Natural Language Parsing

Two views of syntactic structure

1. The linguistic structure of sentences – two views: Constituency = phrase structure grammar = context-free grammars (CFGs)

Phrase structure organizes words into nested constituents

Starting unit: words

the, cat, cuddly, by, door

Words combine into phrases

the cuddly cat, by the door

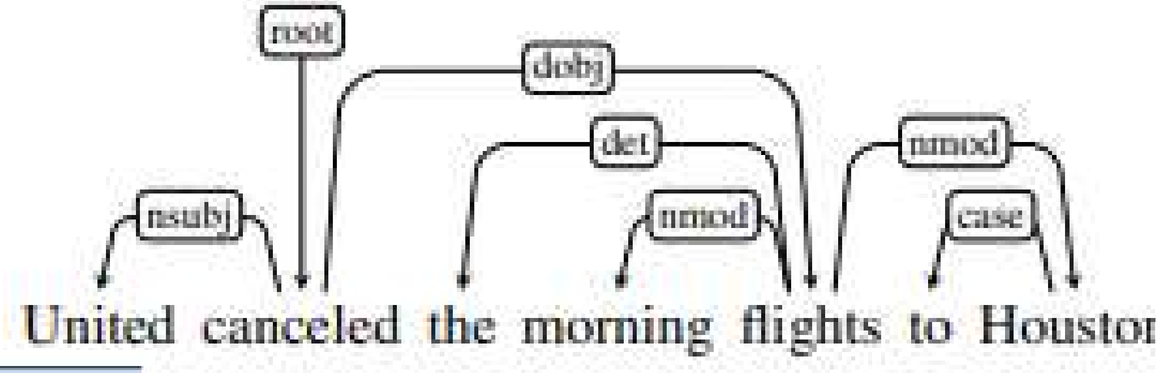
Phrases can combine into bigger phrases

the cuddly cat by the door

head word of a constituent was the central organizing word of a larger constituent (e.g, the primary noun in a noun phrase, or verb in a verb phrase).
Head→dependent

Two views of linguistic structure: Dependency structure

- Dependency structure shows which words depend on (modify, attach to, or are arguments of) which other words.



Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Figure 14.2 Some of the Universal Dependency relations (de Marneffe et al., 2014).

Why do we need sentence structure?

Humans communicate complex ideas by composing words together into bigger units to convey complex meanings

Listeners need to work out what modifies [attaches to] what

A model needs to understand sentence structure in order to be able to interpret language correctly

Prepositional phrase attachment ambiguity


San Jose cops kill man with knife

Text Paper Translate Listen Close

San Jose cops kill man with knife


BBC Sign in News Sport Weather Shop Reel Travel

NEWS

Home Video World US & Canada UK Business Tech Science Stories

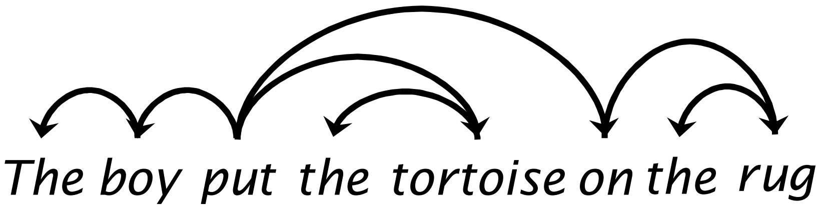
Science & Environment

Scientists count whales from space

By Jonathan Amos
BBC Science Correspondent

Two views of linguistic structure:
2. Dependency structure

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

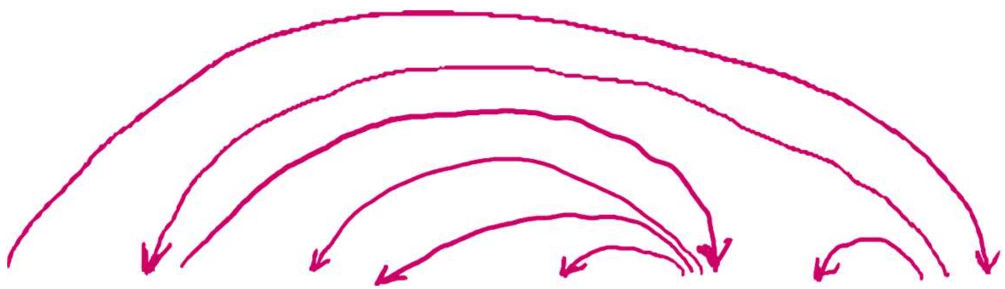


The boy put the tortoise on the rug

Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations (“arrows”) called dependencies

The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc.)



ROOT Discussion of the outstanding issues was completed .

- Some people draw the arrows one way; some the other way!
  - from head to dependent – we follow that convention
- We usually add a fake ROOT so every word is a dependent of precisely 1 other node

The rise of annotated data & Universal Dependencies treebanks

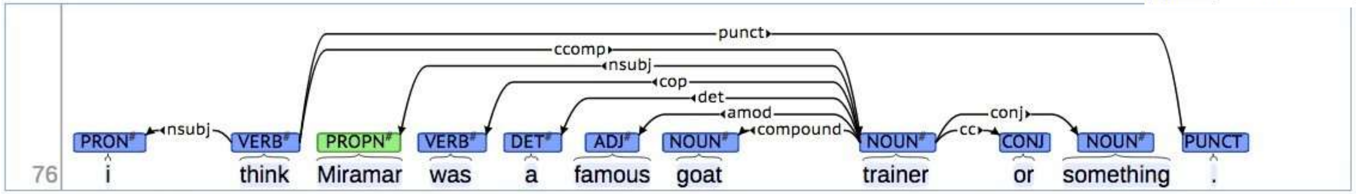
Brown corpus (1967; PoS tagged 1979); Lancaster-IBM Treebank (starting late 1980s); Marcus et al. 1993, The Penn Treebank, *Computational Linguistics*;

Universal Dependencies: http://universaldependencies.org/ inventory of dependency relations that are linguistically motivated, computationally useful, and cross-linguistically applicable.

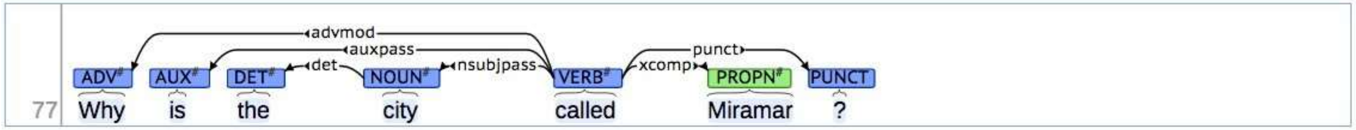
Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Figure 14.2 Some of the Universal Dependency relations (de Marneffe et al., 2014).

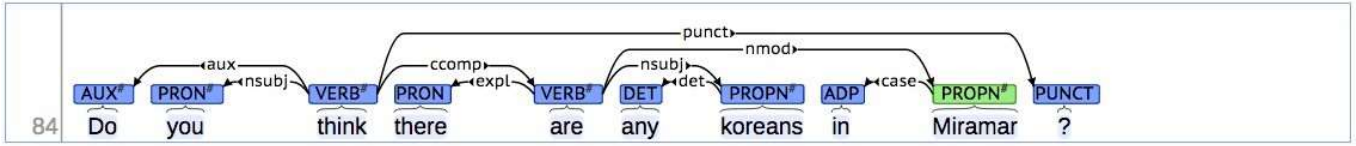
[context] [conllu]



[context] [conllu]



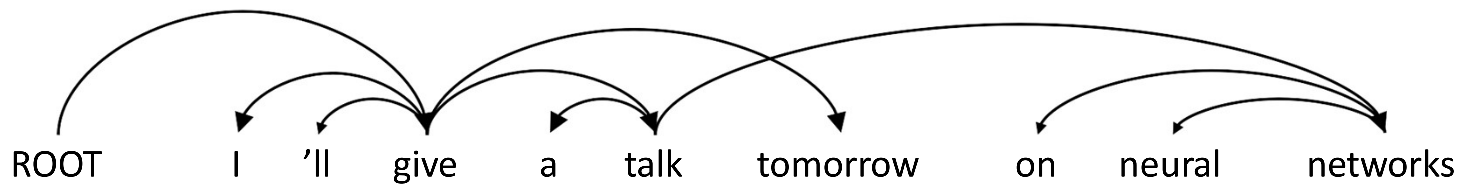
[context] [conllu]



Dependency Parsing

- A sentence is parsed by choosing for each word what other word (including ROOT) it is a dependent of

- Usually some constraints:
  - Only one word is a dependent of ROOT
  - Don’t want cycles A → B, B → A
- This makes the dependencies a tree
- Final issue is whether arrows can cross (be non-projective) or not



MaltParser
[Nivre et al. 2008]

- A simple form of greedy discriminative dependency parser
- The parser does a sequence of bottom up actions
  - Roughly like “shift” or “reduce” in a shift-reduce parser, but the “reduce” actions are specialized to create dependencies with head on left or right
- The parser has:
  - a stack σ, written with top to the right
    - which starts with the ROOT symbol
  - a buffer β, written with top to the left
    - which starts with the input sentence
  - a set of dependency arcs A
    - which starts off empty
  - a set of actions

Basic transition-based dependency parser

Start: σ = [ROOT], $\beta = w_1, \ldots, w_n$, A = $\emptyset$

1. Shift $\qquad \sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
2. Left-Arc$_r$ $\qquad \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j,w_i)\}$
3. Right-Arc$_r$ $\qquad \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_i|\beta, A \cup \{r(w_i,w_j)\}$

Finish: $\beta = \emptyset$

- LEFTARC: Assert a head-dependent relation between the word at the top of the stack and the second word; remove the second word from the stack.
- RIGHTARC: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the top word from the stack;
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.

Notes:

- Unlike the regular presentation of the CFG reduce step, dependencies combine one thing from each of stack and buffer

Example

1. Left-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j,w_i)\}$
   Precondition: $(w_k, r', w_i) \notin A$, $w_i \neq$ ROOT
2. Right-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma|w_i|w_j, \beta, A \cup \{r(w_i,w_j)\}$
3. Reduce $\sigma|w_i, \beta, A \rightarrow \sigma, \beta, A$
   Precondition: $(w_k, r', w_i) \in A$
4. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$

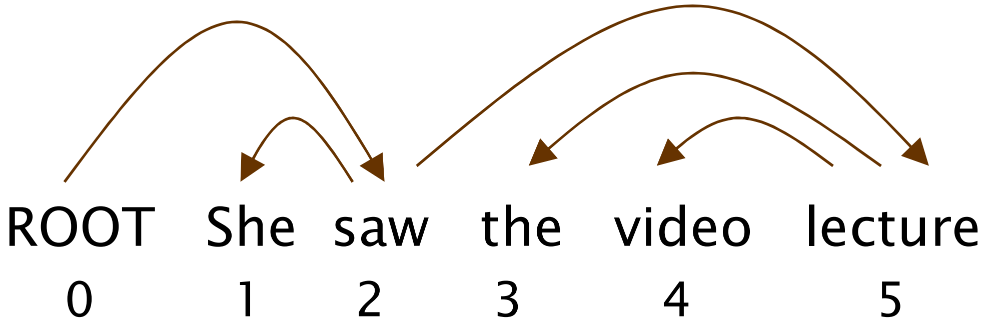
Happy children like to play with their friends .

	[ROOT]	[Happy, children, …]	∅
Shift	[ROOT, Happy]	[children, like, …]	∅
LA$_{amod}$	[ROOT]	[children, like, …]	{amod(children, happy)} = A_1
Shift	[ROOT, children]	[like, to, …]	A_1
LA$_{nsubj}$	[ROOT]	[like, to, …]	$A_1 \cup$ {nsubj(like, children)} = A_2
RA$_{root}$	[ROOT, like]	[to, play, …]	$A_2 \cup${root(ROOT, like) = A_3
Shift	[ROOT, like, to]	[play, with, …]	A_3
LA$_{aux}$	[ROOT, like]	[play, with, …]	$A_3 \cup${aux(play, to) = A_4
RA$_{xcomp}$	[ROOT, like, play]	[with their, …]	$A_4 \cup${xcomp(like, play) = A_5

MaltParser
[Nivre et al. 2008]

- We have left to explain how we choose the next action
- Each action is predicted by a discriminative classifier (often SVM, could be maxent classifier) over each legal move
  - 4 untyped choice; more when typed
  - Features: top of stack word, POS; first in buffer word, POS; etc.

Evaluation of Dependency Parsing: (labeled) dependency accuracy



$$\text{Acc} = \frac{\text{\# correct deps}}{\text{\# of deps}}$$

UAS = 4 / 5 = 80%
LAS = 2 / 5 = 40%

Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	dobj

Parsed

1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

Projectivity

- Dependencies from a CFG tree using heads, must be projective
- But dependency theory normally does allow non-projective structures to account for displaced constituents
  - You can't easily get the semantics of certain constructions right without these nonprojective dependencies

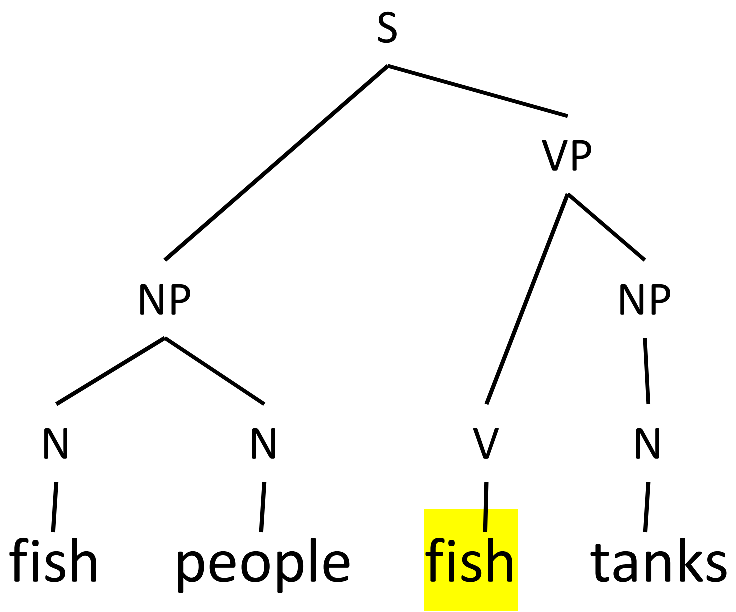
Context-Free Grammar (CFG); Chap 12/13

S $\rightarrow$ NP VP
VP $\rightarrow$ V NP
VP $\rightarrow$ V NP PP
NP $\rightarrow$ NP NP
NP $\rightarrow$ NP PP
NP $\rightarrow$ N
NP $\rightarrow$ *e*
PP $\rightarrow$ P NP

N $\rightarrow$ *people*
N $\rightarrow$ *fish*
N $\rightarrow$ *tanks*
N $\rightarrow$ *rods*
V $\rightarrow$ *people*
V $\rightarrow$ *fish*
V $\rightarrow$ *tanks*
P $\rightarrow$ *with*

people fish tanks

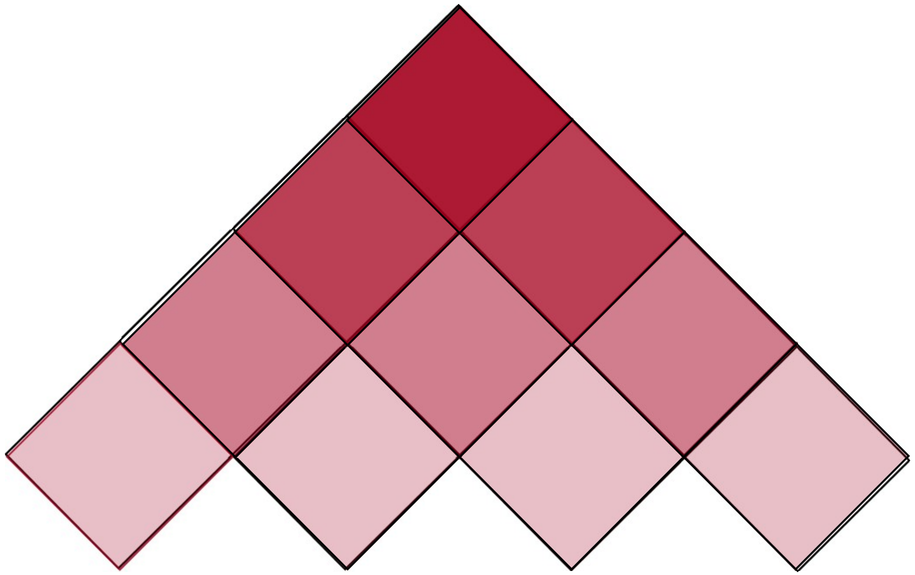
Probabilistic CFG



PCFG

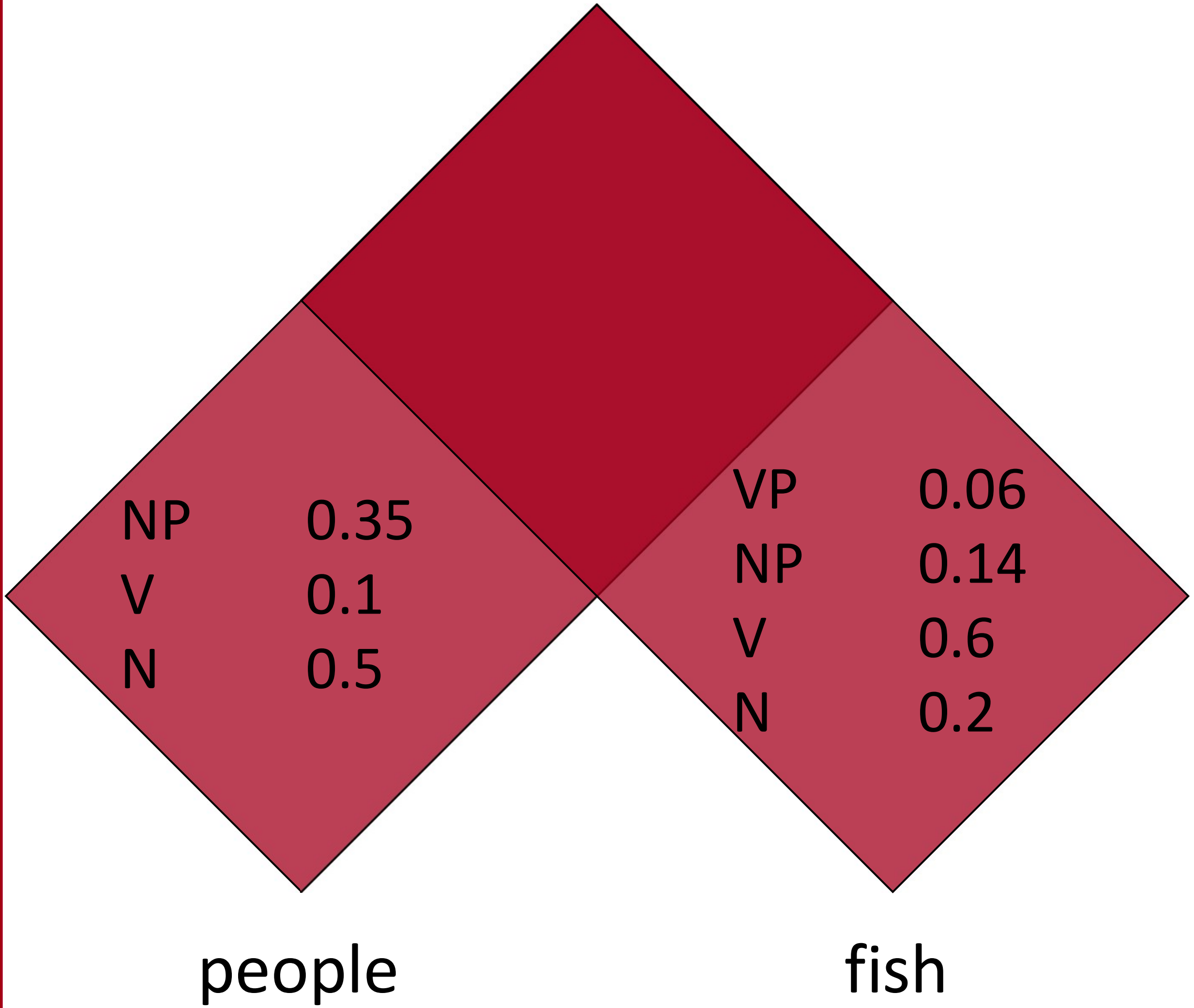
Rule	**Prob θ_i**
$S \rightarrow NP\ VP$	θ_0
$NP \rightarrow NP\ NP$	θ_1
…	
$N \rightarrow$ fish	θ_{42}
$N \rightarrow$ people	θ_{43}
$V \rightarrow$ fish	θ_{44}
…	

Cocke-Kasami-Younger (CKY) Constituency Parsing



fish people fish tanks

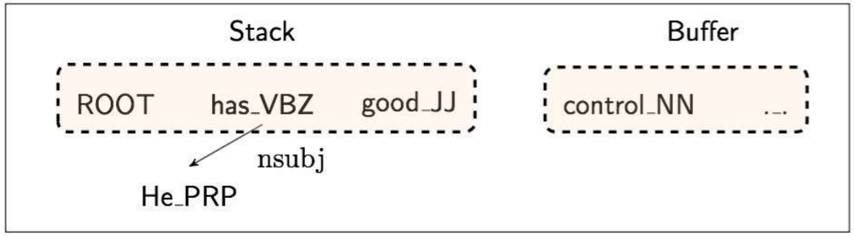
Viterbi (Max) Scores



S $\rightarrow$ NP VP	0.9
S $\rightarrow$ VP	0.1
VP $\rightarrow$ V NP	0.5
VP $\rightarrow$ V	0.1
VP $\rightarrow$ V @VP_V	0.3
VP $\rightarrow$ V PP	0.1
@VP_V $\rightarrow$ NP PP	1.0
NP $\rightarrow$ NP NP	0.1
NP $\rightarrow$ NP PP	0.2
NP $\rightarrow$ N	0.7
PP $\rightarrow$ P NP	1.0

36

Conventional Feature Representation



binary, sparse
dim =10^6–10^7

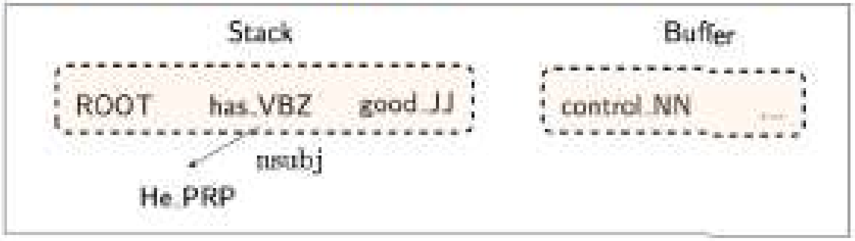
0	0	0	1	0	0	1	0	…	0	0	1	0

Feature templates: usually a combination of 1–3 elements from the configuration

Indicator features

$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$
$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

- We extract a set of tokens based on the stack / buffer positions:



	word		POS		dep.
s_1	good		JJ		Ø
s_2	has		VBZ		Ø
b_1	control		NN		Ø
$lc(s_1)$	Ø	+	Ø	+	Ø
$rc(s_1)$	Ø		Ø		Ø
$lc(s_2)$	He		PRP		nsubj
$rc(s_2)$	Ø		Ø		Ø

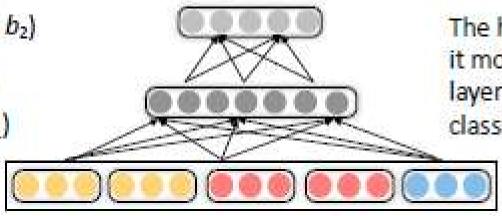
A concatenation of the vector representation of all these is the neural representation of a configuration

Softmax probabilities

Output layer y
$y = \text{softmax}(Uh + b_2)$

Hidden layer h
$h = \text{ReLU}(Wx + b_1)$

Input layer x



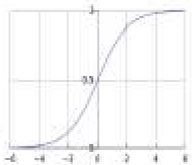
x is result of lookup
$x_{(i,\dots,i+d)} = Le$
lookup + concat

Log loss (cross-entropy error) will be back-propagated to the embeddings

The hidden layer re-represents the input — it moves inputs around in an intermediate layer vector space—so it can be easily classified with a (linear) softmax

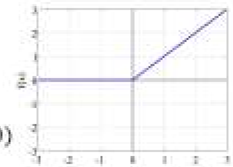
logistic = "sigmoid"

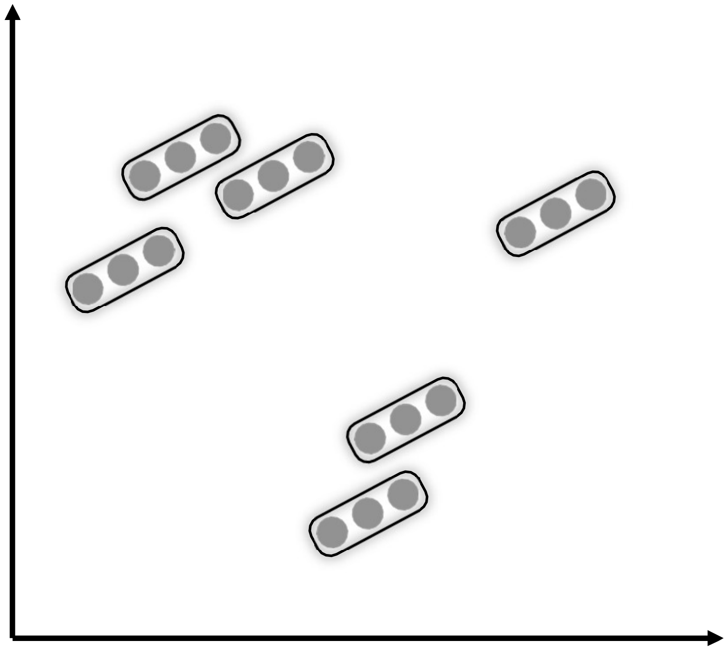
$$f(z) = \frac{1}{1 + \exp(-z)}.$$



ReLU = Rectified Linear Unit

$$\text{ReLU}(z) = \max(z, 0)$$





- Results on English parsing to Stanford Dependencies:
  - Unlabeled attachment score (UAS) = head
  - Labeled attachment score (LAS) = head and label

Parser	UAS	LAS	sent. / s
MaltParser	89.8	87.2	469
MSTParser	91.4	88.1	10
TurboParser	**92.3**	89.6	8
C & M 2014	92.0	**89.7**	**654**

- https://aclanthology.org/D14-1082/