Parsing

2/2565: FRA501 Introduction to Natural Language Processing with Deep learning

Week 07

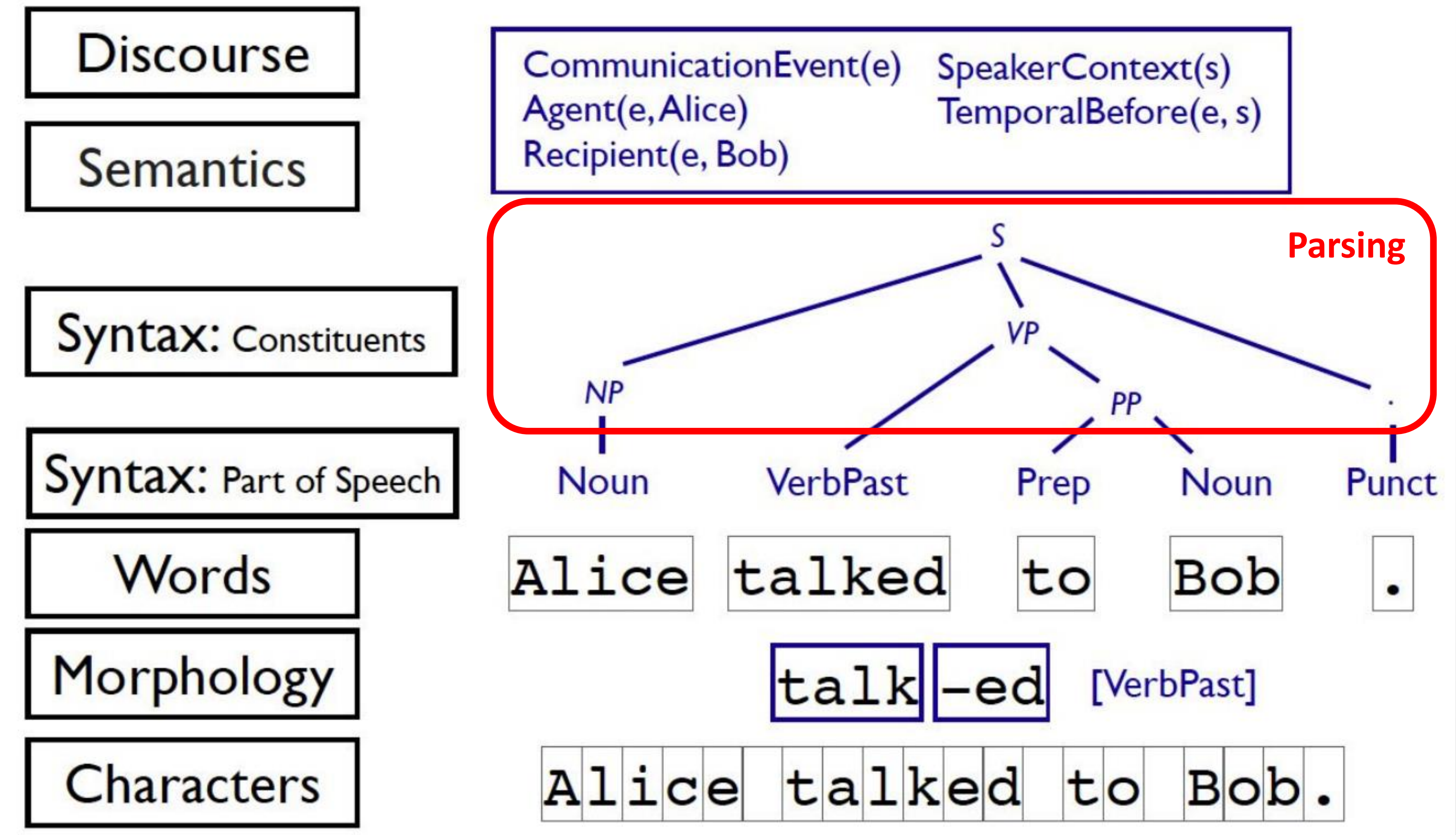
Paisit Khanarsa, Ph.D.

Institute of Field Robotics (FIBO), King Mongkut's University of Technology Thonburi

Outlines

- Introduction to parsing
- Types of grammars
  - Context free grammar
  - Probabilistic context free grammar
    - CYK parser
  - Dependency Grammar
    - Transition-based parsing
    - Recursive neural networks
- Parsing evaluation

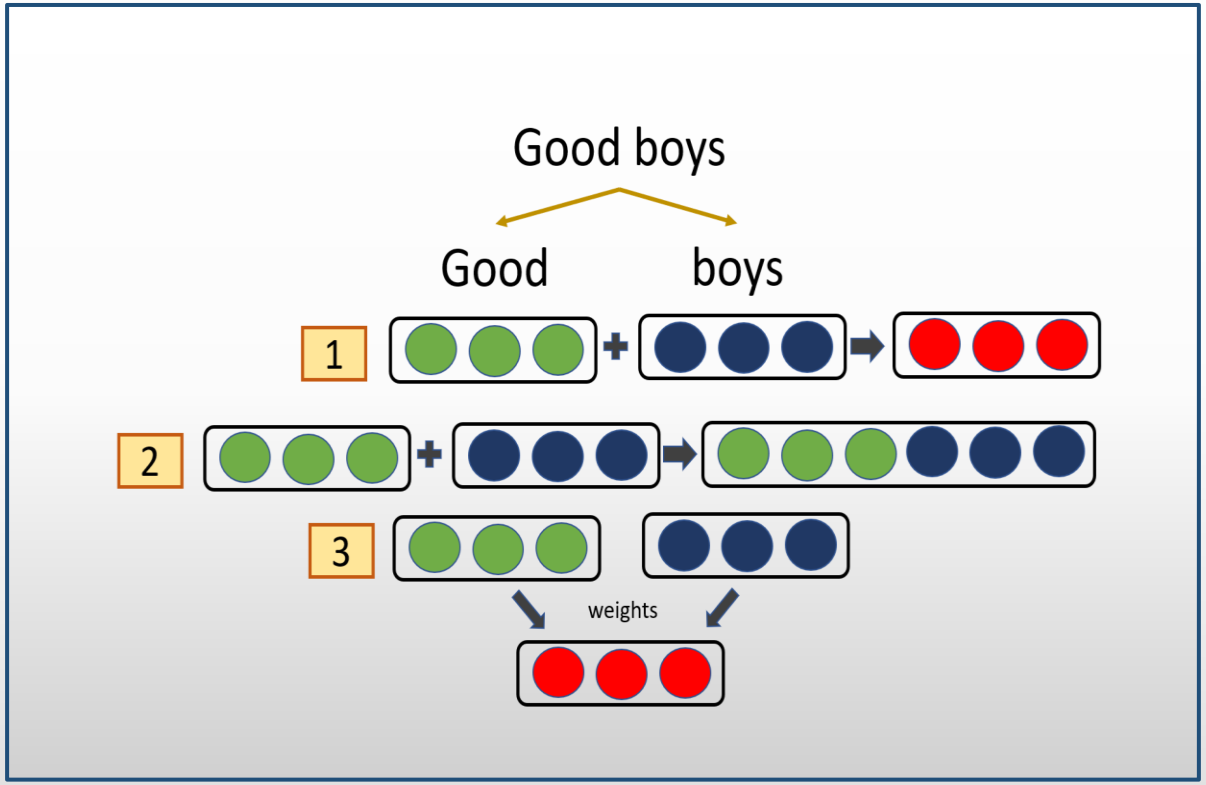
Introduction to parsing



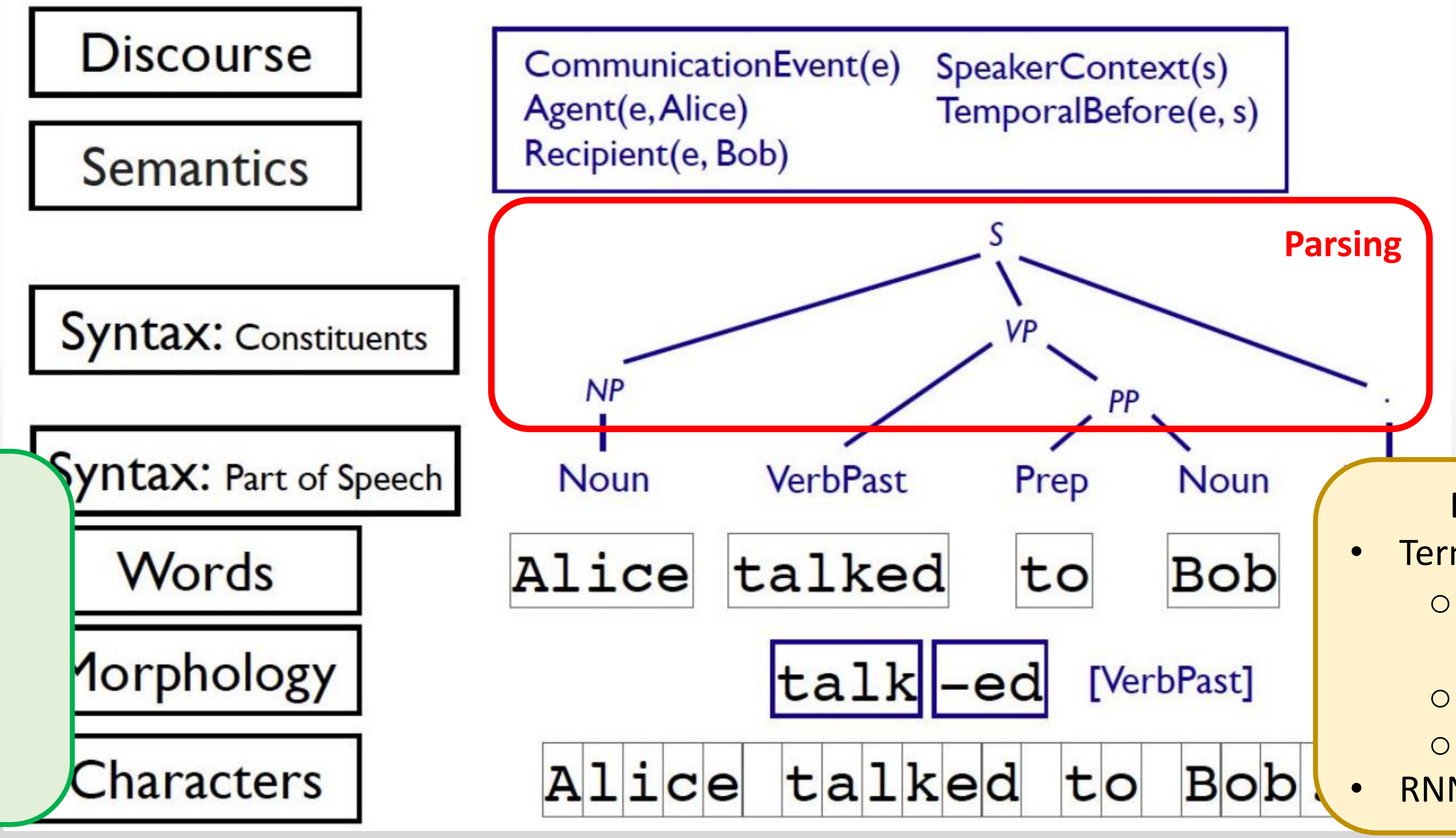
https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_natural_language_processing.htm

Introduction to parsing (cont.)

- Compositionality
- Can create a dense vector representation for a word (e.g. phrase, sentence)
- A larger unit can be created by combining smaller ones.



Introduction to parsing



Word vector

- Co-occurrence
- PPMI
- TFIDF
- Word2Vec
  - CBOW
  - Skip-gram

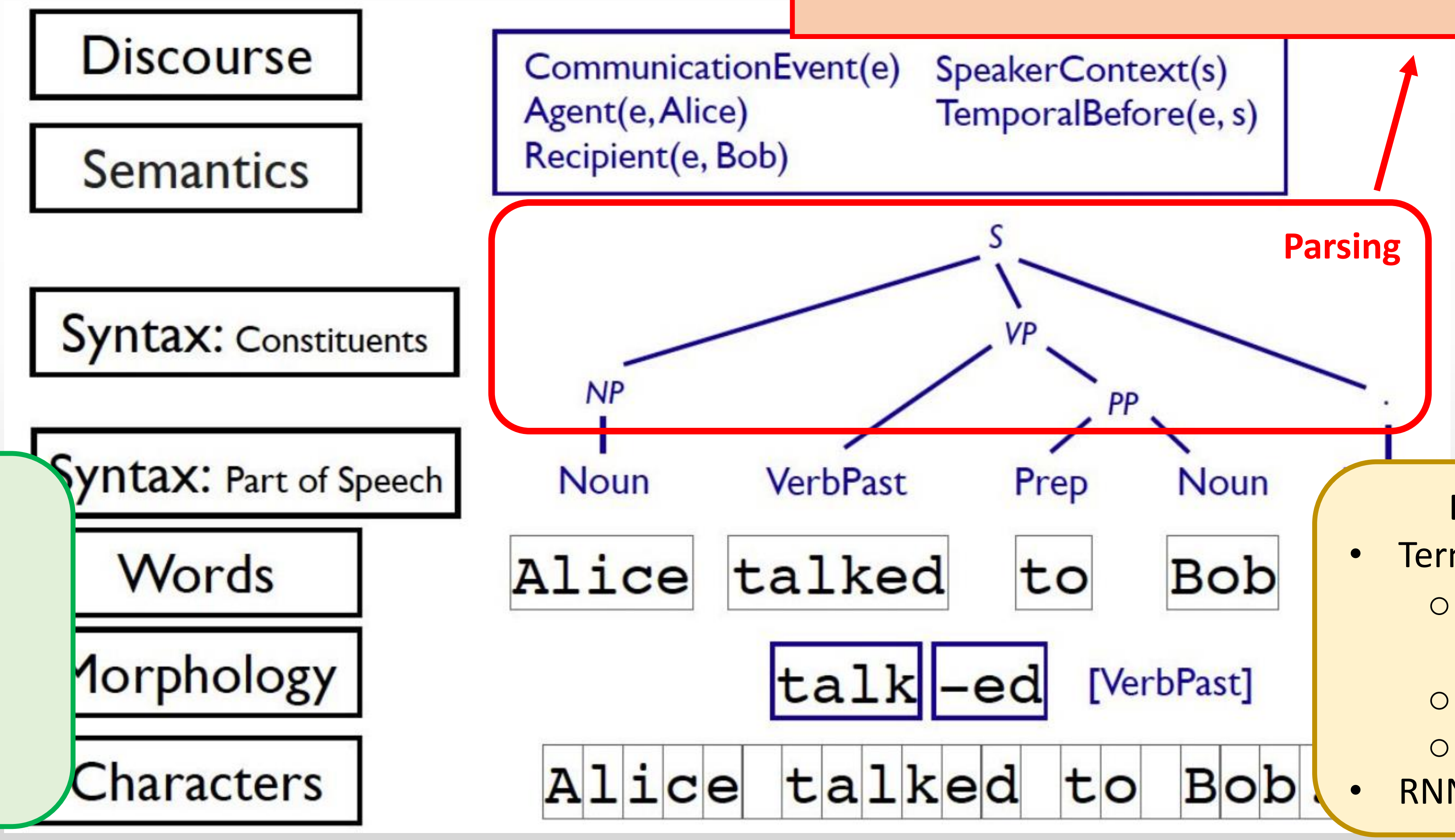
Doc vector

- Term document
  - Bag of words model
  - Doc2Vec
  - LDA2Vec
- RNN

https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_natural_language_processing.htm

Introduction to parsing

In this level, how do we represent things? Keeping word order in mind



Word vector

- Co-occurrence
- PPMI
- TFIDF
- Word2Vec
  - CBOW
  - Skip-gram

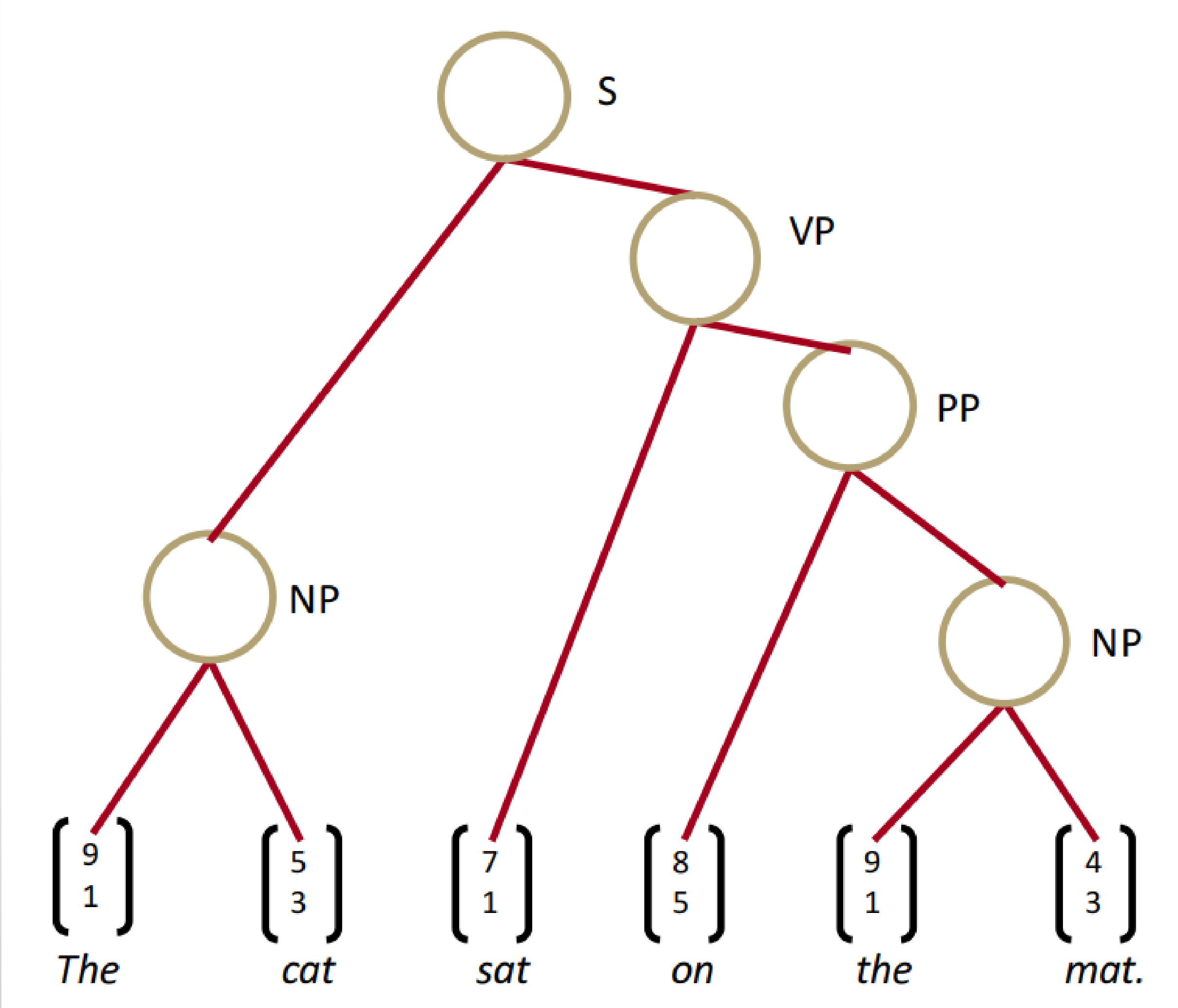
Doc vector

- Term document
  - Bag of words model
  - Doc2Vec
  - LDA2Vec
- RNN

https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_natural_language_processing.htm

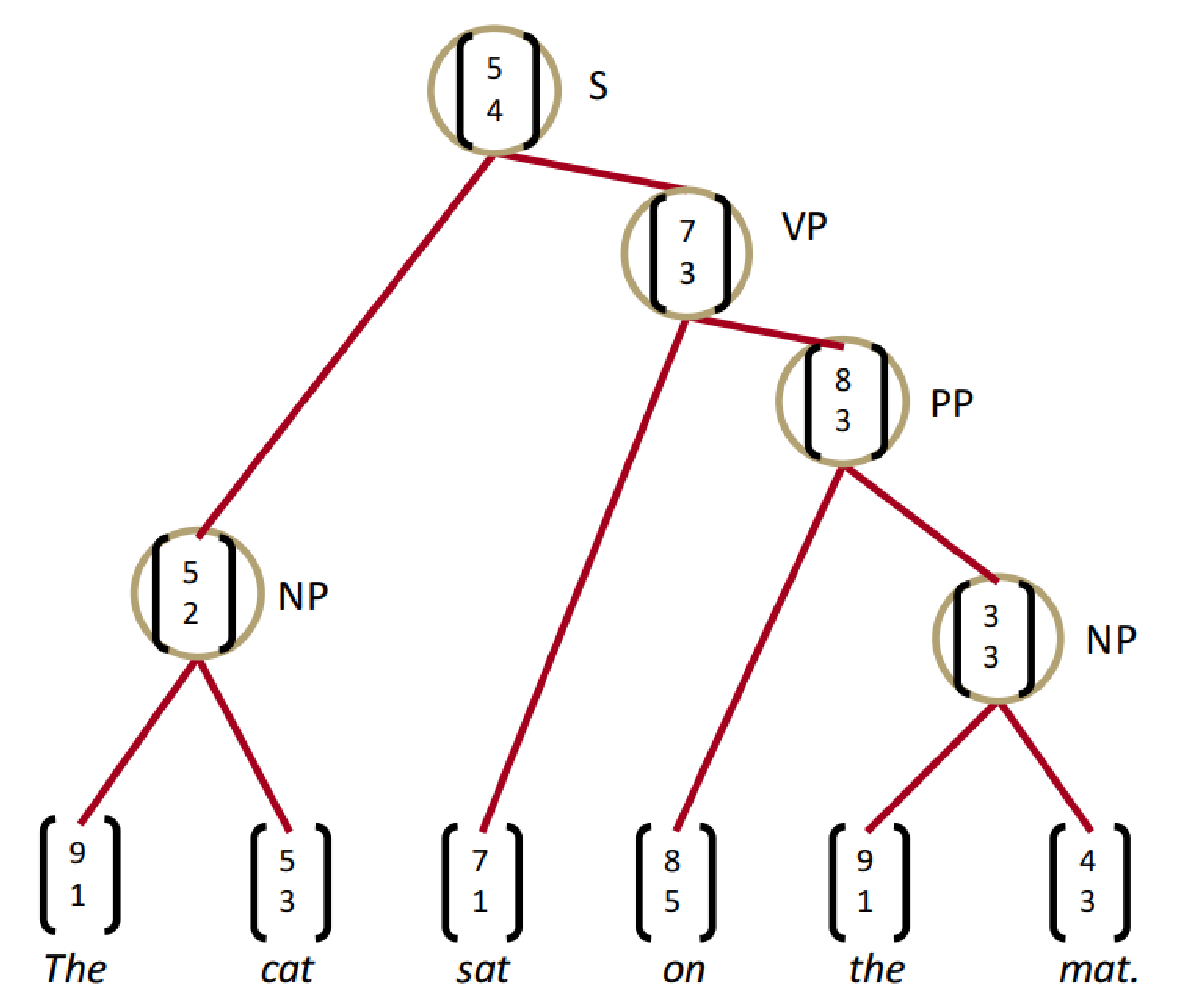
Introduction to parsing (cont.)

- Sentence Parsing



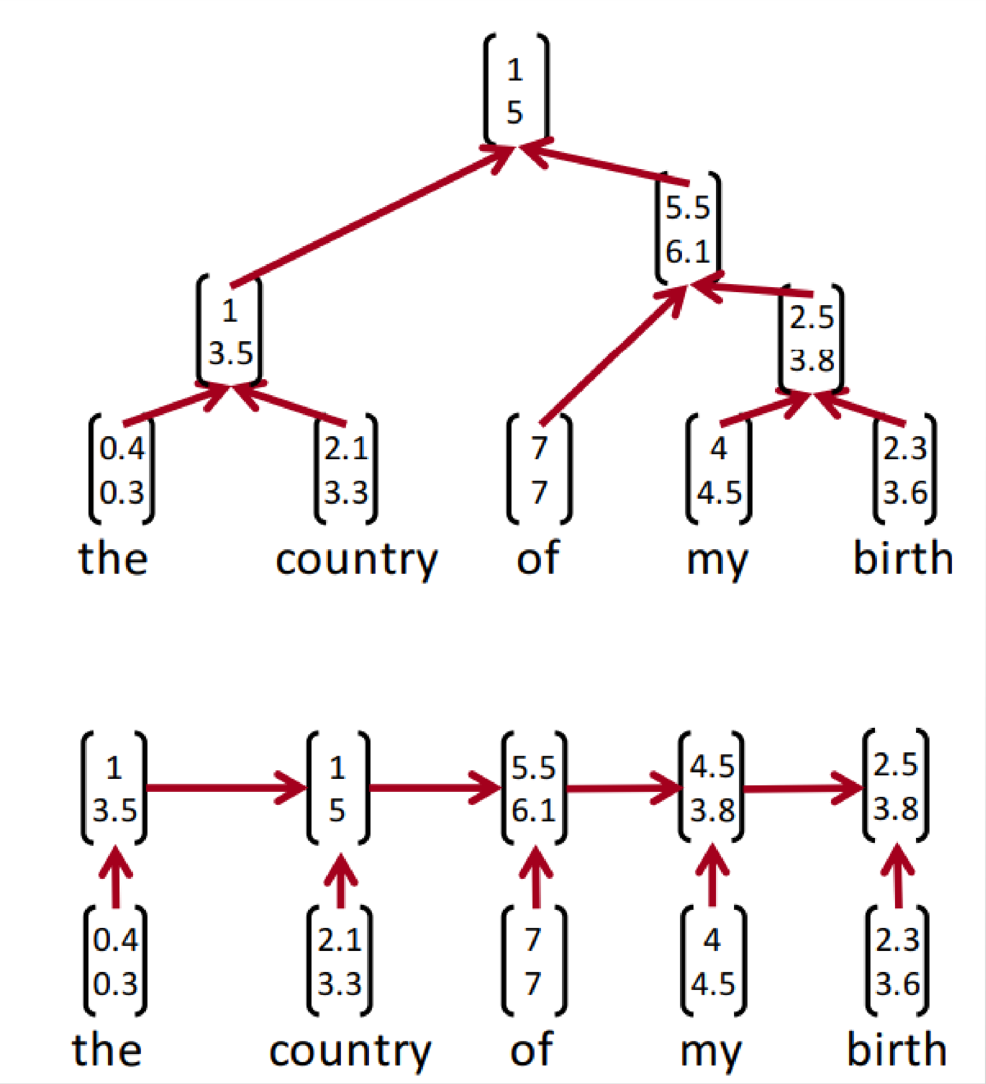
Introduction to parsing (cont.)

- Sentence Parsing



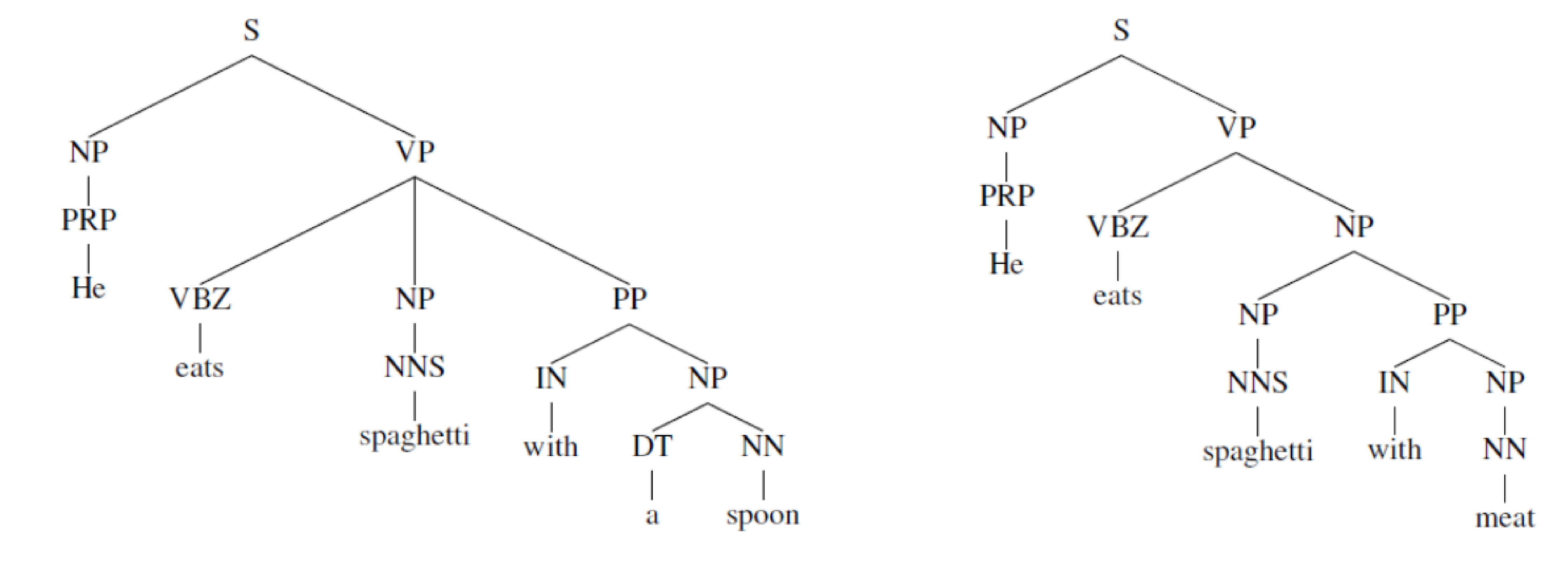
Introduction to parsing (cont.)

- Recursive vs recurrent neural networks



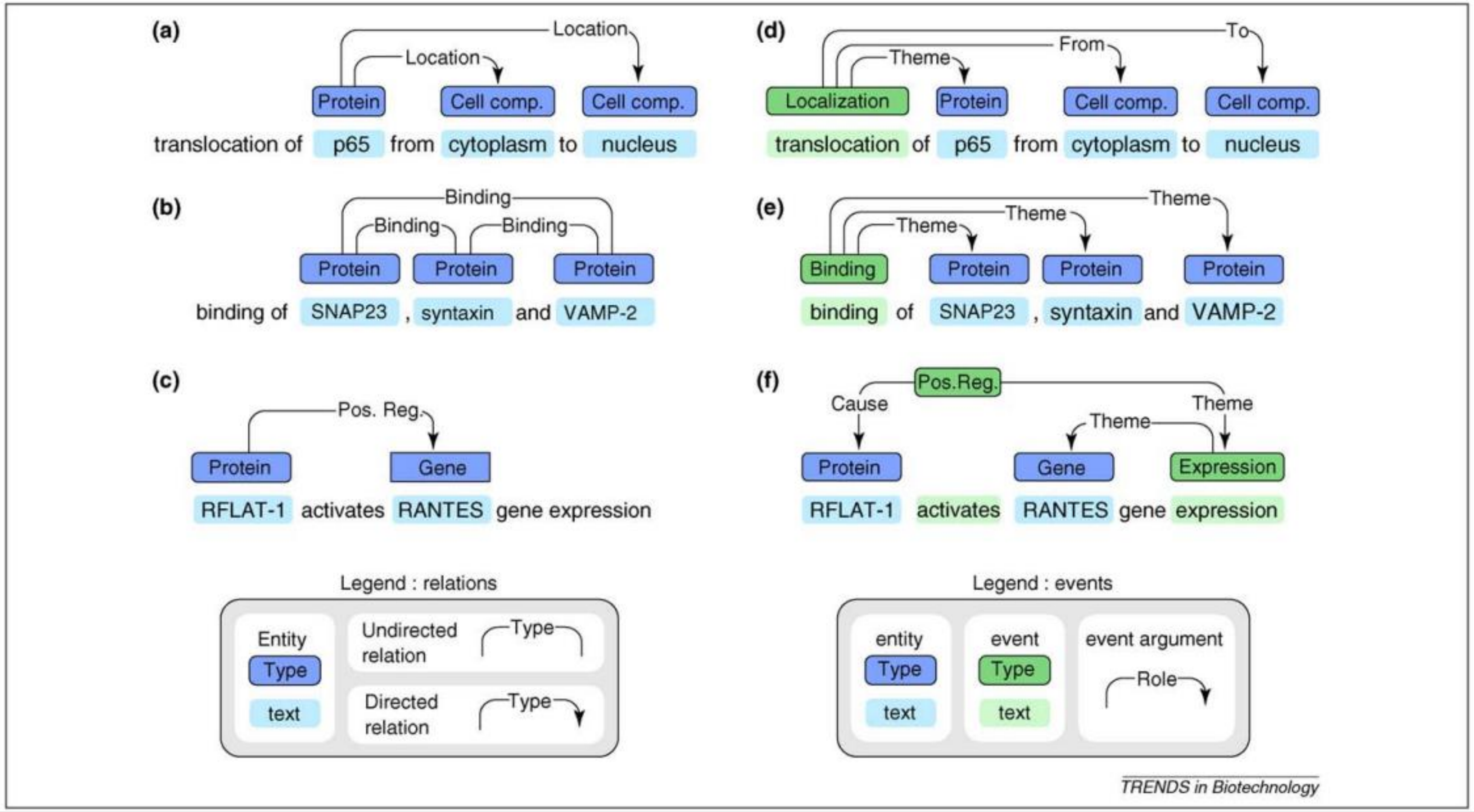
Introduction to parsing (cont.)

- What situations might recursion be preferable in?



Introduction to parsing (cont.)

- Application of parse trees: info extraction



Ananiadou, S., Pyysalo, S., Tsujii, J., & Kell, D. B. (2010). Event extraction for systems biology by text mining the literature. Trends in Biotechnology, 28(7), 381–390.

Types of grammars: Context-Free Grammar (CFG)

- Constituents
  - Groups of words behaving as a single units
    - Ex: Noun phrase
    - Harry the Horse
    - The reason he comes into the house
    - They
    - A high-class spot such as Mindy's
  - Checking for constituents
    - See if they can appear in similar syntactic environments
      - They sit…
      - The reason he comes into the house is…

Types of grammars: Context-Free Grammar (CFG)

- A grammar specifies what kind of parse tree can be generated.
- CFG or Phrase-Structure Grammars assumes the grammar is context-free
  - Most forms of natural language are context-free
    - Thai and English are typically CFG languages
  - Used in many programming languages
- CFG is based on constituent structures

Types of grammars: CFG (cont.)

Note
S: Sentence
VP: Verb phase
NP: Noun phase
PP: prepositional

DT: determiner
Vi: intransitive verb
Vt: transitive verb
NN: noun
IN: preposition

- N = {S, NP, VP, PP, DT, Vi, Vt, NN, IN}: Nontermianals
- S = S : starting symbol
- Σ = {sleeps, saw, man, woman, telescope, the, with, in}: Terminals
- R =

S → NP VP
VP → Vi VP → Vt NP VP → VP PP
NP → DT NN NP → NP PP
PP → IN NP

Vi → sleeps Vt → saw
NN → man NN → woman NN → telescope
DT → the
IN → with IN → in

A CFG is defined by G = (N,S, Σ,R)
Production rules
$X \rightarrow Y_1, .. Y_n$
- Y can be terminal or nonterminal
- The rule only relies on X

Generation using left-most derivation strategy

R =

S → NP VP	Vi → sleeps Vt → saw
VP → Vi VP → Vt NP VP → VP PP	NN → man NN → woman NN → telescope
NP → DT NN NP → NP PP	DT → the
PP → IN NP	IN → with IN → in

- Recursive strategy

```
left_most_derivation(A):
    if terminal(A):
        return A
    rule = choose (rules(A))
    rhs = right_hand_side(rule)
    return concatenate([left_most_derivation(A') for A' in rhs])
```

- To generate we call left_most_derivation(S)
- A string belongs to the language of a CFG if there exist a sequence of left-most derivation that can generate the string

$$L = \{s \in \Sigma^* | s = left\_most\_derivation(S)\}$$

Generation using left-most derivation strategy

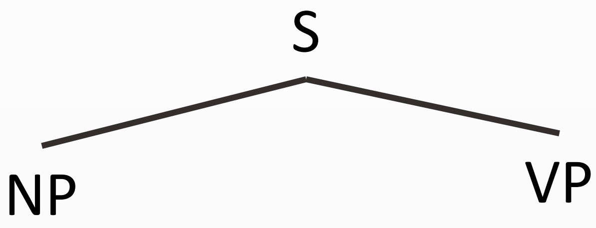
- Example: The woman saw the man with the telescope.

S → NP VP
VP → Vi VP → Vt NP VP → VP PP
NP → DT NN NP → NP PP
PP → IN NP

Vi → sleeps Vt → saw
NN → man NN → woman NN → telescope
DT → the
IN → with IN → in

Generation using left-most derivation strategy

- Example: The woman saw the man with the telescope.

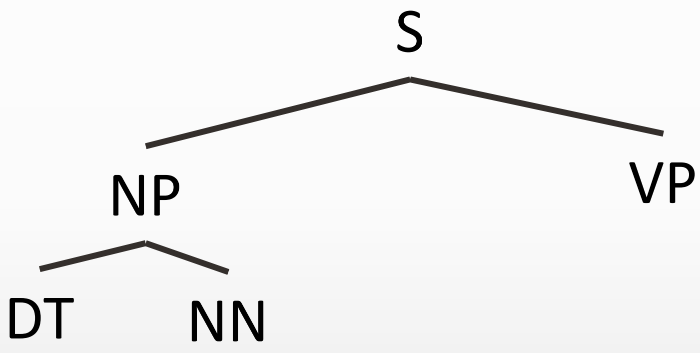


S → NP VP
VP → Vi VP → Vt NP VP → VP PP
NP → DT NN NP → NP PP
PP → IN NP

Vi → sleeps Vt → saw
NN → man NN → woman NN → telescope
DT → the
IN → with IN → in

Generation using left-most derivation strategy

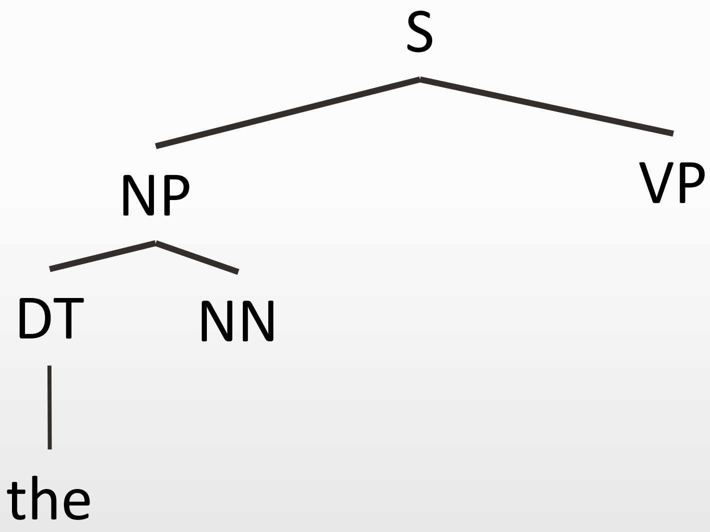
- Example: The woman saw the man with the telescope.



S → NP VP	Vi → sleeps Vt → saw
VP → Vi VP → Vt NP VP → VP PP	NN → man NN → woman NN → telescope
NP → DT NN NP → NP PP	DT → the
PP → IN NP	IN → with IN → in

Generation using left-most derivation strategy

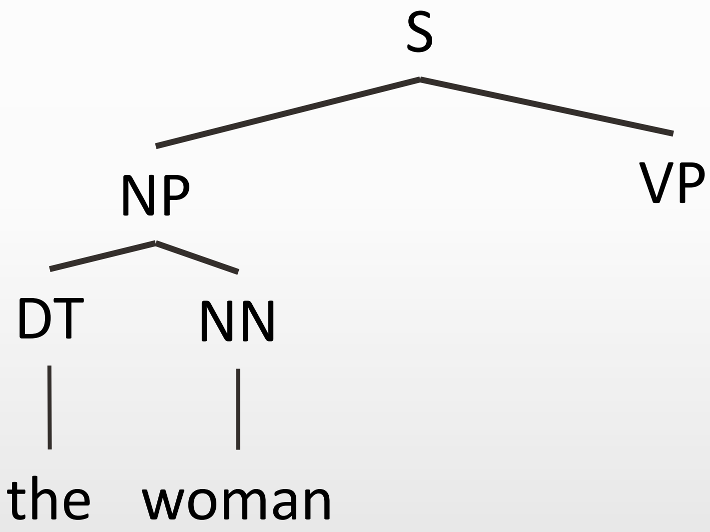
- Example: The woman saw the man with the telescope.



S → NP VP	Vi → sleeps Vt → saw
VP → Vi VP → Vt NP VP → VP PP	NN → man NN → woman NN → telescope
NP → DT NN NP → NP PP	DT → the
PP → IN NP	IN → with IN → in

Generation using left-most derivation strategy

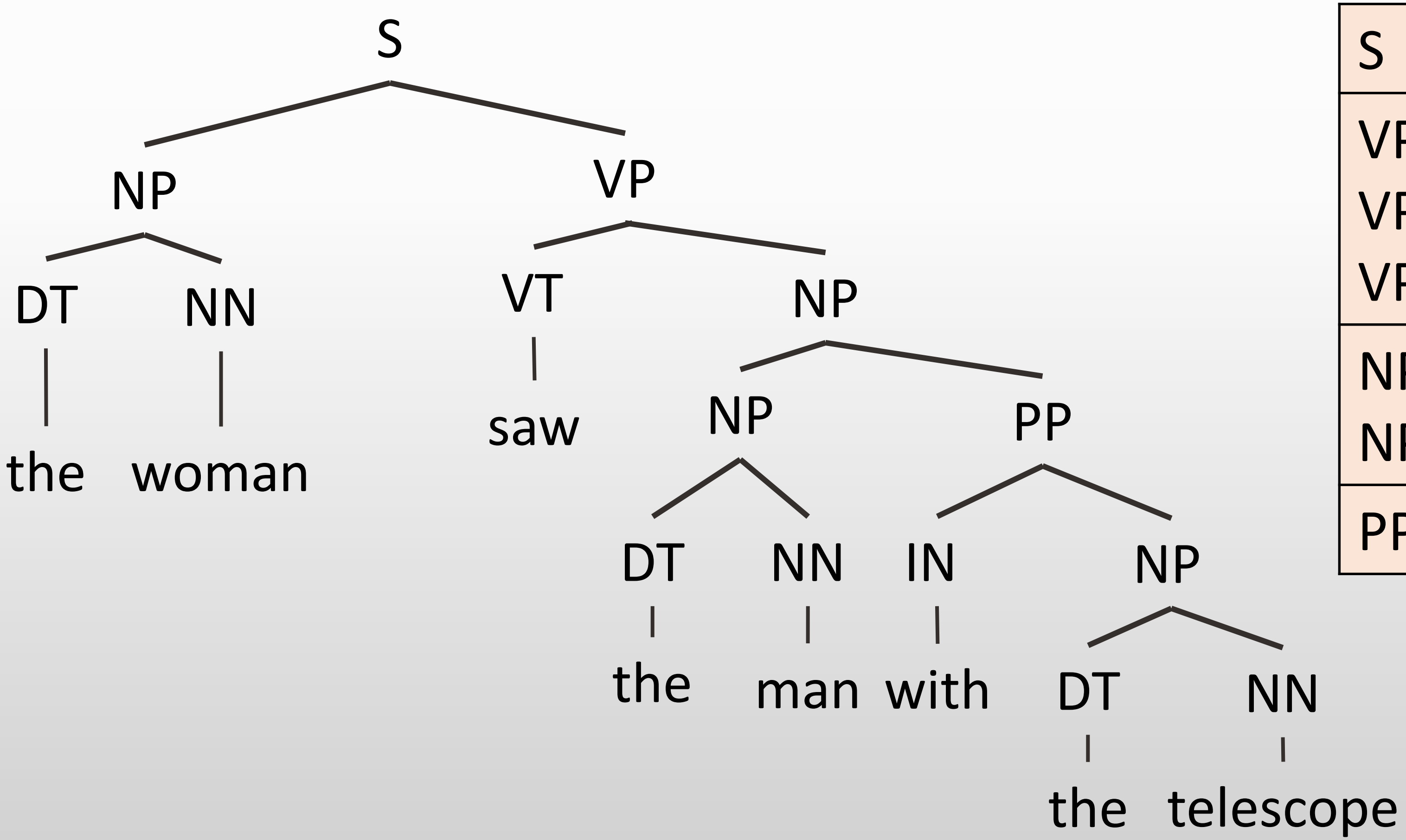
- Example: The woman saw the man with the telescope.



S → NP VP	Vi → sleeps Vt → saw
VP → Vi VP → Vt NP VP → VP PP	NN → man NN → woman NN → telescope
NP → DT NN NP → NP PP	DT → the
PP → IN NP	IN → with IN → in

Generation using left-most derivation strategy

- Example: The woman saw the man with the telescope.

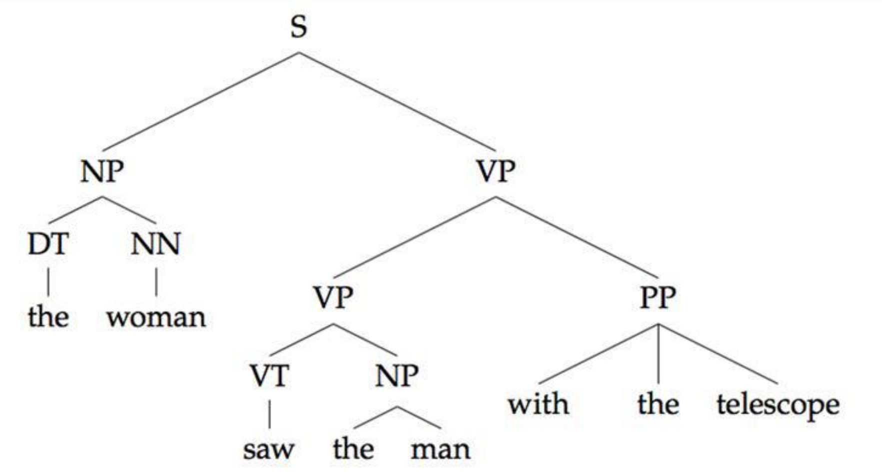


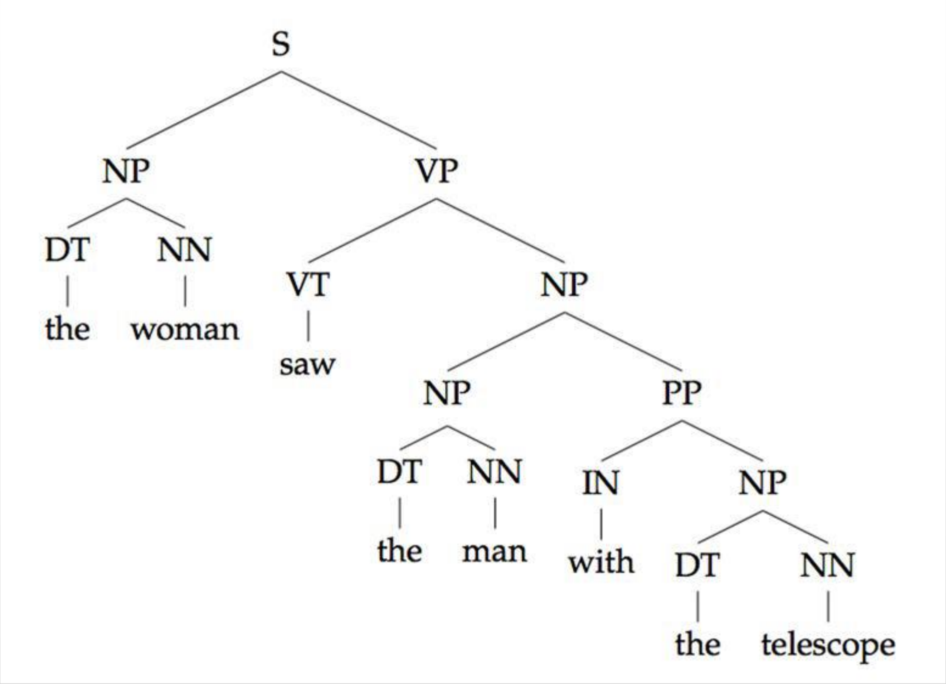
S → NP VP
VP → Vi VP → Vt NP VP → VP PP
NP → DT NN NP → NP PP
PP → IN NP

Vi → sleeps Vt → saw
NN → man NN → woman NN → telescope
DT → the
IN → with IN → in

Generation using left-most derivation strategy

- Ambiguities
  - There can be multiple derivations for the same string
  - These sentences are ambiguous as each parse represents a different meaning





Types of grammars: Probabilistic Context-Free Grammar (PCFG)

- Production rules now have probabilities

S → NP VP	1.0
VP → Vi VP → Vt NP VP → VP PP	0.4 0.4 0.2
NP → DT NN NP → NP PP	0.3 0.7
PP → IN NP	1.0

Vi → sleeps Vt → saw	1.0 1.0
NN → man NN → woman NN → telescope	0.7 0.2 0.1
DT → the	1.0
IN → with IN → in	0.5 0.5

The probability of a (sentence, parse tree) pair

$$p(S,T) = \prod_{i=1}^{m} p(\alpha_i \rightarrow \beta_i | \alpha_i)$$

m : the number of transitions

Example: $P(NN \rightarrow man|NN) = 0.7$

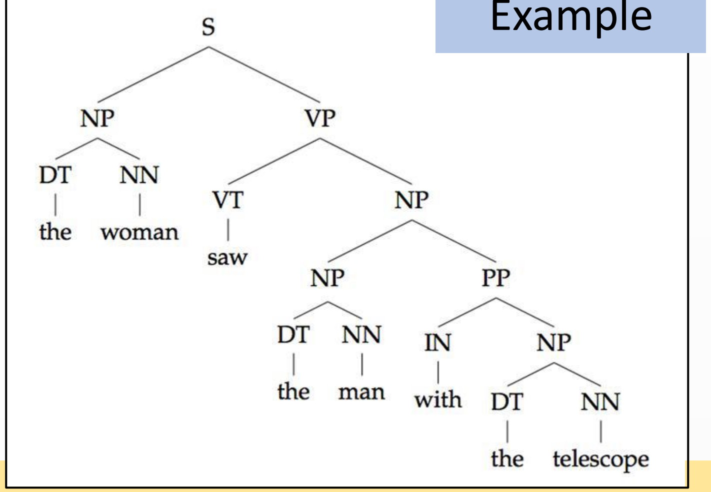
Types of grammars: Probabilistic Context-Free Grammar (PCFG)

- Production rules now have probabilities

S → NP VP	1.0
VP → Vi	0.4
VP → Vt NP	0.4
VP → VP PP	0.2
NP → DT NN	0.3
NP → NP PP	0.7
PP → IN NP	1.0

Vi → sleeps	1.0
Vt → saw	1.0
NN → man	0.7
NN → woman	0.2
NN → telescope	0.1
DT → the	1.0
IN → with	0.5
IN → in	0.5

Example



The probability of a (sentence, parse tree) pair

$$p(S,T) = \prod_{i=1}^{m} p(\alpha_i \rightarrow \beta_i | \alpha_i)$$

m : the number of transitions

Example: $P(NN \rightarrow man | NN) = 0.7$

$$P(S,T) = [1.0 \times 0.3 \times 1.0 \times 0.2] \times [0.4 \times 1.0 \times 0.7 \times (0.3 \times 1.0 \times 0.7) \times (1.0 \times 0.5 \times 0.3 \times 1.0 \times 0.1)]$$

Estimating transition probabilities

- Counts from training set
  - Example:

$$P(S \rightarrow NP\ VP\ |S) = \frac{count(S \rightarrow NP\ VP)}{count(S)}$$

PCFG tasks

- What is the most likely parse?
  - $argmax_T P(T, S)$
- What is the probability of the sentence?
  - $P(S) = \sum_T P(T, S)$

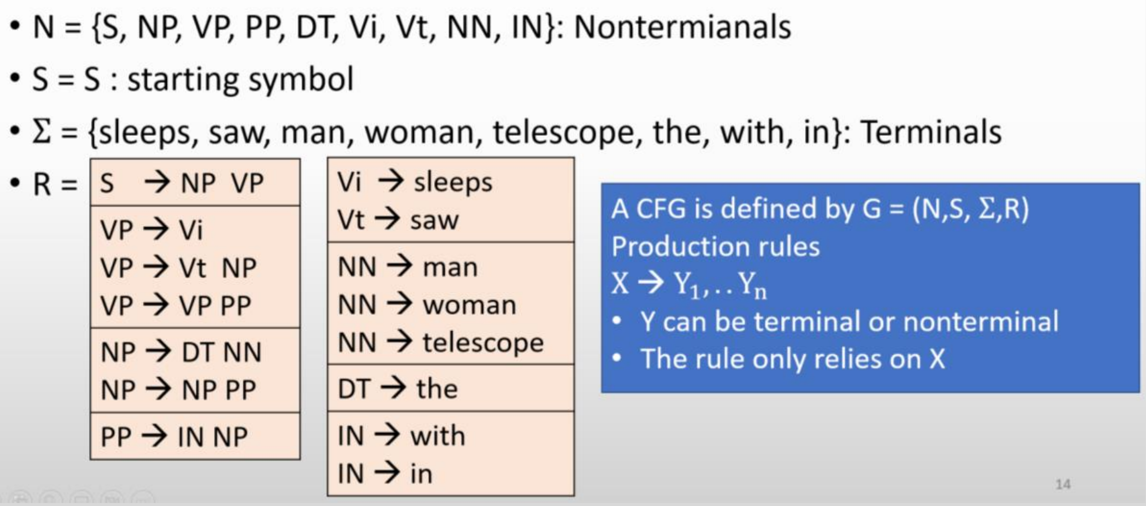
CYK algorithm
(Cocke–Younger–Kasami algorithm)

Chomsky Normal Form

- CYK can be used if the CFG is in Chomsky Normal Form.
- A CFG is in Chomsky Normal Form if each rule either converts to two nonterminals or a single terminal.

$$X \rightarrow Y_1\, Y_2 \quad , \quad X \rightarrow y$$

- Any CFG can be converted to CNF
- Example: NP -> DT, ADJ, NN
  - NP -> DT, ADJP
  - ADJP -> ADJ, NN

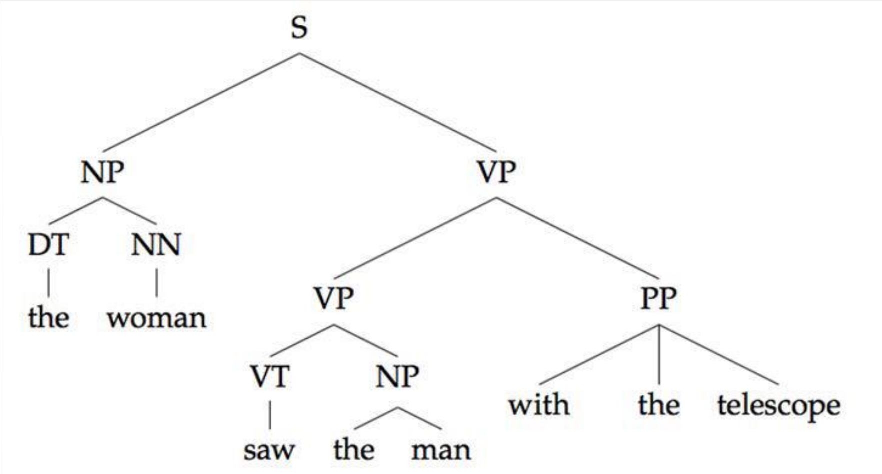


CYK algorithm for parsing

- $\pi(i, j, N)$: probability that words i to j can be generated by nonterminal N
- Base case: $\pi(i, i, N) = P(N \rightarrow w_i | N)$
- Inductive case:

$$\pi(i, j, N) = \max_{k,P,Q} P(N \rightarrow P\ Q | N) \cdot \pi(i, k, P) \cdot \pi(k+1, j, Q)$$

where $k \in \{i, \dots, j-1\}, P \in \mathcal{N}$ and $Q \in \mathcal{N}$



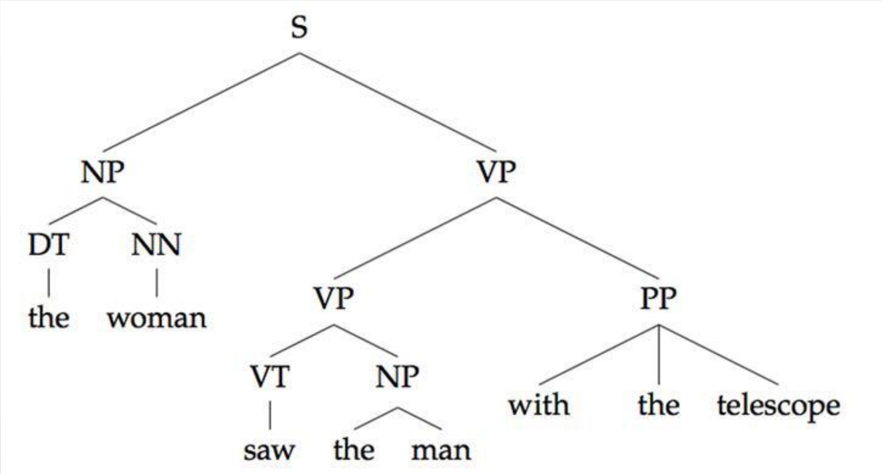
CYK algorithm for LM

- $\pi(i, j, N)$: probability that words i to j can be generated by nonterminal N
- Base case: $\pi(i, i, N) = P(N \rightarrow w_i | N)$
- Inductive case:

$$\pi(i, j, N) = \sum_{k,P,Q} P(N \rightarrow P\ Q | N) \cdot \pi(i, k, P) \cdot \pi(k + 1, j, Q)$$

where $k \in \{i, \ldots, j - 1\}, P \in \mathcal{N}$ and $Q \in \mathcal{N}$

$argmax_T P(T, S)$ vs $\sum_T P(T, S)$



Example CYK with PCFG for LM

- Find $P(\text{"abc"}) = ?$
- N = {A,B}
- Σ = {a,b,c}
- S = {A}
- $\pi(i, j, N)$

A → A B	0.8
A → a	0.2
B → BB	0.7
B → b	0.1
B → c	0.2

A

a			
b			
c			

B

a			
b			
c			

Example CYK with PCFG for LM

- Find $P(\text{"abc"}) = ?$
- N = {A,B}
- Σ = {a,b,c}
- S = {A}
- $\pi(i, i, N) = P(N \rightarrow w_i | N)$

A → A B	0.8
A → a	0.2
B → BB	0.7
B → b	0.1
B → c	0.2

A

a	0.2		
b		0	
c			0

B

a	0		
b		0.1	
c			0.2

Example CYK with PCFG for LM

- Find $P(\text{"abc"}) = ?$
- N = {A,B}
- Σ = {a,b,c}
- S = {A}
- $\pi(i,i,N) = P(N \rightarrow w_i|N)$
- $\pi(i,j,N) = \sum_{k,P,Q} P(N \rightarrow P\ Q|N) \cdot \pi(i,k,P) \cdot \pi(k+1,j,Q)$

A → A B A → a	0.8 0.2
B → BB B → b B → c	0.7 0.1 0.2

A

a	0.2		
b		0	
c			0

B

a	0		
b		0.1	
c			0.2

Example CYK with PCFG for LM

- Find $P(\text{"abc"}) = ?$
- N = {A,B}
- Σ = {a,b,c}
- S = {A}
- $\pi(i,i,N) = P(N \to w_i | N)$
- $\pi(i,j,N) = \sum_{k,P,Q} P(N \to P\ Q | N) \cdot \pi(i,k,P) \cdot \pi(k+1,j,Q)$

A → A B	0.8
A → a	0.2
B → BB	0.7
B → b	0.1
B → c	0.2

A

a	0.2	0.016	
b		0	
c			0

$$\begin{aligned}\pi(1,2,A) = {} & P(A \to AA) \cdot P(A \to a|A) \cdot P(A \to b|A) + \\ & P(A \to AB) \cdot P(A \to a|A) \cdot P(B \to b|B) + \\ & P(A \to BA) \cdot P(B \to a|B) \cdot P(A \to b|B) + \\ & P(A \to BB) \cdot P(B \to a|B) \cdot P(B \to b|B) \\ = {} & 0 + P(A \to AB) \cdot \pi(1,1,A) \cdot \pi(2,2,B) + 0 + 0 \\ = {} & 0.8 \cdot 0.2 \cdot 0.1 = 0.016\end{aligned}$$

Example CYK with PCFG for LM

- Find $P(\text{"abc"}) = ?$
- N = {A,B}
- Σ = {a,b,c}
- S = {A}
- $\pi(i,i,N) = P(N \rightarrow w_i | N)$
- $\pi(i,j,N) = \sum_{k,P,Q} P(N \rightarrow P\ Q | N) \cdot \pi(i,k,P) \cdot \pi(k+1,j,Q)$

A → A B	0.8
A → a	0.2
B → BB	0.7
B → b	0.1
B → c	0.2

$$\begin{aligned}\pi(1,2,B) &= P(B \rightarrow BB) \cdot P(B \rightarrow a|B) \cdot P(B \rightarrow b|B) \\ &= P(B \rightarrow BB) \cdot \pi(1,1,B) \cdot \pi(2,2,B) \\ &= 0.7 \cdot 0 \cdot 0.1 = 0\end{aligned}$$

A

c			0

B

a	0	0	
b		0.1	
c			0.2

Example CYK with PCFG for LM

- Find $P(\text{"abc"}) = ?$
- N = {A,B}
- Σ = {a,b,c}
- S = {A}
- $\pi(i,i,N) = P(N \to w_i | N)$
- $\pi(i,j,N) = \sum_{k,P,Q} P(N \to P\ Q | N) \cdot \pi(i,k,P) \cdot \pi(k+1,j,Q)$

A → A B	0.8
A → a	0.2
B → BB	0.7
B → b	0.1
B → c	0.2

A

a	0.2	0.016	
b		0	0

$$\pi(2,3,A) = P(A \to AB) \cdot \pi(2,2,A) \cdot \pi(3,3,B) = 0.8 \cdot 0 \cdot 0.2 = 0$$

B

a	0	0	
b		0.1	0.014
c			0.2

$$\pi(2,3,B) = P(B \to BB) \cdot \pi(2,2,B) \cdot \pi(3,3,B) = 0.7 \cdot 0.1 \cdot 0.2 = 0.014$$

Example CYK with PCFG for LM

- Find $P(\text{"abc"}) = ?$
- N = {A,B}
- Σ = {a,b,c}
- S = {A}
- $\pi(i,i,N) = P(N \rightarrow w_i|N)$
- $\pi(i,j,N) = \sum_{k,P,Q} P(N \rightarrow P\ Q|N) \cdot \pi(i,k,P) \cdot \pi(k+1,j,Q)$

A → A B	0.8
A → a	0.2
B → BB	0.7
B → b	0.1
B → c	0.2

A

a	0.2	0.016	0.0048
b		0	0

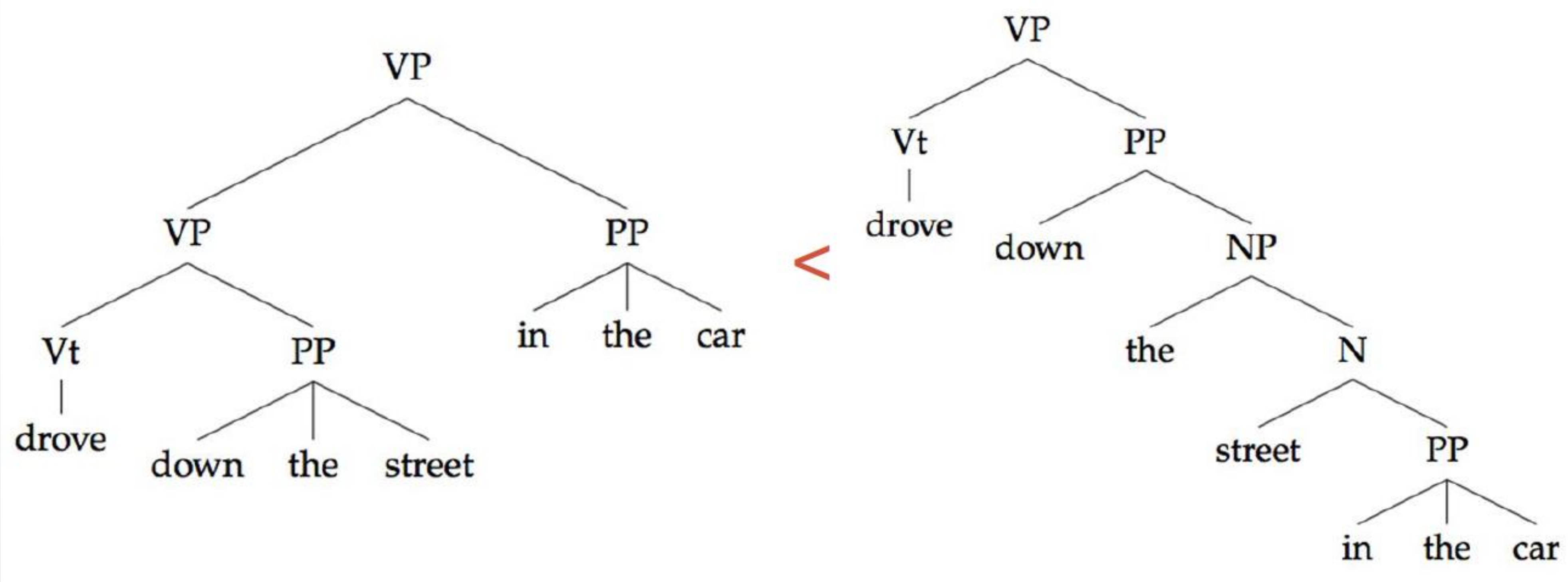
B

a	0	0	
b		0.1	0.014

$$\pi(1,3,A) = P(A \rightarrow AB) \cdot \pi(1,2,A) \cdot \pi(3,3,B) + P(A \rightarrow AB) \cdot \pi(1,1,A) \cdot \pi(2,3,B)$$
$$= 0.8 \cdot 0.016 \cdot 0.2 + 0.8 \cdot 0.2 \cdot 0.014 = 0.0048$$

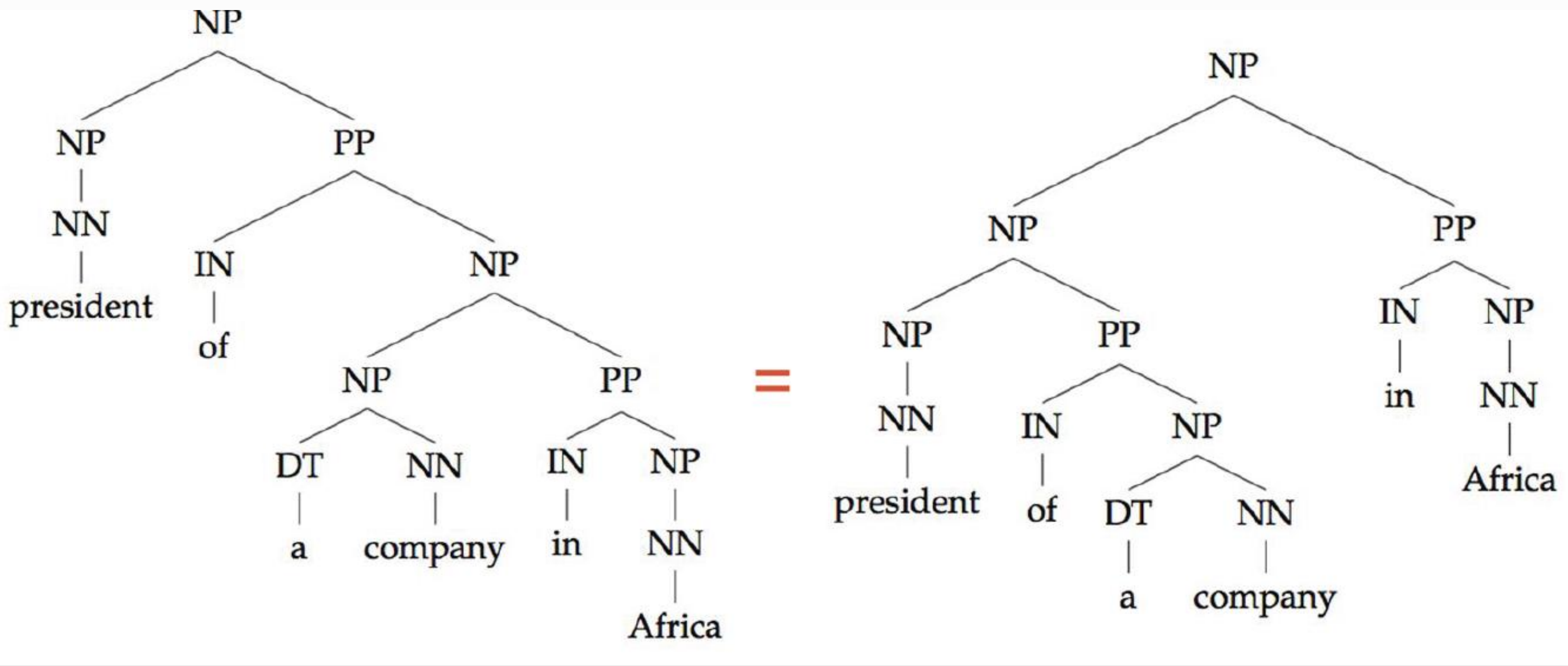
PCFG weakness

- Lack of sensitivity to lexical info



PCFG weakness

- Lack of sensitivity to structural frequency



Dependency grammar

- CFG is based on constituency relation
- In dependency grammar the structure is composed of lexical items (words) linked by edges to form a tree
- Assumptions
  - Each words in a sentence is related or modifies another word
  - All words have a direct or indirect relation to the main verb

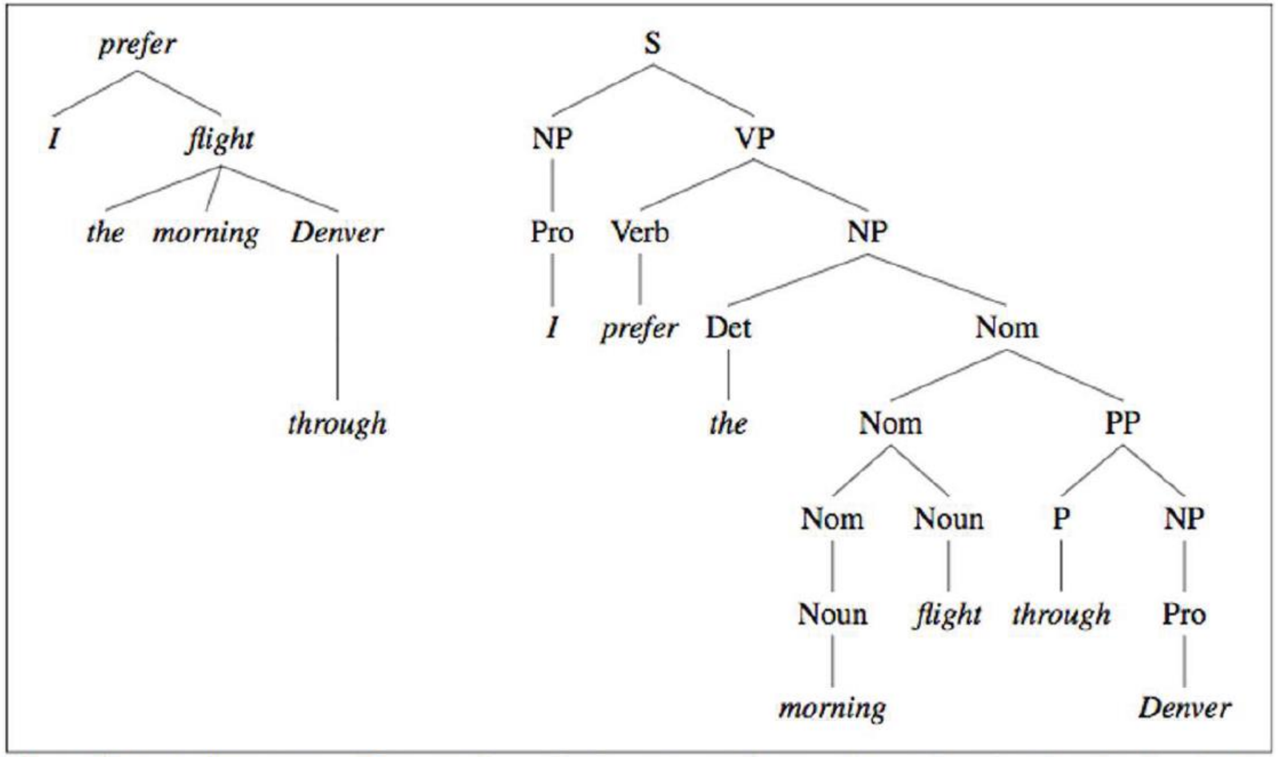
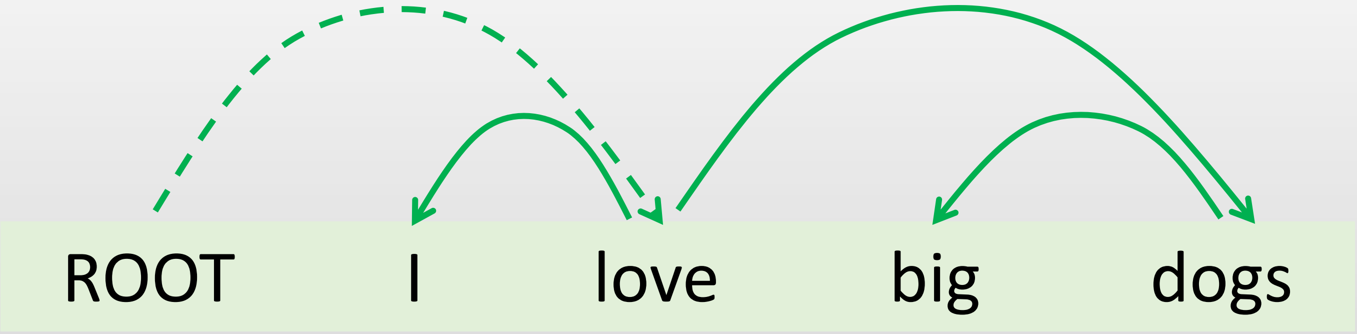


Figure 14.1 A dependency-style parse alongside the corresponding constituent-based analysis for *I prefer the morning flight through Denver.*

Dependency grammar

- Example:
  - Add ROOT node as the root of the tree
  - The main verb always point to ROOT
  - Each arc can have a category for the relationship.
  - Each word can have a PoS label

A -> B means A governs B
or B depends on A
or A is the head of B



Constituency structures vs dependency structures

- Constituency structures use more nodes to represent sentences at different levels.
- Constituency structures explicitly label non-terminal nodes
- Constituency structures encode more info than dependency structures
- You can convert constituency structures to dependency structures
    - Dependency parsers trained on this is usually better than dependency parsers trained on original dependency structures

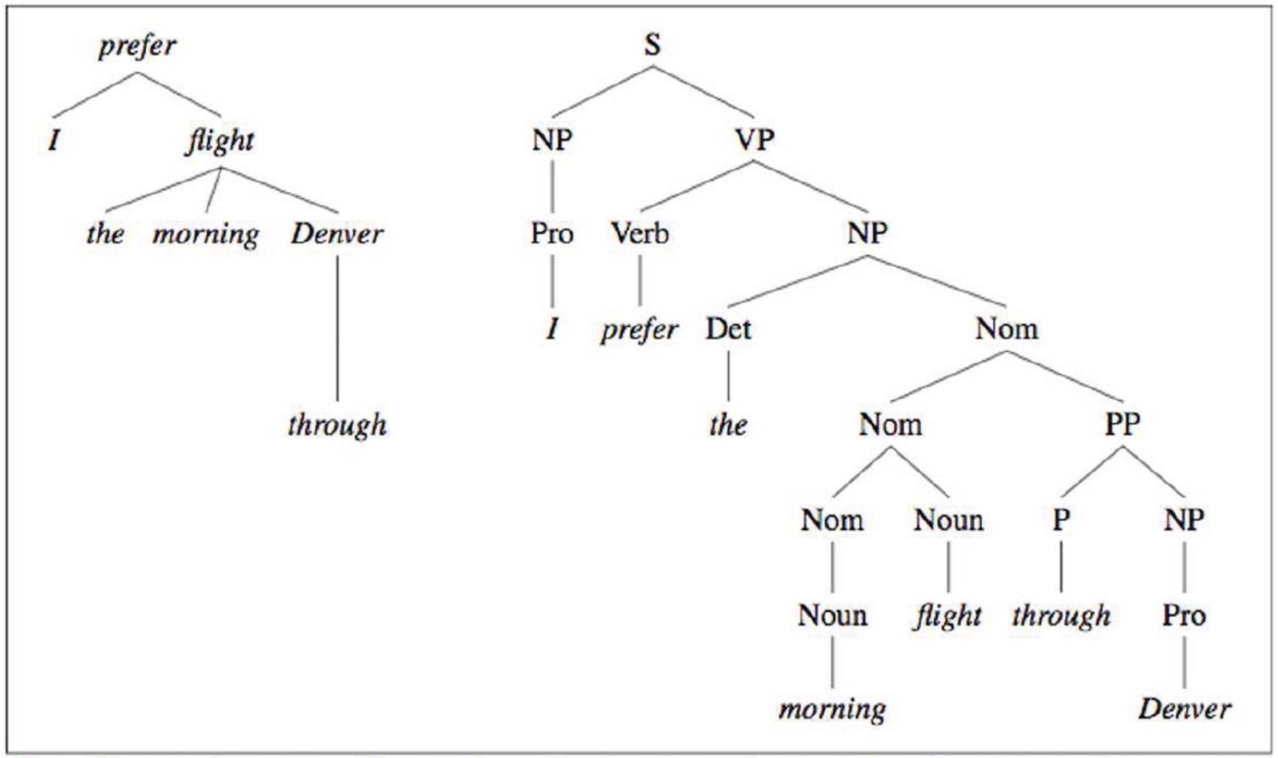
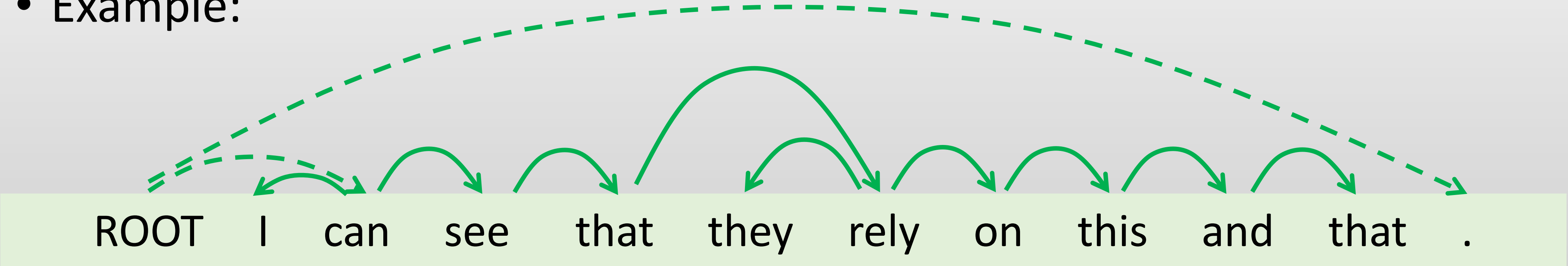


Figure 14.1 A dependency-style parse alongside the corresponding constituent-based analysis for *I prefer the morning flight through Denver.*

Criteria for heads (basics)

- Head, H. Dependent D
- D modifies H
  - Big (D) dogs (H), willow (D) tree (H)
- H can often replace D
  - I love big (D) dogs (H) -> I love dogs
- H is obligatory while D sometimes is optional
- H determines whether D is obligatory
  - Sarah sneezed (H) vs George kicks (H) the chair (D)
- More criterias! Mostly depends on corpus
- Example:

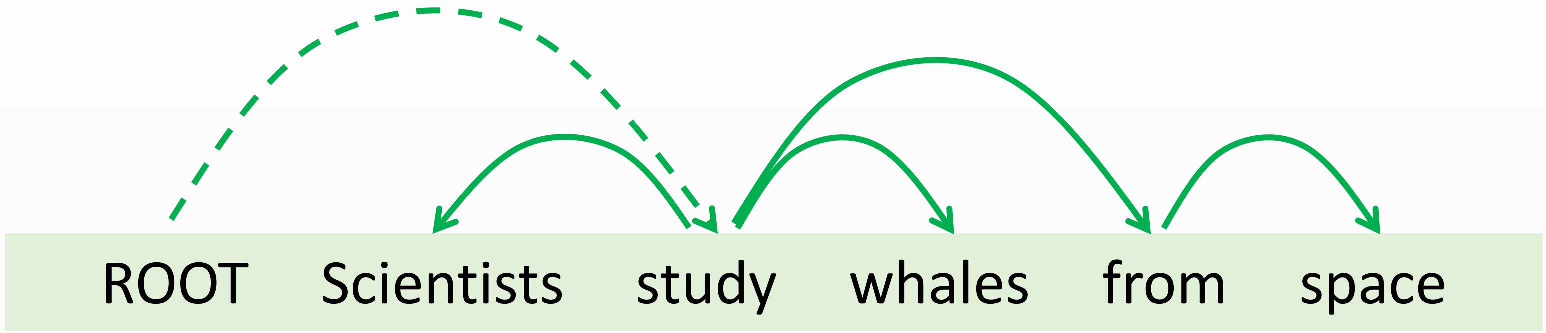


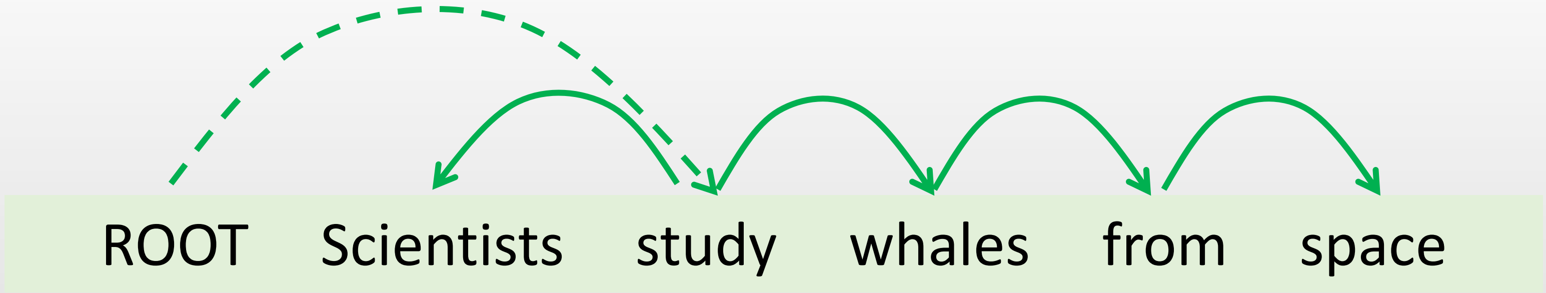
Dependency: Examples

ROOT Scientists study whales from space

ROOT Scientists study whales from space

Dependency: Examples



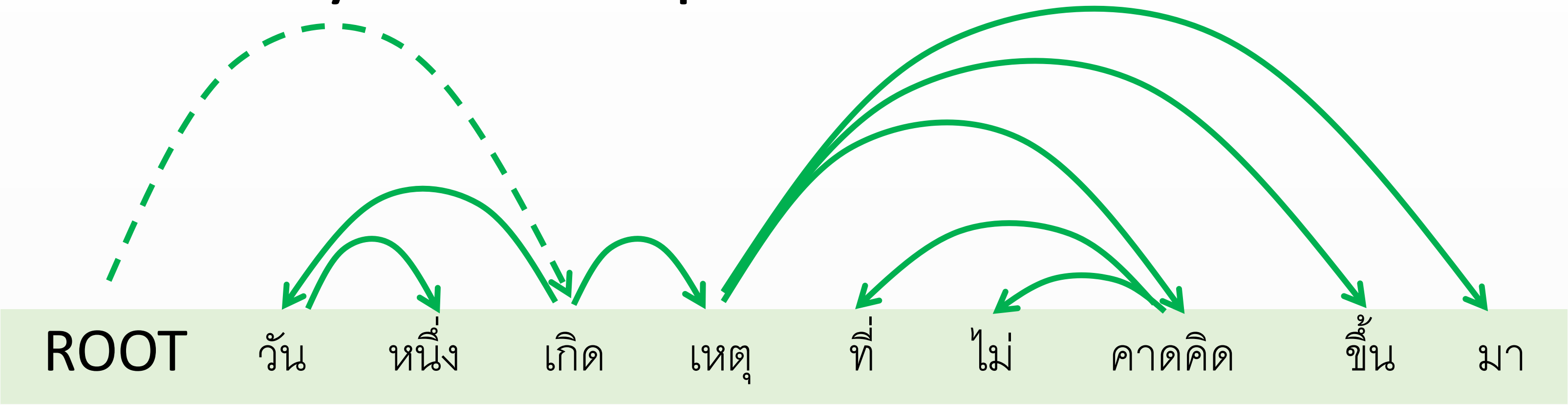


Dependency: Examples

ROOT	วัน	หนึ่ง	เกิด	เหตุ	ที่	ไม่	คาดคิด	ขึ้น	มา

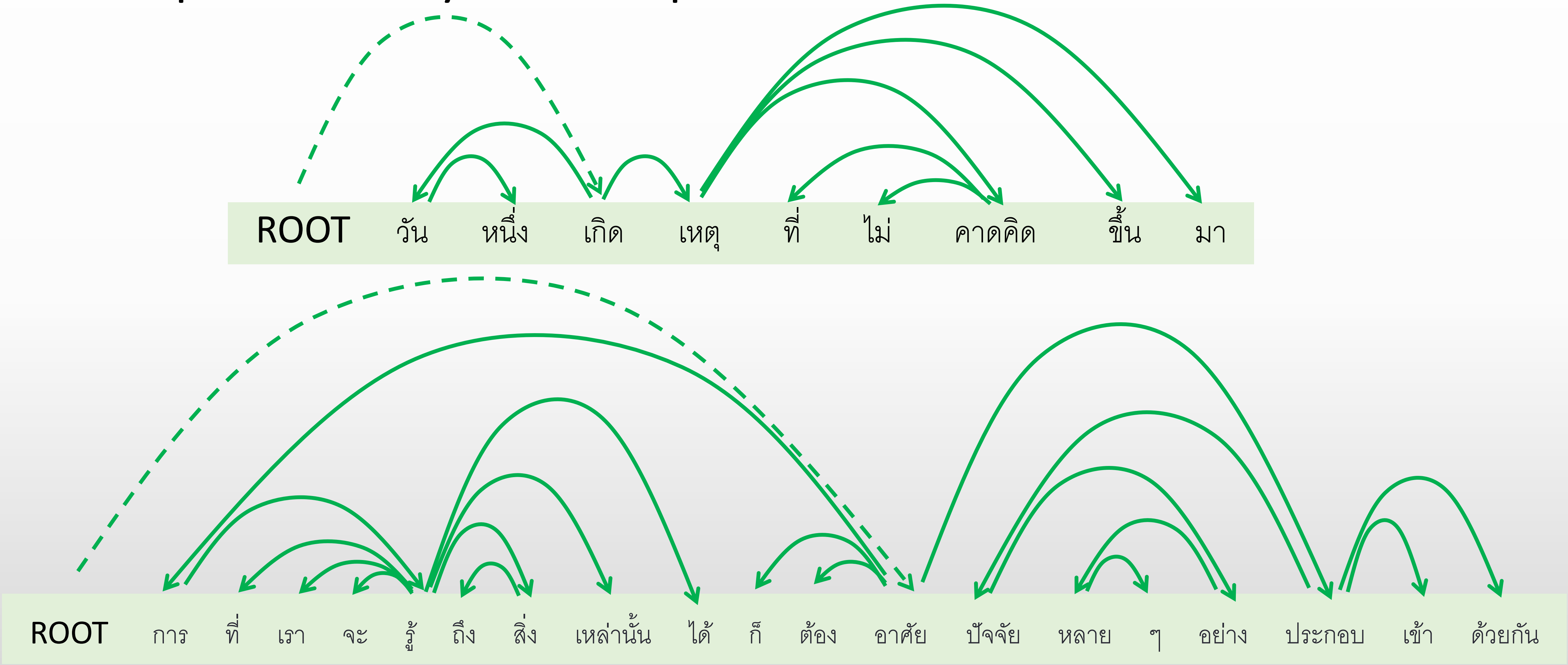
ROOT	การ	ที่	เรา	จะ	รู้	ถึง	สิ่ง	เหล่านั้น	ได้	ก็	ต้อง	อาศัย	ปัจจัย	หลาย	ๆ	อย่าง	ประกอบ	เข้า	ด้วยกัน

Dependency: Examples



ROOT การ ที่ เรา จะ รู้ ถึง สิ่ง เหล่านั้น ได้ ก็ ต้อง อาศัย ปัจจัย หลาย ๆ อย่าง ประกอบ เข้า ด้วยกัน

Dependency: Examples

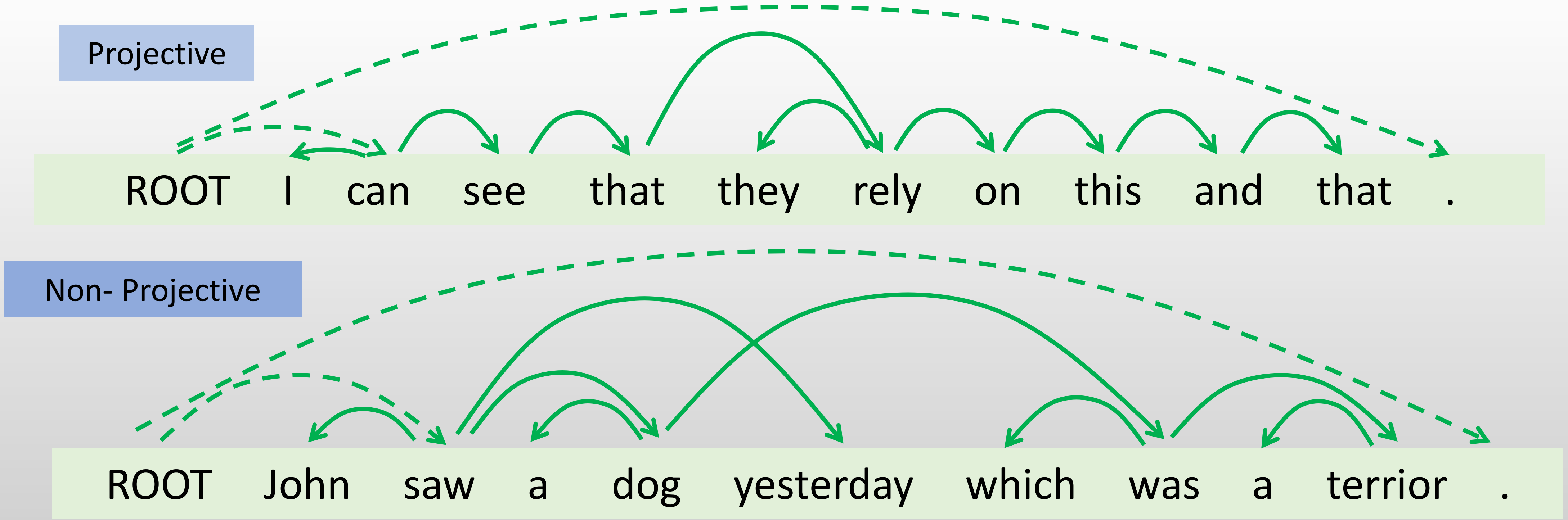


Dependency graph requirements

- Syntactic structure is complete (connectedness, spanning)
- Hierarchical (acyclic)
- Every word has a single head

Projectivity

- A dependency graph is projective if the arcs do not cross
- English and Thai are mostly projective.
- Some languages are more non-projective than others, for example German, Dutch, Czech.



Transition-based parsing (Nivre 2007)

- Use a stack and buffer data structure and sequentially add edges
- Characteristics
  - Greedy algo. Only goes left to right. No backtracking.
  - Requires projectivity
  - The algo is closely related to how human parse sentences (left to right one word at a time instead of looking at the sentence as a whole)

Arc-standard Transition-based parsing

- A stack σ, written with top of the stack to the right
  - Starts with the ROOT symbol
- A buffer β, written with top to the left
  - Starts with the input sentence
- A set of dependency arcs A
  - Starts of empty
- A set of actions

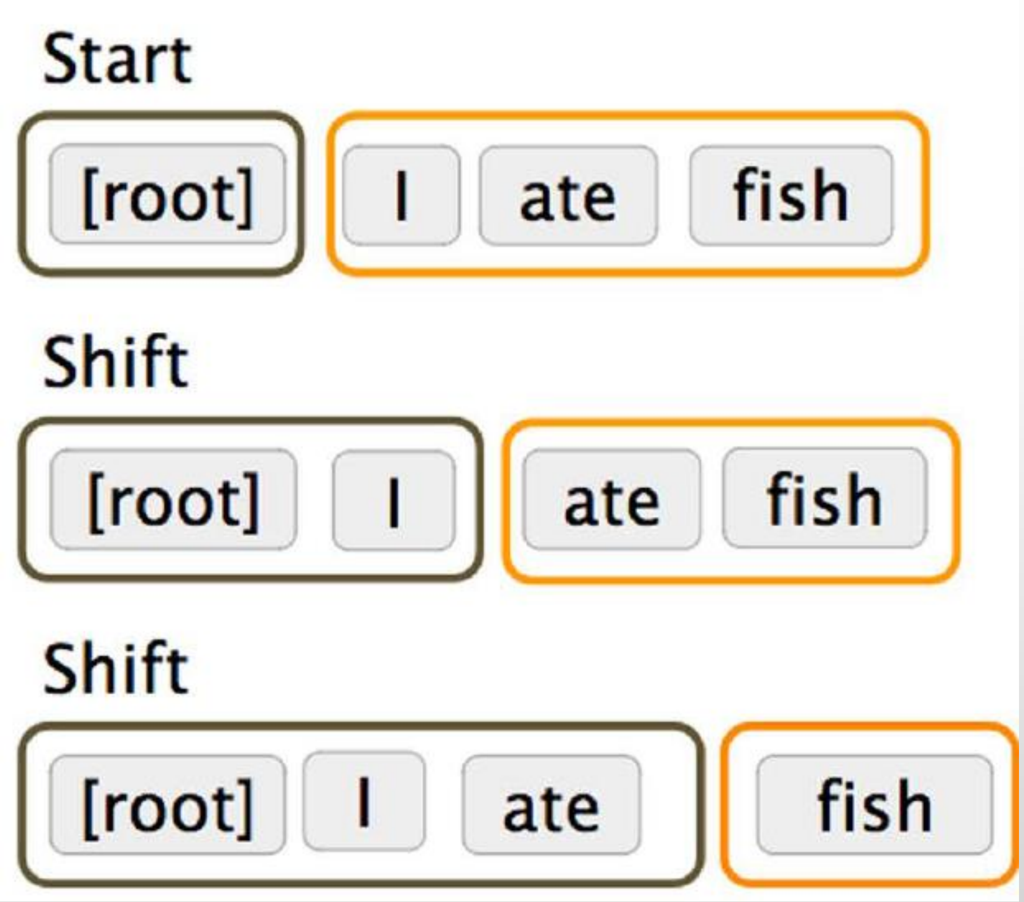
$\sigma = [ROOT]$
$\beta = w_1, w_2, \dots, w_n$
$A = \emptyset$ (Set of Arc)

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

- Finishes when β becomes empty and $\sigma = [ROOT]$

Arc-standard Transition-based parsing

- Example: I ate fish



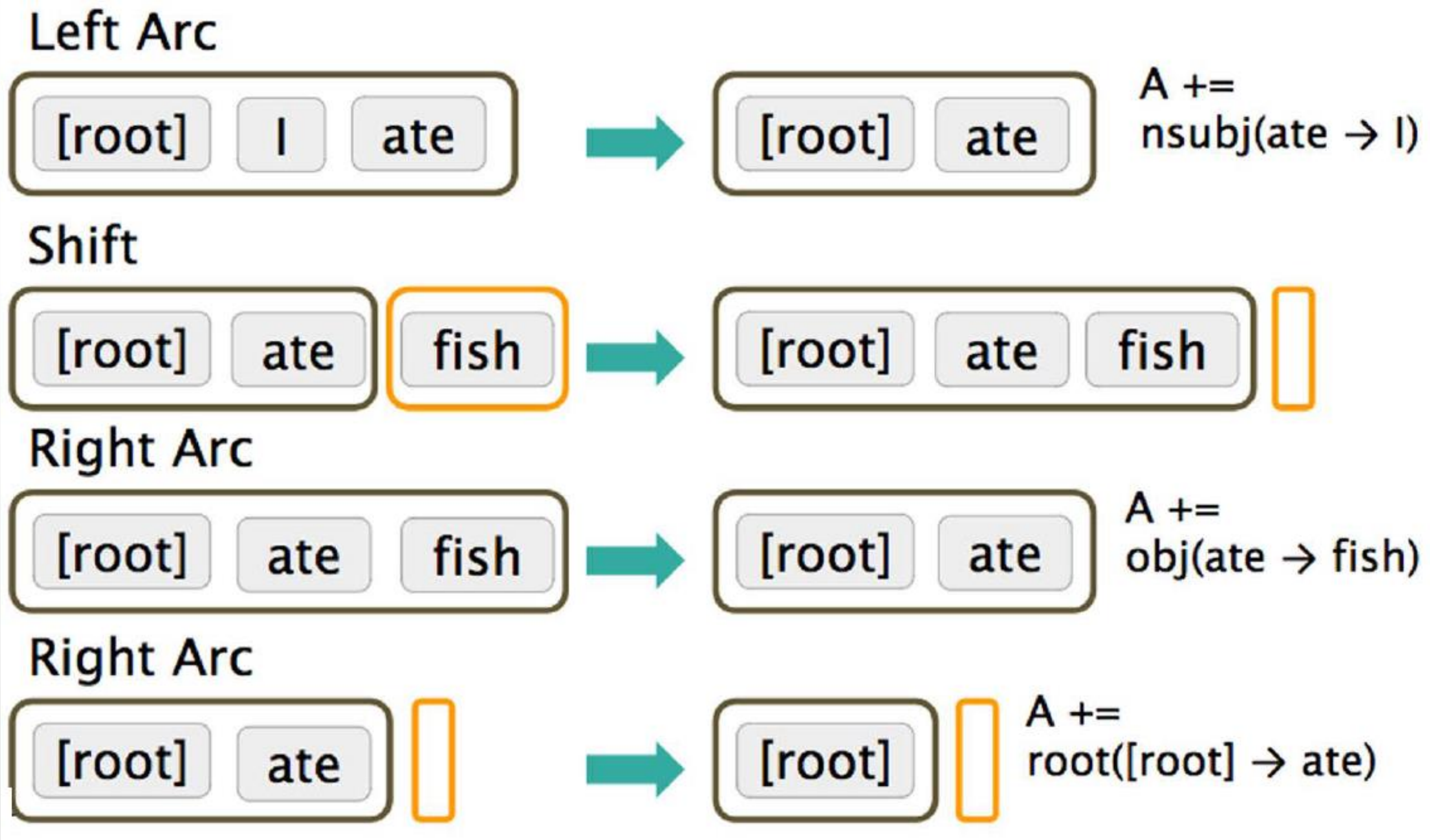
Start: σ=$[ROOT]$, $\beta = w_1, w_2, \ldots, w_n$, A=$\emptyset$

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

Finish: $\beta = \emptyset$ and $\sigma = [ROOT]$

Arc-standard Transition-based parsing

- Example: I ate fish



Start: σ=$[ROOT]$, $\beta = w_1, w_2, \ldots, w_n$, A=$\emptyset$

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

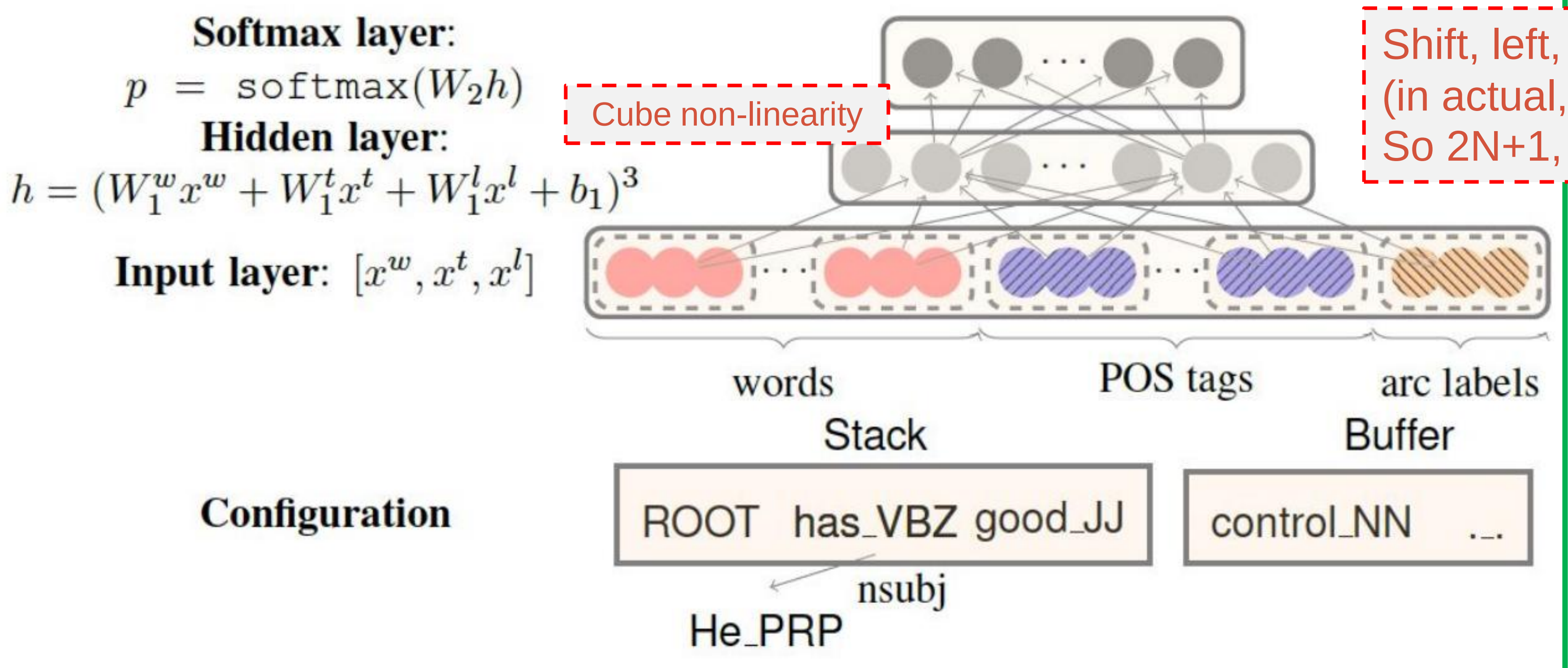
Finish: β = $\emptyset$ and $\sigma = [ROOT]$

Discriminative parsing

- How to choose an action?
  - Shift, left-arc, right-arc
- Each action is predicted by a discriminative classifier (SVM, logistic regression, Neural networks) over legal moves
  - Features: top two word from stack, POS, children info; first word in buffer, POS, children info; etc.
- Greedy and no beamsearch
  - But you can include beamsearch (modern parsers do)

Discriminative parsing

Discriminative parsing with neural networks



Cube non-linearity

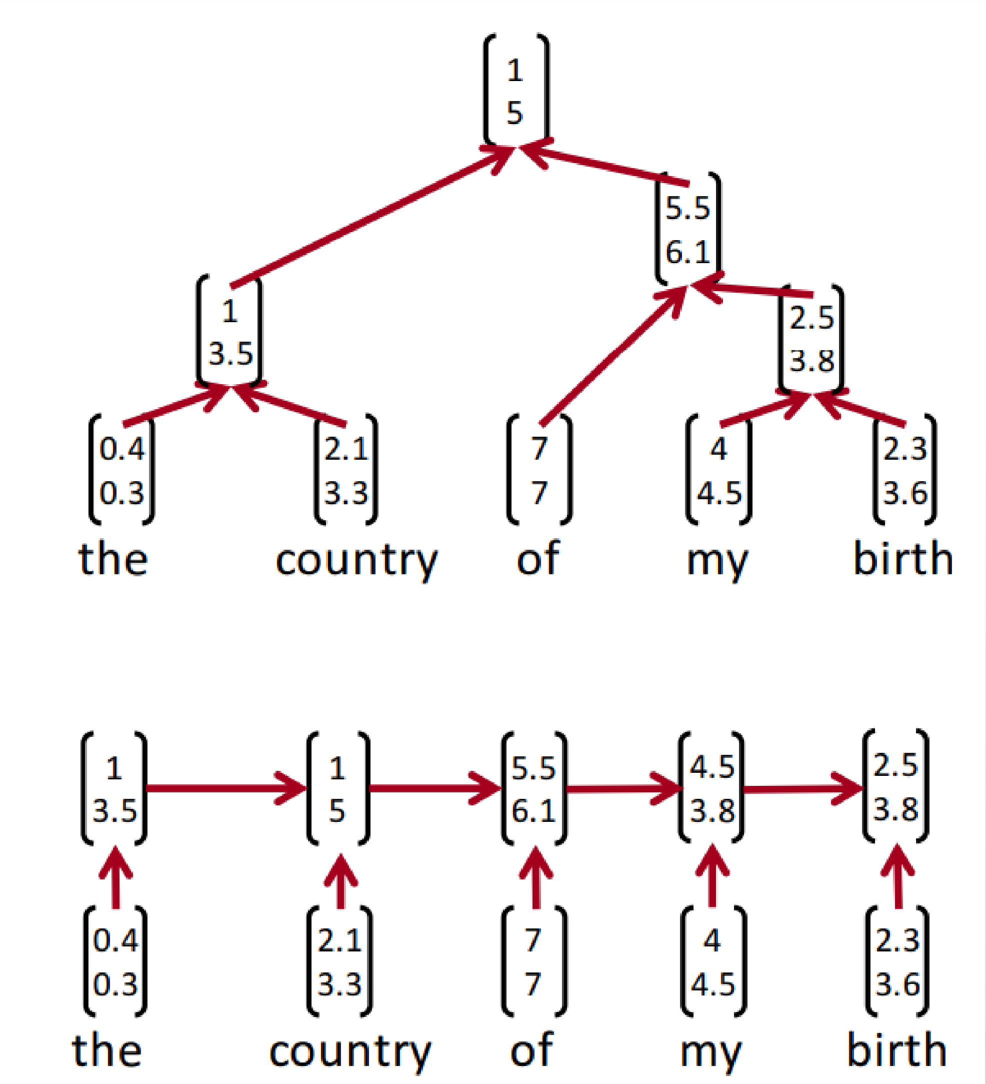
Shift, left, right
(in actual, left/right + type of dependency)
So 2N+1, where N = dependency types

POS and arc labels are also embeddings

https://courses.engr.illinois.edu/cs546/sp2020/Slides/Lecture17.pdf

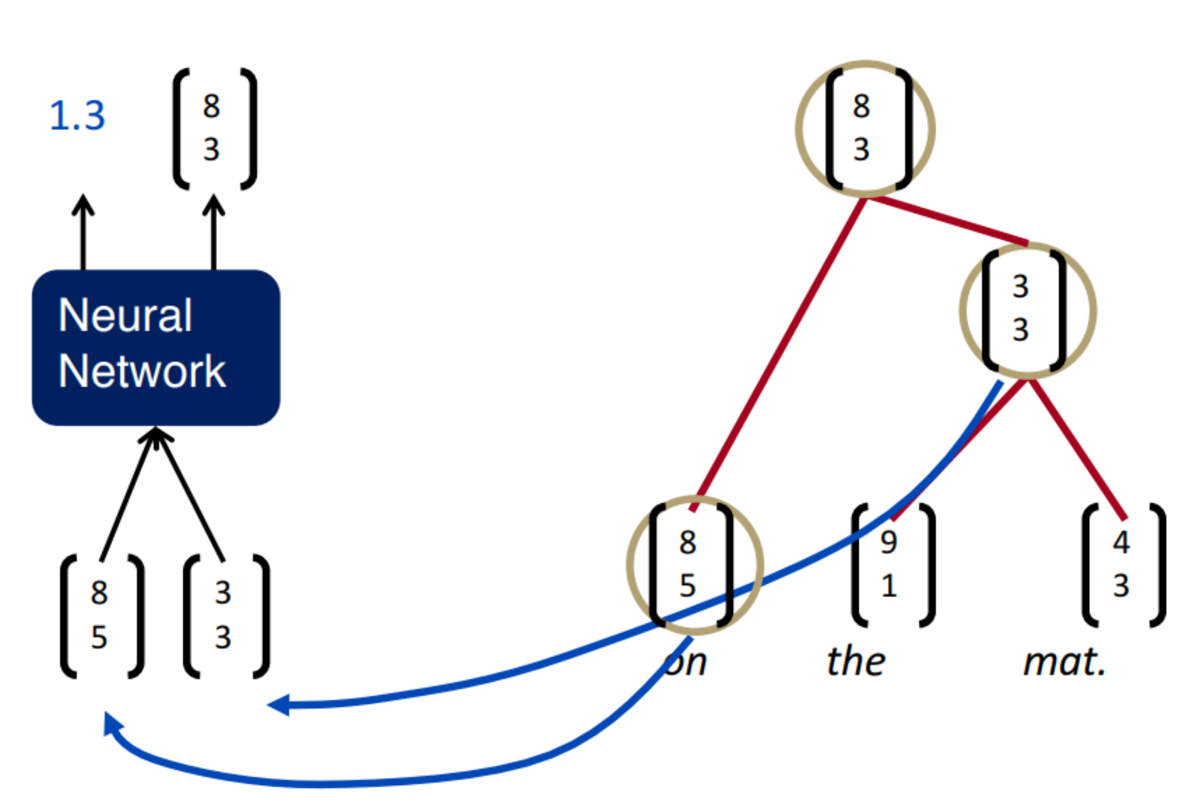
Recursive neural networks

- Not really used in parsing anymore but interesting concept
- Recursive vs Recurrent



Recursive neural networks

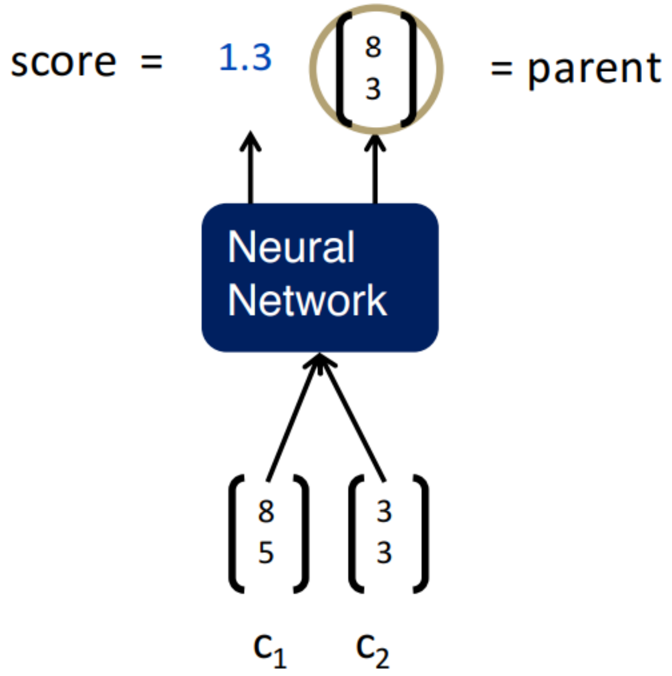
- Concept: try different connections, see if which one gives the highest score (graph-based dependency parsers)
- Inputs: two candidate children representations
  - Output:
    - Semantic representation of the parent
    - Score of new node



http://cs224d.stanford.edu/lectures/CS224d-Lecture10.pdf

Recursive neural networks

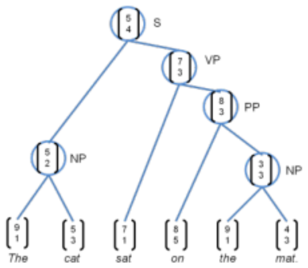
- Shared recursive structure



score = $U^{\mathsf{T}} p$

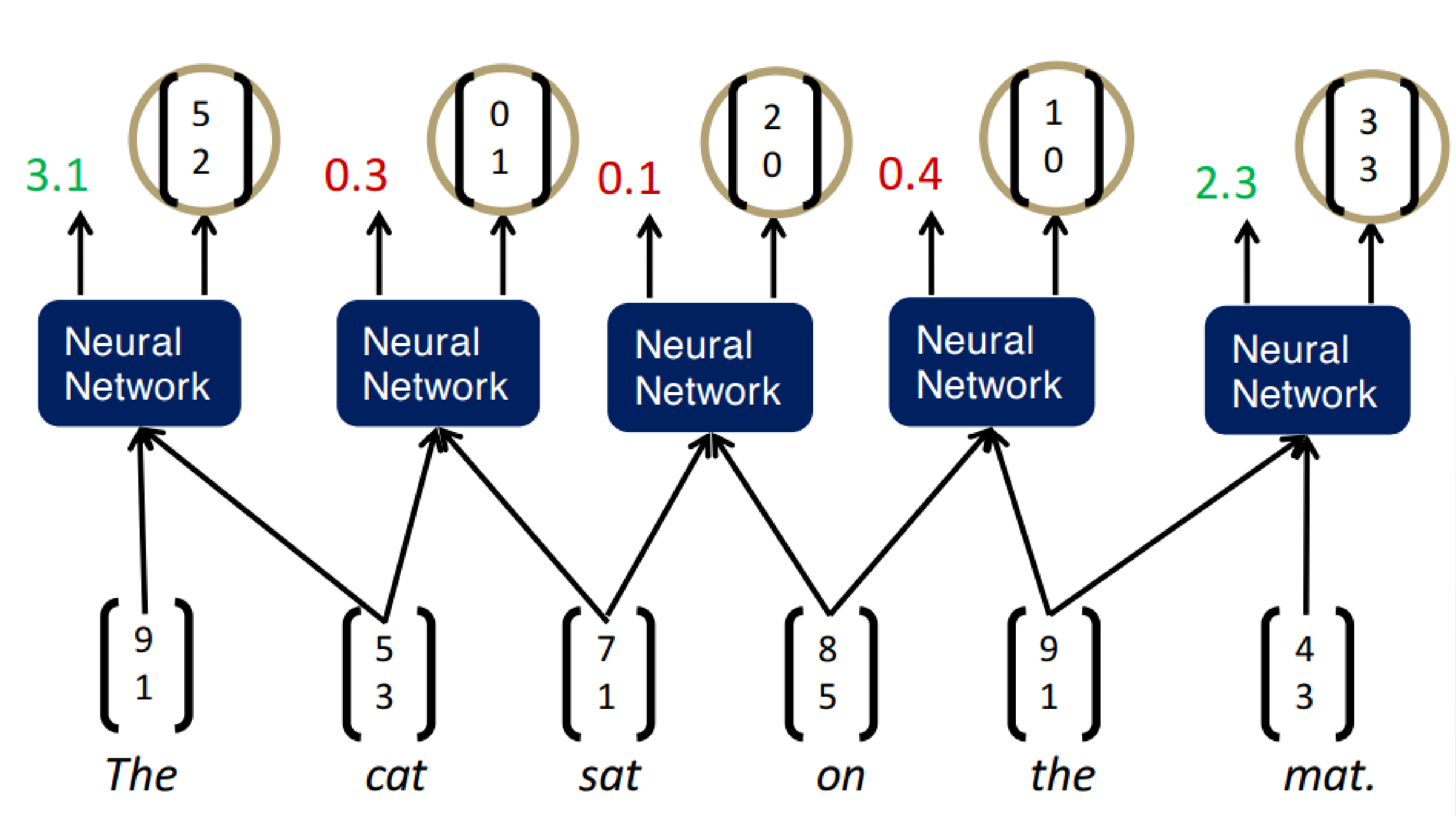
$$\boxed{p = \tanh\left(W\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b\right),}$$

Same *W* parameters at all nodes of the tree



Recursive neural networks

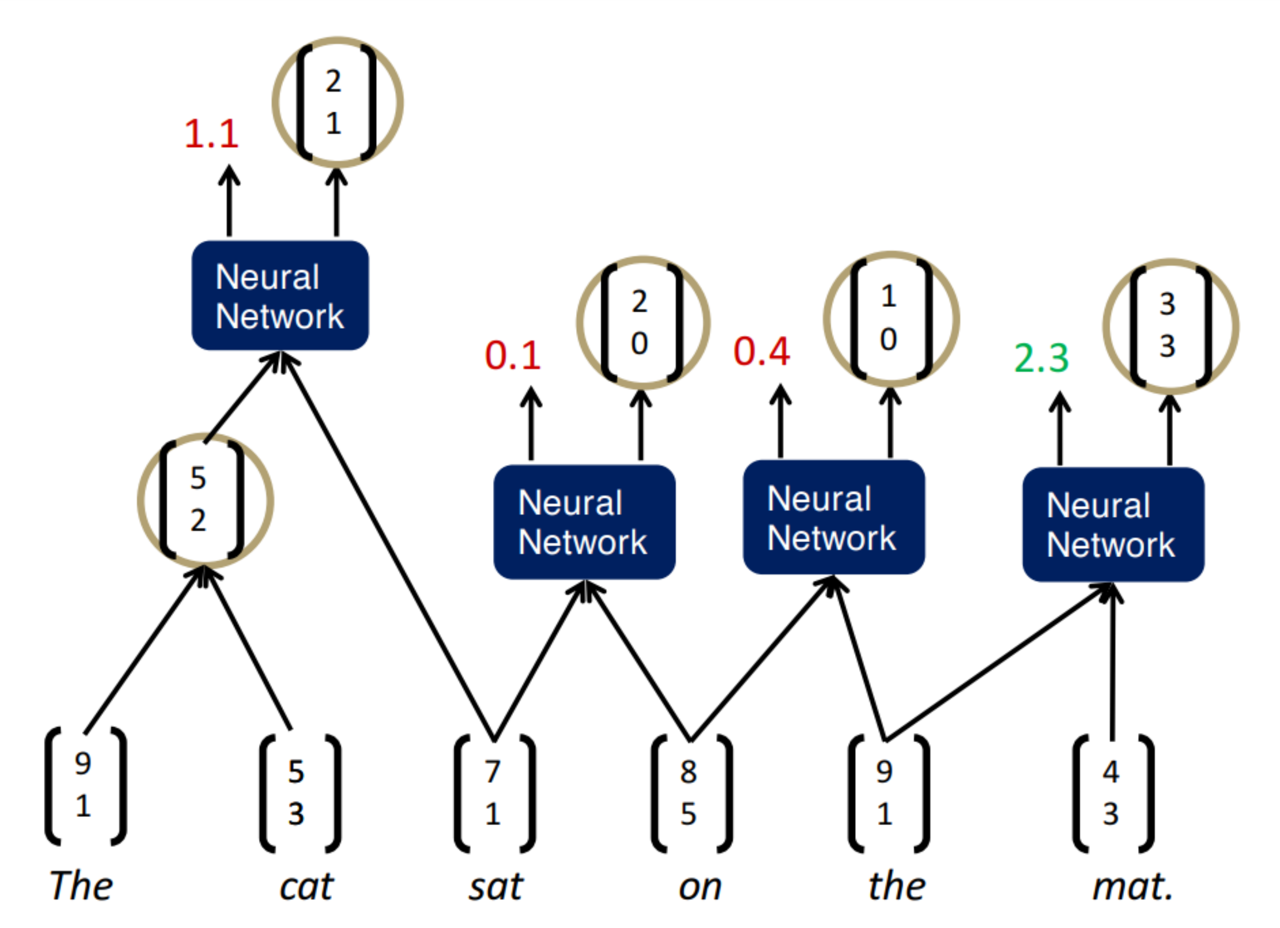
- Parsing with recursive networks



http://cs224d.stanford.edu/lectures/CS224d-Lecture10.pdf

Recursive neural networks

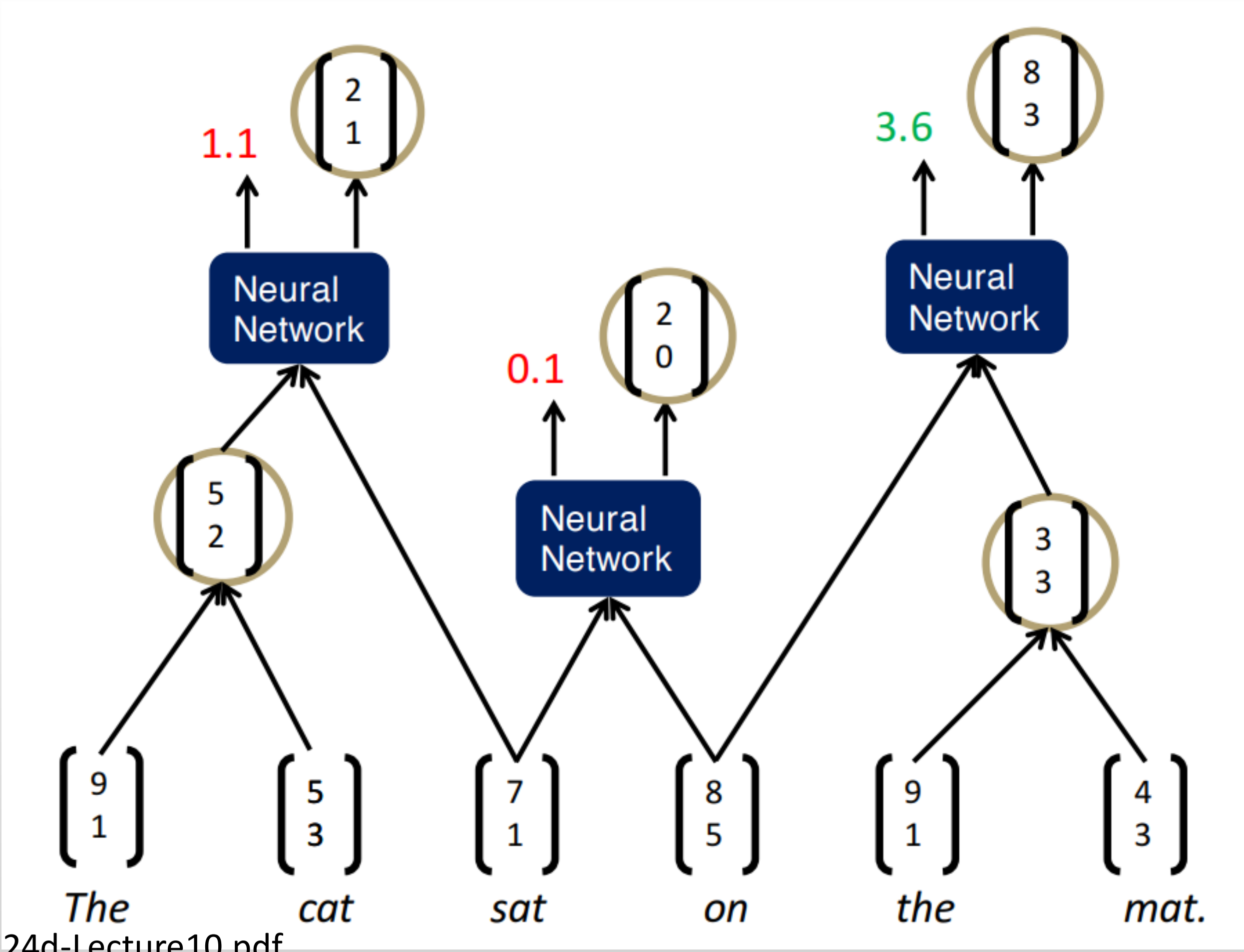
- Parsing with recursive networks



http://cs224d.stanford.edu/lectures/CS224d-Lecture10.pdf

Recursive neural networks

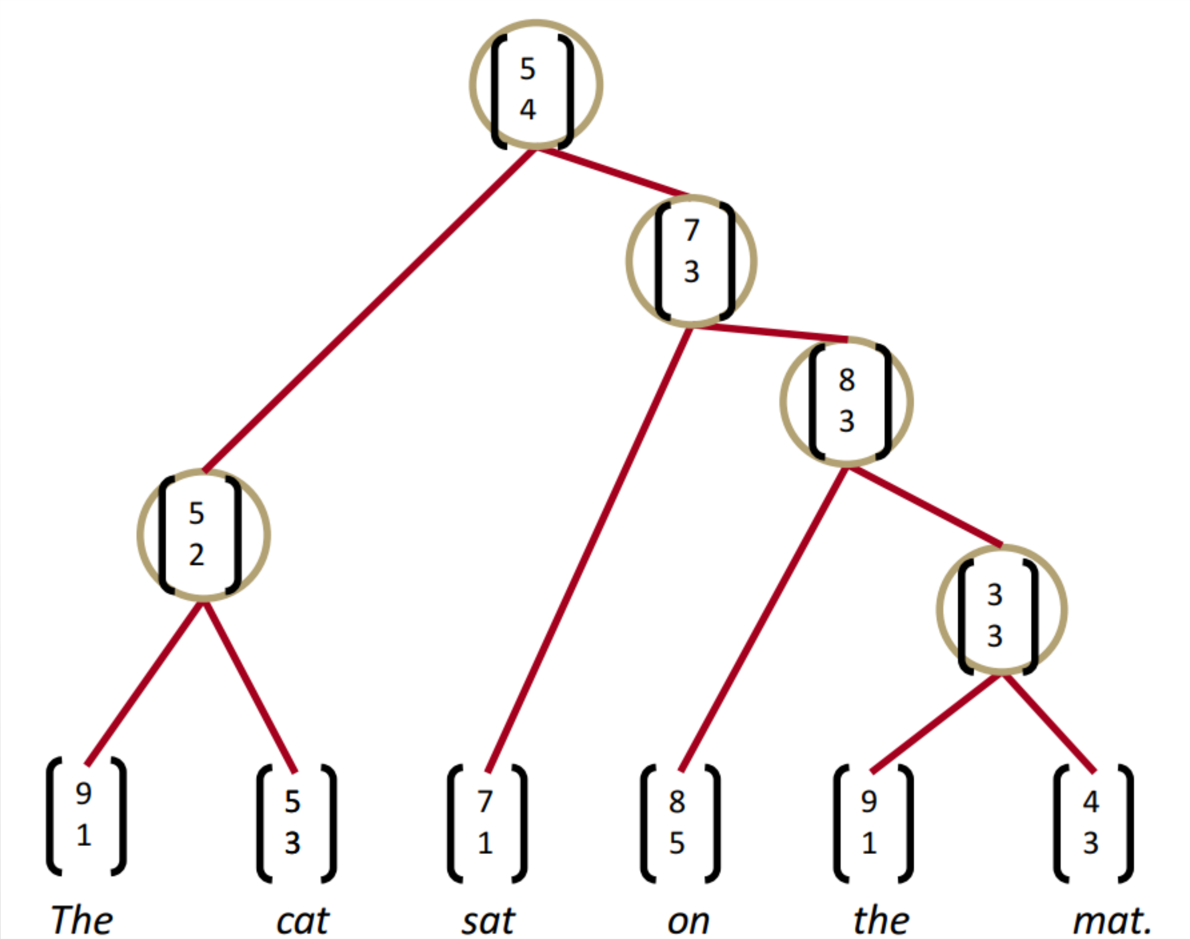
- Parsing with recursive networks



http://cs224d.stanford.edu/lectures/CS224d-Lecture10.pdf

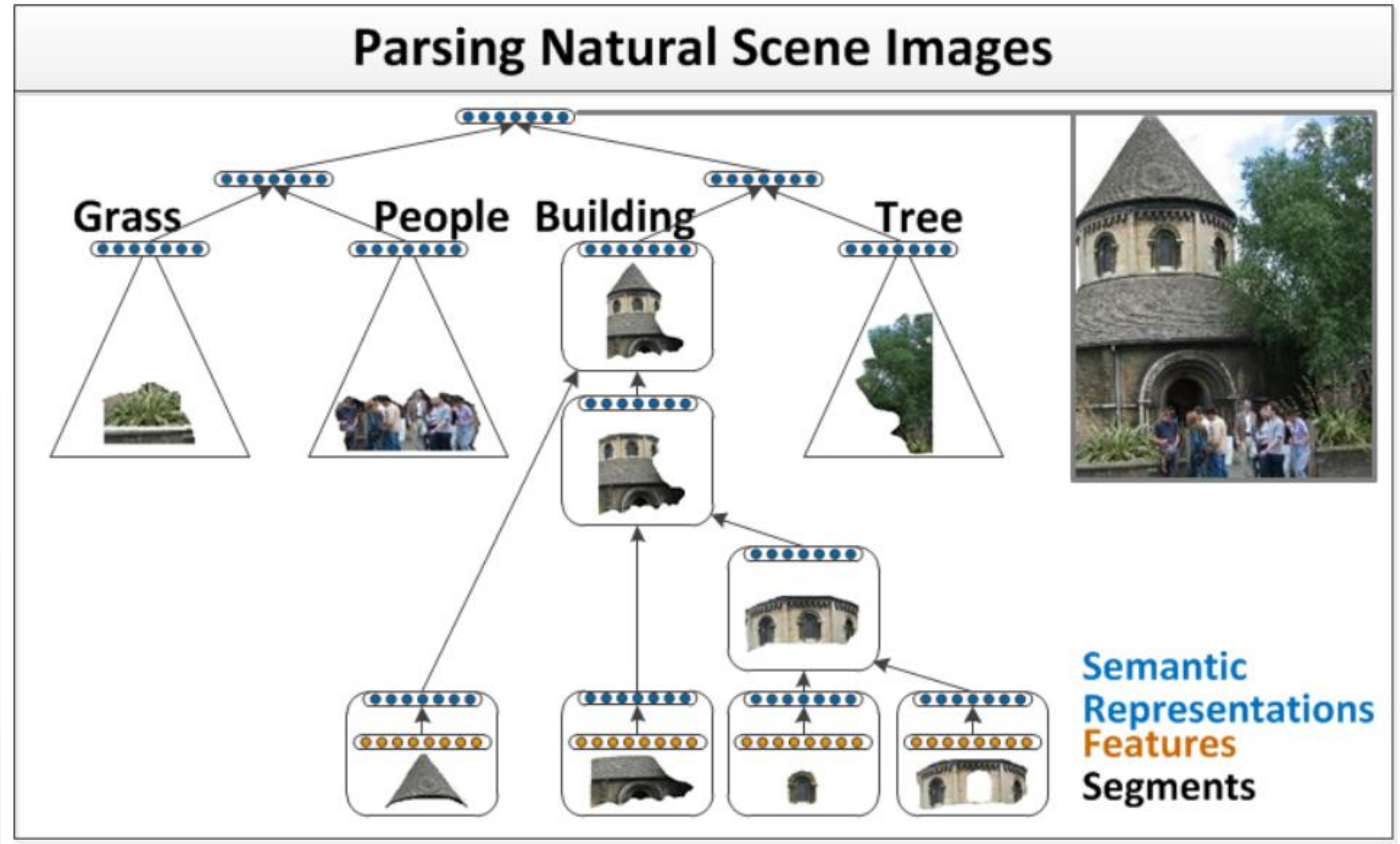
Recursive neural networks

- Parsing with recursive networks



http://cs224d.stanford.edu/lectures/CS224d-Lecture10.pdf

Recursive neural networks: Scene parsing



https://ai.stanford.edu/~ang/papers/icml11-ParsingWithRecursiveNeuralNetworks.pdf

Resources for parsing

- Universal Dependencies (UD) is a framework for consistent annotation of grammar (parts of speech, morphological features, and syntactic dependencies) across different human languages.

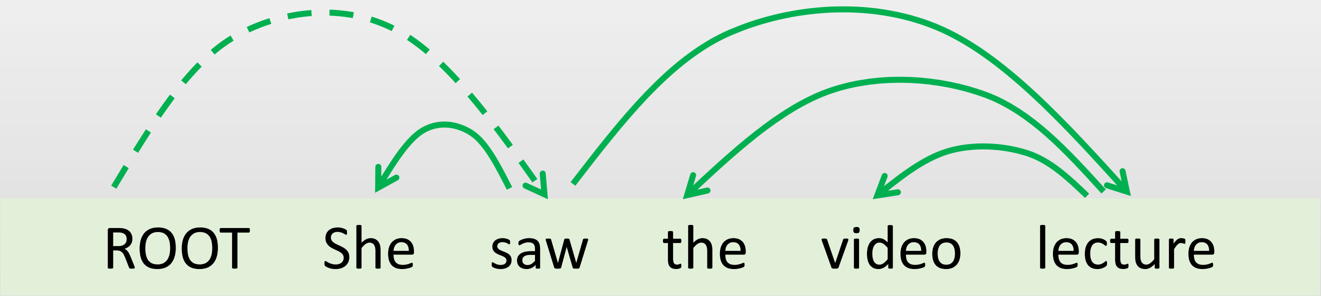
Universal Dependencies

Serbian	1	97K	IE, Slavic
Sinhala	1	<1K	IE, Indic
Skolt Sami	1	2K	Uralic, Sami
Slovak	1	106K	IE, Slavic
Slovenian	2	296K	IE, Slavic
Soi	1	<1K	IE, Iranian
South Levantine Arabic	1	<1K	Afro-Asiatic, Semitic
Spanish	3	1,022K	IE, Romance
Swedish	3	206K	IE, Germanic
Swedish Sign Language	1	1K	Sign Language
Swiss German	1	1K	IE, Germanic
Tagalog	2	1K	Austronesian, Central Philippine
Tamil	2	12K	Dravidian
Tatar	1	2K	Turkic, Northwestern
Teko	1	1K	Tupian, Maweti-Guarani
Telugu	1	6K	Dravidian
Thai	1	22K	Tai-Kadai
Tupinamba	1	4K	Tupian, Maweti-Guarani
Turkish	9	736K	Turkic, Southwestern
Turkish German	1	37K	Code switching
Ukrainian	1	122K	IE, Slavic

https://universaldependencies.org/

Parsing evaluation

- Labeled parsing accuracy (LAS)
- Unlabeled parsing accuracy (UAS)
- Acc = #correct deps/# of deps
- Example:
  - UAS = 4/5 = 0.8
  - LAS = 2/5 = 0.4



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

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

HW3

https://drive.google.com/drive/folders/1XQ5fSlgqqupXVtwCkwtYPQmLpqizXB7M?usp=share_link