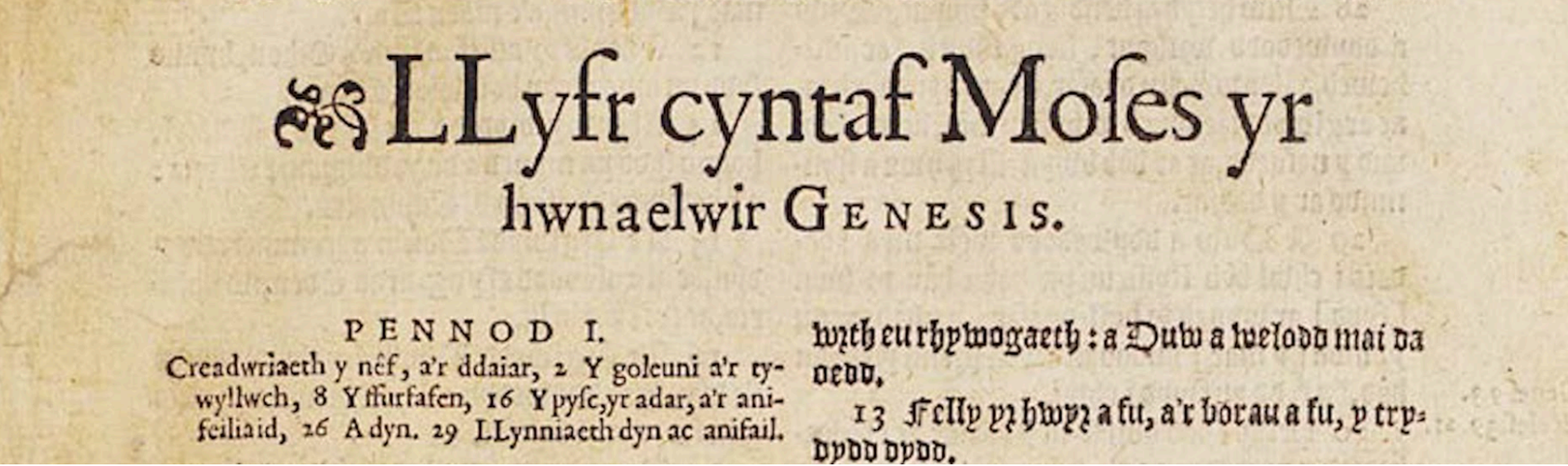

LLyfr cyntaf Moſes yr
hwnaelwir GENESIS.

PENNOD I.

Creadwriaeth y nêf, a'r ddaiar, 2 Y goleuni a'r tywyllwch, 8 Yffurfafen, 16 Ypyſc,yr adar, a'r anifeiliaid, 26 A dyn. 29 LLynniaeth dyn ac anifail.

wyth eu rhyfwogaeth: a Duw a welodd mai da oedd.

13 Felly yr hwyr a fu, a'r borau a fu, y trydydd dydd.

Natural Language Processing

Info 159/259

Lecture 14: Dependency parsing (March 7, 2023)

David Bamman, UC Berkeley

PCFGs

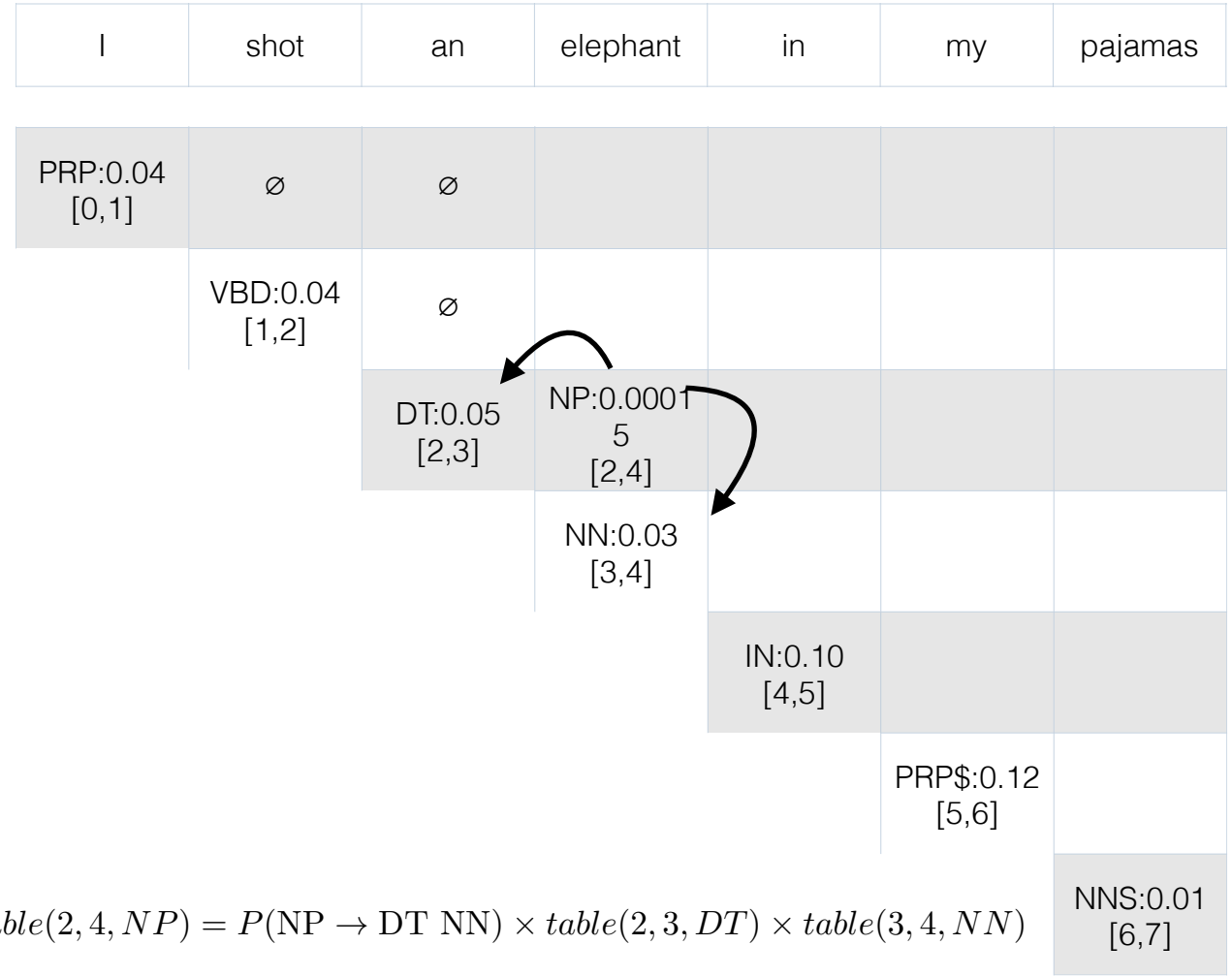
- A PCFG gives us a mechanism for assigning scores (here, probabilities) to different parses for the same sentence.
- But we often care about is finding the single best parse with the highest probability.
- We calculate the max probability parse using CKY by storing the probability of each phrase within each cell as we build it up.

$$table(i, j, A) = P(A \rightarrow BC) \times table(i, k, B) \times table(k, j, C)$$

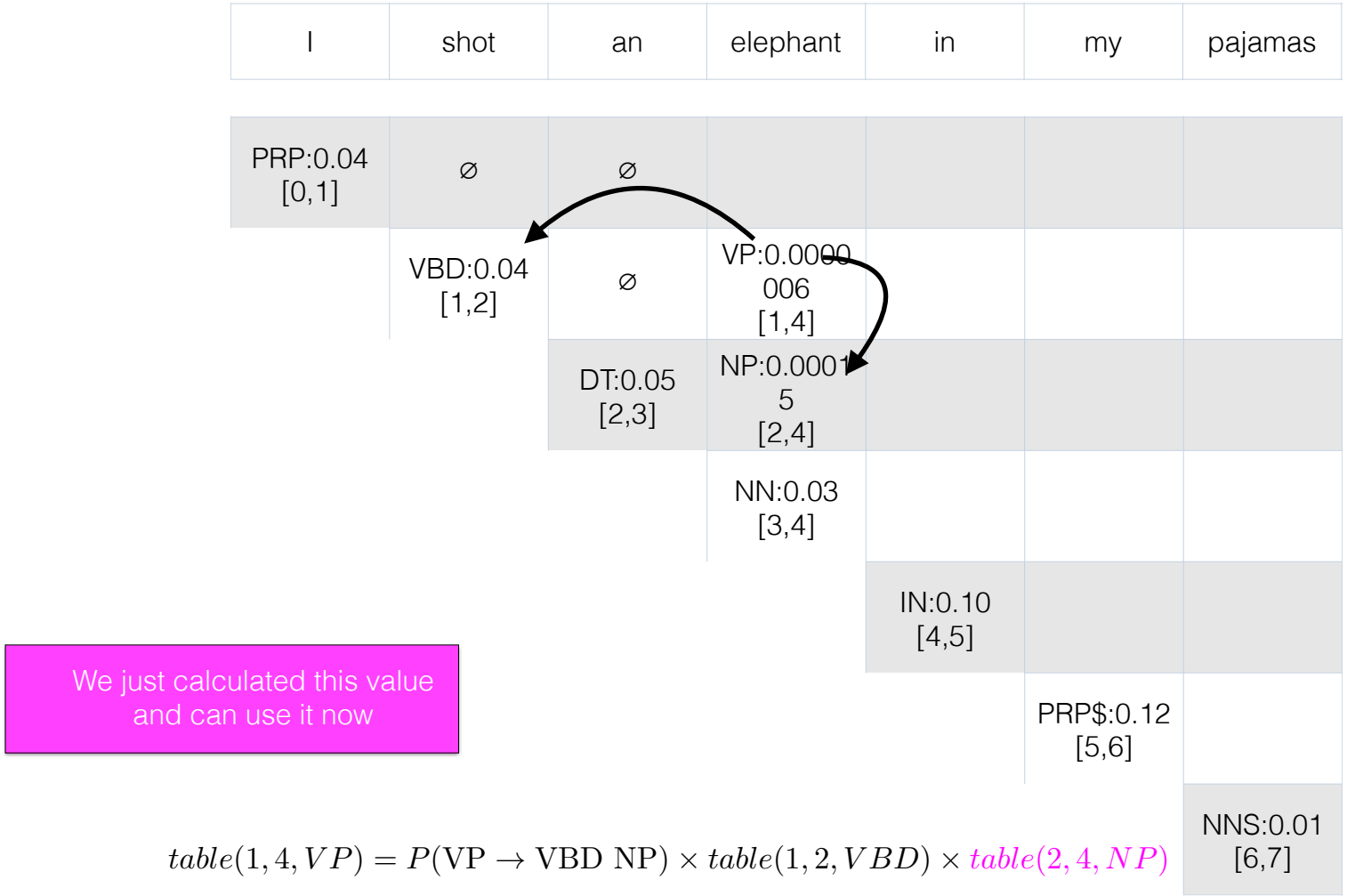
I	shot	an	elephant	in	my	pajamas
PRP:0.04 [0,1]						
	VBD:0.04 [1,2]					
		DT:0.05 [2,3]				
			NN:0.03 [3,4]			
				IN:0.10 [4,5]		
					PRP$:0.12 [5,6]	
						NNS:0.01 [6,7]

Probability of a terminal (word) given its tag

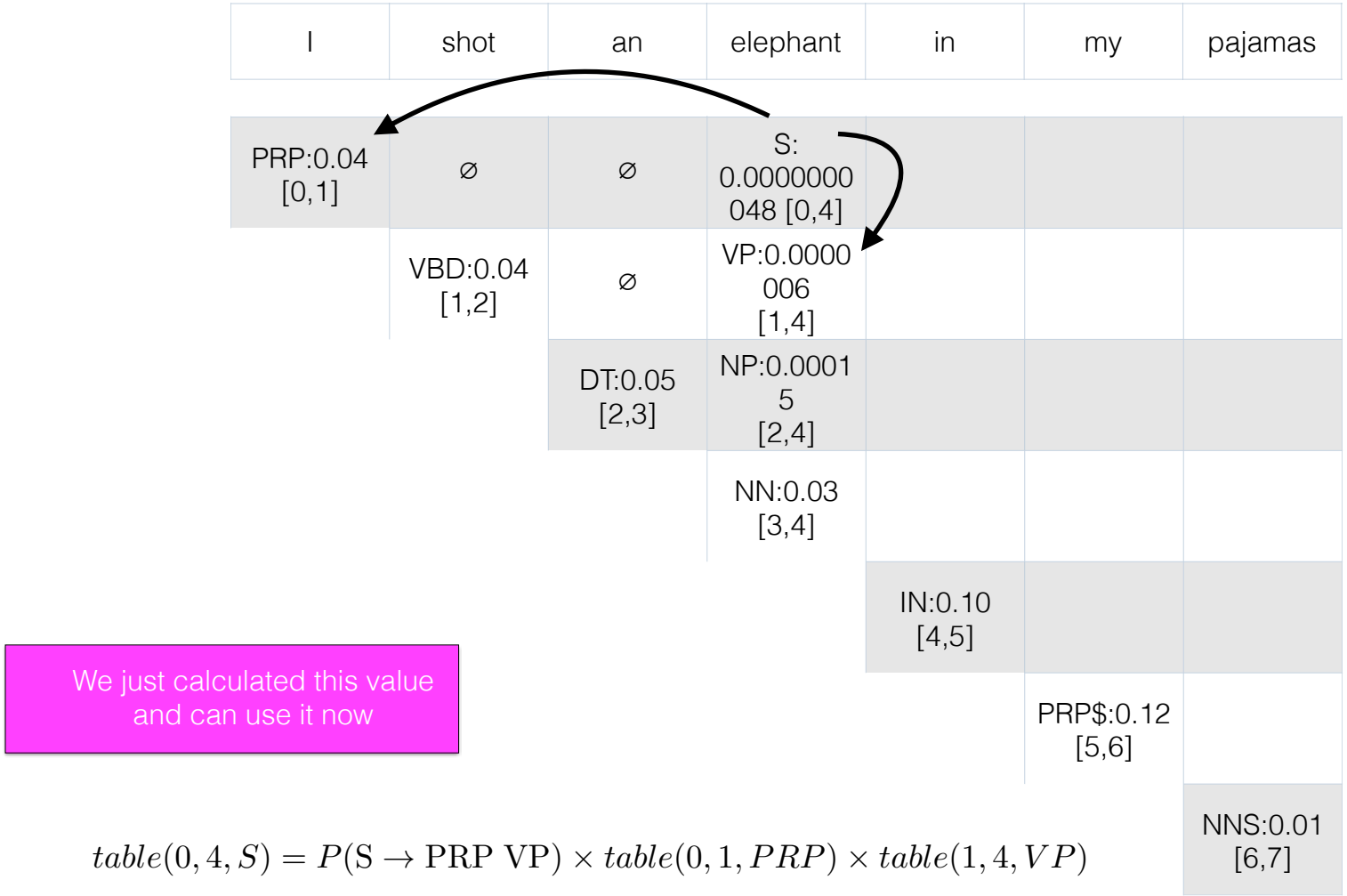
$$P(A \rightarrow \beta)$$



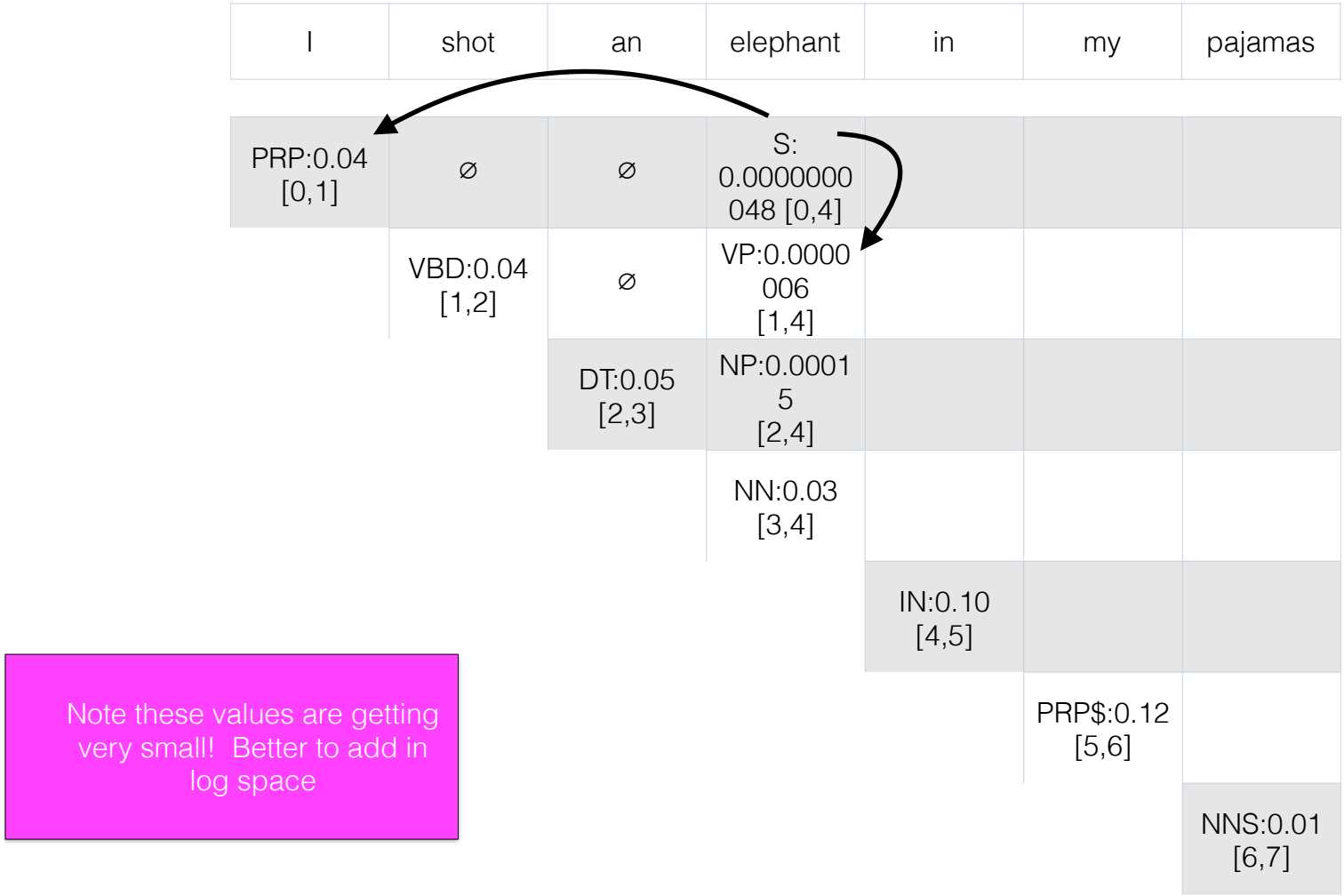
$$table(2, 4, NP) = P(\text{NP} \rightarrow \text{DT NN}) \times table(2, 3, DT) \times table(3, 4, NN)$$

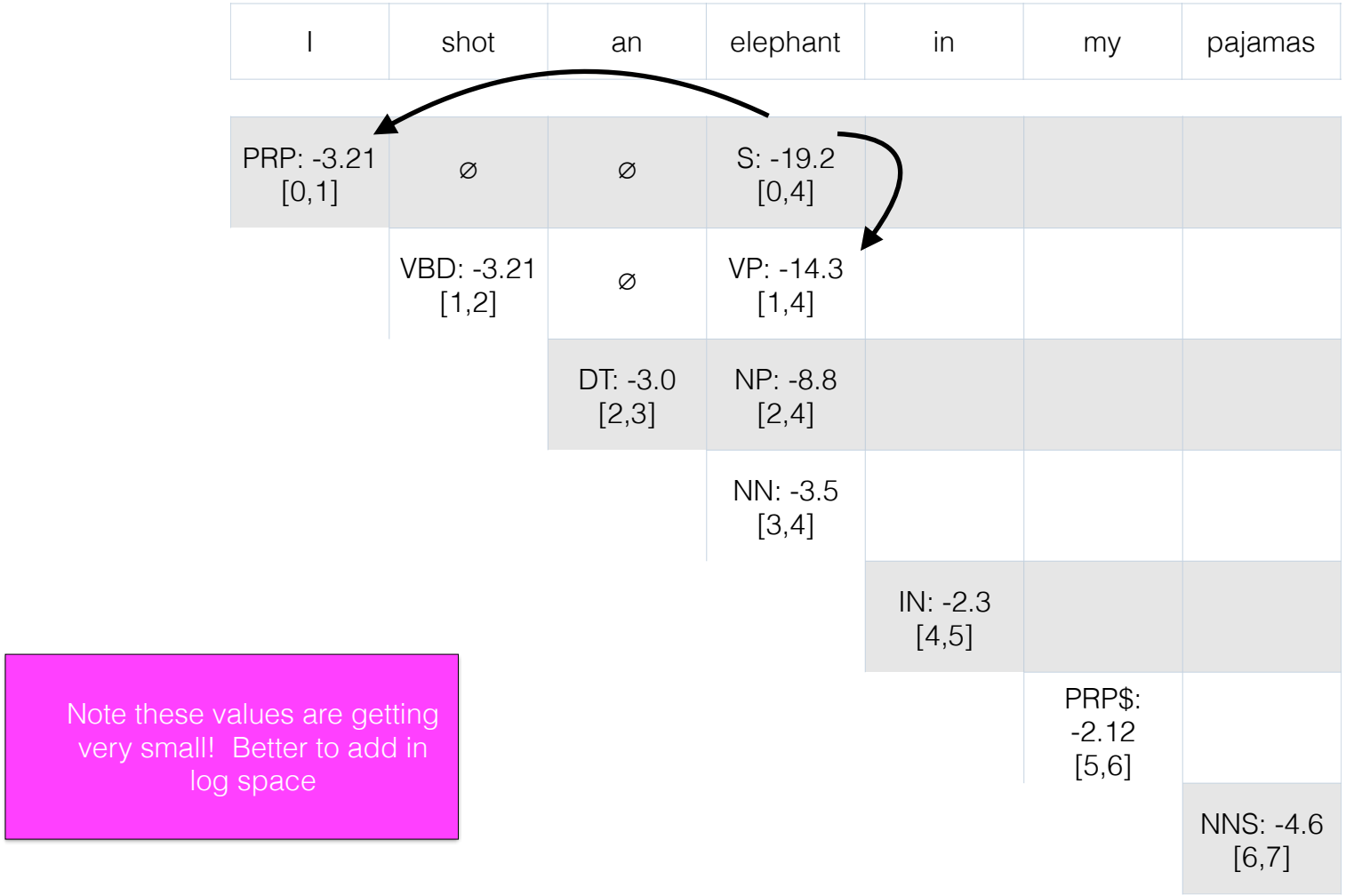


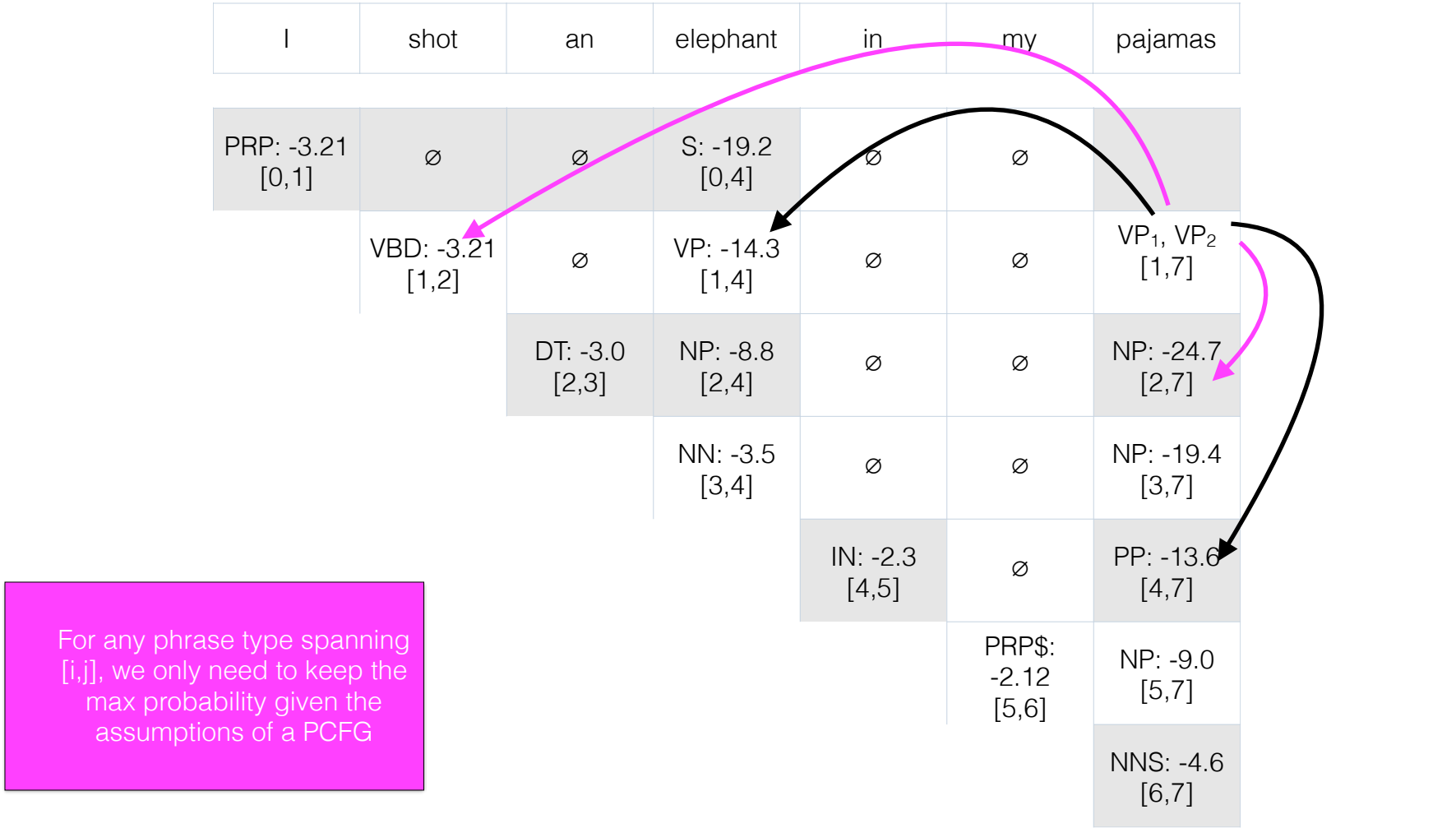
$$table(1,4,VP) = P(\text{VP} \rightarrow \text{VBD NP}) \times table(1,2,VBD) \times table(2,4,NP)$$



$$table(0, 4, S) = P(\text{S} \rightarrow \text{PRP VP}) \times table(0, 1, PRP) \times table(1, 4, VP)$$


I
shot
an
elephant
in
my
pajamas
PRP:0.04 [0,1]
∅
∅
S: 0.0000000048 [0,4]
VBD:0.04 [1,2]
∅
VP:0.0000006 [1,4]
DT:0.05 [2,3]
NP:0.00015 [2,4]
NN:0.03 [3,4]
IN:0.10 [4,5]
PRP$:0.12 [5,6]
NNS:0.01 [6,7]
Note these values are getting very small! Better to add in log space


I
shot
an
elephant
in
my
pajamas
PRP: -3.21
[0,1]
∅
∅
S: -19.2
[0,4]
VBD: -3.21
[1,2]
∅
VP: -14.3
[1,4]
DT: -3.0
[2,3]
NP: -8.8
[2,4]
NN: -3.5
[3,4]
IN: -2.3
[4,5]
PRP$:
-2.12
[5,6]
NNS: -4.6
[6,7]
Note these values are getting very small! Better to add in log space

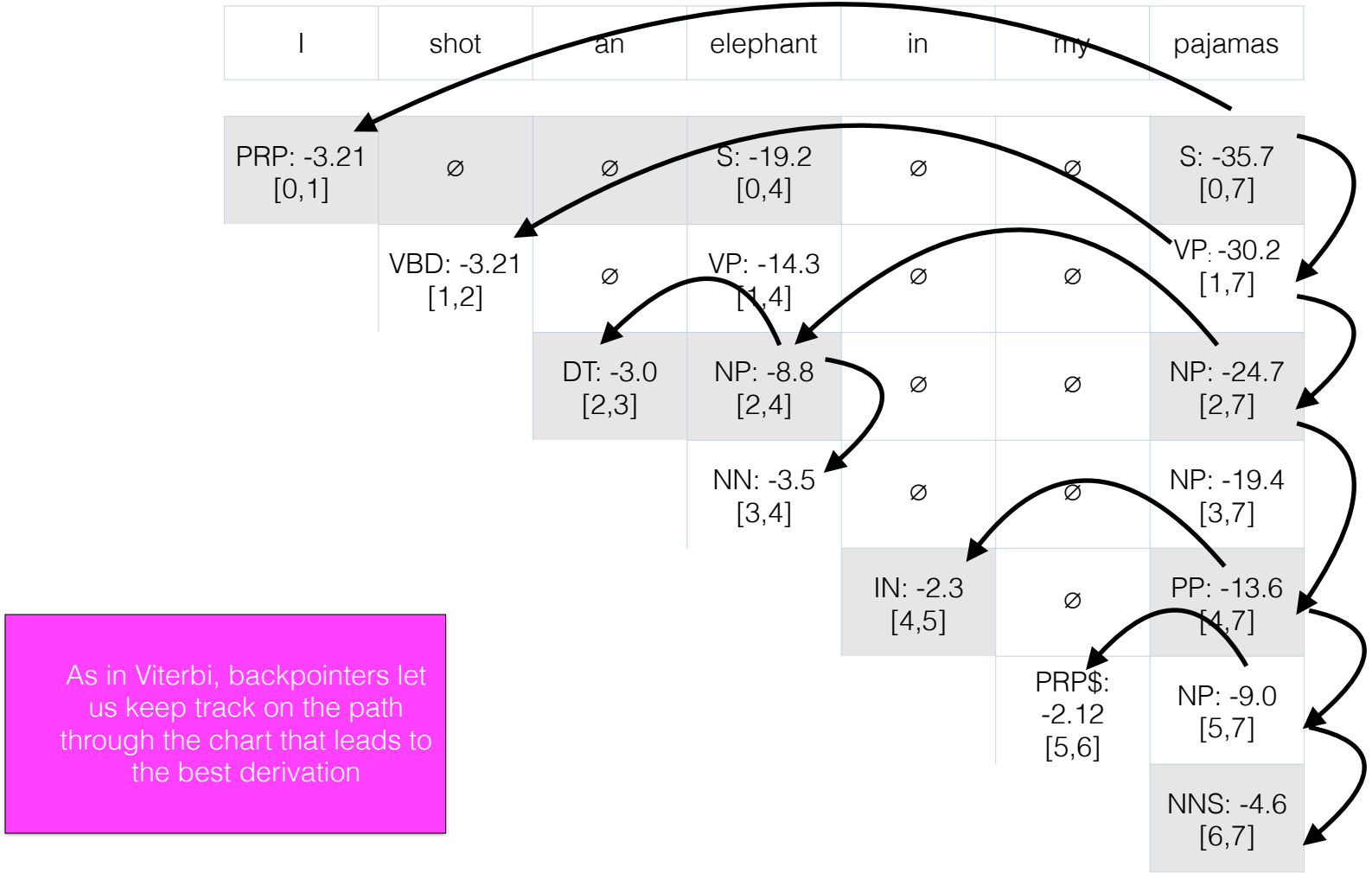

I
shot
an
elephant
in
my
pajamas
PRP: -3.21 [0,1]
∅
∅
S: -19.2 [0,4]
∅
∅
VBD: -3.21 [1,2]
∅
VP: -14.3 [1,4]
∅
∅
VP1, VP2 [1,7]
DT: -3.0 [2,3]
NP: -8.8 [2,4]
∅
∅
NP: -24.7 [2,7]
NN: -3.5 [3,4]
∅
∅
NP: -19.4 [3,7]
IN: -2.3 [4,5]
∅
PP: -13.6 [4,7]
PRP$: -2.12 [5,6]
NP: -9.0 [5,7]
NNS: -4.6 [6,7]
For any phrase type spanning [i,j], we only need to keep the max probability given the assumptions of a PCFG

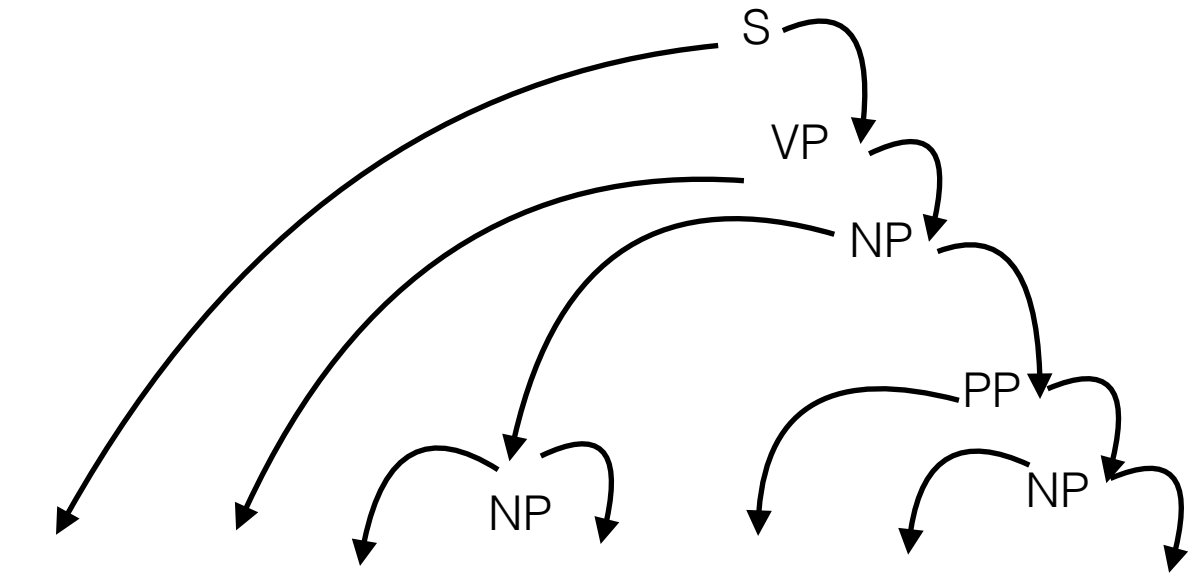
I	shot	an	elephant	in	my	pajamas
PRP: -3.21 [0,1]	∅	∅	S: -19.2 [0,4]	∅	∅	
	VBD: -3.21 [1,2]	∅	VP: -14.3 [1,4]	∅	∅	VP: -30.2 [1,7]
		DT: -3.0 [2,3]	NP: -8.8 [2,4]	∅	∅	NP: -24.7 [2,7]
			NN: -3.5 [3,4]	∅	∅	NP: -19.4 [3,7]
				IN: -2.3 [4,5]	∅	PP: -13.6 [4,7]
					PRP$: -2.12 [5,6]	NP: -9.0 [5,7]
						NNS: -4.6 [6,7]

For any phrase type spanning [i,j], we only need to keep the max probability given the assumptions of a PCFG

I	shot	an	elephant	in	my	pajamas
PRP: -3.21 [0,1]	∅	∅	S: -19.2 [0,4]	∅	∅	S: -35.7 [0,7]
	VBD: -3.21 [1,2]	∅	VP: -14.3 [1,4]	∅	∅	VP: -30.2 [1,7]
		DT: -3.0 [2,3]	NP: -8.8 [2,4]	∅	∅	NP: -24.7 [2,7]
			NN: -3.5 [3,4]	∅	∅	NP: -19.4 [3,7]
				IN: -2.3 [4,5]	∅	PP: -13.6 [4,7]
					PRP$: -2.12 [5,6]	NP: -9.0 [5,7]
						NNS: -4.6 [6,7]

For any phrase type spanning [i,j], we only need to keep the max probability given the assumptions of a PCFG


I
shot
an
elephant
in
my
pajamas
PRP: -3.21 [0,1]
S: -19.2 [0,4]
S: -35.7 [0,7]
VBD: -3.21 [1,2]
VP: -14.3 [1,4]
VP: -30.2 [1,7]
DT: -3.0 [2,3]
NP: -8.8 [2,4]
NP: -24.7 [2,7]
NN: -3.5 [3,4]
NP: -19.4 [3,7]
IN: -2.3 [4,5]
PP: -13.6 [4,7]
PRP$: -2.12 [5,6]
NP: -9.0 [5,7]
NNS: -4.6 [6,7]
As in Viterbi, backpointers let us keep track on the path through the chart that leads to the best derivation


S
VP
NP
PP
NP
NP
PRP
VBD
DT
NN
IN
PRP
NNS
I
shot
an
elephant
in
my
pajamas

Dependency syntax

- Syntactic structure = asymmetric, binary relations between words.

Tesnier 1959; Nivre 2005

Trees

- A dependency structure is a directed graph G = (V,A) consisting of a set of vertices *V* and arcs *A* between them. Typically constrained to form a tree:
  - Single root vertex with no incoming arcs
  - Every vertex has exactly one incoming arc except root (single head constraint)
  - There is a unique path from the root to each vertex in V (acyclic constraint)

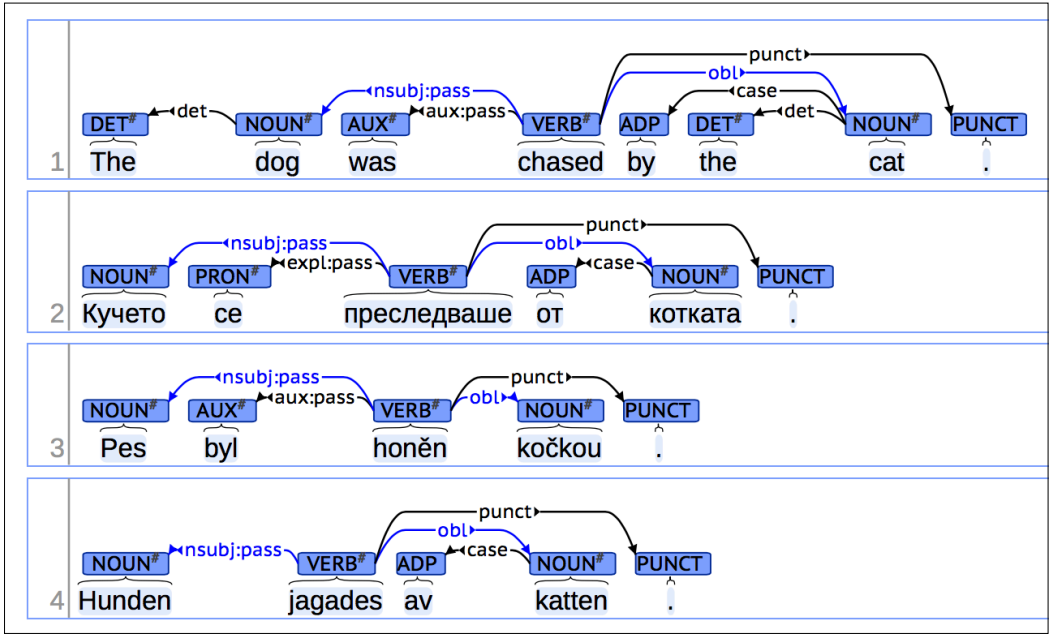
Universal Dependencies

English

Bulgarian

Czech

Swedish

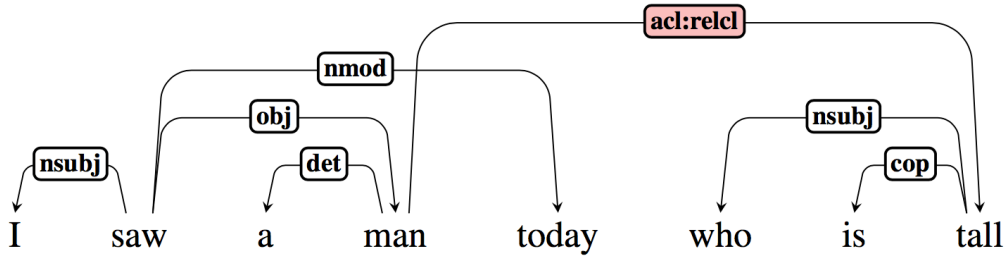


http://universaldependencies.org

Dependency parsing

- Transition-based parsing
  - O(n)
  - Only projective structures (pseudo-projective [Nivre and Nilsson 2005])
- Graph-based parsing
  - $O(n^2)$
  - Projective and non-projective trees

Projectivity



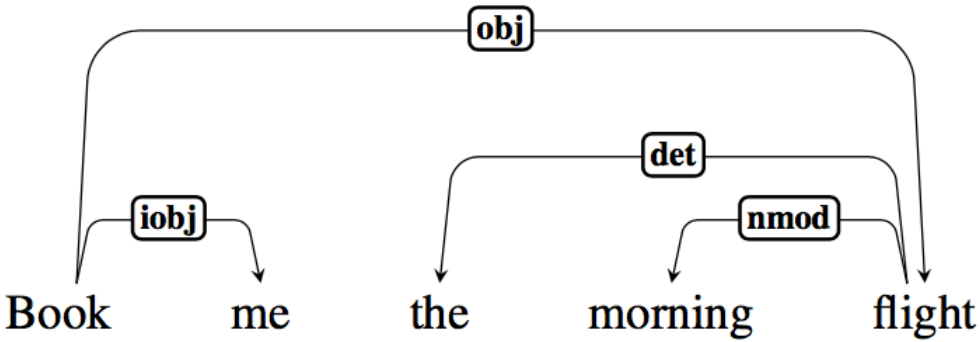
- An arc between a head and dependent is projective if there is a path from the head to every word between the head and dependent. Every word between head and dependent is a descendent of the head.

Transition-based parsing

- Basic idea: parse a sentence into a dependency tree by training a local classifier to predict a parser's next action from its current configuration.

Configuration

- Stack
- Input buffer of words
- Arcs in a parsed dependency tree
- Parsing = sequences of transitions through space of possible configurations


obj
det
iobj
nmod
Book
me
the
morning
flight

∅ book me the morning flight

stack	action	arc

∅ book me the morning flight

stack	action	arc

LeftArc(label): assert relation between head at $stack_1$ and dependent at $stack_2$; remove $stack_2$

RightArc(label): assert relation between head at $stack_2$ and dependent at $stack_1$; remove $stack_1$

Shift: Remove word from front of input buffer (∅) and push it onto stack

book me the morning flight

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (∅) and dependent at $stack_2$: remove $stack_2$	
	RightArc(label): assert relation between head at $stack_2$ and dependent at $stack_1$ (∅); remove $stack_1$ (∅)	
∅	 Shift: Remove word from front of input buffer (book) and push it onto stack	

If we remove an element from the stack, it can't have any further dependents

me the morning flight

stack	action	arc
book	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	
∅	RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	
	☞ Shift: Remove word from front of input buffer (me) and push it onto stack	

the morning flight

stack	action	arc
		iobj(book, me)
	LeftArc(label): assert relation between head at $stack_1$ (me) and dependent at $stack_2$ (book); remove $stack_2$ (book)	
me	☞ RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (me); remove $stack_1$ (me)	
book		
∅	Shift: Remove word from front of input buffer (the) and push it onto stack	

the morning flight

stack	action	arc
		iobj(book, me)
book	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	
∅	RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	
	Shift: Remove word from front of input buffer (the) and push it onto stack	

morning flight

stack	action	arc
		iobj(book, me)
	LeftArc(label): assert relation between head at $stack_1$ (the) and dependent at $stack_2$ (book): remove $stack_2$ (book)	
the	RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (the); remove $stack_1$ (the)	
book		
∅	☞ Shift: Remove word from front of input buffer (morning) and push it onto stack	

flight

stack	action	arc
morning the book ∅	LeftArc(label): assert relation between head at $stack_1$ (morning) and dependent at $stack_2$ (the); remove $stack_2$ (the)	*iobj(book, me)*
	RightArc(label): assert relation between head at $stack_2$ (the) and dependent at $stack_1$ (morning); remove $stack_1$ (morning)	
	☞ Shift: Remove word from front of input buffer (flight) and push it onto stack	

stack	action	arc
flight	☞ LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (morning): remove $stack_2$ (morning)	*iobj(book, me)*
morning	RightArc(label): assert relation between head at $stack_2$ (morning) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	*nmod(flight, morning)*
the	~~Shift: Remove word from front of input buffer and push it onto stack~~	
book		
∅		

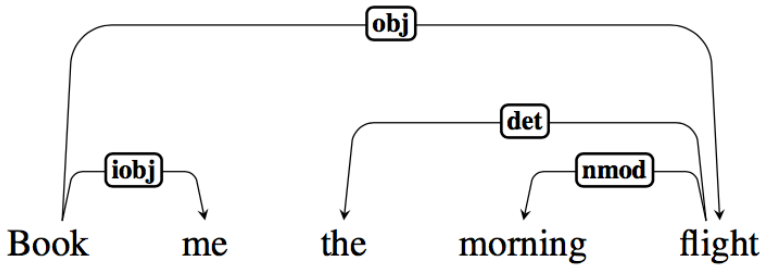
stack		action	arc
flight	☞	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (the); remove $stack_2$ (the)	*iobj(book, me)*
			nmod(flight, morning)
the			*det(flight, the)*
book		RightArc(label): assert relation between head at $stack_2$ (the) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	
∅		~~Shift: Remove word from front of input buffer and push it onto stack~~	

stack	action	arc
flight	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (book): remove $stack_2$ (book)	*iobj(book, me)*
		nmod(flight, morning)
		det(flight, the)
book	☞ RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	*obj(book, flight)*
∅	~~Shift: Remove word from front of input buffer and push it onto stack~~	

This is our parse

stack	action	arc
		iobj(book, me)
	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	*nmod(flight, morning)*
		det(flight, the)
	☞ RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	*obj(book, flight)*
book		*root(∅, book)*
∅	~~Shift: Remove word from front of input buffer and push it onto stack~~	

This is our parse



arc

iobj(book, me)

nmod(flight, morning)

det(flight, the)

obj(book, flight)

root(ø, book)

Let's go back to this earlier configuration

the morning flight

stack	action	arc
me	LeftArc(label): assert relation between head at $stack_1$ (me) and dependent at $stack_2$ (book); remove $stack_2$ (book)	
book	RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (me); remove $stack_1$ (me)	
∅	Shift: Remove word from front of input buffer (the) and push it onto stack	

- This is a multiclass classification problem: given the current configuration — i.e., the elements in the stack, the words in the buffer, and the arcs created so far, what's the best transition?

Output space $\mathcal{Y}$ =

Shift
LeftArc(nsubj)
RightArc(nsubj)
LeftArc(det)
RightArc(det)
LeftArc(obj)
RightArc(obj)
…

Features are scoped over the stack, buffer, and arcs created so far

stack

me

book

buffer

the morning flight

arc

feature	example
$stack_1$ = me	1
$stack_2$ = book	1
$stack_1$ POS = PRP	1
$buffer_1$ = the	1
$buffer_2$ = morning	1
$buffer_1$ = today	0
$buffer_1$ POS = RB	0
$stack_1$ = me AND $stack_2$ = book	1
$stack_1$ = PRP AND $stack_2$ = VB	1
iobj(book,*) in arcs	0

Use any multiclass classification model

- Logistic regression
- SVM
- NB
- Neural network

feature	example	β
$stack_1$ = me	1	0.7
$stack_2$ = book	1	1.3
$stack_1$ POS = PRP	1	6.4
$buffer_1$ = the	1	-1.3
$buffer_2$ = morning	1	-0.07
$buffer_1$ = today	0	0.52
$buffer_1$ POS = RB	0	-2.1
$stack_1$ = me AND $stack_2$ =	1	0
$stack_1$ = PRP AND $stack_2$ =	1	-0.1
iobj(book,*) in arcs	0	3.2

Training

We're training to predict the parser action —Shift, RightArc(label), LeftArc(label)—given the featurized configuration

Configuration features	Label
<stack1 = me, 1>, <stack2 = book, 1>, <stack1 POS = PRP, 1>, <buffer1 = the, 1>,	Shift
<stack1 = me, 0>, <stack2 = book, 0>, <stack1 POS = PRP, 0>, <buffer1 = the, 0>,	RightArc(det)
<stack1 = me, 0>, <stack2 = book, 1>, <stack1 POS = PRP, 0>, <buffer1 = the, 0>,	RightArc(nsubj)

Neural Shift-Reduce Parsing

- We can train a neural shift-reduce parser by just changing how we:
  - represent the configuration
  - predict the label from that representation
- Otherwise training and prediction remains the same.

Chen and Manning (2014), “A Fast and Accurate Dependency Parser using Neural Networks”

Neural Shift-Reduce Parsing

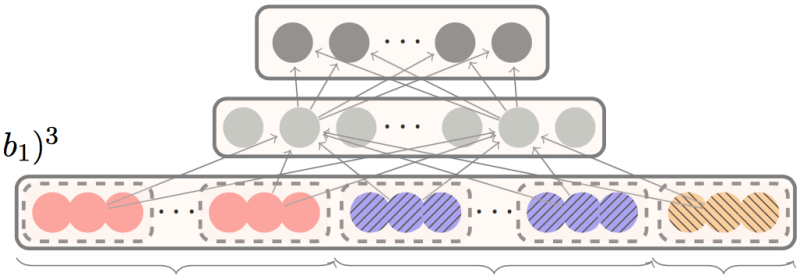
Softmax layer:

$$p = \texttt{softmax}(W_2 h)$$

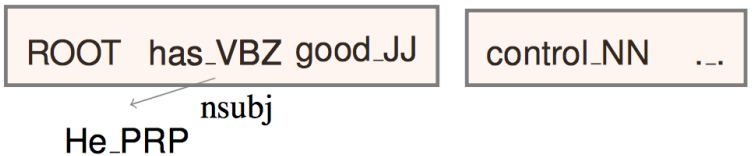
Hidden layer:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

Input layer: $[x^w, x^t, x^l]$



Configuration



Chen and Manning (2014), "A Fast and Accurate Dependency Parser using Neural Networks"

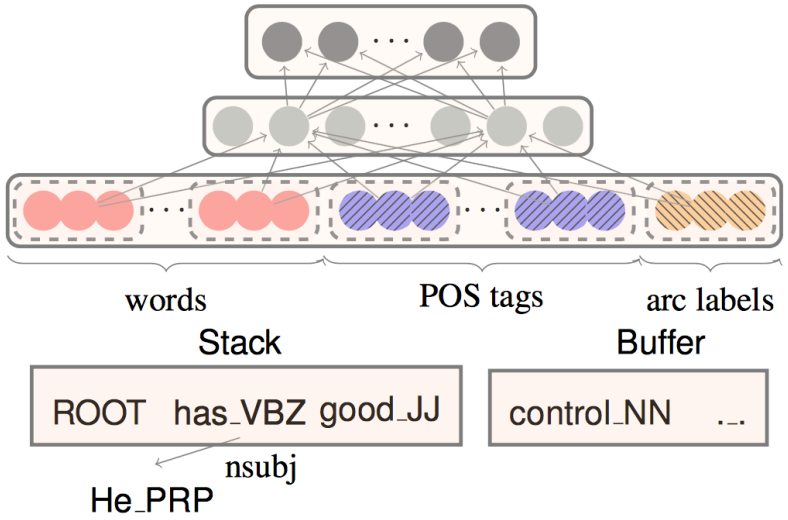
Neural Shift-Reduce Parsing

Representation for configuration:

- Embeddings for words/POS tags on top of stack
- Embeddings for words/POS tags at front of buffer
- Embeddings for existing arc labels

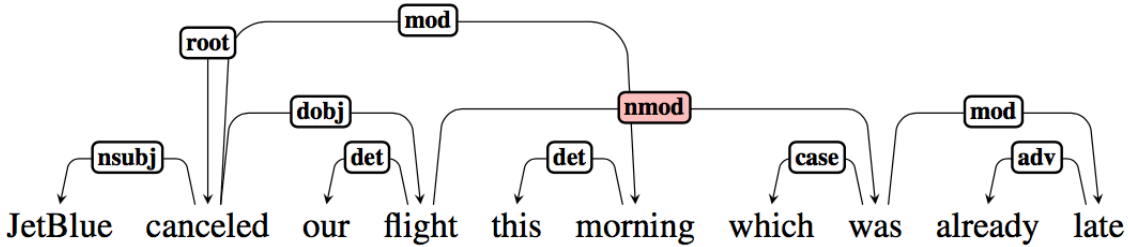
Classifier:

- Feed-forward neural network (input representation has a fixed dimensionality)



Chen and Manning (2014), "A Fast and Accurate Dependency Parser using Neural Networks"

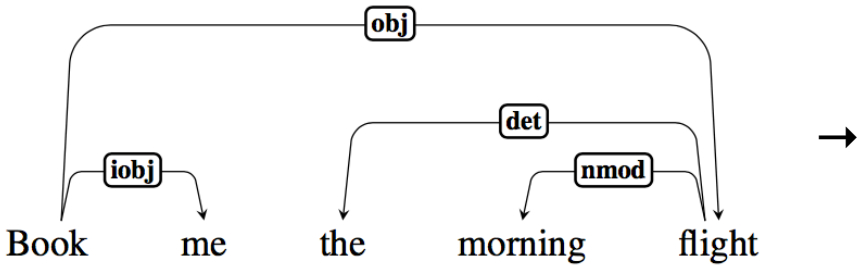
Training data



Our training data comes from treebanks (native dependency syntax or converted to dependency trees).

Oracle

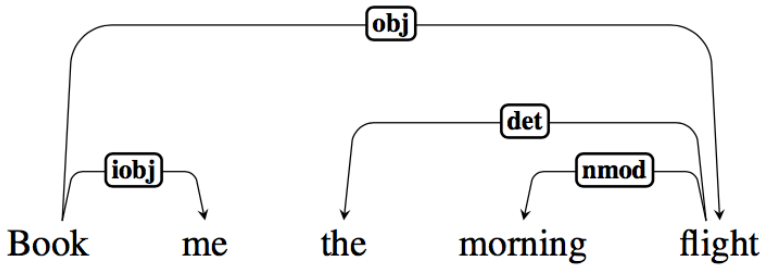
- An algorithm for converting a gold-standard dependency tree into a series of actions a transition-based parser should follow to yield the tree.



→

Configuration features	Label
<stack1 = “”>, <stack2 = “”>, <stack1 POS = “”>, <buffer1 = ∅>,	Shift
<stack1 = ∅>, <stack2 = “”>, <stack1 POS = ∅>, <buffer1 = book>,	Shift
<stack1 = book>, <stack2 =∅>, <stack1 POS = VB>, <buffer1 = me>,	Shift

This is our parse



arc

iobj(book, me)

nmod(flight, morning)

det(flight, the)

obj(book, flight)

root(ø, book)

∅ book me the morning flight

stack	action	gold tree
		iobj(book, me)
		nmod(flight, morning)
		det(flight, the)
		obj(book, flight)
		root(∅, book)

ø book me the morning flight

stack	action	gold tree
	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	*iobj(book, me)*
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*nmod(flight, morning)*
	Else shift: Remove word from front of input buffer and push it onto stack	*det(flight, the)*
		obj(book, flight)
		root(ø, book)

root(∅, book) exists but book has dependents in gold tree!

book me the morning flight

stack	action	gold tree
	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	*iobj(book, me)*
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*nmod(flight, morning)*
	Else shift: Remove word from front of input buffer and push it onto stack	*det(flight, the)*
		obj(book, flight)
∅		*root(∅, book)*

iobj(book, me) exists and me has no dependents in gold tree

me the morning flight

stack	action	gold tree
	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	*iobj(book, me)*
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*nmod(flight, morning)*
book	Else shift: Remove word from front of input buffer and push it onto stack	*det(flight, the)*
		obj(book, flight)
∅		*root(∅, book)*

the morning flight

stack	action	gold tree
me book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

morning flight

stack	action	gold tree
the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

flight

stack	action	gold tree
morning the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

nmod(flight,morning)

stack	action	gold tree
flight morning the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* ✅ *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

det(flight,the)

stack	action	gold tree
flight	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
		✅ *nmod(flight, morning)*
the	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	✅ *det(flight, the)*
book		*obj(book, flight)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

obj(book,flight)

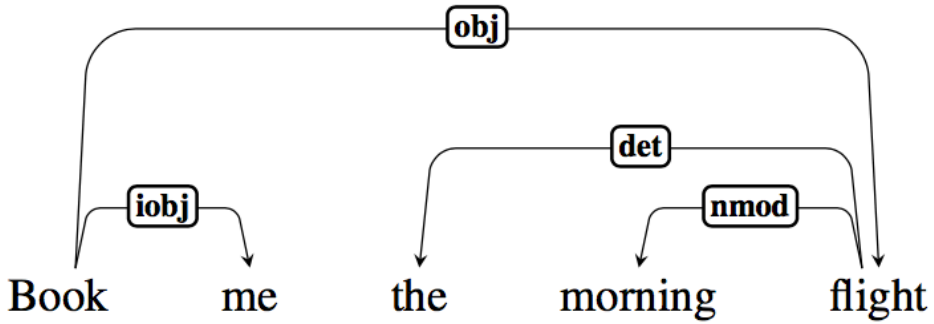
stack	action	gold tree
flight	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
		✅ *nmod(flight, morning)*
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	✅ *det(flight, the)*
book		✅ *obj(book, flight)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

root(∅, book) *and* book has no more dependents we haven't seen

stack	action	gold tree
book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* ✅ *nmod(flight, morning)* ✅ *det(flight, the)* ✅ *obj(book, flight)* ✅ *root(∅, book)*

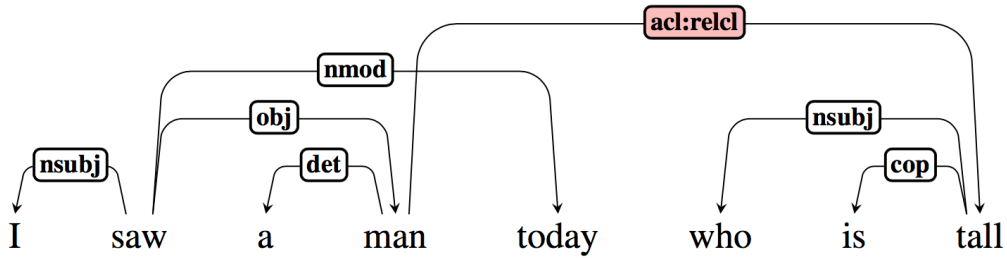
With only ∅ left on the stack and nothing in the buffer, we're done

stack	action	gold tree
∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* ✅ *nmod(flight, morning)* ✅ *det(flight, the)* ✅ *obj(book, flight)* ✅ *root(∅, book)*



Shift
Shift
Shift
RightArc(iobj)
Shift
Shift
Shift
LeftArc(nmod)
LeftArc(det)
RightArc(obj)
RightArc(root)

Projectivity



- What happens if you run an oracle on a sentence with a non-projective parse tree?

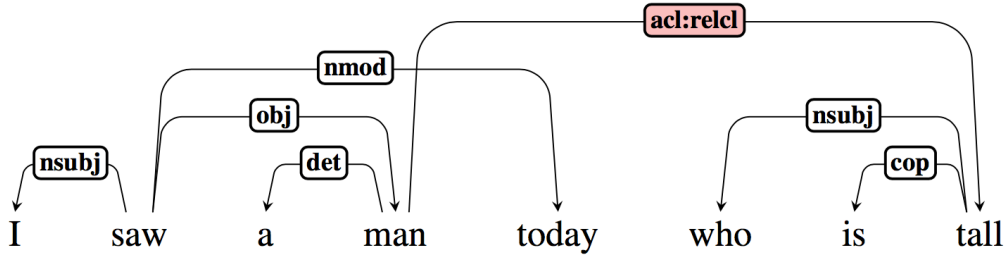
Graph-based parsing

- For a given sentence S, we want to find the highest-scoring tree among all possible trees for that sentence $\mathcal{G}_S$

$$\hat{T}(S) = \arg\max_{t \in \mathcal{G}_S} \text{score}(t, S)$$

- Edge-factored scoring: the total score of a tree is the sum of the scores for all of its edges (arcs):

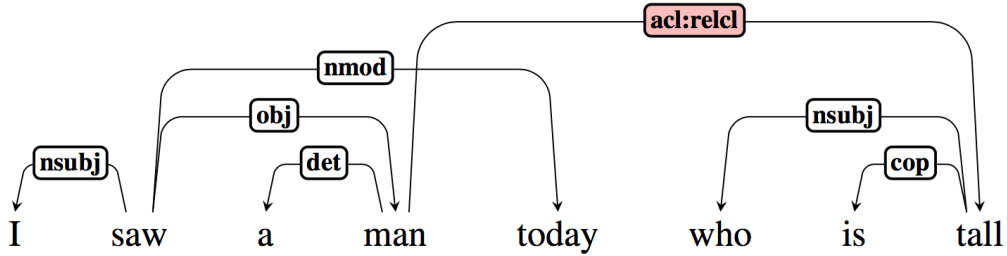
$$\text{score}(t, S) = \sum_{e \in t} \text{score}(e)$$



Edge-factored features

- Word form of head/dependent
- POS tag of head/dependent
- Distributed representation of h/d
- Distance between h/d
- POS tags between h/d
- Head to left of dependent?

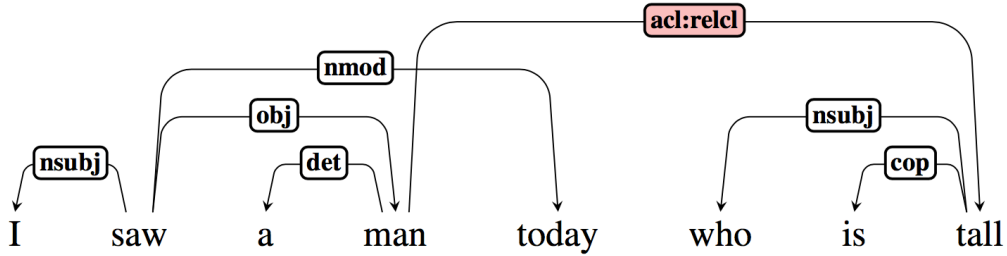
$head_t$ = man	1
$child_t$ = tall	1
distance	4
$child_{pos}$ = JJ and $head_{pos}$ = NN	1
$child_{pos}$ = NN and $head_{pos}$ = JJ	0



$$\text{score}(e) = \sum_{i=1}^{F} x_i \beta_i$$

Feature value

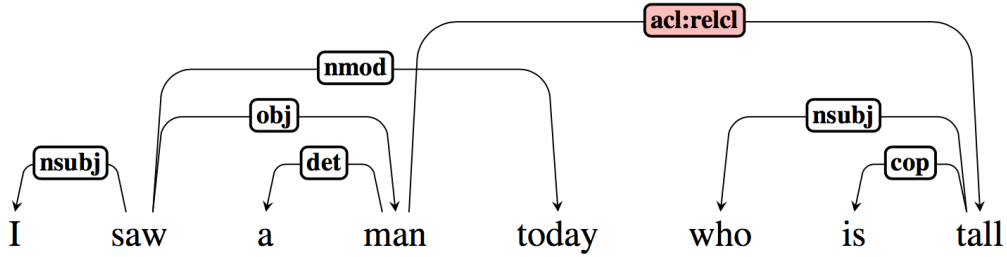
Learned coefficient for that feature

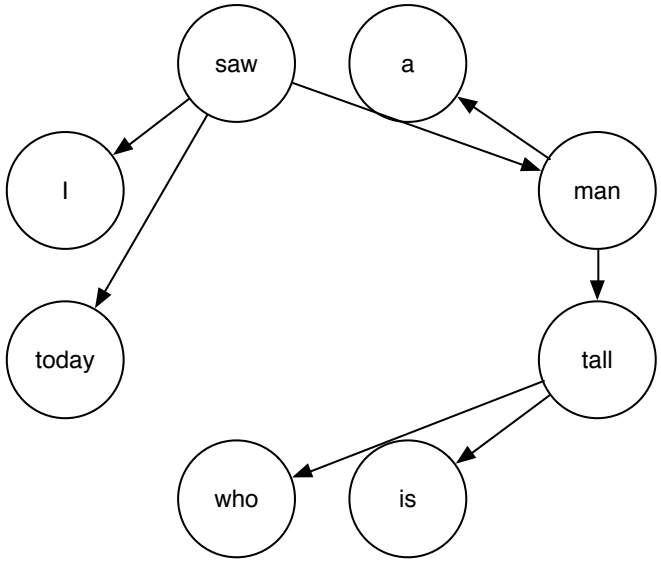


$$\text{score}(e) = \sum_{i=1}^{F} x_i \beta_i$$

	x	β
head_t = man	1	3.7
child_t = tall	1	1.3
distance	4	0.7
child_{pos} = JJ and head_{pos} = NN	1	0.3
child_{pos} = NN and head_{pos} = JJ	0	-2.7

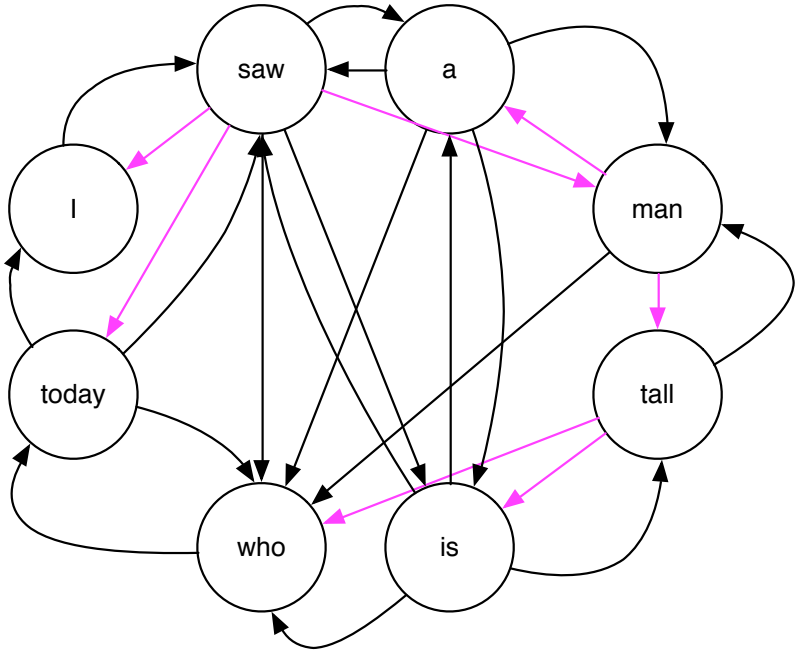
$$\text{score}(e) = 8.1$$


acl:relcl
nmod
obj
nsubj
det
nsubj
cop
I
saw
a
man
today
who
is
tall


saw
a
I
man
today
tall
who
is

MST Parsing

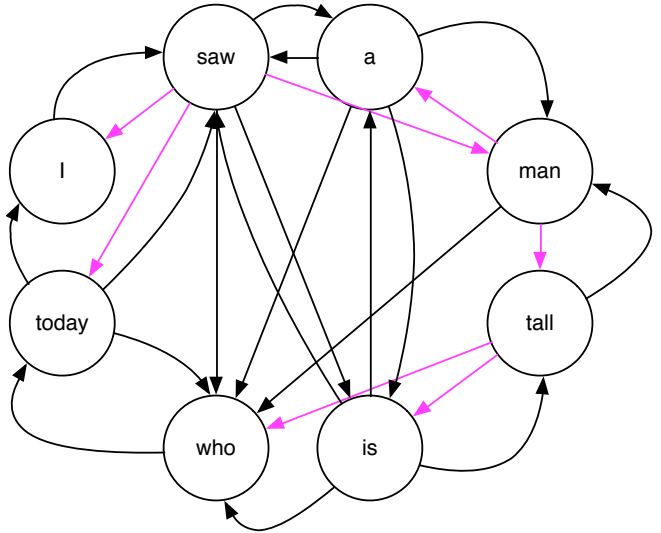
- We start out with a fully connected graph with a score for each edge
- N^2 edges total



(Assume one edge connects each node as dependent and node as head, N^2 total)

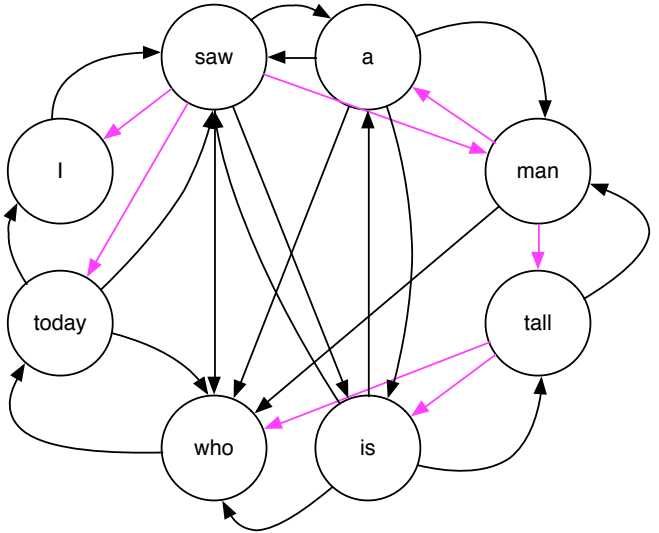
MST Parsing

- From this graph G, we want to find a spanning tree (tree that spans G [includes all the vertices in G])
- If the edges have weights, the best parse is the maximal spanning tree (the spanning tree with the highest total weight).



MST Parsing

- To find the MST of any graph, we can use the Chu-Liu-Edmonds algorithm in $O(n^3)$ time.
- More efficient Gabow et al. find the MST in $O(n^2+n \log n)$



Learning

ϕ is our feature vector scoped over the source dependent, target head and entire sentence x

$$\hat{T}(S) = \arg\max_{t \in \mathcal{G}_S} \text{score}(t, S)$$

both are vectors

$$\hat{T}(S) = \arg\max_{t \in \mathcal{G}_S} \sum_{e \in E} \phi(e, x)^\top \beta$$

$$\hat{T}(S) = \arg\max_{t \in \mathcal{G}_S} \left[\sum_{e \in E} \phi(e, x) \right]^\top \beta$$

Learning

$$\hat{T}(S) = \arg\max_{t \in \mathcal{G}_S} \text{score}(t, S)$$

- Given this formulation, we want to learn weights for β that make the score for the gold tree higher than for all other possible trees.
- That's expensive, so let's just try to make the score for the gold tree higher than the single best tree we predict (if it's wrong)

Learning

$$\left[\sum_{e \in E} \phi(e, x)\right]^\top \beta = \Phi_{gold}(E, x)^\top \beta$$

score for gold tree in treebank

$$\left[\sum_{e \in \hat{E}} \phi(e, x)\right]^\top \beta = \hat{\Phi}_{pred}\left(\hat{E}, x\right)^\top \beta$$

score for argmax tree in our model

Learning

- We can optimize this using SGD by taking the derivative with respect to beta.

$$\Phi_{gold}(E, x)^\top \beta - \hat{\Phi}_{pred}(\hat{E}, x)^\top \beta$$

$$= \left[\Phi_{gold}(E, x) - \hat{\Phi}_{pred}(\hat{E}, x)\right]^\top \beta$$

$$\frac{\partial}{\partial \beta}\left[\Phi_{gold}(E, x) - \hat{\Phi}_{pred}(\hat{E}, x)\right]^\top \beta = \Phi_{gold}(E, x) - \hat{\Phi}_{pred}(\hat{E}, x)$$

Structured Perceptron

Algorithm 1 Structured perceptron

1: **function** PERCEPTRONUPDATE(x, E, β)
2: $\Phi_{gold}(E, x) \leftarrow$ createFeatures(x, E)
3: $\hat{E} \leftarrow$ CLE (x, β)
4: $\hat{\Phi}_{pred}(\hat{E}, x) \leftarrow$ createFeatures(x, $\hat{E}$)
5: $\beta \leftarrow \beta + \Phi_{gold}(E, x) - \hat{\Phi}_{pred}(\hat{E}, x)$
6: **end function**

Create feature vector from true tree

Use CLE to find best tree given current β

Create feature vector from best predicted tree

Update β with the difference between the feature vectors