	$\leq 40$ Words (2245 sentences)				
	LR	LP	CBs	0 CBs	$\leq 2$ CBs
(Magerman 95)	84.6%	84.9%	1.26	56.6%	81.4%
(Collins 96)	85.8%	86.3%	1.14	59.9%	83.6%
Model 1	87.4%	88.1%	0.96	65.7%	86.3%
Model 2	88.1%	88.6%	0.91	66.5%	86.9%
Model 3	88.1%	88.6%	0.91	66.4%	86.9%



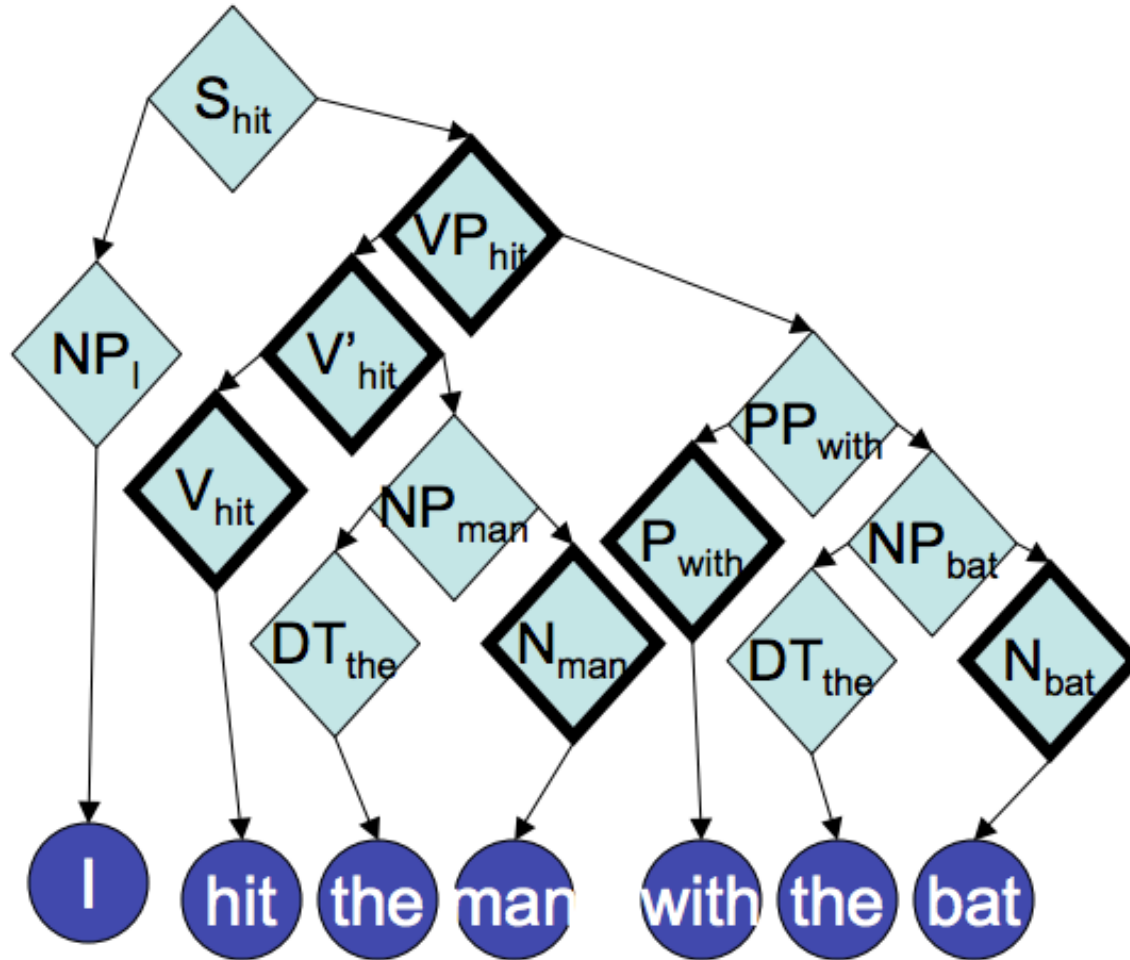
# Head Rules

- how are heads decided?
- most researchers use deterministic head rules (Magerman/Collins)
- for a PCFG rule  $A \rightarrow B_1 \dots B_N$ , these head rules say which of  $B_1 \dots B_N$  is the head of the rule
- examples:
  - $S \rightarrow NP \underline{VP}$
  - $VP \rightarrow \underline{VBD} NP PP$
  - $NP \rightarrow DT JJ \underline{NN}$

# Head Annotation

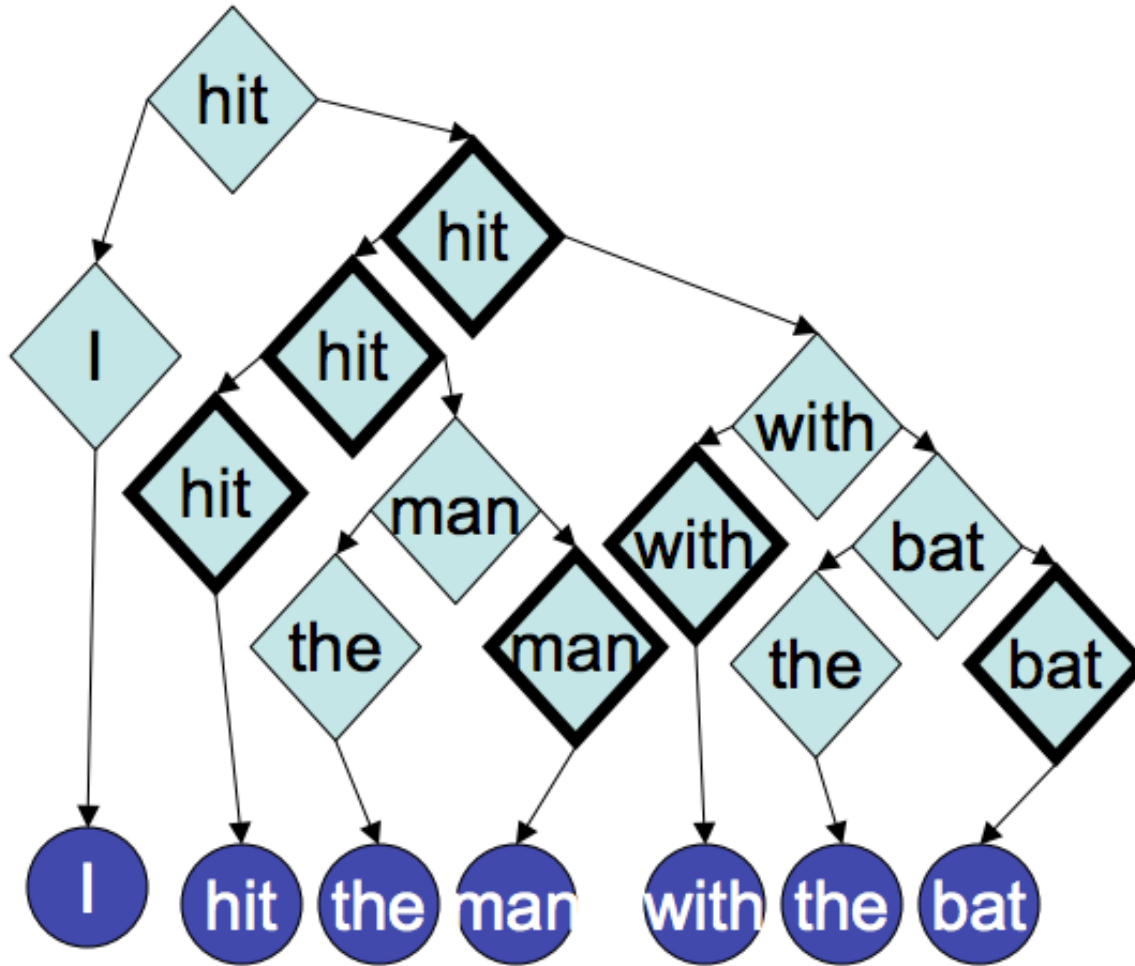


# Lexical Head Annotation



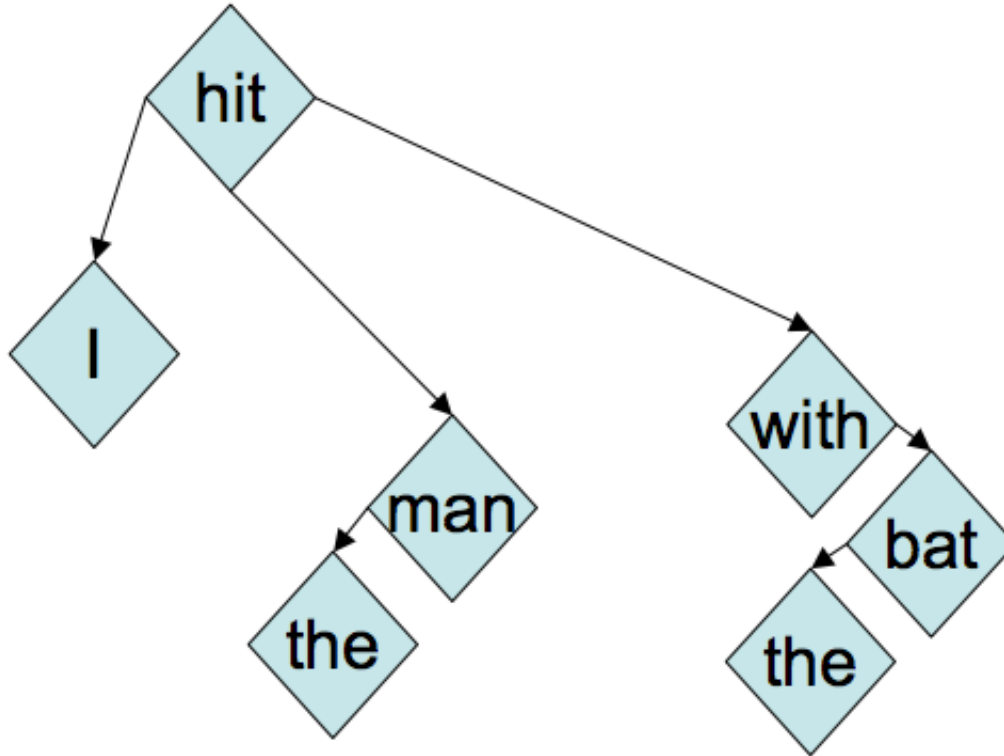
# Lexical Head Annotation → Dependencies

remove  
nonlexical  
parts:

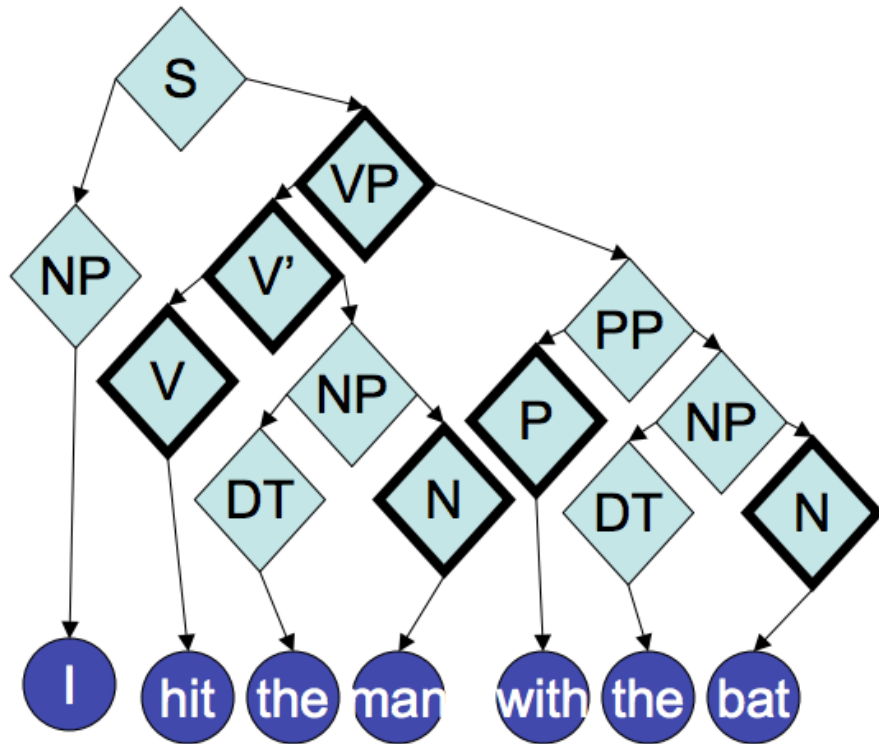


# Dependencies

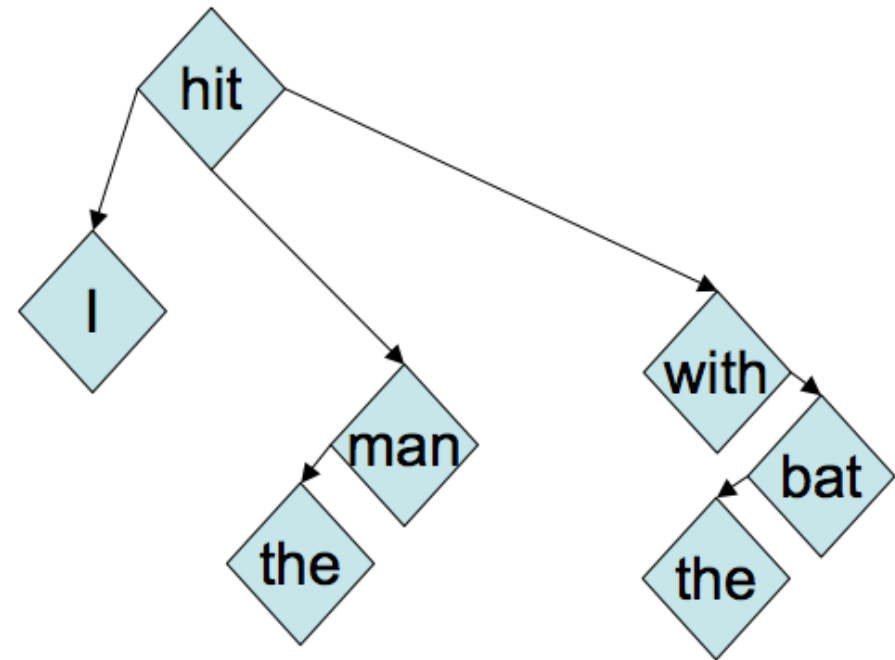
merge  
redundant  
nodes:



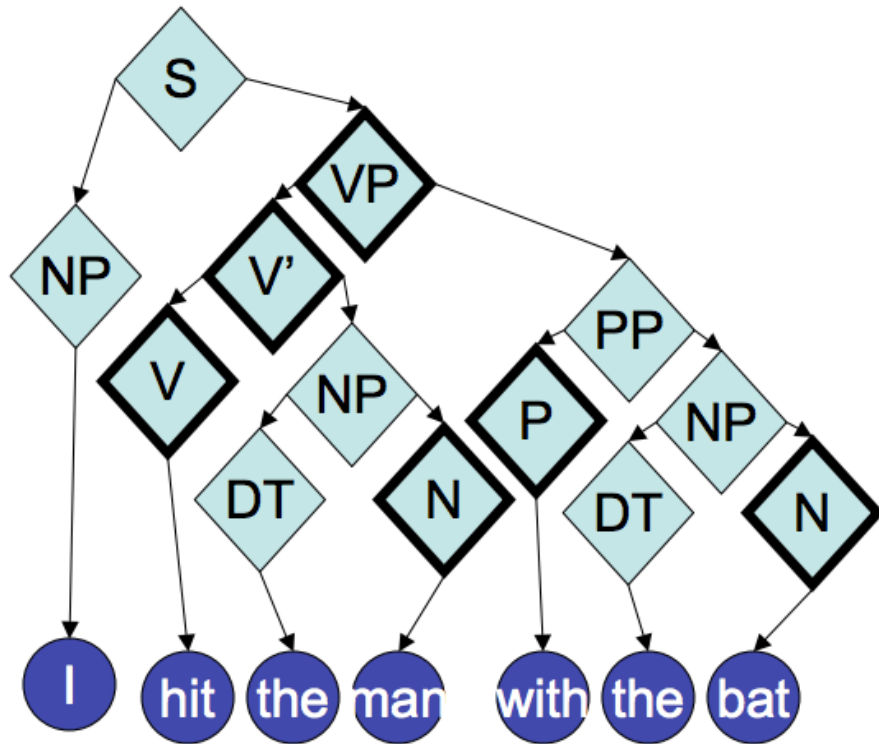
## constituent parse:



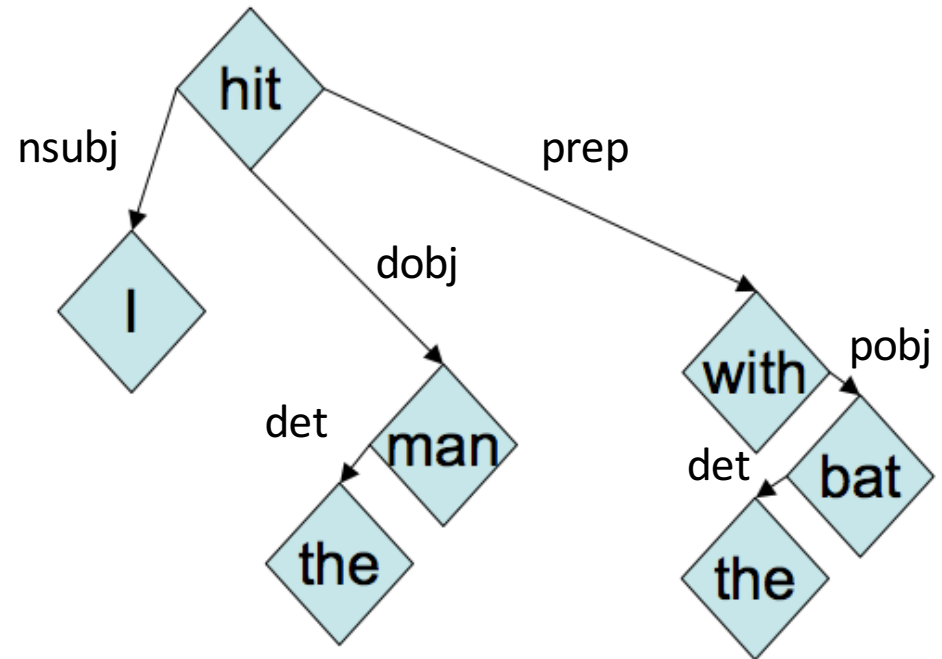
## dependency parse:



## constituent parse:

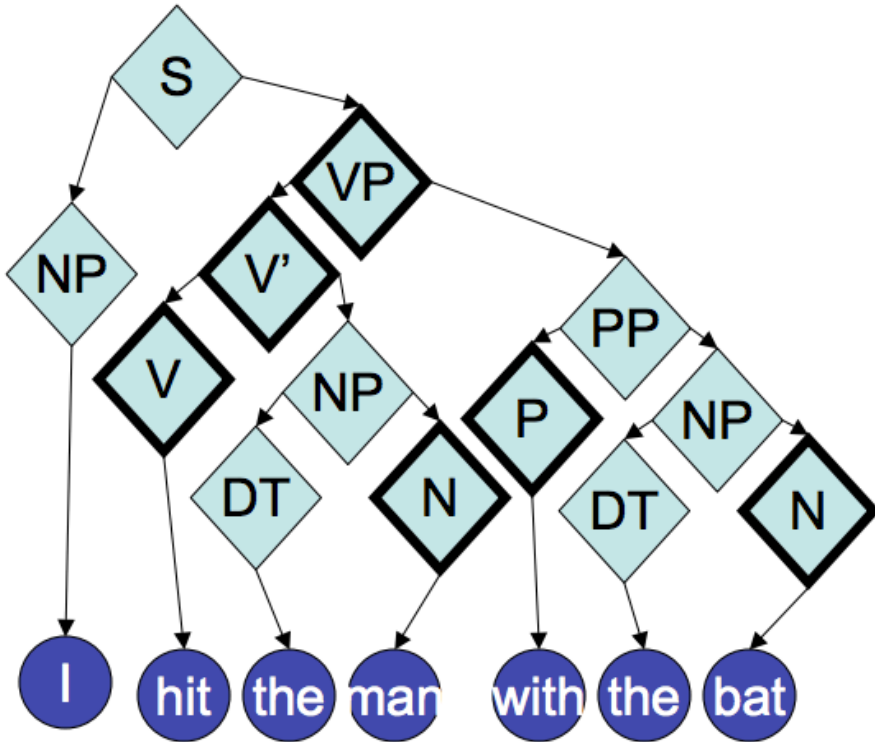


## labeled dependency parse:

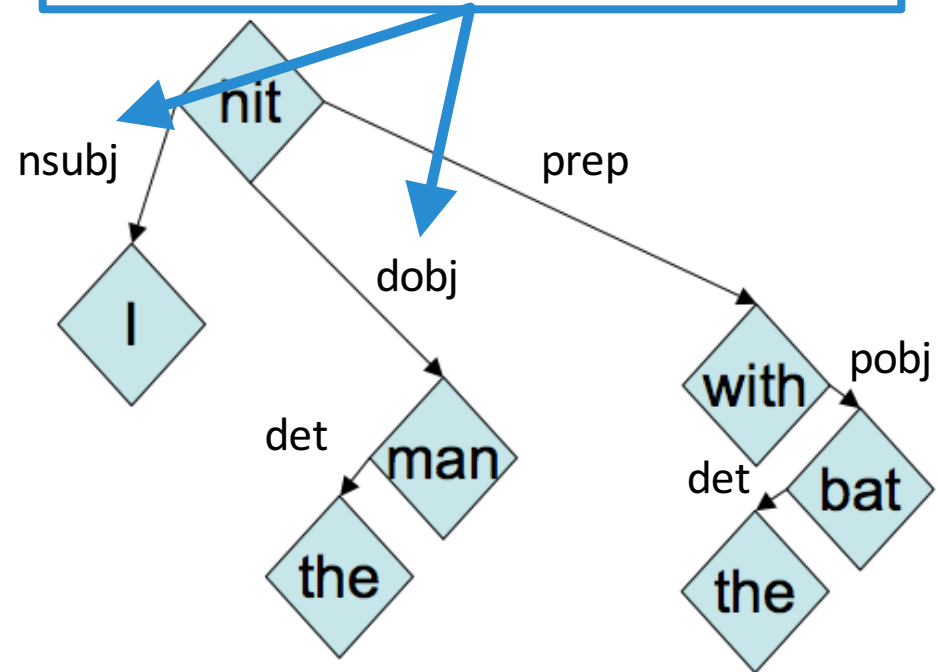


nsubj = "nominal subject"  
dobj = "direct object"  
prep = "preposition modifier"  
pobj = "object of preposition"  
det = "determiner"

# constituent parse:



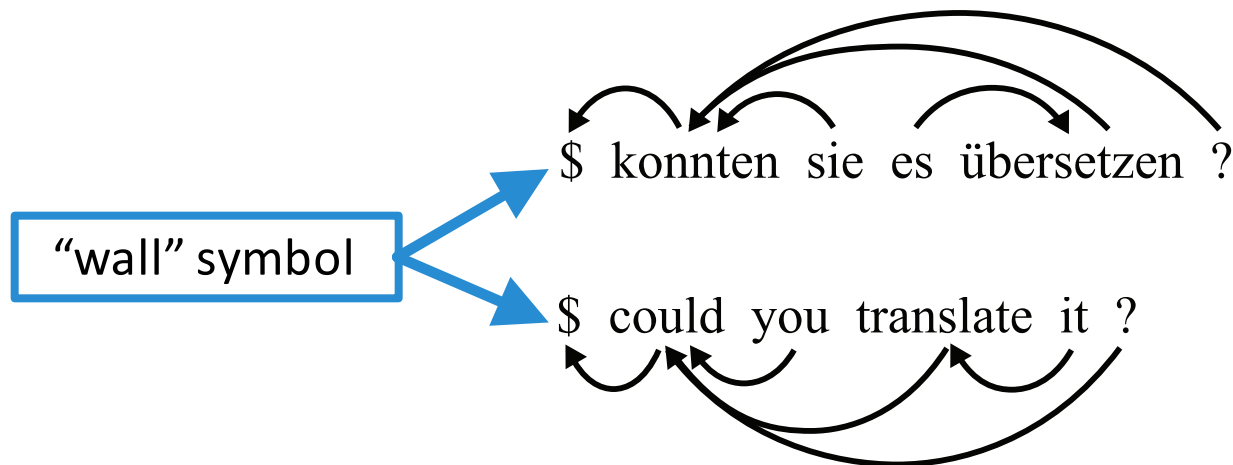
captures some semantic relationships



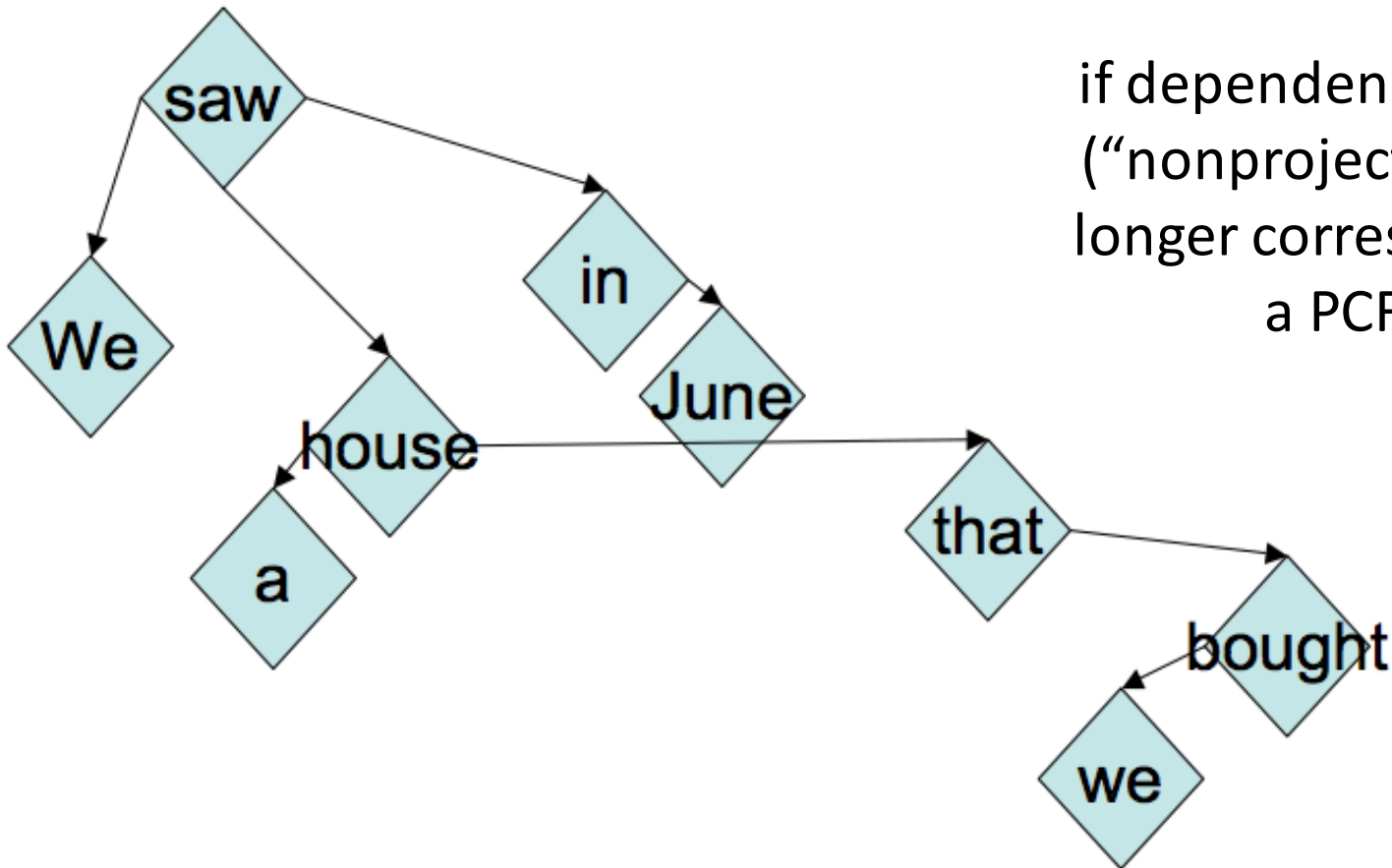
nsubj = "nominal subject"  
dobj = "direct object"  
prep = "preposition modifier"  
pobj = "object of preposition"  
det = "determiner"



- how (unlabeled) dependency trees are typically drawn:
  - root of tree is represented by \$ (“wall symbol”)
  - arrows drawn entirely above (or below) sentence
  - arrows are directed from child to parent (or from parent to child); you will see both in practice— don’t get confused!



# Crossing Dependencies



if dependencies cross  
("nonprojective"), no  
longer corresponds to  
a PCFG

## Projective vs. Nonprojective Dependency Parsing

- English dependency treebanks are mostly projective
  - but when focusing more on semantic relationships, often becomes more nonprojective
- some (relatively) free word order languages, like Czech, are fairly nonprojective
- nonprojective parsing can be formulated as a minimum spanning tree problem
- projective parsing cannot

# Dependency Parsing

- several widely-used algorithms
- different guarantees but similar performance in practice
- graph-based:
  - dynamic programming (Eisner, 1997)
  - minimum spanning tree (McDonald et al., 2005)
- transition-based:
  - shift-reduce (Nivre, *inter alia*)

# Dependency Parsers

- Stanford parser
- TurboParser
- Joakim Nivre's MALT parser
- Ryan McDonald's MST parser
- and many others for many non-English languages

# Complexity Comparison

- constituent parsing:  $O(Gn^3)$ 
  - parsing complexity depends on grammar structure (“grammar constant”  $G$ )
  - since it has lots of nonterminal-only rules at the top of the tree, there are many rule probabilities to estimate
- dependency parsing:  $O(n^3)$ 
  - operates directly on words, so parsing complexity has no grammar constant
  - features designed on possible dependencies (pairs of words) and larger structures
  - transition-based parsing algorithms are  $O(n)$ , though not optimal; also, non-projective parsing is faster

# Applications of Dependency Parsing

- widely used for NLP tasks because:
  - faster than constituent parsing
  - captures more semantic information
- text classification (features on dependencies)
- syntax-based machine translation
- relation extraction
  - e.g., extract relation between Sam Smith and AI Tech:  
*Sam Smith was named new CEO of AI Tech.*
  - use dependency path between *Sam Smith* and *AI Tech*:
    - Smith → named, named ← CEO, CEO ← of, of ← AI Tech

# Summary: two types of grammars

- phrase structure / constituent grammars
  - inspired mostly by Chomsky and others
  - only appropriate for certain languages (e.g., English)
- dependency grammars
  - closer to a semantic representation; some have made this more explicit
  - problematic for certain syntactic structures (e.g., conjunctions, nesting of noun phrases, etc.)
- both are widely used in NLP
- you can find constituent parsers and dependency parsers for several languages online



# Review

# Modeling, Inference, Learning

**modeling:** define score function



$$\text{classify}(x, \theta) = \underset{y}{\operatorname{argmax}} \text{ score}(x, y, \theta)$$

- **Modeling:** How do we assign a score to an  $(x, y)$  pair using parameters  $\theta$ ?

# Modeling, Inference, Learning

**inference:** solve  $\operatorname{argmax}$

**modeling:** define score function

$$\operatorname{classify}(x, \theta) = \operatorname{argmax}_y \operatorname{score}(x, y, \theta)$$

- **Inference:** How do we efficiently search over the space of all labels?

# Modeling, Inference, Learning

**inference:** solve  $\operatorname{argmax}$

**modeling:** define score function

$$\operatorname{classify}(x, \theta) = \operatorname{argmax}_y \operatorname{score}(x, y, \theta)$$

**learning:** choose  $\theta$

- **Learning:** How do we choose  $\theta$ ?

# Applications

# Applications of our Classification Framework

text classification:

$$\text{classify}_{\text{text}}^{\text{linear}}(\mathbf{x}, \boldsymbol{\theta}) = \operatorname{argmax}_{y \in \mathcal{L}} \sum_i \theta_i f_i(\mathbf{x}, y)$$

$$\mathcal{L} = \{\text{objective, subjective}\}$$

$x$	$y$
the hulk is an anger fueled monster with incredible strength and resistance to damage .	objective
in trying to be daring and original , it comes off as only occasionally satirical and never fresh .	subjective

# Applications of our Classification Framework

word sense classifier for *bass*:

$$\text{classify}_{\text{bassWSD}}^{\text{linear}}(\mathbf{x}, \boldsymbol{\theta}) = \operatorname{argmax}_{y \in \mathcal{L}_{\text{bass}}} \sum_i \theta_i f_i(\mathbf{x}, y)$$

$$\mathcal{L}_{\text{bass}} = \{\text{bass}_1, \text{bass}_2, \dots, \text{bass}_8\}$$

x	y
he's a bass in the choir .	<i>bass</i> <sub>3</sub>
our bass is line-caught from the Atlantic .	<i>bass</i> <sub>4</sub>

- **S: (n) bass** (the lowest part of the musical range)
- **S: (n) bass, bass part** (the lowest part in polyphony)
- **S: (n) bass, basso** (an adult male singer with a low voice)
- **S: (n) sea bass, bass** (the lean flesh of a saltwater fish of the family Serranidae)
- **S: (n) freshwater bass, bass** (any of various fish with lean flesh (especially of the genus *Micropterus*))
- **S: (n) bass, bass voice, basso** (the lowest adult male voice)
- **S: (n) bass** (the member with the lowest range in an ensemble of instruments)
- **S: (n) bass** (nontechnical name for any of numerous species of freshwater spiny-finned fishes)

# Applications of our Classification Framework

skip-gram model as a classifier:

$$\text{classify}_{\text{skipgram}}(x, \theta) = \operatorname{argmax}_{y \in \mathcal{L}} \theta^{(\text{in}, x)} \cdot \theta^{(\text{out}, y)}$$

$\mathcal{L} = V$  (the entire vocabulary)

$x$	$y$
agriculture	<s>
agriculture	is
agriculture	the

corpus (English Wikipedia):

*agriculture is the traditional mainstay of the cambodian economy .*

*but benares has been destroyed by an earthquake .*

...



# Simplest kind of structured prediction: Sequence Labeling

## Part-of-Speech Tagging

determiner	verb (past)	prep.	proper noun	proper noun	poss.	adj.	noun
Some	questioned	if	Tim	Cook	's	first	product
modal	verb	det.	adjective	noun	prep.	proper noun	punc.
would	be	a	breakaway	hit	for	Apple	.

# Formulating segmentation tasks as sequence labeling via B-I-O labeling:

## Named Entity Recognition

O O O B-PERSON I-PERSON O O O  
Some questioned if Tim Cook 's first product

O O O O O O B-ORGANIZATION O  
would be a breakaway hit for Apple .

**B = “begin”**

**I = “inside”**

**O = “outside”**

# Applications of our Classifier Framework so far

task	input ( $x$ )	output ( $y$ )	output space ( $\mathcal{L}$ )	size of $\mathcal{L}$
text classification	a sentence	gold standard label for $x$	pre-defined, small label set (e.g., {positive, negative})	2-10
word sense disambiguation	instance of a particular word (e.g., <i>bass</i> ) with its context	gold standard word sense of $x$	pre-defined sense inventory from WordNet for <i>bass</i>	2-30
learning skip-gram word embeddings	instance of a word in a corpus	a word in the context of $x$ in a corpus	vocabulary	$ V $
part-of-speech tagging	a sentence	gold standard part-of-speech tags for $x$	all possible part-of-speech tag sequences with same length as $x$	$ P ^{ x }$

# Applications of Classifier Framework (continued)

task	input ( $x$ )	output ( $y$ )	output space ( $\mathcal{L}$ )	size of $\mathcal{L}$
named entity recognition	a sentence	gold standard named entity labels for $x$ (BIO tags)	all possible BIO label sequences with same length as $x$	$ P ^{ x }$
constituent parsing	a sentence	gold standard constituent parse (labeled bracketing) of $x$	all possible labeled bracketings of $x$	exponential in length of $x$ (Catalan number)
dependency parsing	a sentence	gold standard dependency parse (labeled directed spanning tree) of $x$	all possible labeled directed spanning trees of $x$	exponential in length of $x$

- each application draws from particular linguistic concepts and must address different kinds of linguistic ambiguity/variability:
  - **word sense**: sense granularity, relationships among senses, word sense ambiguity
  - **word vectors**: distributional properties, sense ambiguity, different kinds of similarity
  - **part-of-speech**: tag granularity, tag ambiguity
  - **parsing**: constituent/dependency relationships, attachment & coordination ambiguities

# Modeling

# model families

- linear models
  - lots of freedom in defining features, though feature engineering required for best performance
  - learning uses optimization of a loss function
  - one can (try to) interpret learned feature weights
- stochastic/generative models
  - linear models with simple “features” (counts of events)
  - learning is easy: count & normalize (but smoothing needed)
  - easy to generate samples
- neural networks
  - can usually get away with less feature engineering
  - learning uses optimization of a loss function
  - hard to interpret (though we try!), but often works best

# special case of linear models: stochastic/generative models

model	tasks	context expansion
<i>n</i> -gram language models	language modeling (for MT, ASR, etc.)	increase <i>n</i>
hidden Markov models	part-of-speech tagging, named entity recognition, word clustering	increase order of HMM (e.g., bigram HMM → trigram HMM)
probabilistic context-free grammars	constituent parsing	increase size of rules, e.g., flattening, parent annotation, etc.

- all use MLE + smoothing (though probably different kinds of smoothing)
- all assign probability to sentences (some assign probability jointly to pairs of <sentence, something else>)
- all have the same trade-off of increasing “context” (feature size) and needing more data / better smoothing



# Feature Engineering for Text Classification

$$\text{score}(\mathbf{x}, y, \boldsymbol{\theta}) = \sum_i \theta_i f_i(\mathbf{x}, y)$$

- Two features:

$$f_1(\mathbf{x}, y) = \mathbb{I}[y = \text{positive}] \wedge \mathbb{I}[\mathbf{x} \text{ contains } \textit{great}]$$

$$f_2(\mathbf{x}, y) = \mathbb{I}[y = \text{negative}] \wedge \mathbb{I}[\mathbf{x} \text{ contains } \textit{great}]$$

where  $\mathbb{I}[S] = 1$  if  $S$  is true, 0 otherwise

# Higher-Order Binary Feature Templates

unigram binary template:

$$f^{u,b}(\mathbf{x}, y) = \mathbb{I}[y = \text{label}] \wedge \mathbb{I}[\mathbf{x} \text{ contains } \textit{word}]$$

bigram binary template:

$$f^{b,b}(\mathbf{x}, y) = \mathbb{I}[y = \text{label}] \wedge \mathbb{I}[\mathbf{x} \text{ contains } \textit{“word1 word2”}]$$

trigram binary features

...

# Unigram **Count** Features

- a “count” feature returns the count of a particular word in the text
- unigram count feature template:

$$f^{\text{u,c}}(\mathbf{x}, y) = \begin{cases} \sum_{i=1}^{|\mathbf{x}|} \mathbb{I}[x_i = \text{word}], & \text{if } \mathbb{I}[y = \text{label}] \\ 0, & \text{otherwise} \end{cases}$$

# Feature Count Cutoffs

- problem: some features are extremely rare
- solution: only keep features that appear at least  $k$  times in the training data

# 2-transformation (1-layer) network

$$\mathbf{z}^{(1)} = g \left( W^{(0)} \mathbf{x} + \mathbf{b}^{(0)} \right)$$

$$\mathbf{s} = g \left( W^{(1)} \mathbf{z}^{(1)} + \mathbf{b}^{(1)} \right)$$



vector of label scores

- we'll call this a “2-transformation” neural network, or a “1-layer” neural network
- input vector is  $\mathbf{x}$
- score vector is  $\mathbf{s}$
- one hidden vector  $\mathbf{z}^{(1)}$  (“hidden layer”)

# 1-layer neural network for sentiment classification

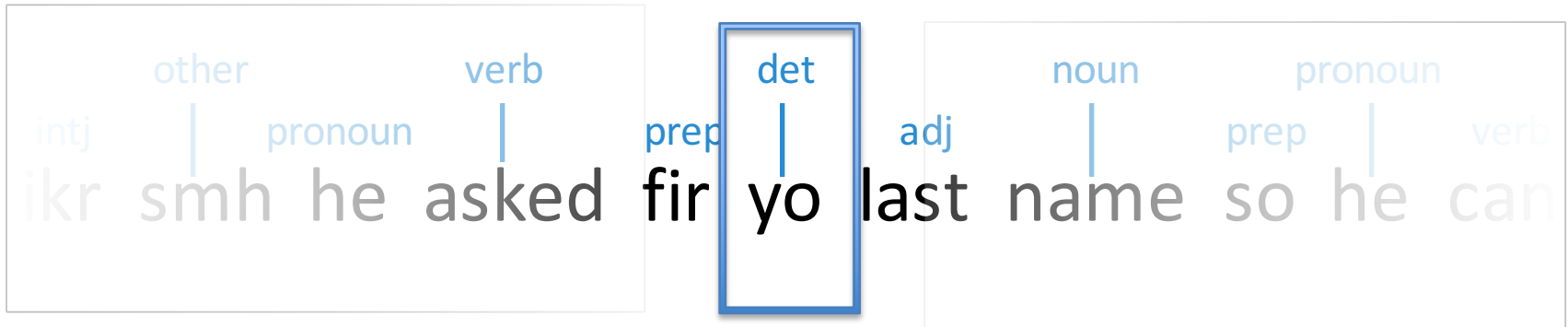
$$\mathbf{z}^{(1)} = g \left( W^{(0)} \mathbf{x} + \mathbf{b}^{(0)} \right)$$

$$\mathbf{s} = g \left( W^{(1)} \mathbf{z}^{(1)} + \mathbf{b}^{(1)} \right)$$



$$\mathbf{s} = \begin{bmatrix} \text{score}(\mathbf{x}, \text{positive}, \boldsymbol{\theta}) \\ \text{score}(\mathbf{x}, \text{negative}, \boldsymbol{\theta}) \end{bmatrix}$$

# Neural Networks for Twitter Part-of-Speech Tagging



- let's use the center word + two words to the right:

$$\mathbf{x} = [0.4 \quad \dots \quad 0.9 \quad 0.2 \quad \dots \quad 0.7 \quad 0.3 \quad \dots \quad 0.6]^\top$$

vector for *yo*      vector for *last*      vector for *name*

- if *name* is to the right of *yo*, then *yo* is probably a form of *your*
- but our  $\mathbf{x}$  above uses separate dimensions for each position!
  - i.e., *name* is two words to the right
  - what if *name* is one word to the right?

# Convolution

$\mathbf{c}$  = “feature map”, has an entry for each word position in context window / sentence

$$\mathbf{x} = [0.4 \ \dots \ 0.9 \ 0.2 \ \dots \ 0.7 \ 0.3 \ \dots \ 0.6]^\top$$

vector for *yo*      vector for *last*      vector for *name*

$$c_1 = \mathbf{w} \cdot \mathbf{x}_{1:d}$$

$$c_2 = \mathbf{w} \cdot \mathbf{x}_{d+1:2d}$$

$$c_3 = \mathbf{w} \cdot \mathbf{x}_{2d+1:3d}$$



# Pooling

$\mathbf{c}$  = “feature map”, has an entry for each word position in context window / sentence

how do we convert this into a fixed-length vector?

use **pooling**:

max-pooling: returns maximum value in  $\mathbf{c}$

average pooling: returns average of values in  $\mathbf{c}$

vector for *yo*    vector for *last*    vector for *name*

$$c_1 = \mathbf{w} \cdot \mathbf{x}_{1:d}$$

$$c_2 = \mathbf{w} \cdot \mathbf{x}_{d+1:2d}$$

$$c_3 = \mathbf{w} \cdot \mathbf{x}_{2d+1:3d}$$

# Pooling

$c$  = “feature map”, has an entry for each word position in context window / sentence

how do we convert this into a fixed-length vector?

use **pooling**:

max-pooling: returns maximum value in  $c$

average pooling: returns average of values in  $c$

vector for *yo*    vector for *last*    vector for *name*

$$c_1 = w \cdot x_{1:d}$$

then, this single filter  $w$  produces a single feature value (the output of some kind of pooling).

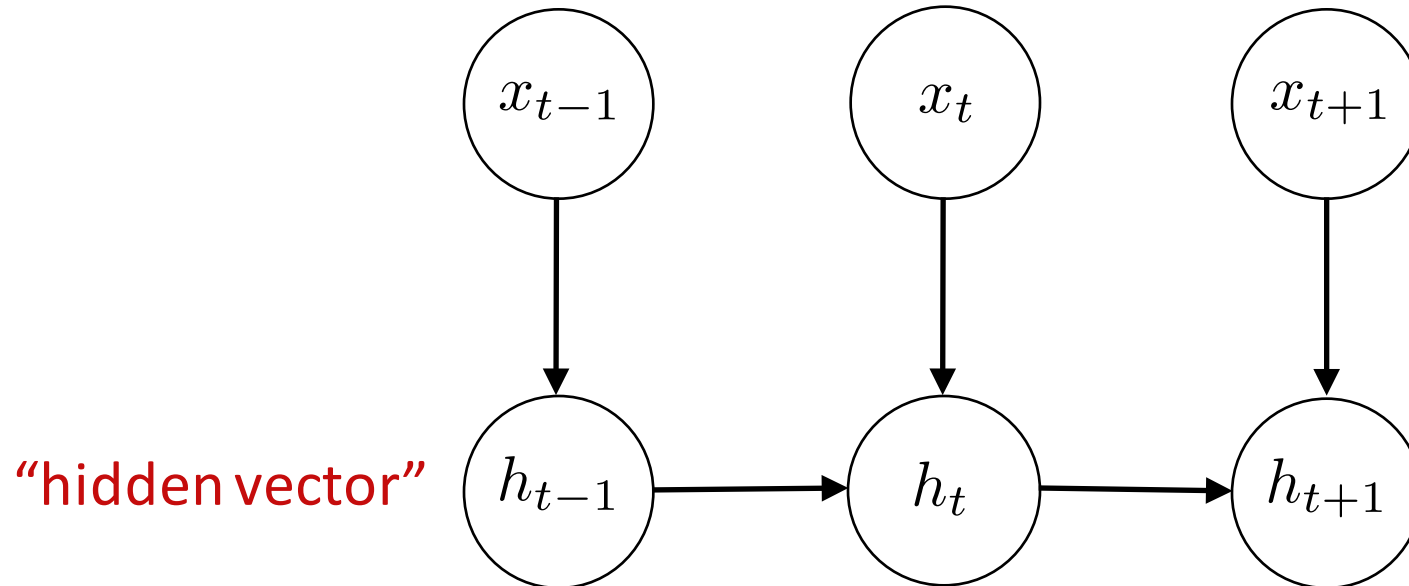
in practice, we use many filters of many different lengths (e.g.,  $n$ -grams rather than words).

# Convolutional Neural Networks

- convolutional neural networks (**convnets** or **CNNs**) use filters that are “convolved with” (matched against all positions of) the input
- think of convolution as “perform the same operation everywhere on the input in some systematic order”
- “convolutional layer” = set of filters that are convolved with the input vector (whether  $\mathbf{x}$  or hidden vector)
- could be followed by more convolutional layers, or by a type of pooling
- often used in NLP to convert a sentence into a feature vector

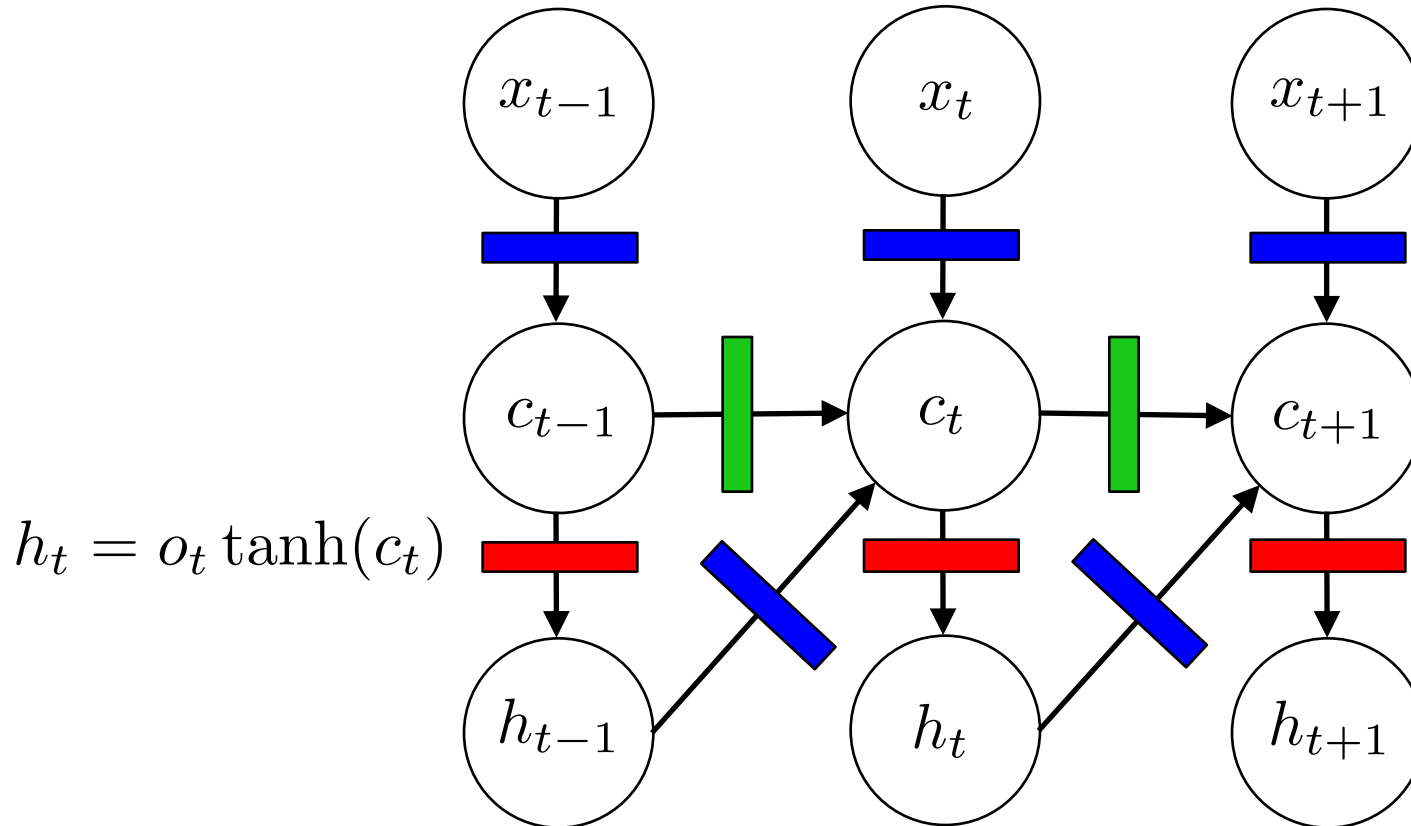
# Recurrent Neural Networks

$$h_t = \tanh \left( W^{(xh)} x_t + W^{(hh)} h_{t-1} + b^{(h)} \right)$$



# Long Short-Term Memory (LSTM) Recurrent Neural Networks

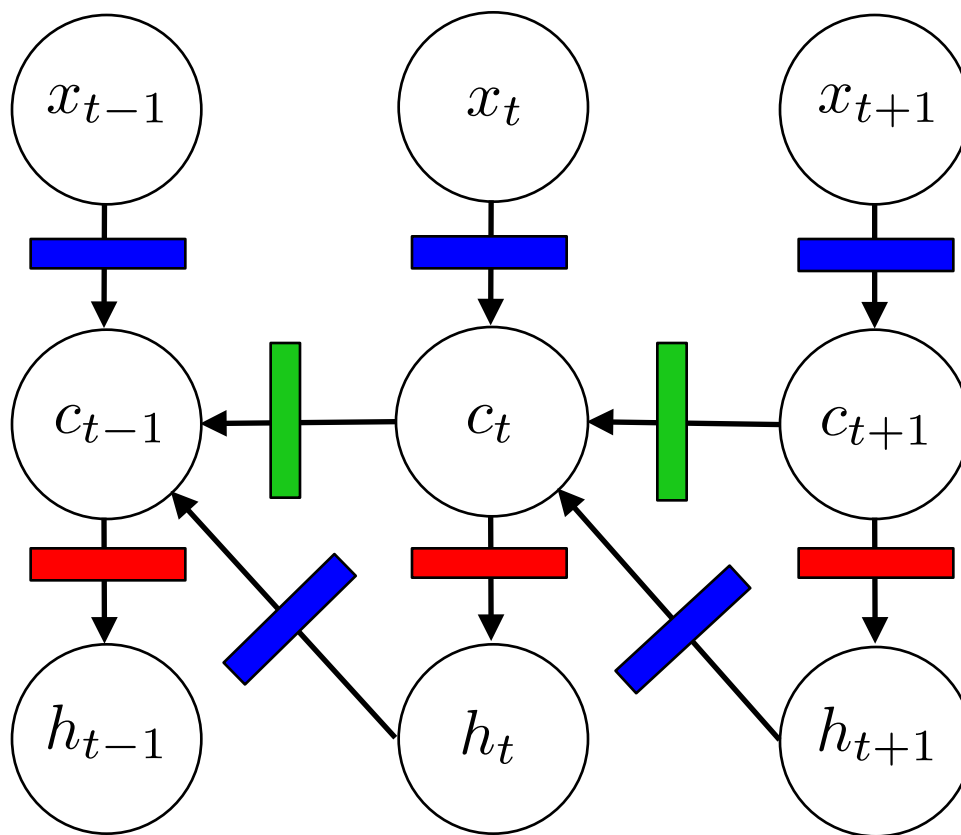
$$c_t = f_t c_{t-1} + i_t \tanh \left( W^{(xc)} x_t + W^{(hc)} h_{t-1} + b^{(c)} \right)$$



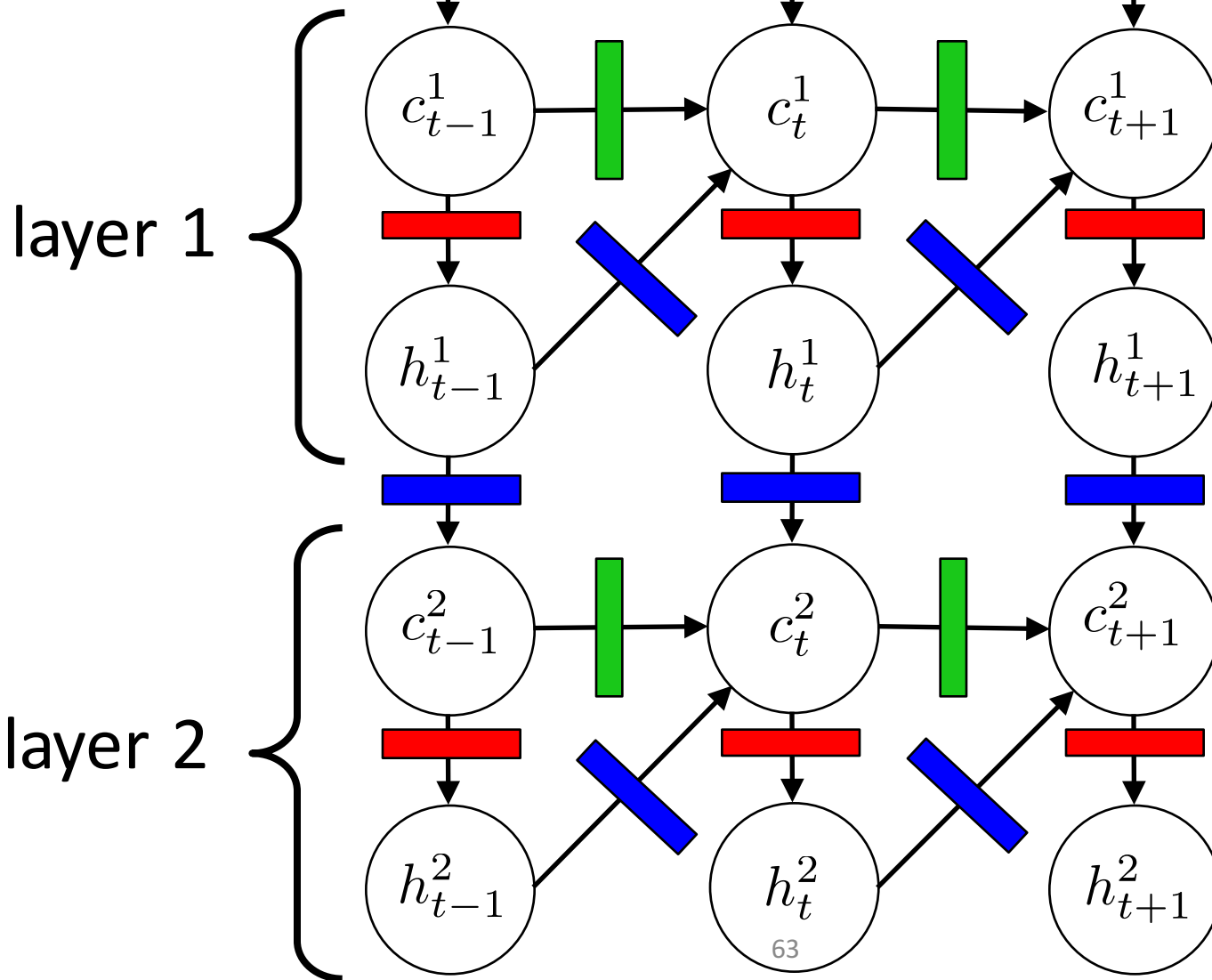
# Backward & Bidirectional LSTMs

bidirectional:

if shallow, just use forward and backward LSTMs in parallel, concatenate final two hidden vectors, feed to softmax

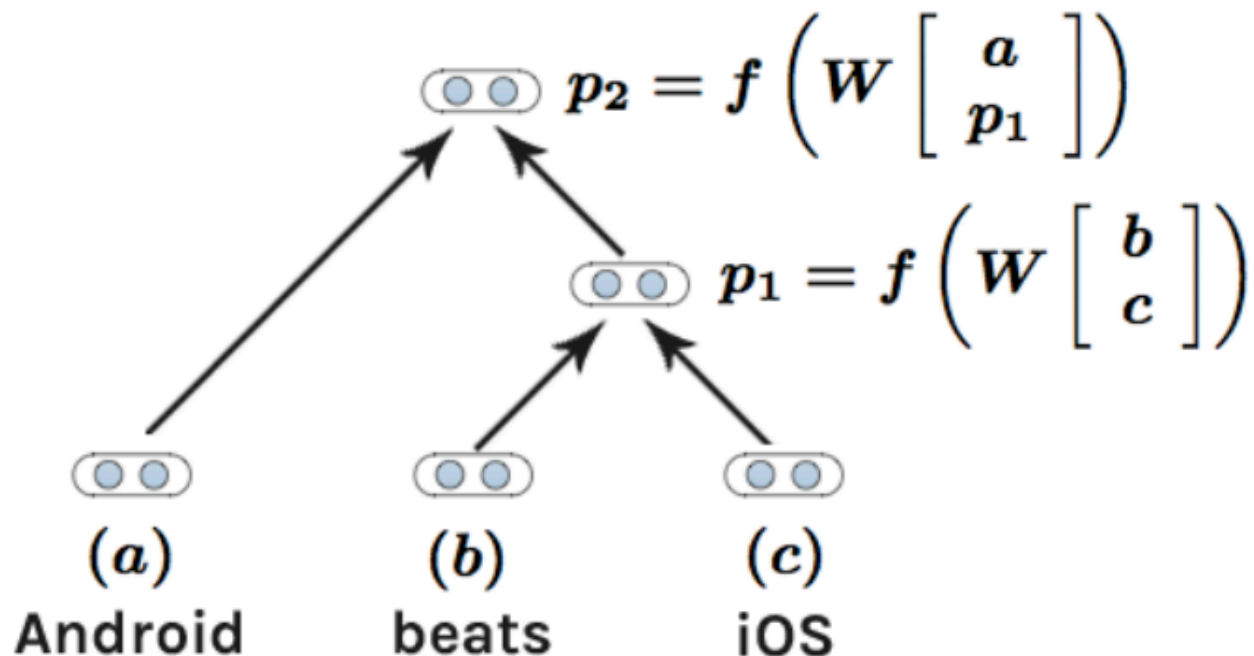


# Deep LSTM (2-layer)



# Recursive Neural Networks for NLP

- first, run a constituent parser on the sentence
- convert the constituent tree to a binary tree (each rewrite has exactly two children)
- construct vector for sentence recursively at each rewrite (“split point”):





# Learning

# Cost Functions

- **cost function**: scores output against a gold standard

$$\text{cost} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}_{\geq 0}$$

- should reflect the evaluation metric for your task
- usual conventions:  $\text{cost}(y, y) = 0$        $\text{cost}(y, y') = \text{cost}(y', y)$
- for classification, what cost should we use?

$$\text{cost}(y, y') = \mathbb{I}[y \neq y']$$

# Empirical Risk Minimization

(Vapnik et al.)

- replace expectation with sum over examples:

$$\hat{\theta} = \operatorname{argmin}_{\theta} \mathbb{E}_{P(x,y)} [\operatorname{cost}(y, \operatorname{classify}(x, \theta))]$$



$$\hat{\theta} = \operatorname{argmin}_{\theta} \sum_{i=1}^{|\mathcal{T}|} \operatorname{cost}(y^{(i)}, \operatorname{classify}(x^{(i)}, \theta))$$

# Empirical Risk Minimization

(Vapnik et al.)

- replace expectation with sum over examples:

$$\hat{\theta} = \operatorname{argmin}_{\theta} \mathbb{E}_{P(x,y)} [\operatorname{cost}(y, \operatorname{classify}(x, \theta))]$$



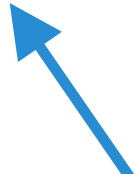
$$\hat{\theta} = \operatorname{argmin}_{\theta} \sum_{i=1}^{|\mathcal{T}|} \operatorname{cost}(y^{(i)}, \operatorname{classify}(x^{(i)}, \theta))$$

problem: NP-hard even for binary classification with linear models

# Empirical Risk Minimization with Surrogate Loss Functions

- given training data:  $\mathcal{T} = \{\langle \mathbf{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{|\mathcal{T}|}$   
where each  $y^{(i)} \in \mathcal{L}$  is a label
- we want to solve the following:

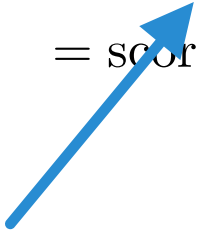
$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$



many possible loss  
functions to consider  
optimizing

# Loss Functions

name	loss	where used
cost (“0-1”)	$\text{cost}(y, \text{classify}(\mathbf{x}, \boldsymbol{\theta}))$	intractable, but underlies “direct error minimization”
perceptron	$-\text{score}(\mathbf{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \boldsymbol{\theta})$	perceptron algorithm (Rosenblatt, 1958)
hinge	$-\text{score}(\mathbf{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} (\text{score}(\mathbf{x}, y', \boldsymbol{\theta}) + \text{cost}(y, y'))$	support vector machines, other large-margin algorithms
log	$-\log p_{\boldsymbol{\theta}}(y   \mathbf{x})$ $= \text{score}(\mathbf{x}, y, \boldsymbol{\theta}) + \log \sum_{y' \in \mathcal{L}} \exp\{\text{score}(\mathbf{x}, y', \boldsymbol{\theta})\}$	logistic regression, conditional random fields, maximum entropy models



$$p_{\boldsymbol{\theta}}(y | \mathbf{x}) = \frac{\exp\{\text{score}(\mathbf{x}, y, \boldsymbol{\theta})\}}{\sum_{y' \in \mathcal{L}} \exp\{\text{score}(\mathbf{x}, y', \boldsymbol{\theta})\}}$$

# (Sub)gradients of Losses for Linear Models

name	entry $j$ of (sub)gradient of loss for linear model
cost ("0-1")	not subdifferentiable in general
perceptron	$-f_j(\mathbf{x}, y) + f_j(\mathbf{x}, \hat{y})$ , where $\hat{y} = \text{classify}(\mathbf{x}, \boldsymbol{\theta})$
hinge	$-f_j(\mathbf{x}, y) + f_j(\mathbf{x}, \tilde{y})$ , where $\tilde{y} = \text{costClassify}(\mathbf{x}, y, \boldsymbol{\theta})$
log	

$$\text{classify}(\mathbf{x}, \boldsymbol{\theta}) = \operatorname{argmax}_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \boldsymbol{\theta})$$

$$\text{costClassify}(\mathbf{x}, y, \boldsymbol{\theta}) = \operatorname{argmax}_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \boldsymbol{\theta}) + \text{cost}(y, y')$$

# (Sub)gradients of Losses for Linear Models

name	entry $j$ of (sub)gradient of loss for linear model
cost ("0-1")	not subdifferentiable in general
perceptron	$-f_j(\mathbf{x}, y) + f_j(\mathbf{x}, \hat{y})$ , where $\hat{y} = \text{classify}(\mathbf{x}, \boldsymbol{\theta})$
hinge	$-f_j(\mathbf{x}, y) + f_j(\mathbf{x}, \tilde{y})$ , where $\tilde{y} = \text{costClassify}(\mathbf{x}, y, \boldsymbol{\theta})$
log	$-f_j(\mathbf{x}, y) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\cdot \mathbf{x})}[f_j(\mathbf{x}, \cdot)]$

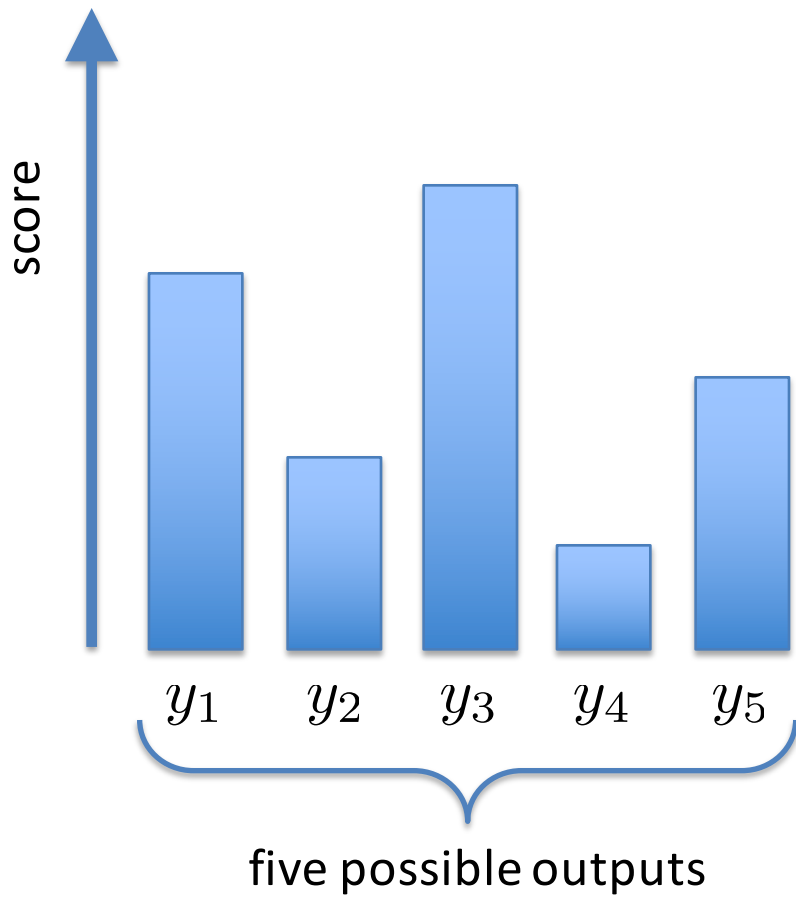
expectation of feature value with respect to distribution over  $y$  (where distribution is defined by  $\theta$ )

alternative notation:

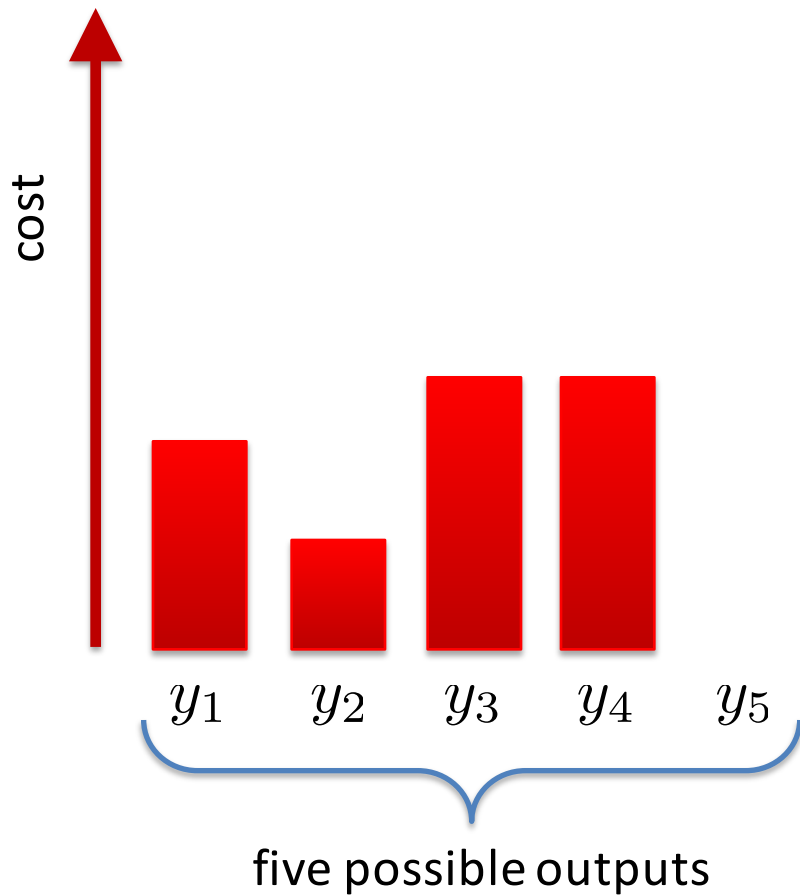
$$-f_j(\mathbf{x}, y) + \mathbb{E}_{y' \sim p_{\boldsymbol{\theta}}(Y|\mathbf{x})}[f_j(\mathbf{x}, y')]$$



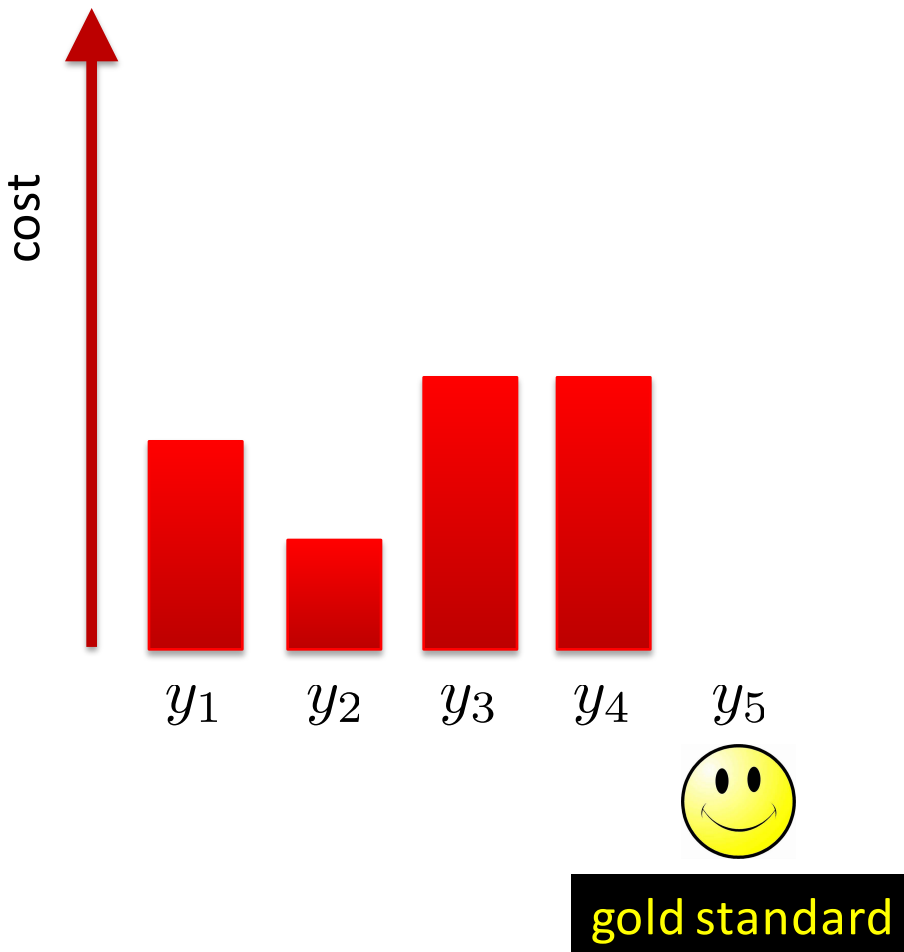
# Visualization



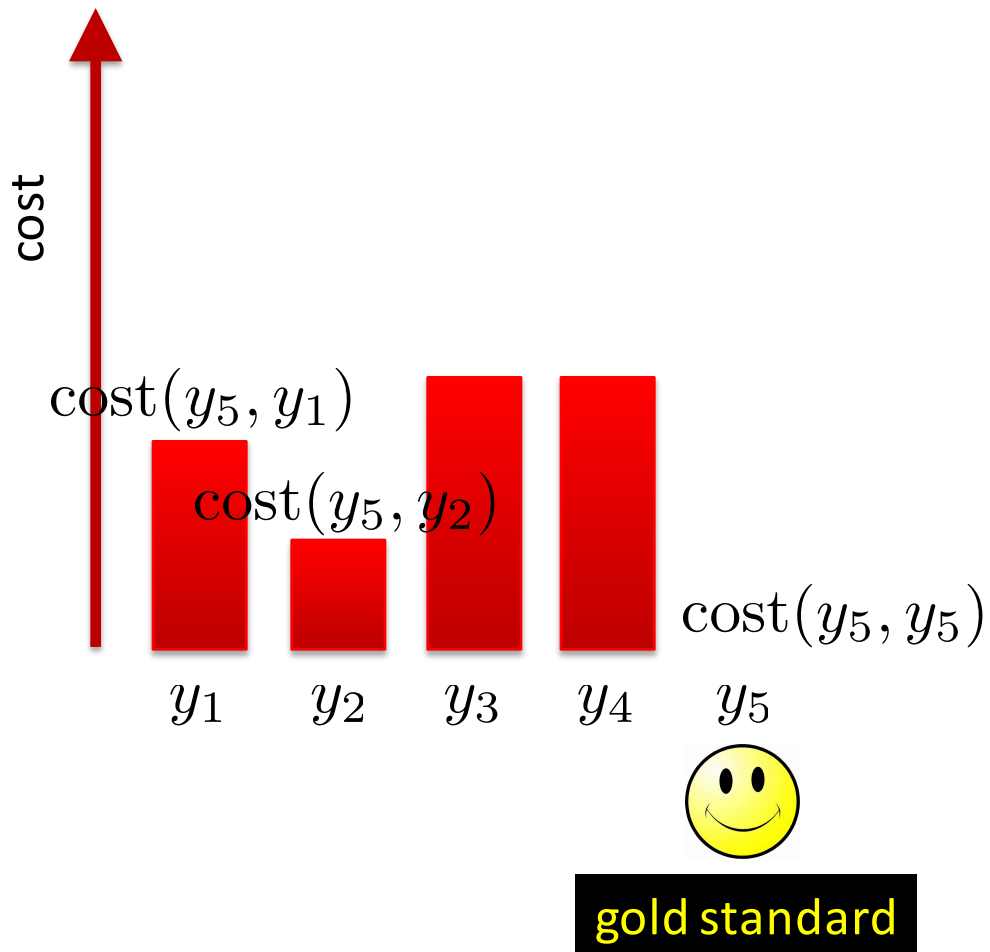
# Visualization



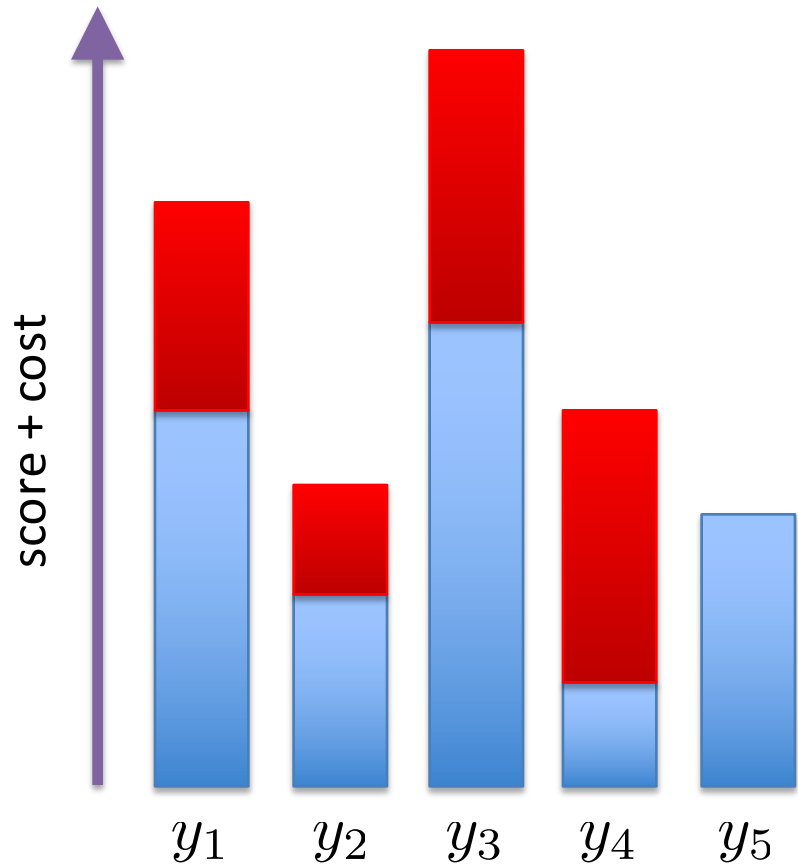
# Visualization



# Visualization



# Visualization



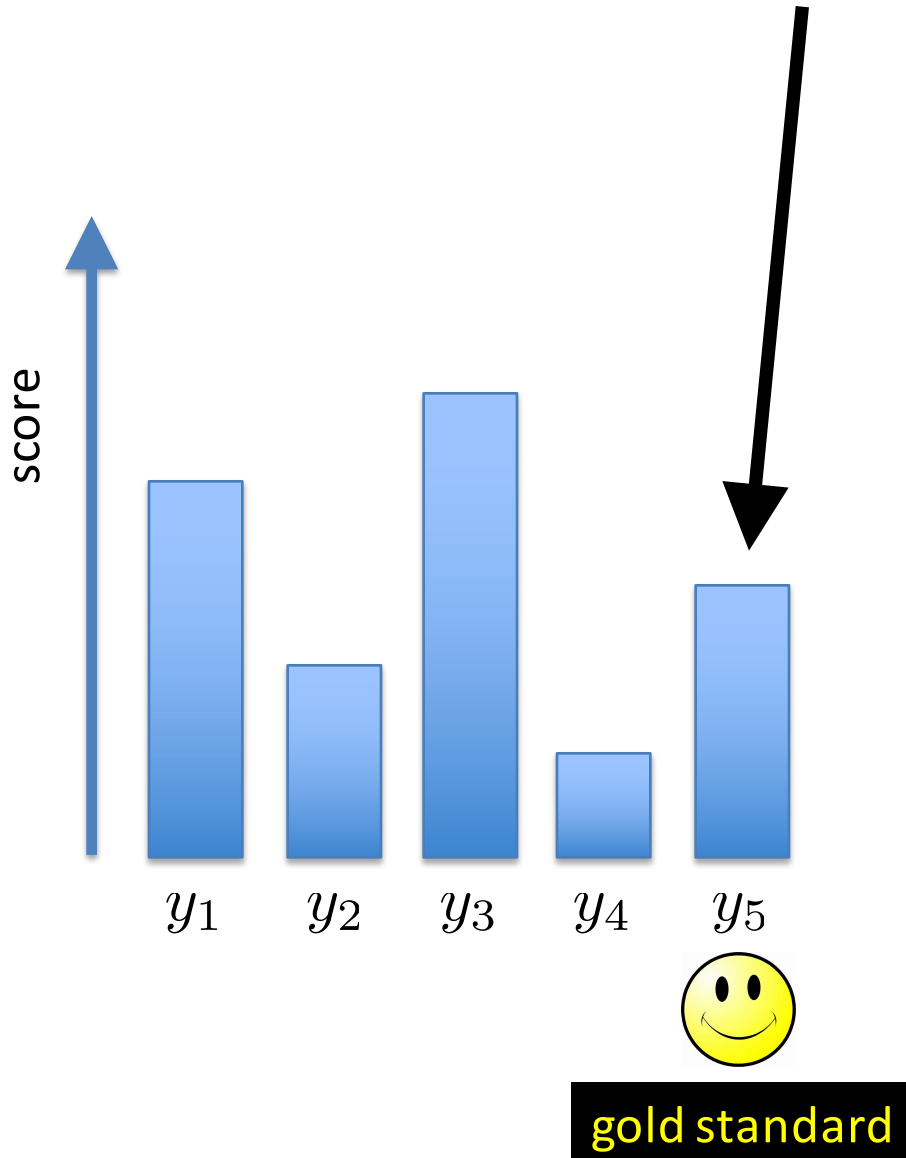
gold standard

perceptron loss:

$$\text{loss}_{\text{perc}}(\mathbf{x}, y, \boldsymbol{\theta}) = -\text{score}(\mathbf{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \boldsymbol{\theta})$$

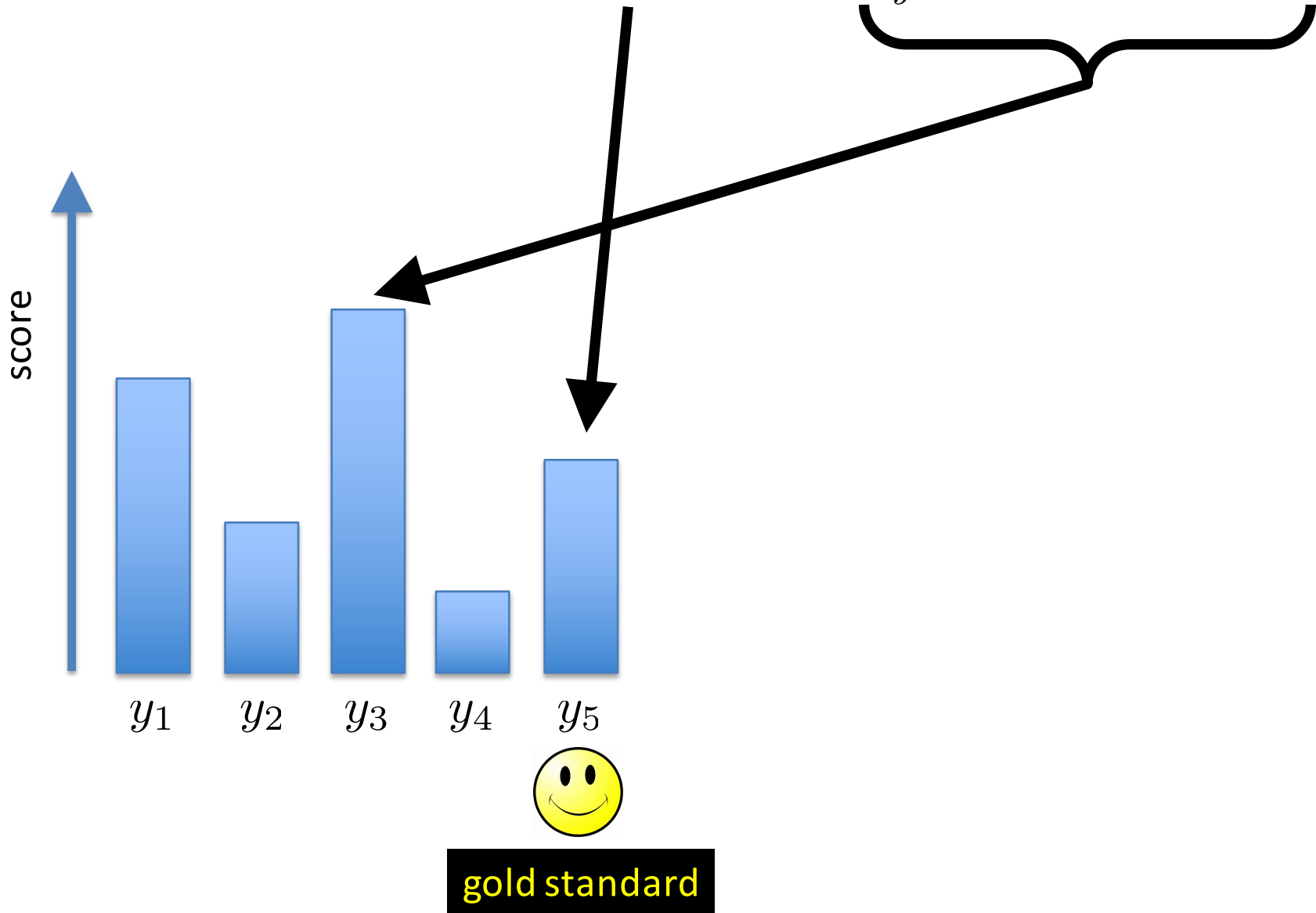
perceptron loss:

$$\text{loss}_{\text{perc}}(\mathbf{x}, y, \boldsymbol{\theta}) = -\text{score}(\mathbf{x}, y, \boldsymbol{\theta}) + \underbrace{\max_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \boldsymbol{\theta})}$$



perceptron loss:

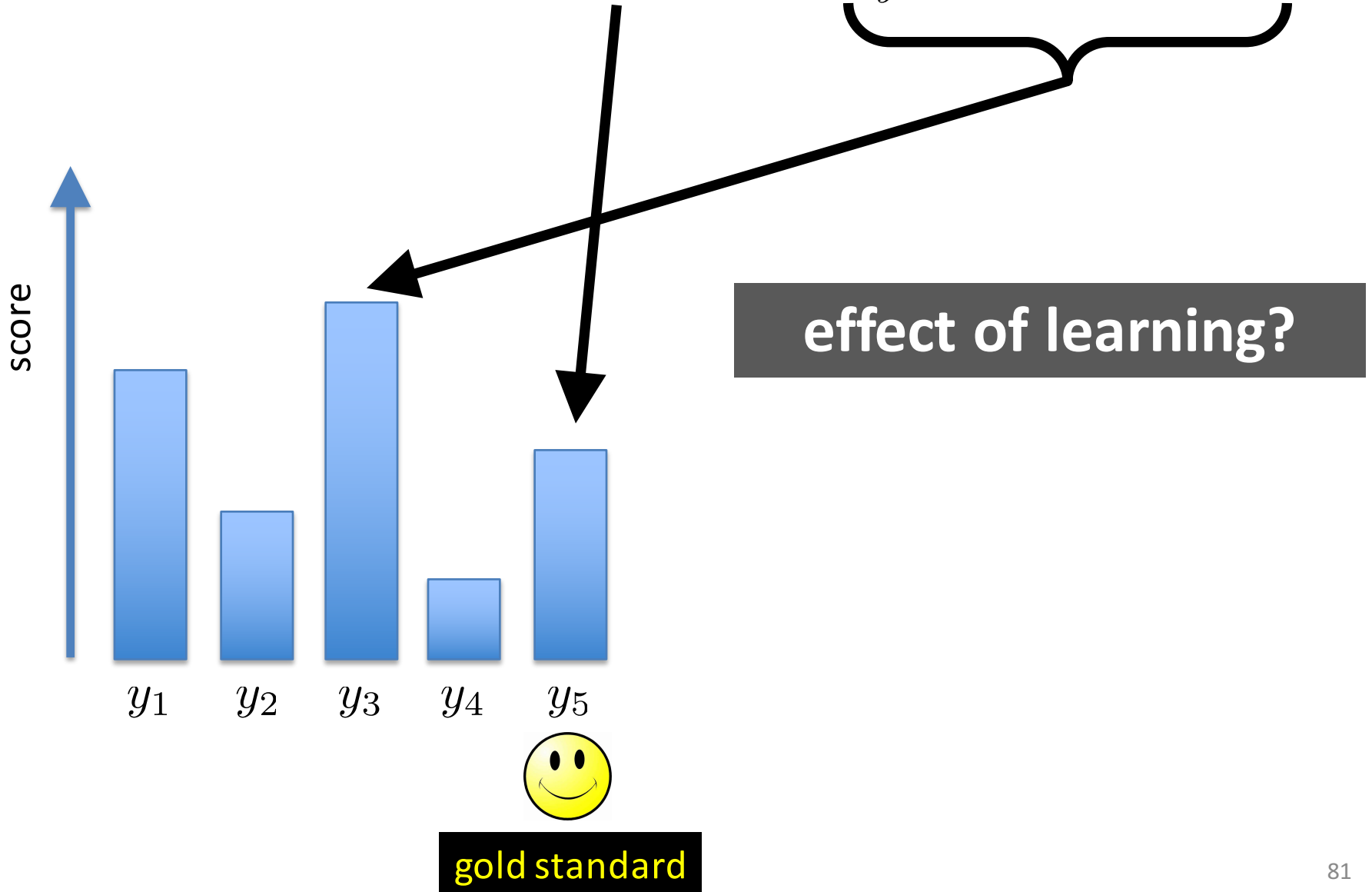
$$\text{loss}_{\text{perc}}(\mathbf{x}, y, \boldsymbol{\theta}) = -\text{score}(\mathbf{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \boldsymbol{\theta})$$





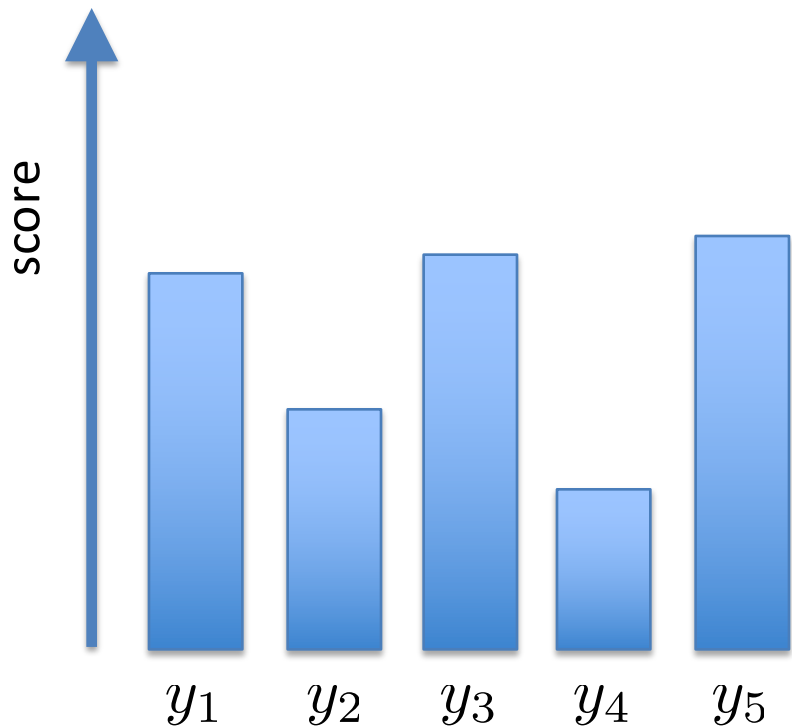
perceptron loss:

$$\text{loss}_{\text{perc}}(\mathbf{x}, y, \boldsymbol{\theta}) = -\text{score}(\mathbf{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \boldsymbol{\theta})$$



perceptron loss:

$$\text{loss}_{\text{perc}}(\mathbf{x}, y, \boldsymbol{\theta}) = -\text{score}(\mathbf{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \boldsymbol{\theta})$$



gold standard

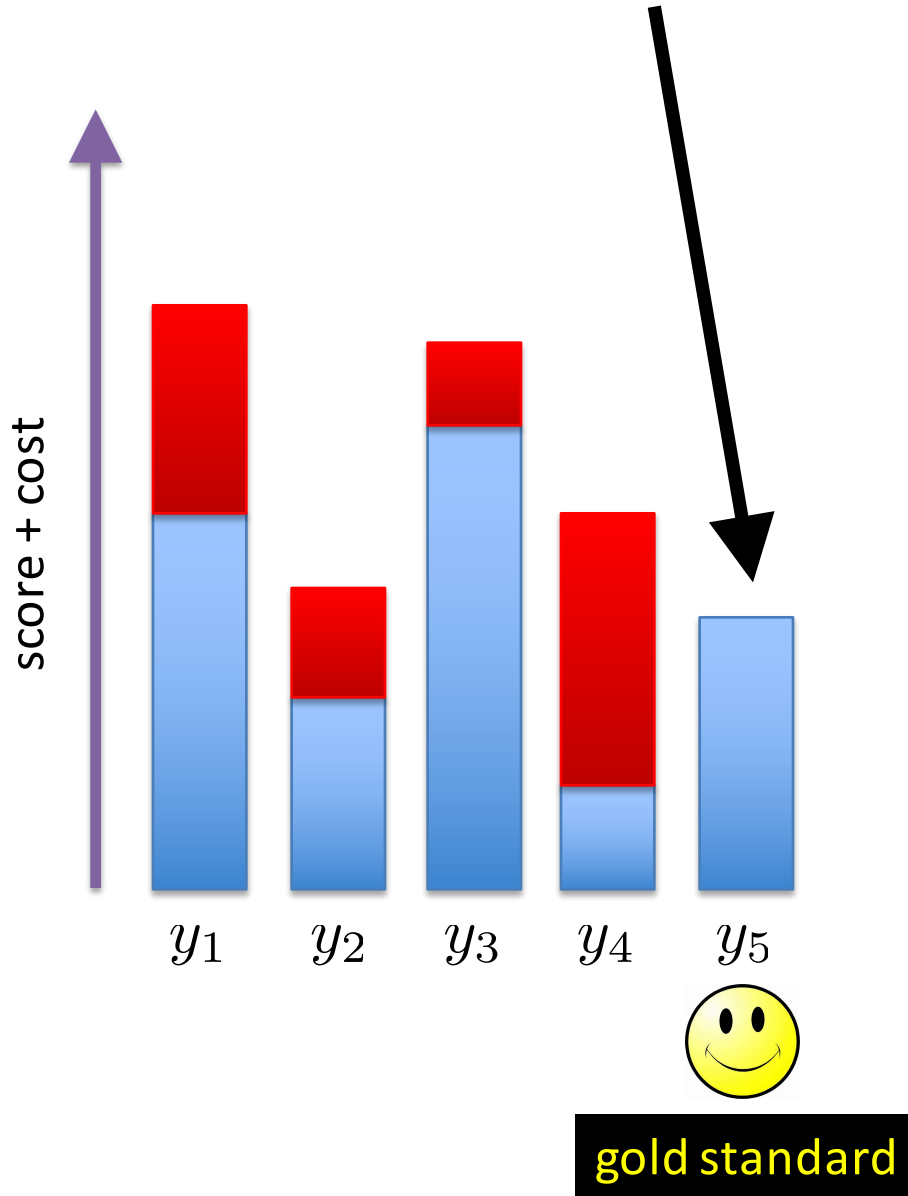
effect of learning:  
gold standard will  
have highest score

hinge loss:

$$\text{loss}_{\text{hinge}}(\mathbf{x}, y, \boldsymbol{\theta}) = -\text{score}(\mathbf{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} (\text{score}(\mathbf{x}, y', \boldsymbol{\theta}) + \text{cost}(y, y'))$$

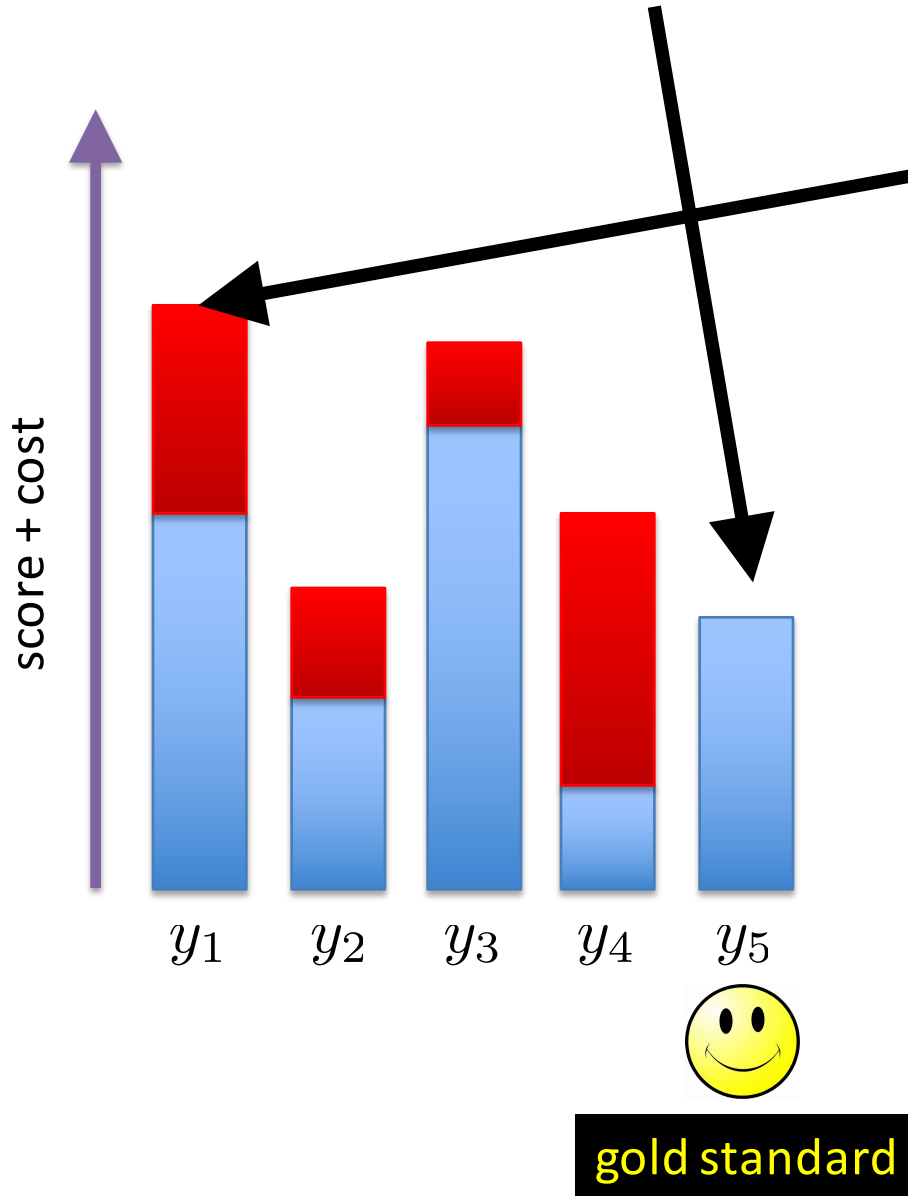
hinge loss:

$$\text{loss}_{\text{hinge}}(\mathbf{x}, y, \theta) = -\text{score}(\mathbf{x}, y, \theta) + \underbrace{\max_{y' \in \mathcal{L}} (\text{score}(\mathbf{x}, y', \theta) + \text{cost}(y, y'))}_{\text{hinge loss}}$$



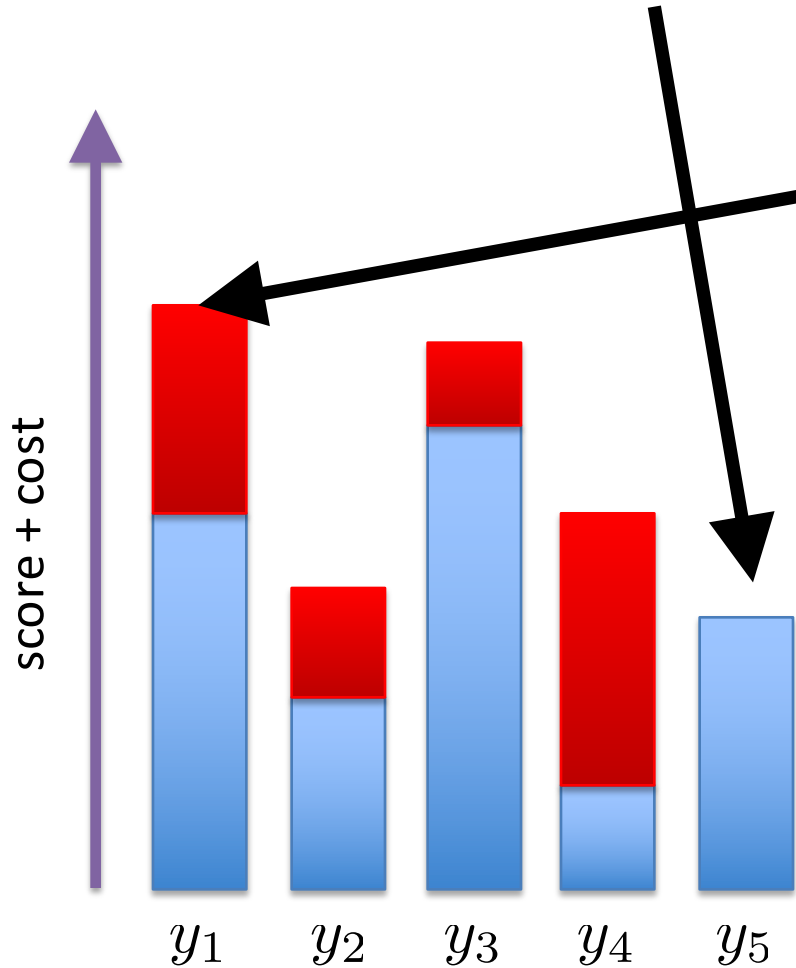
hinge loss:

$$\text{loss}_{\text{hinge}}(\mathbf{x}, y, \theta) = -\text{score}(\mathbf{x}, y, \theta) + \max_{y' \in \mathcal{L}} (\text{score}(\mathbf{x}, y', \theta) + \text{cost}(y, y'))$$



hinge loss:

$$\text{loss}_{\text{hinge}}(\mathbf{x}, y, \theta) = -\text{score}(\mathbf{x}, y, \theta) + \max_{y' \in \mathcal{L}} (\text{score}(\mathbf{x}, y', \theta) + \text{cost}(y, y'))$$



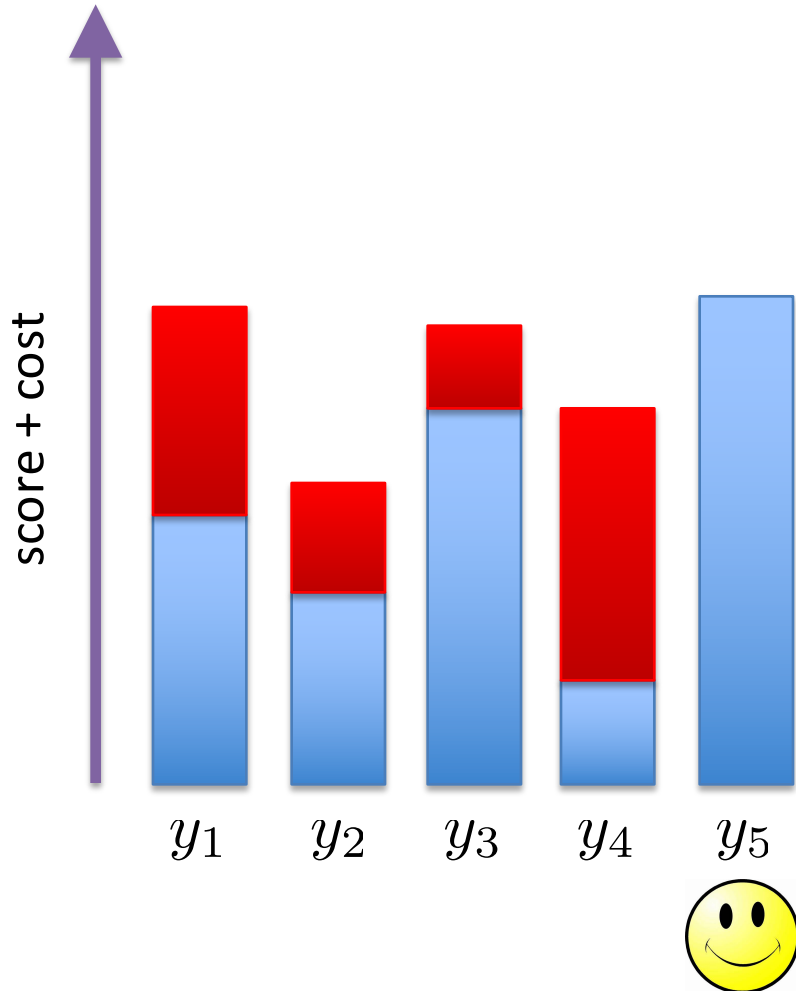
effect of learning?



gold standard

hinge loss:

$$\text{loss}_{\text{hinge}}(\mathbf{x}, y, \boldsymbol{\theta}) = -\text{score}(\mathbf{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} (\text{score}(\mathbf{x}, y', \boldsymbol{\theta}) + \text{cost}(y, y'))$$



**effect of learning:**  
score of gold standard  
will be higher than  
score+cost of all  
others

gold standard

# Regularized Empirical Risk Minimization

- given training data:  $\mathcal{T} = \{\langle \mathbf{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{|\mathcal{T}|}$

where each  $y^{(i)} \in \mathcal{L}$  is a label

- we want to solve the following:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) + \underbrace{\lambda R(\boldsymbol{\theta})}_{\text{regularization term}}$$

regularization  
strength



regularization  
term



# Regularization Terms

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) + \lambda R(\boldsymbol{\theta})$$

- most common: penalize large parameter values
- intuition: large parameters might be instances of overfitting
- examples:

**$L_2$  regularization:**  $R_{L_2}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2 = \sum_i \theta_i^2$

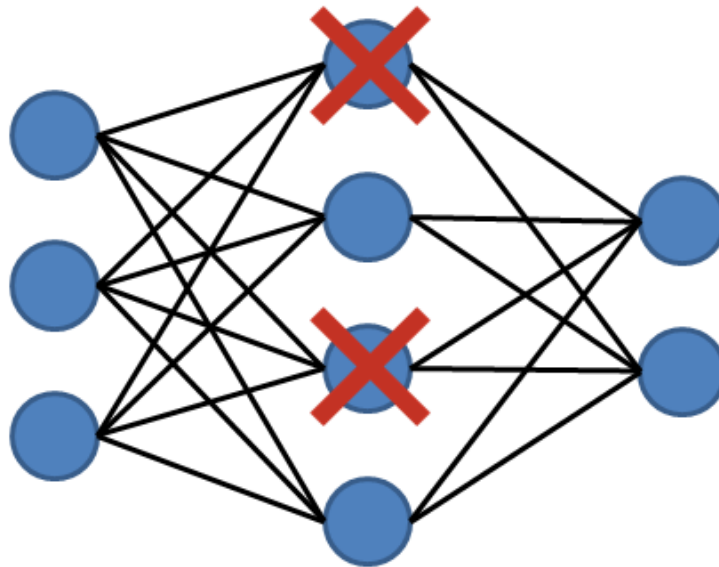
(also called Tikhonov regularization  
or ridge regression)

**$L_1$  regularization:**  $R_{L_1}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum_i |\theta_i|$

(also called basis pursuit or LASSO)

# Dropout

- popular regularization method for neural networks
- randomly “drop out” (set to zero) some of the vector entries in the layers



# Inference

# Exponentially-Large Search Problems

**inference: solve** argmax

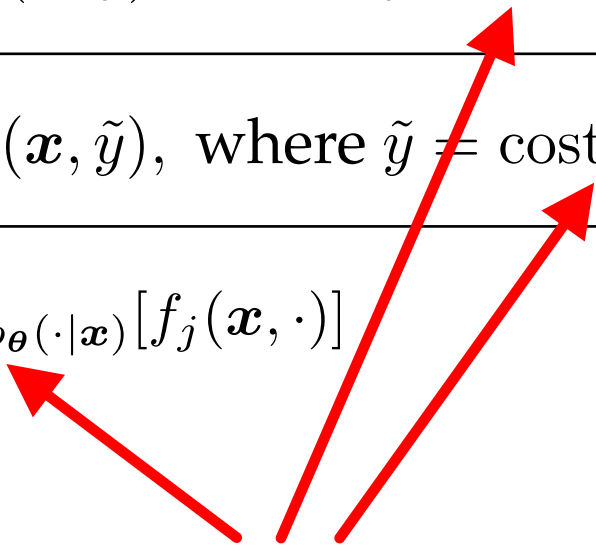
$$\text{classify}(x, \theta) = \underset{y}{\text{argmax}} \text{ score}(x, y, \theta)$$

- when output is a sequence or tree, this argmax requires iterating over an exponentially-large set

# Learning requires solving exponentially-hard problems too!

loss	entry $j$ of (sub)gradient of loss for linear model
perceptron	$-f_j(\mathbf{x}, y) + f_j(\mathbf{x}, \hat{y})$ , where $\hat{y} = \text{classify}(\mathbf{x}, \boldsymbol{\theta})$
hinge	$-f_j(\mathbf{x}, y) + f_j(\mathbf{x}, \tilde{y})$ , where $\tilde{y} = \text{costClassify}(\mathbf{x}, y, \boldsymbol{\theta})$
log	$-f_j(\mathbf{x}, y) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\cdot \mathbf{x})}[f_j(\mathbf{x}, \cdot)]$

computing each of these terms  
requires iterating through every  
possible output



# Dynamic Programming (DP)

- what is dynamic programming?
  - a family of algorithms that break problems into smaller pieces and reuse solutions for those pieces
  - only applicable when the problem has certain properties (**optimal substructure** and **overlapping sub-problems**)
- in this class, we use DP to iterate over exponentially-large output spaces in polynomial time
- we focus on a particular type of DP algorithm: **memoization**

# Implementing DP algorithms

- even if your goal is to compute a sum or a max, focus first on **counting mode** (count the number of unique outputs for an input)
- memoization = recursion + saving/reusing solutions
  - start by defining recursive equations
  - “**memoize**” by creating a table to store all intermediate results from recursive equations, use them when requested

# Inference in HMMs

$$\text{classify}(\mathbf{x}, \boldsymbol{\theta}) = \underset{\mathbf{y}}{\operatorname{argmax}} p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}) = \underset{\mathbf{y}}{\operatorname{argmax}} \prod_{i=1}^{|\mathbf{x}|} p_{\boldsymbol{\tau}}(y_i | y_{i-1}) p_{\boldsymbol{\eta}}(x_i | y_i)$$

- since the output is a sequence, this argmax requires iterating over an exponentially-large set
- last week we talked about using dynamic programming (DP) to solve these problems
- for HMMs (and other sequence models), the for solving this is called the **Viterbi algorithm**



# Viterbi Algorithm

- recursive equations + memoization:

**base case:**

returns probability of sequence starting with label  $y$  for first word



$$V(1, y) = p_{\eta}(x_1 | y) p_{\tau}(y | \langle s \rangle)$$

$$V(m, y) = \max_{y' \in \mathcal{L}} ( p_{\eta}(x_m | y) p_{\tau}(y | y') V(m - 1, y') )$$



**recursive case:**

computes probability of max-probability label sequence that ends with label  $y$  at position  $m$

**final value is in:**  $V(|\mathbf{x}| + 1, \langle /s \rangle)$

# Viterbi Algorithm

- space and time complexity?
- can be read off from the recursive equations:

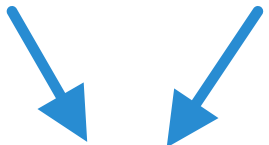
**space complexity:**

size of memoization table, which is # of unique indices of recursive equations

length of  
sentence

\*

number  
of labels


$$V(m, y) = \max_{y' \in \mathcal{L}} ( p_{\eta}(x_m | y) p_{\tau}(y | y') V(m - 1, y') )$$

**so, space complexity is  $O(|x| |L|)$**

# Viterbi Algorithm

- space and **time** complexity?
- can be read off from the recursive equations:

**time complexity:**

size of memoization table \* complexity of computing each entry

length of sentence \* number of labels \* each entry requires iterating through the labels

$$V(m, y) = \max_{y' \in \mathcal{L}} (p_{\eta}(x_m | y) p_{\tau}(y | y') V(m - 1, y'))$$

so, time complexity is  $O(|x| |L| |L|) = O(|x| |L|^2)$

# Feature Locality

- **feature locality**: how “big” are your features?
- when designing efficient inference algorithms (whether w/ DP or other methods), we need to be mindful of this
- features can be arbitrarily big in terms of the input, but not in terms of the *output*!
- the features in HMMs are small in both the input and output sequences (only two pieces at a time)