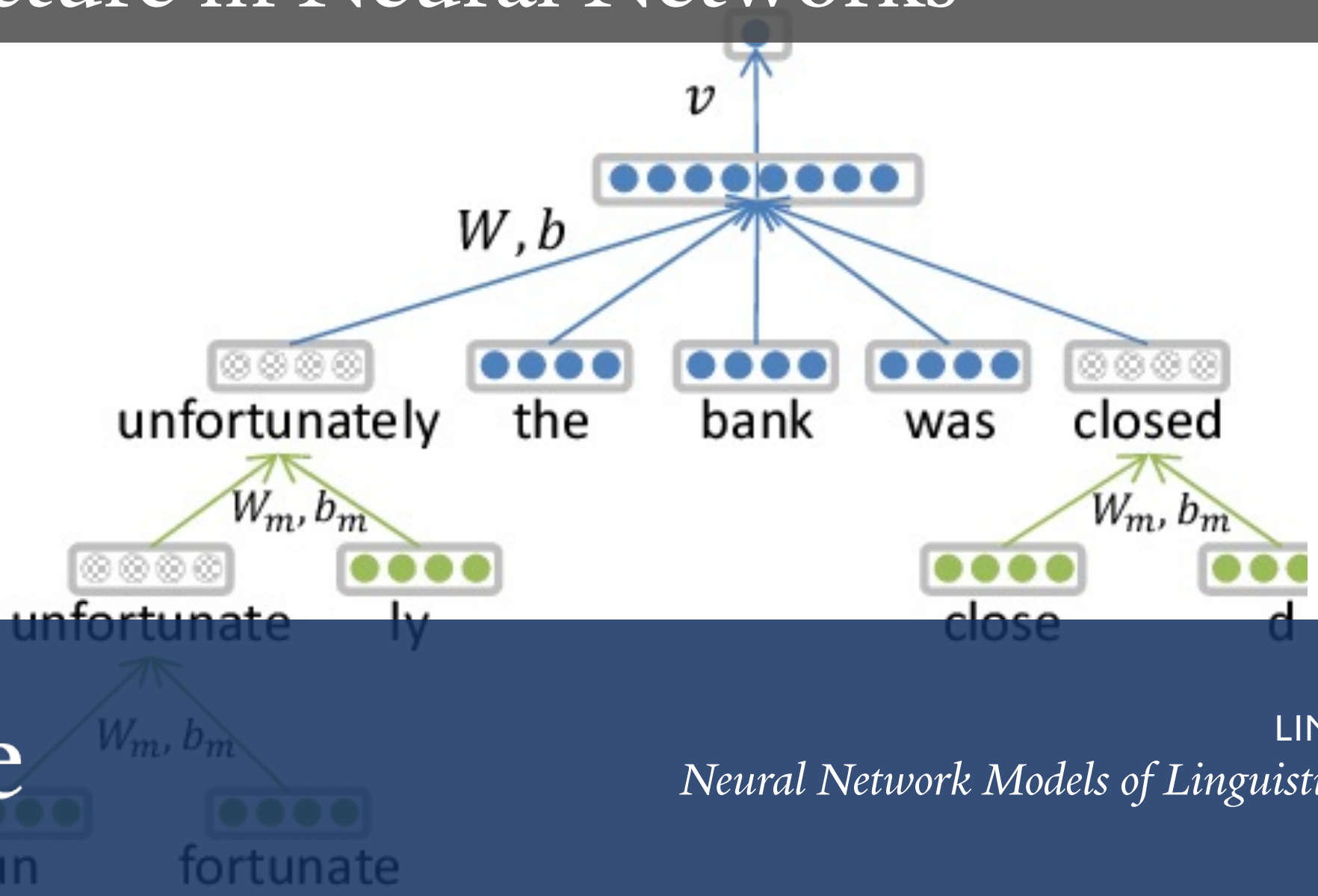
Tuesday • November 2, 2021

Structure in Neural Networks



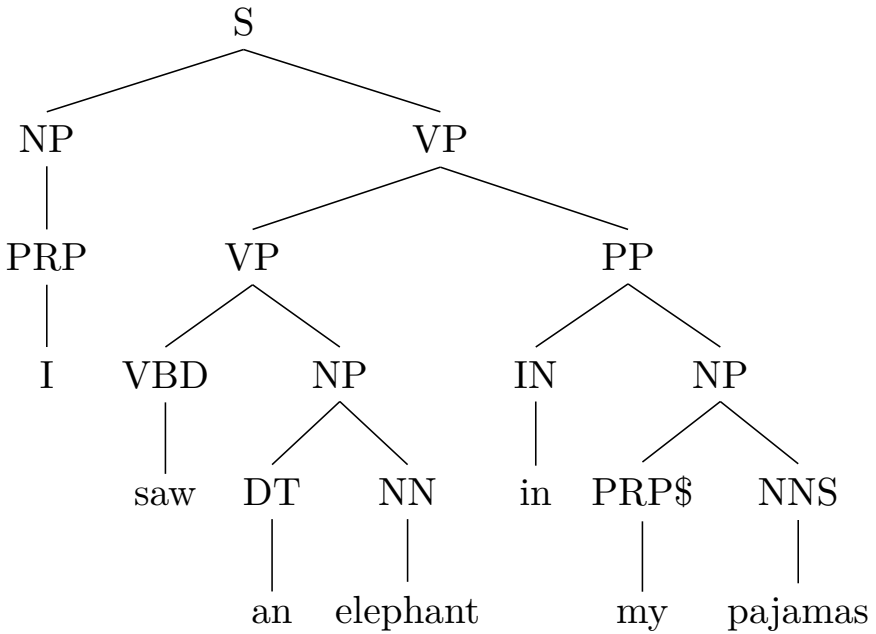
Yale

compositionality explains how the meaning of a sentence is derived from its individual components and their arrangement.

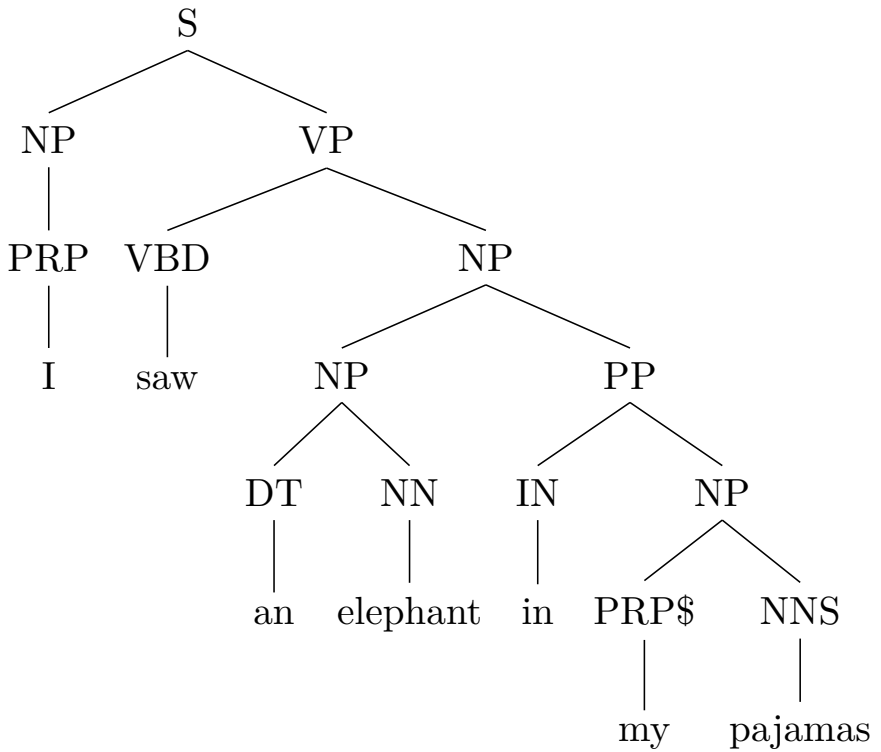
Why structure? Interpretation!

natural language can still contain ambiguity, which can lead to multiple interpretations of the same sentence.

- Compositionality: the meaning of the whole is determined by its parts and **the way they are put together**



I saw an elephant while I was wearing pajamas.



I saw an elephant that was inside my pajamas.

Why syntactic structure? Scope!

The placement of "only" changes the scope of the word "only" restricts or emphasizes., which in turn alters the meaning of the sentence

- Only
  - The bird only eats sunflower seeds.
    "only" modifies the verb "eats."
    the bird doesn't perform any other actions with sunflower seeds besides eating them.
  - The bird eats only sunflower seeds.
    both" only" modifies the noun phrase "sunflower seeds."
  - The bird eats sunflower seeds only.
    the bird exclusively eats sunflower seeds and doesn't consume any other type of food.
  - Only the bird eats sunflower seeds.

"only" modifies subject "the bird."
no other creature or entity eats sunflower seeds, just the bird.

- Not
  - That movie that is streaming on Netflix is not really worth watching.
  - That movie that is not streaming on Netflix is really worth watching.
  - That movie is not expected to be worth watching.

Why syntactic structure? Distribution!

ambiguity: prepositional phrase "in my pajamas" could be modifying either the verb "saw" or the noun "elephant."

- Structure characterizes distributional regularities

1. I saw an elephant while I was wearing pajamas.
2. I saw an elephant that was inside my pajamas.

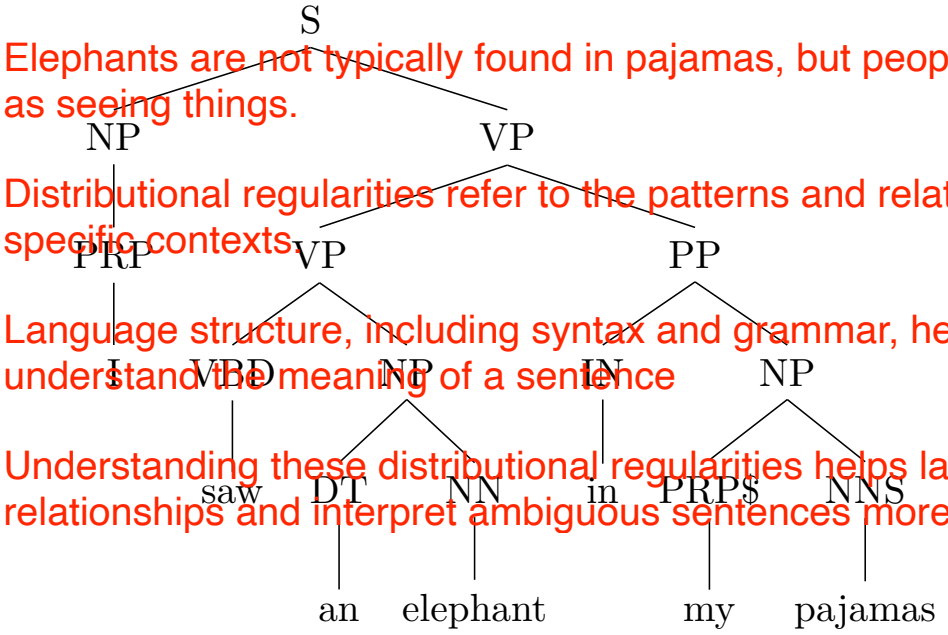
first interpretation would be considered more likely due to the distributional regularities associated with the words "elephant" and "pajamas."

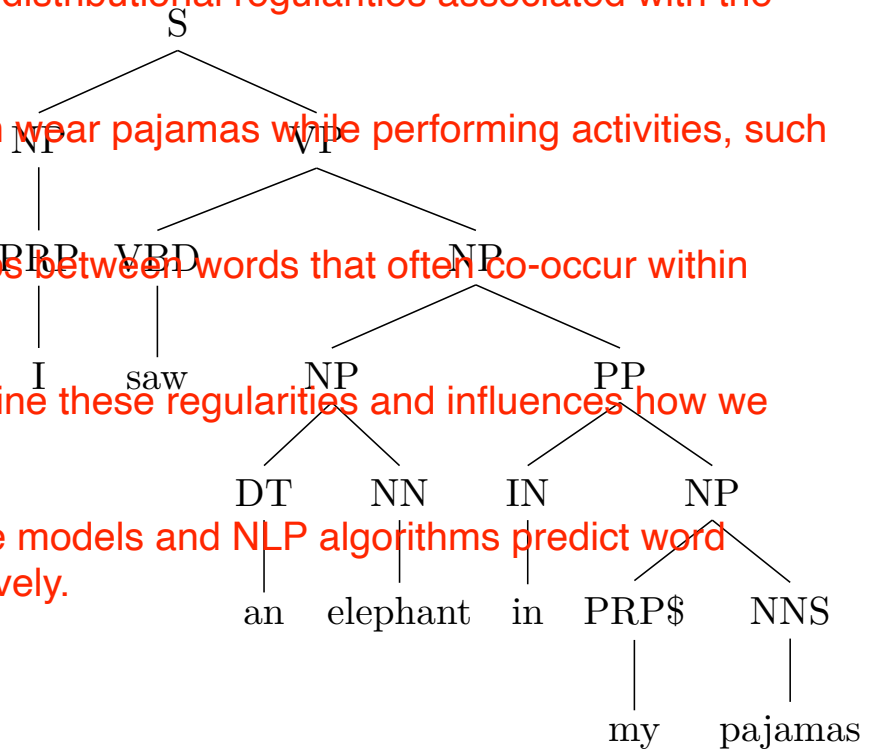
Elephants are not typically found in pajamas, but people often wear pajamas while performing activities, such as seeing things.

Distributional regularities refer to the patterns and relationships between words that often co-occur within specific contexts.

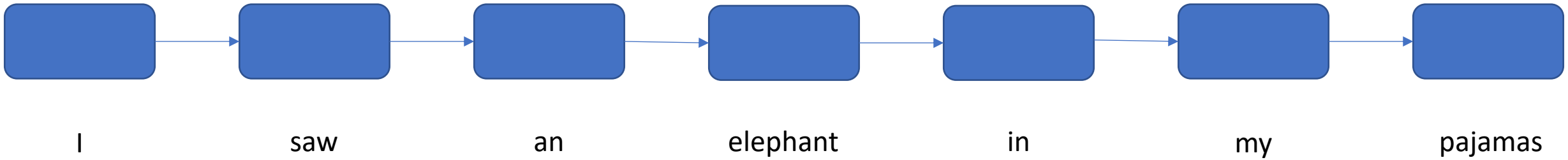
Language structure, including syntax and grammar, helps define these regularities and influences how we understand the meaning of a sentence

Understanding these distributional regularities helps language models and NLP algorithms predict word relationships and interpret ambiguous sentences more effectively.



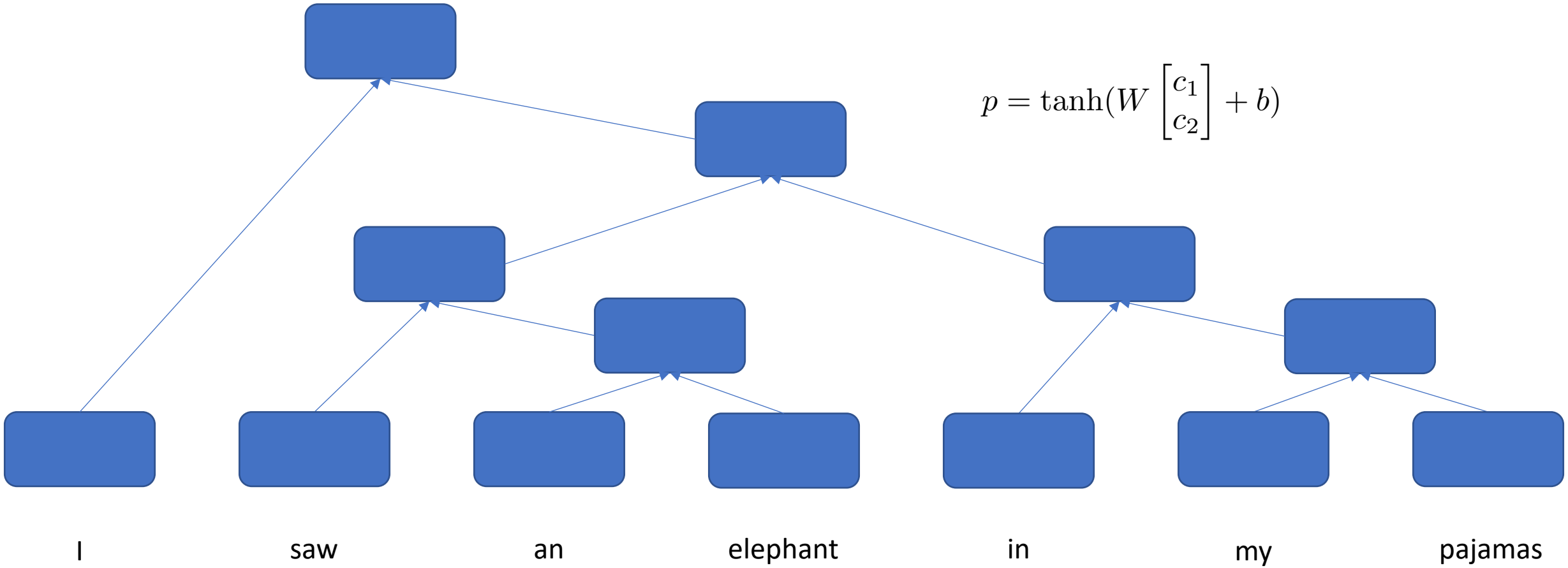


Incorporating structure in neural networks



Incorporating structure in neural networks

- Recursive Neural Networks (Socher et al.)



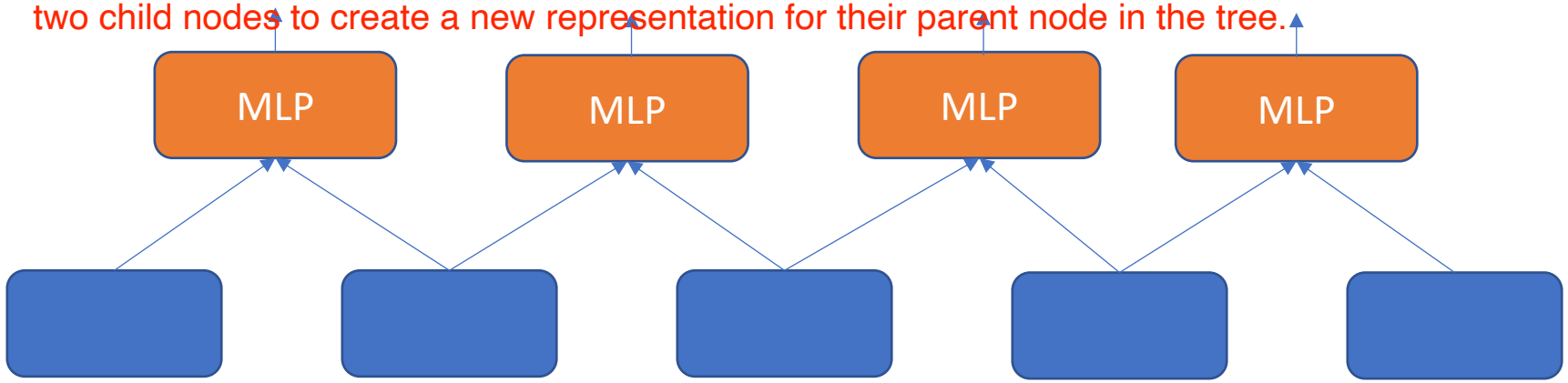
Recursive Neural Networks

Parsing is the process of analyzing and assigning a structure to a given input (e.g., a sentence in natural language). In NLP, parsing often refers to syntactic parsing, which involves determining the syntactic structure of a sentence, such as a parse tree or a dependency graph. The parse tree reveals the hierarchical organization of the sentence, capturing the relationships between words, phrases, and clauses.

- Where does the structure come from?
  - Parsing (structure assignment) through binary composition

Binary composition is an approach to combine two elements (e.g., words or phrases) into a single, more complex element, such as a phrase or a larger syntactic constituent.

In the context of tree-structured models like Recursive Neural Networks (RecNNs) or Recursive Neural Tensor Networks (RNTNs), binary composition refers to the process of combining the representations of two child nodes to create a new representation for their parent node in the tree.

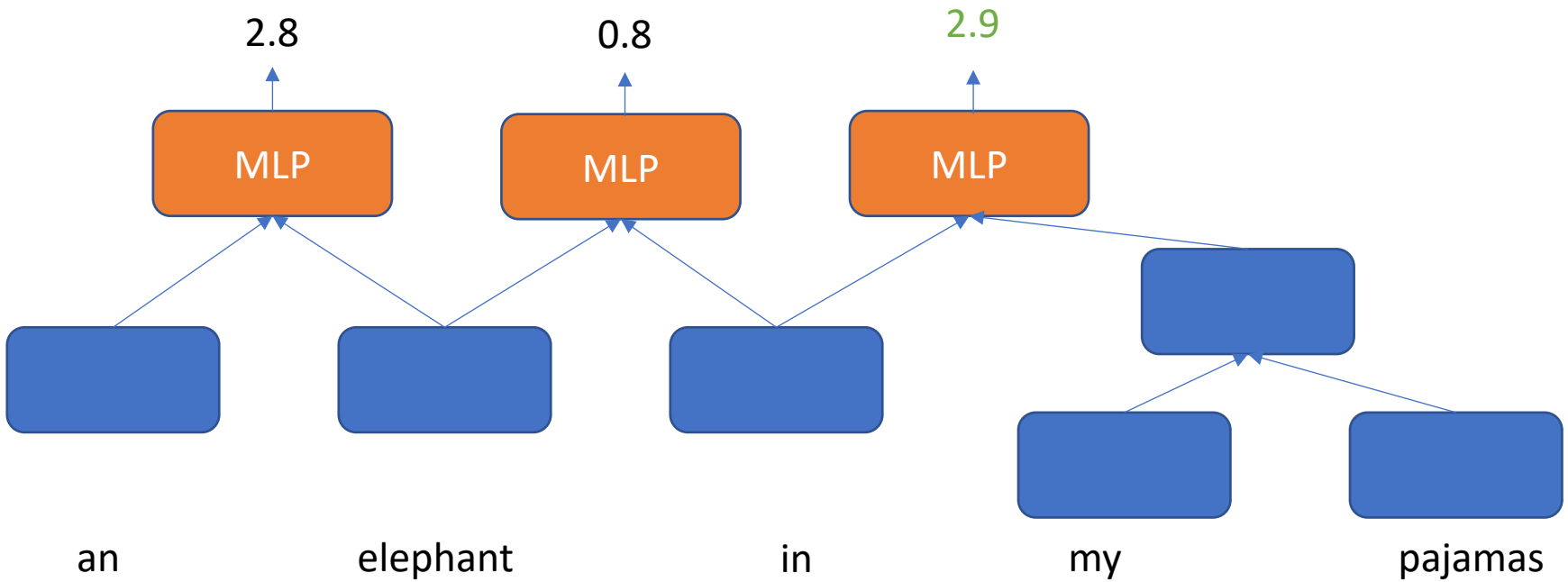


… an elephant in my pajamas

Recursive Neural Networks

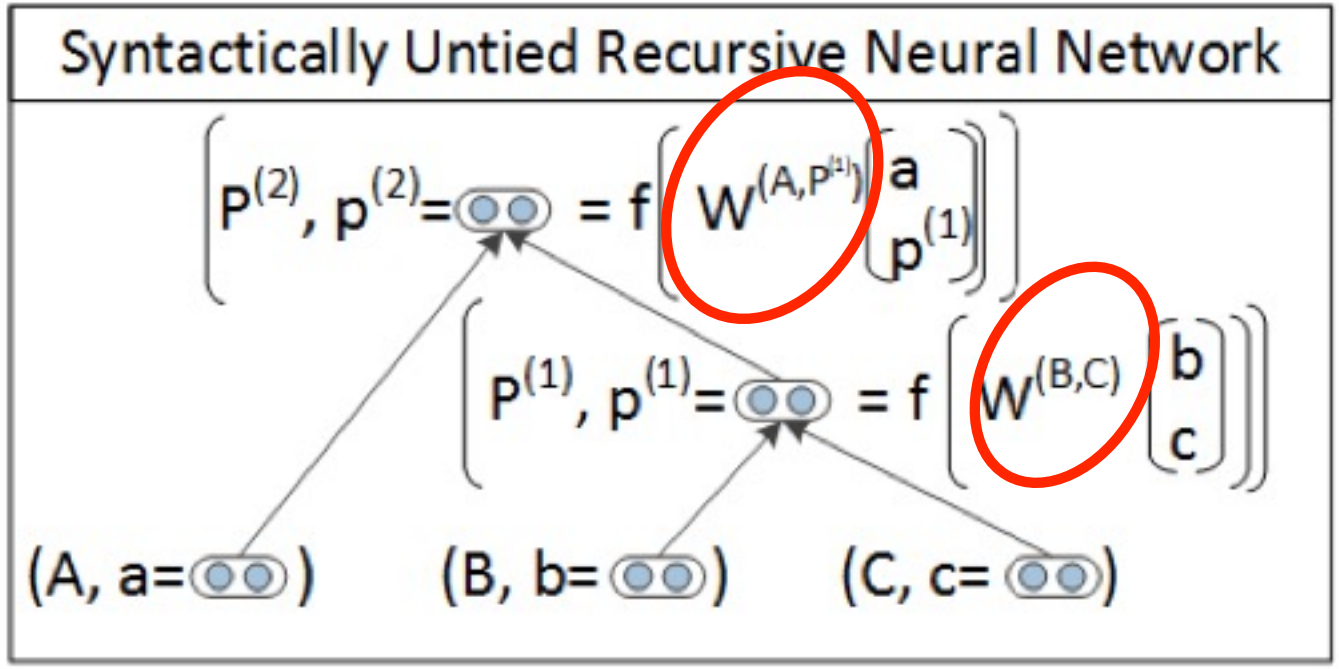
- Where does the structure come from?
  - Parsing through binary combination

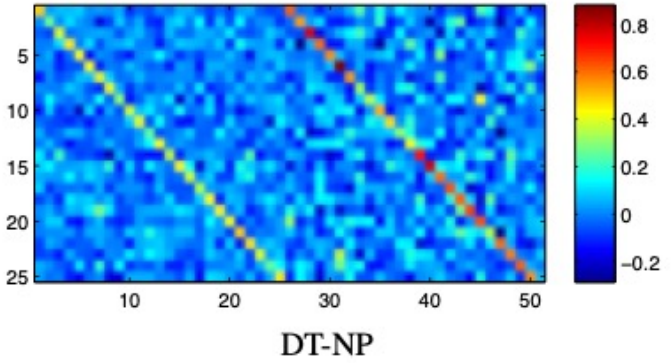
- Greedy parsing
- Beam search

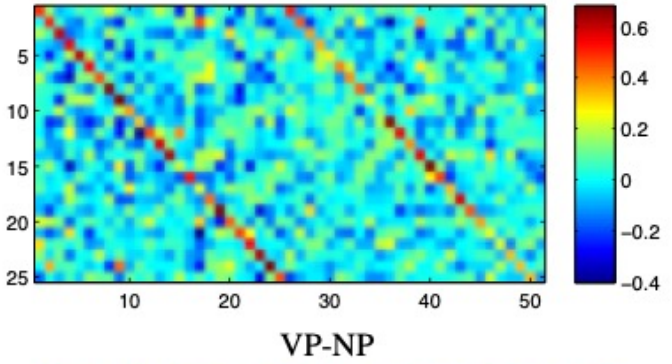


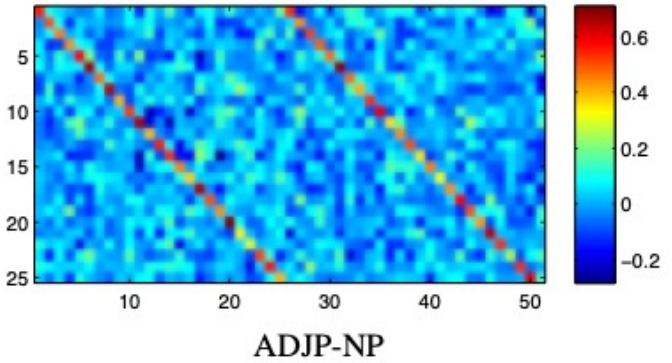
Recursive Neural Networks

- Composition matrix need not be uniform



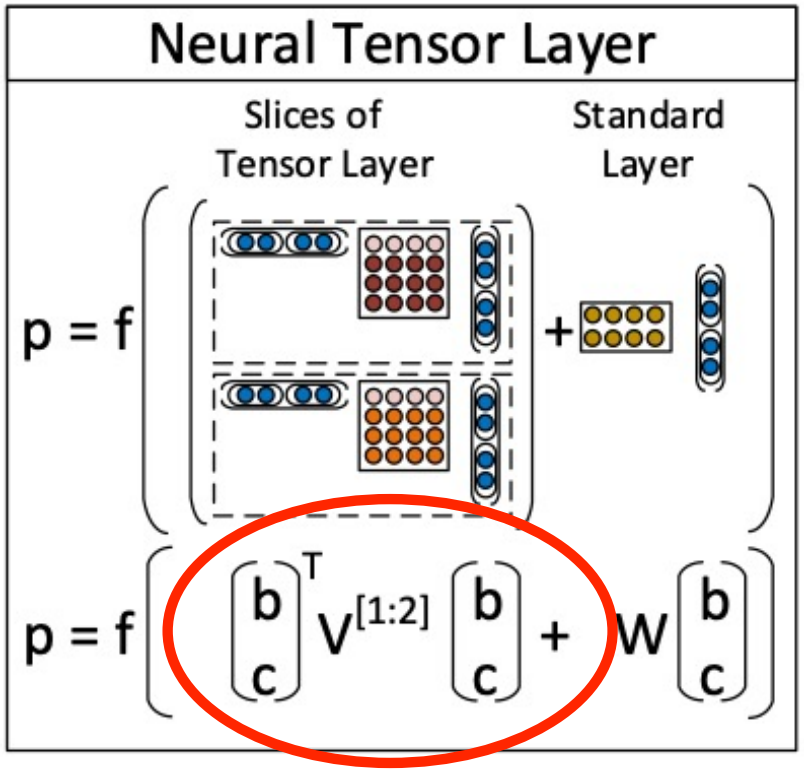






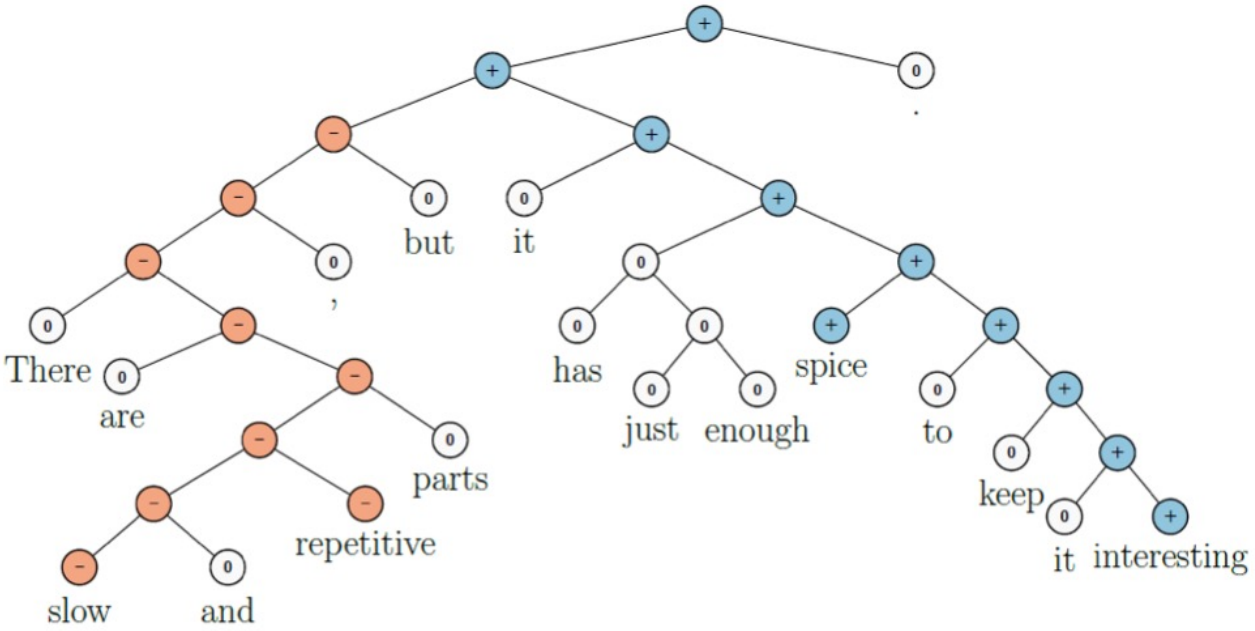
Recursive Neural Networks

- Multiplicative interactions (Recursive Neural Tensor Network)



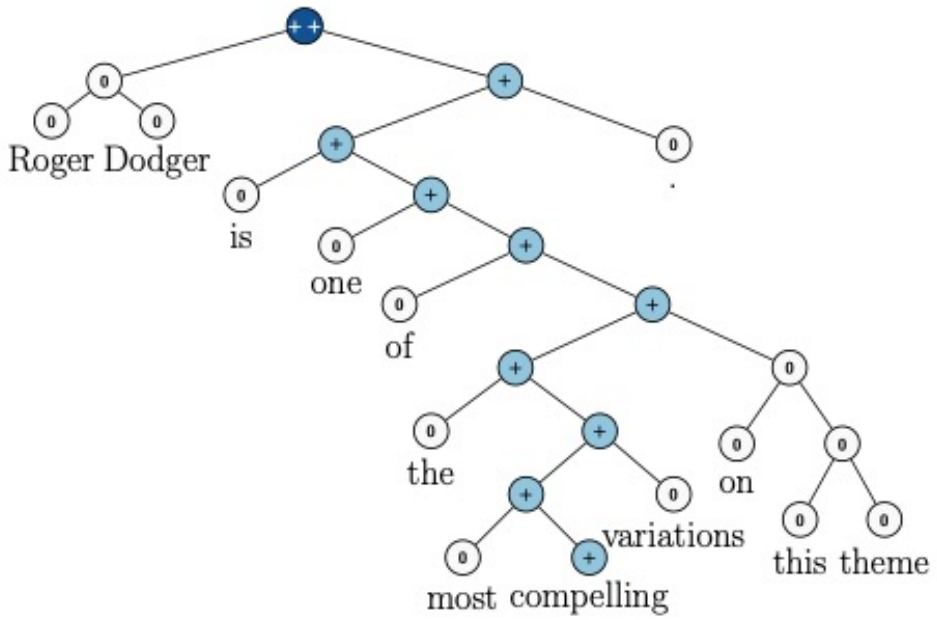
Applications to Sentiment Analysis

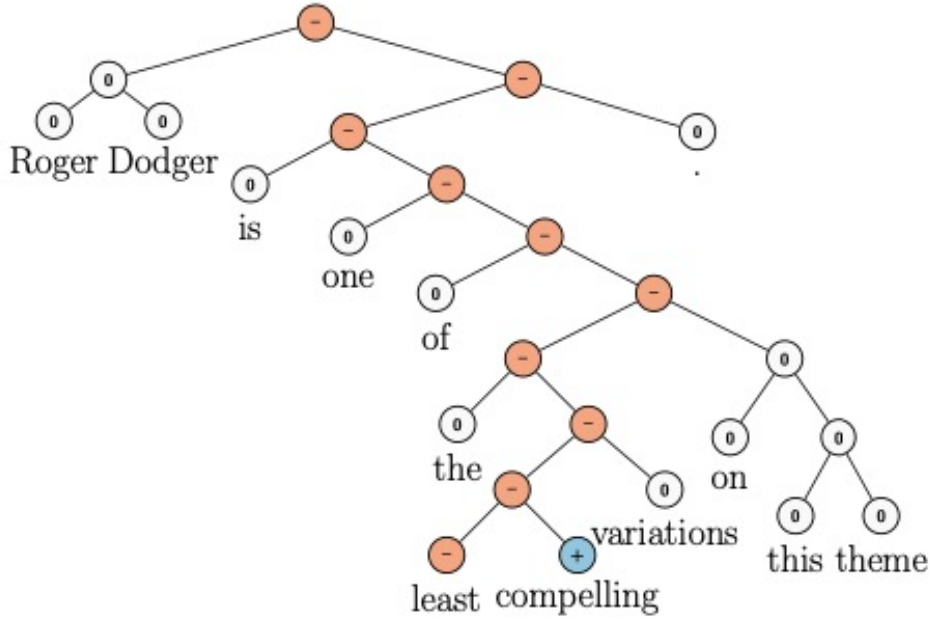
- Stanford Sentiment Treebank
  - 215,154 phrases labeled for sentiment in 11,855 sentences (5 levels)



Applications to Sentiment Analysis

- Stanford Sentiment Treebank
  - 215,154 phrases labeled for sentiment in 11,855 sentences (5 levels)

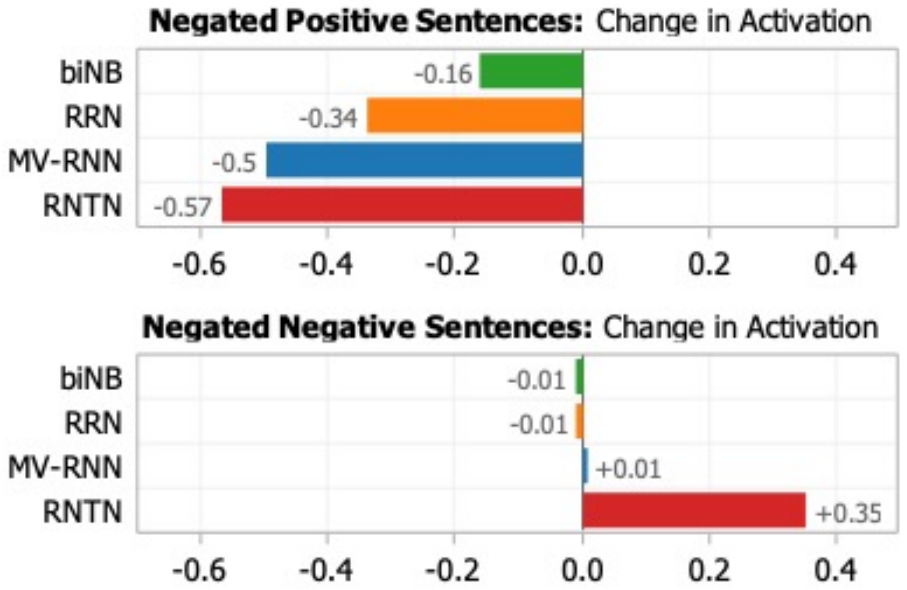




Applications to Sentiment Analysis

- Performance

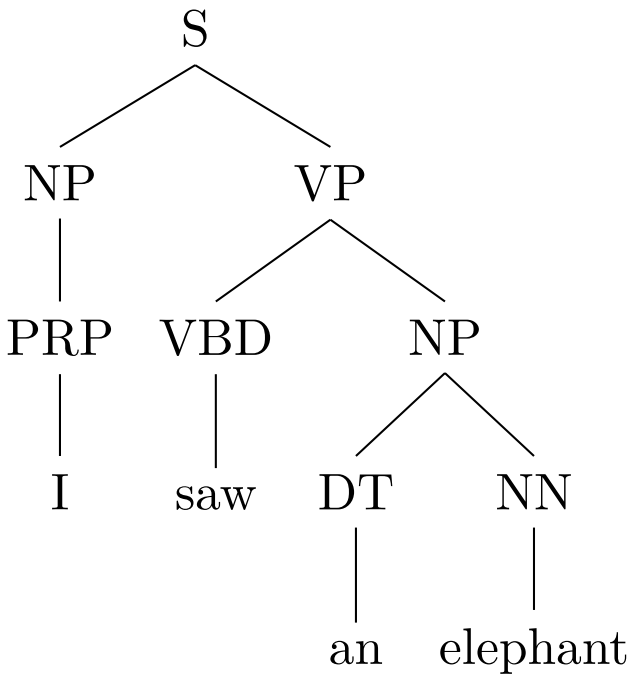
Model	Fine-grained		Positive/Negative	
	All	Root	All	Root
NB	67.2	41.0	82.6	81.8
SVM	64.3	40.7	84.6	79.4
BiNB	71.0	41.9	82.7	83.1
VecAvg	73.3	32.7	85.1	80.1
RNN	79.0	43.2	86.1	82.4
MV-RNN	78.7	44.4	86.8	82.9
RNTN	**80.7**	**45.7**	**87.6**	**85.4**



- BERT gets 92% for binary classification (but with much more pre-training data and many more parameters)

Adding Structure, indirectly

Hierarchical Structure and Push-down Automata

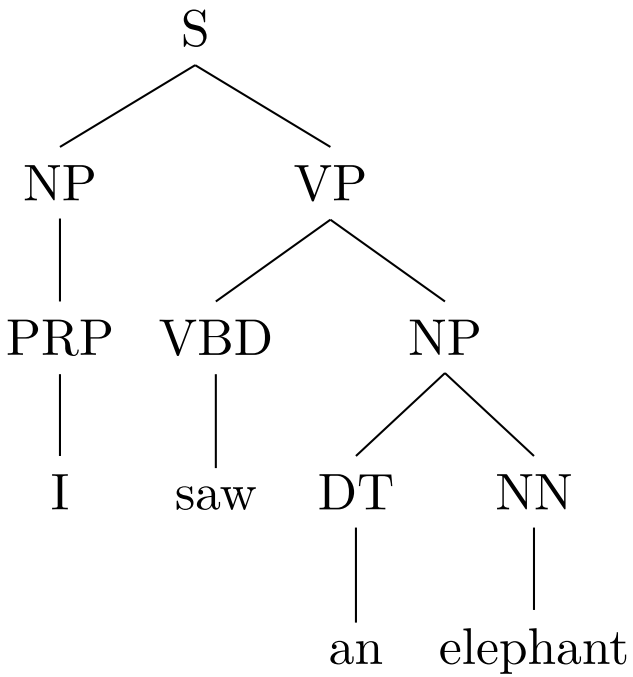


I	saw	an	elephant

Buffer

Push-down Stack

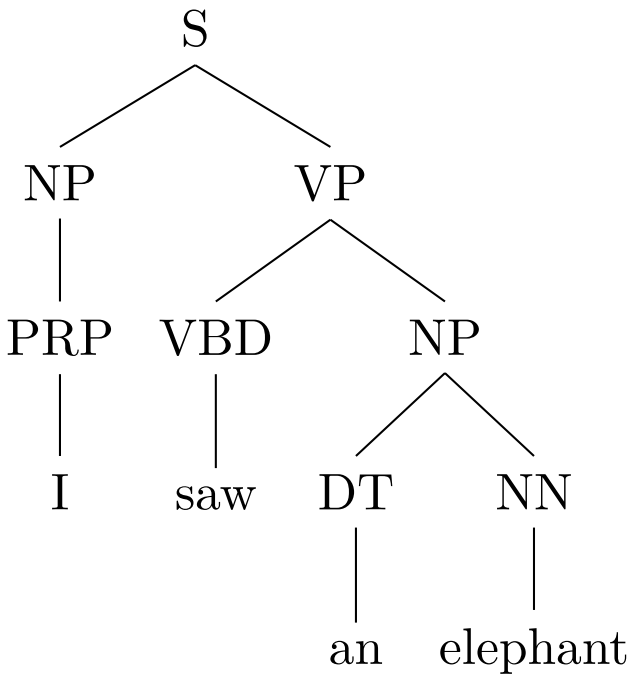
Hierarchical Structure and Push-down Automata



SHIFT

saw	an	elephant	

Hierarchical Structure and Push-down Automata

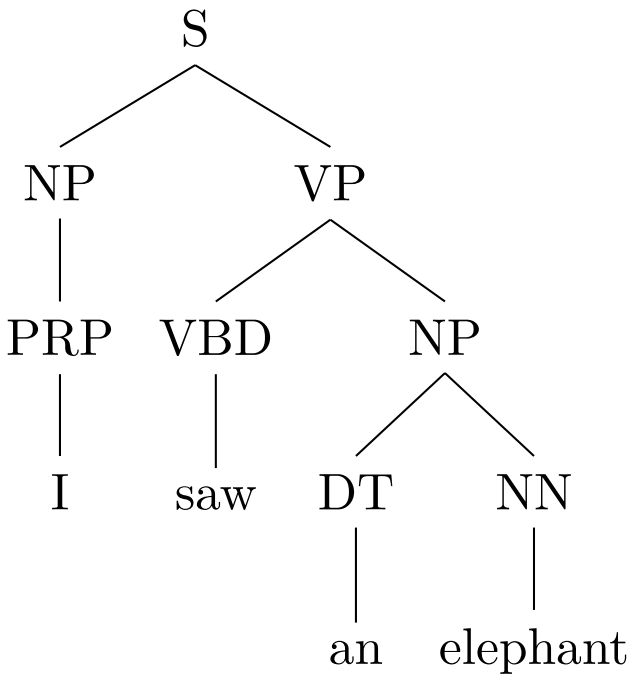


REDUCE

saw	an	elephant	

[PRP I]

Hierarchical Structure and Push-down Automata

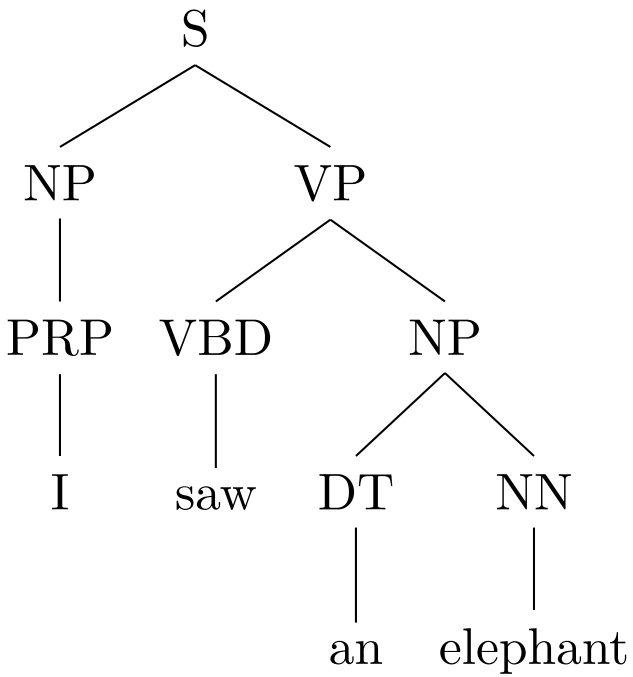


REDUCE

saw	an	elephant	

[NP [PRP I]]

Hierarchical Structure and Push-down Automata



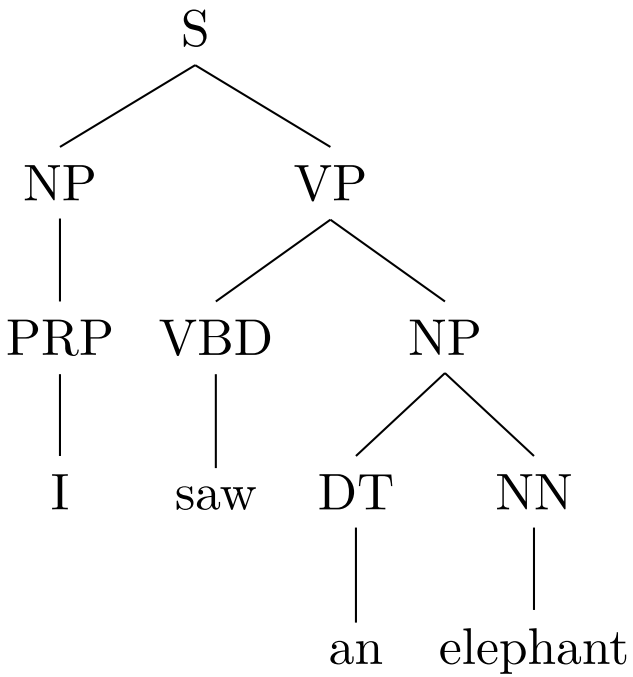
SHIFT

an	**elephant**		

saw

[NP [PRP I]]

Hierarchical Structure and Push-down Automata



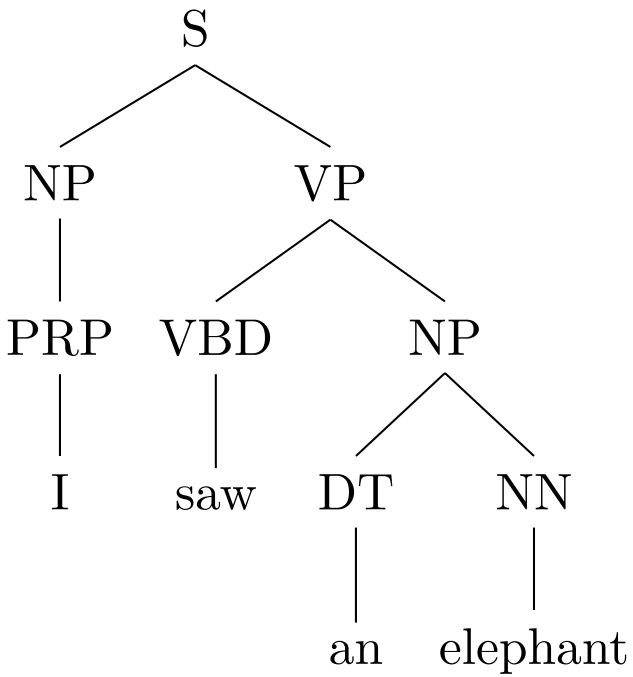
REDUCE

an	**elephant**		

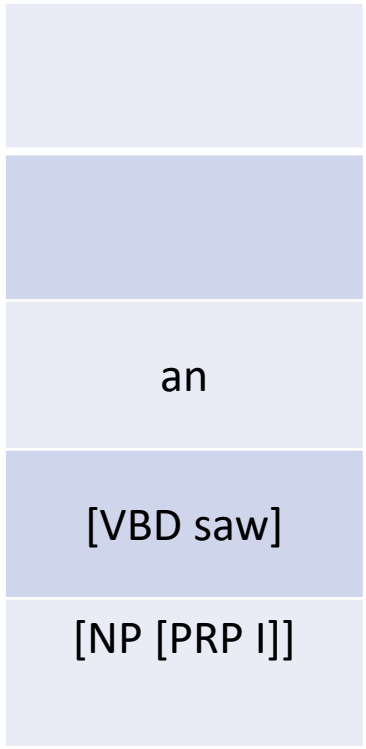
[VBD saw]

[NP [PRP I]]

Hierarchical Structure and Push-down Automata

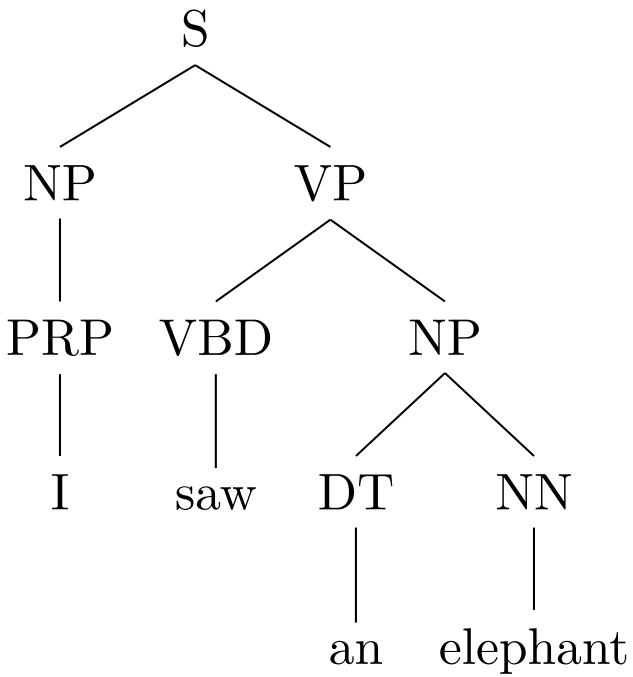


SHIFT



elephant			

Hierarchical Structure and Push-down Automata



REDUCE

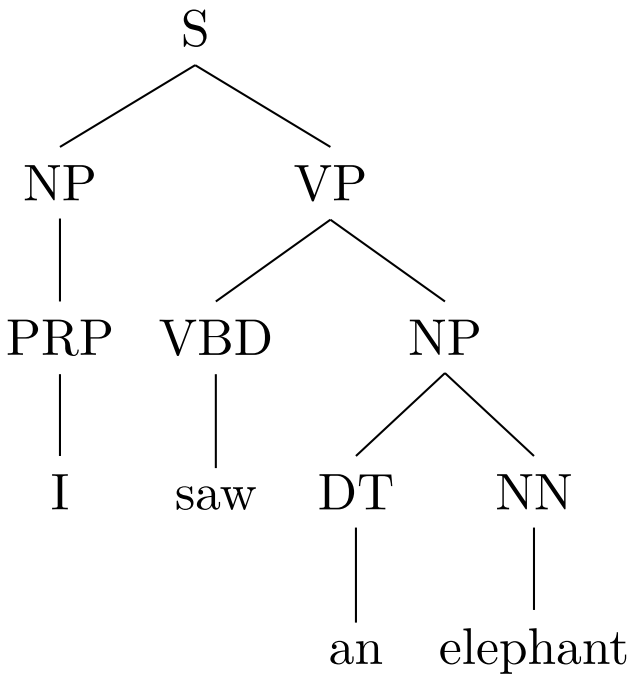
elephant				

[DT an]

[VBD saw]

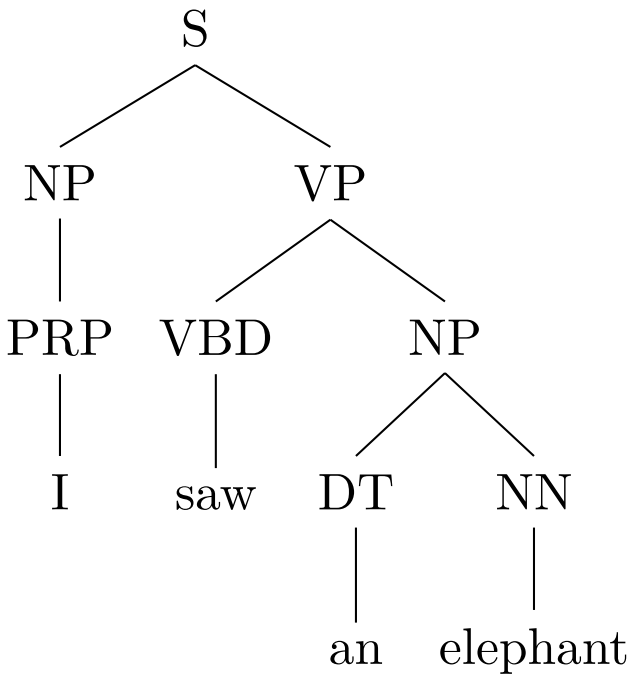
[NP [PRP I]]

Hierarchical Structure and Push-down Automata



SHIFT

Hierarchical Structure and Push-down Automata



REDUCE

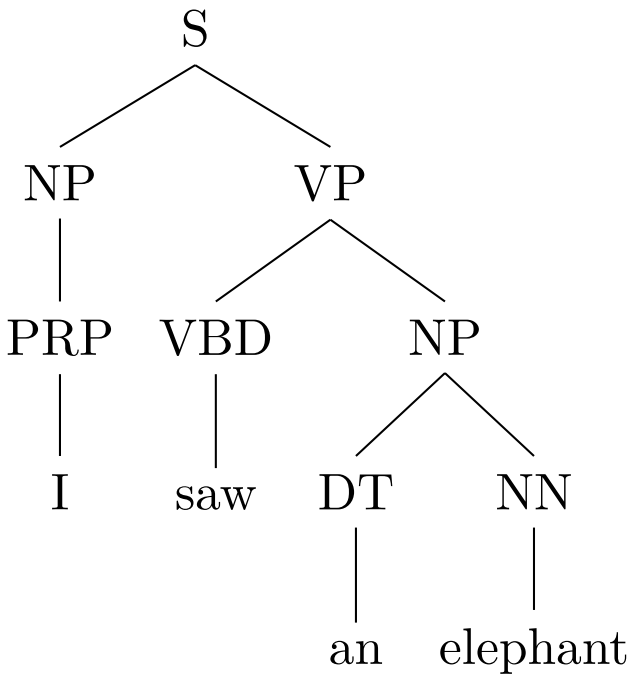
[NN elephant]

[DT an]

[VBD saw]

[NP [PRP I]]

Hierarchical Structure and Push-down Automata



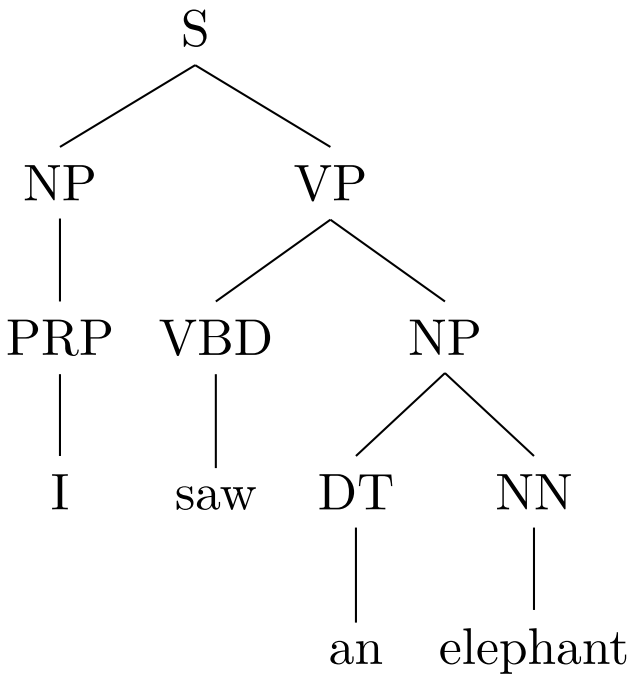
REDUCE

[NP [DT an] [NN elephant]]

[VBD saw]

[NP [PRP I]]

Hierarchical Structure and Push-down Automata

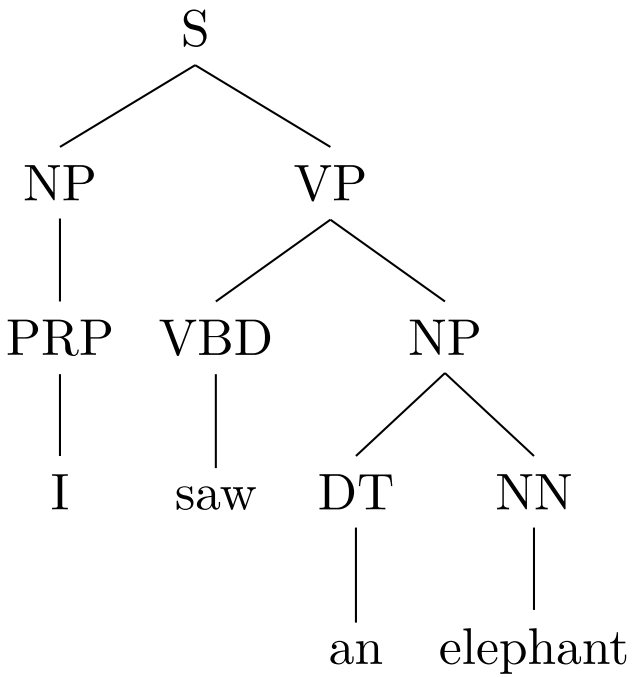


REDUCE

[VP [VBD saw] [NP [DT an] [NN elephant]]]

[NP [PRP I]]

Hierarchical Structure and Push-down Automata



REDUCE

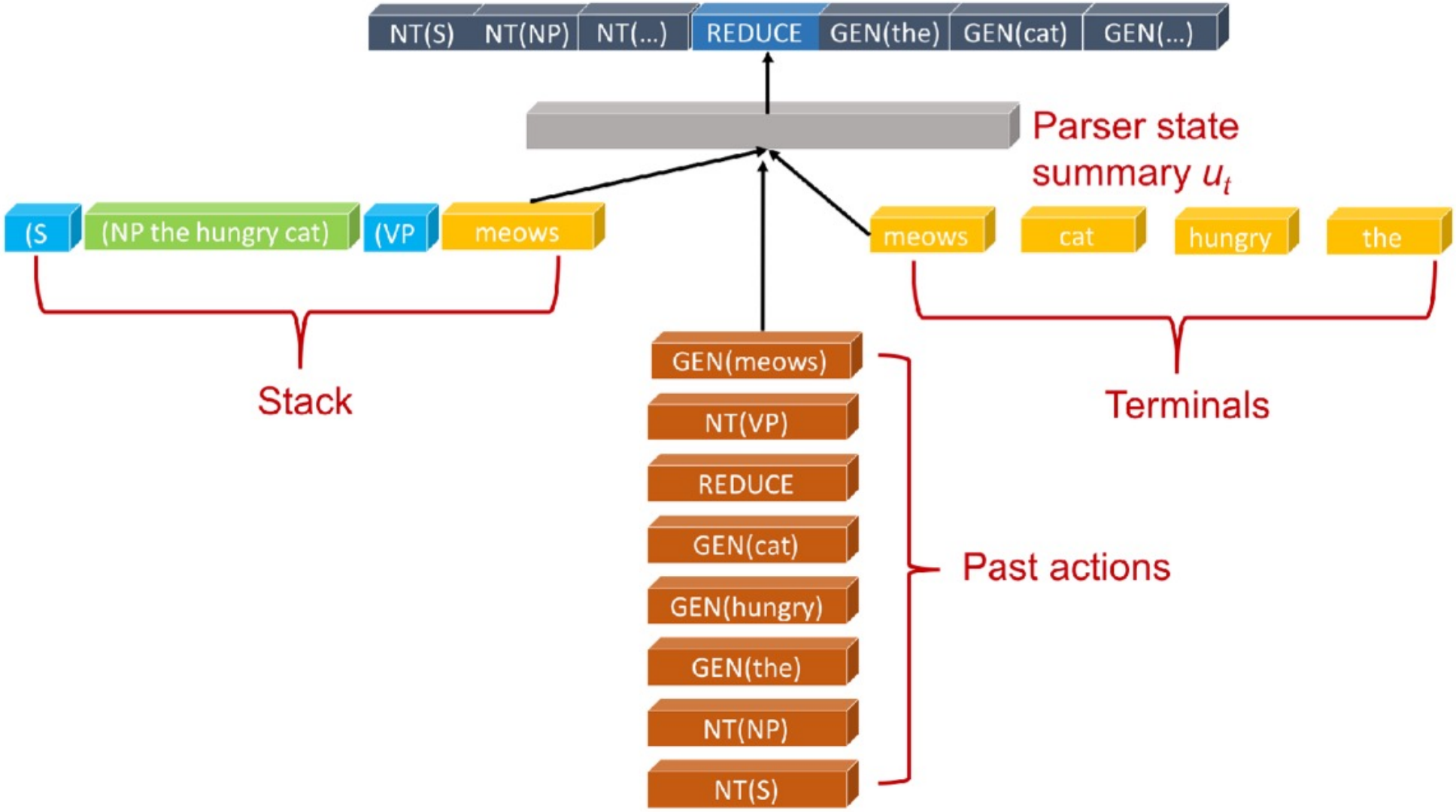
[S [NP [PRP I]] [VP [VBD saw] [NP [DT an] [NN elephant]]]]

Recurrent Neural Network Grammars (RNNGs): generating with a push-down stack

(S (NP the hungry cat) (VP meows) .)

No. Steps	Stack	Terminals	Action
0			NT(S)
1	(S		NT(NP)
2	(S | (NP		GEN(*the*)
3	(S | (NP | *the*	*the*	GEN(*hungry*)
4	(S | (NP | *the* | *hungry*	*the hungry*	GEN(*cat*)
5	(S | (NP | *the* | *hungry* | *cat*	*the hungry cat*	REDUCE
6	(S | (NP *the hungry cat*)	*the hungry cat*	NT(VP)

RNNG architecture



Encoding the context

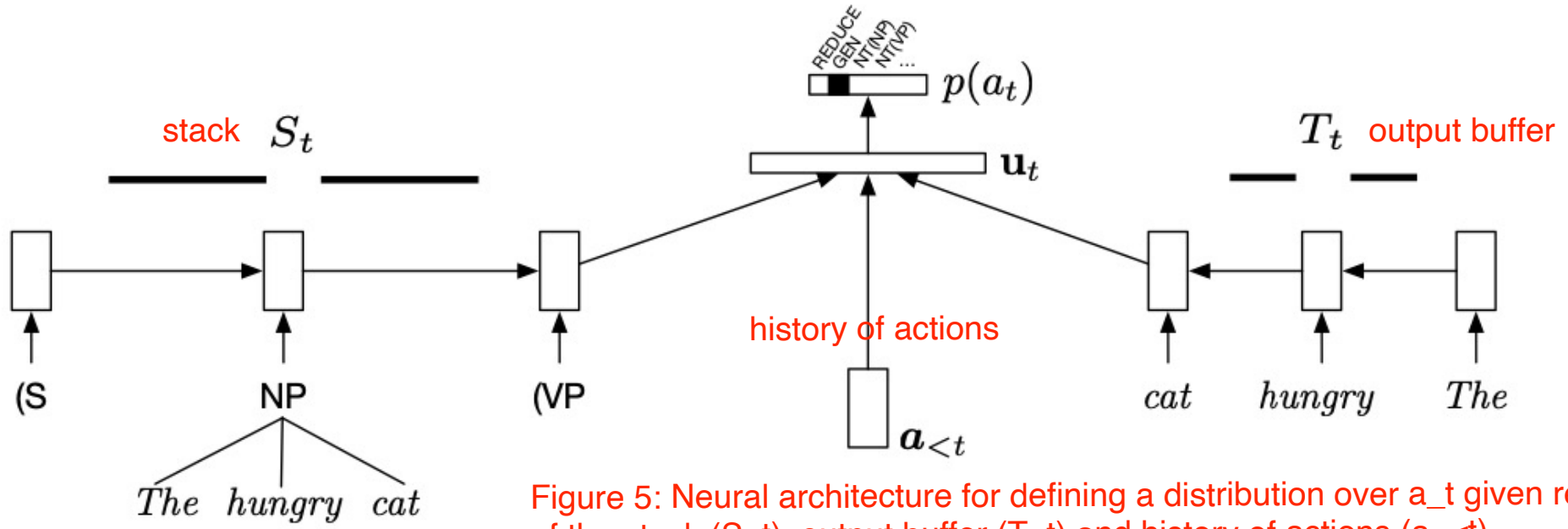


Figure 5: Neural architecture for defining a distribution over a_t given representations of the stack (S_t), output buffer (T_t) and history of actions (a_<t).
generator state at line 7 of Figure 4. (S l (NP The hungry cat) l (VP

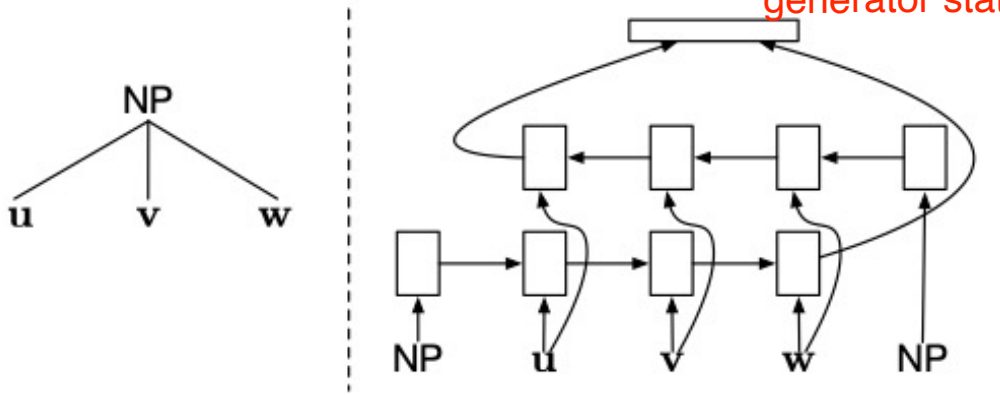
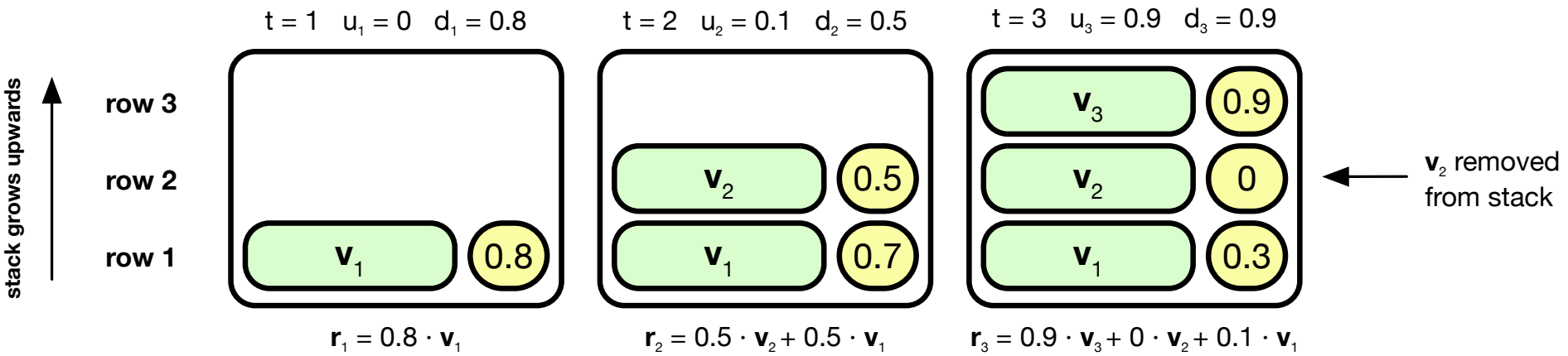


Figure 4: Syntactic composition function based on bidirectional LSTMs that is executed during a REDUCE operation; the
network on the right models the structure on the left.

RNNGs as language models

Model	**test ppl** (PTB)	**test ppl** (CTB)
IKN 5-gram	169.3	255.2
LSTM LM	113.4	207.3
RNNG	102.4	171.9

Putting the stack in the model (Greffenstette et al., 2015, Hao et al. 2018)



Inputs: vector v

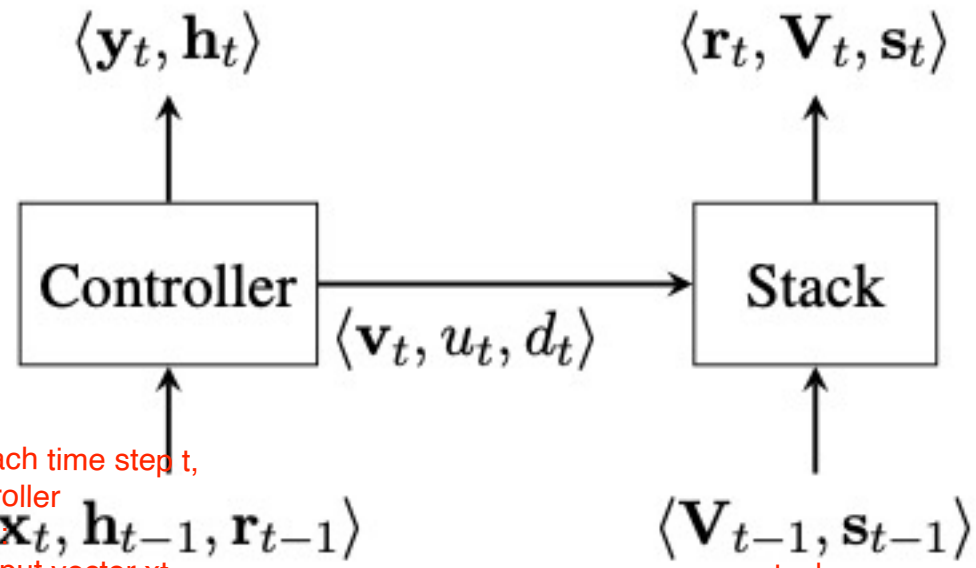
pop amount u

push amount v

Output: vector r

A continuous push-down stack

A Neural Stack model consists of 2 components:
a controller executing computation of the network
a stack implementing data model of the network.



At each time step t,
controller
input:
an input vector xt
a recurrent state vector ht−1
a read vector rt−1 representing material at top of stack at the end of previous time step.

output:
an output yt
a new recurrent state vector ht
a tuple <vt, ut, dt> containing instructions for manipulating the stack

stack
input:
a tuple <vt, ut, dt> instruction
output:
r_t: material at the top of the stack after popping and pushing operations
a recurrent state matrix Vt: contents of the stack
a strength vector st

Linear controller

$$u_t = \sigma\left(\mathbf{W}_u \cdot \left[\begin{array}{c|c} \mathbf{x}_t & \mathbf{r}_{t-1} \end{array}\right]^\top + b_u\right)$$

$$d_t = \sigma\left(\mathbf{W}_d \cdot \left[\begin{array}{c|c} \mathbf{x}_t & \mathbf{r}_{t-1} \end{array}\right]^\top + b_d\right)$$

$$\mathbf{v}_t = \sigma\left(\mathbf{W}_\mathbf{v} \cdot \left[\begin{array}{c|c} \mathbf{x}_t & \mathbf{r}_{t-1} \end{array}\right]^\top + \mathbf{b}_\mathbf{v}\right)$$

$$\mathbf{y}_t = \mathbf{W}_\mathbf{y} \cdot \left[\begin{array}{c|c} \mathbf{x}_t & \mathbf{r}_{t-1} \end{array}\right]^\top + \mathbf{b}_\mathbf{y}$$

The linear controller is a feedforward network consisting of a single linear layer.
The network output is directly extracted from the linear layer
stack instructions are passed through the sigmoid function

Transduction (mapping) tasks (Hao et al., 2018)

- String Reversal
- XOR evaluation
- Parenthesis prediction

$$S \rightarrow S\ T \mid T\ S \mid T$$
$$T \rightarrow (\ T\ ) \mid (\ )$$
$$T \rightarrow [\ T\ ] \mid [\ ]$$

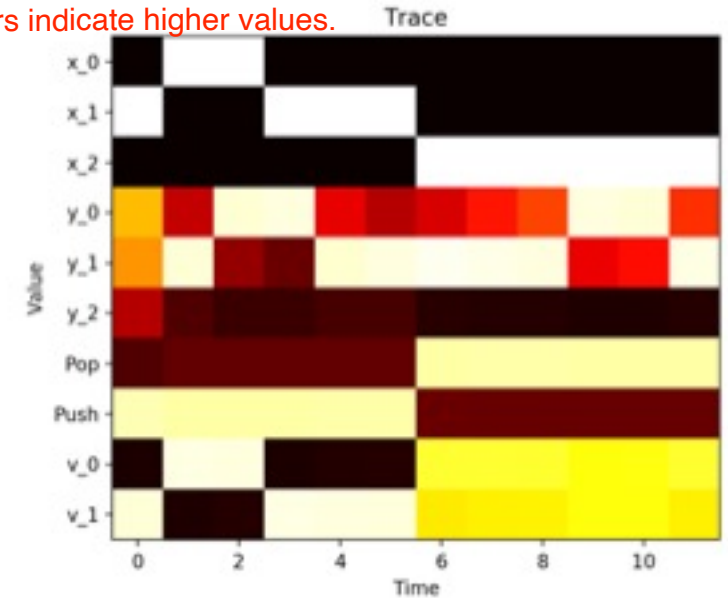
Mapping tasks results

Task	Buffered	Controller	Stack	Min	Med	Max	Min	Med	Max
Reversal	No	Linear	Yes	49.9	**100.0**	**100.0**	49.3	**100.0**	**100.0**
Reversal	Yes	Linear	Yes	55.3	98.7	99.4	49.5	60.4	74.7
Reversal	No	LSTM	Yes	81.2	89.3	94.4	**67.2**	71.0	73.7
Reversal	No	LSTM	No	**83.0**	86.5	92.5	64.8	68.6	73.3
XOR	No	Linear	Yes	51.1	53.5	54.4	50.7	51.9	51.9
XOR	No	LSTM	Yes	**100.0**	**100.0**	**100.0**	**99.7**	**100.0**	**100.0**
XOR	Yes	Linear	Yes	51.0	99.8	**100.0**	50.4	96.0	99.1
Delayed XOR	No	Linear	Yes	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Parenthesis	No	Linear	Yes	72.8	97.0	99.3	59.9	80.3	83.2
Parenthesis	No	Linear	No	70.0	71.8	73.3	59.9	60.5	60.7
Parenthesis	No	LSTM	Yes	**100.0**	**100.0**	**100.0**	**85.8**	**86.8**	**88.9**
Parenthesis	No	LSTM	No	**100.0**	**100.0**	**100.0**	83.5	85.8	88.0

Diagrams of network computation on the Reversal task with linear and LSTM controllers.
In each diagram, the input may consist of 2 or 4 distinct alphabet symbols, but only the symbols 0 and 1 are included in the output.

Reversal: looking inside the network

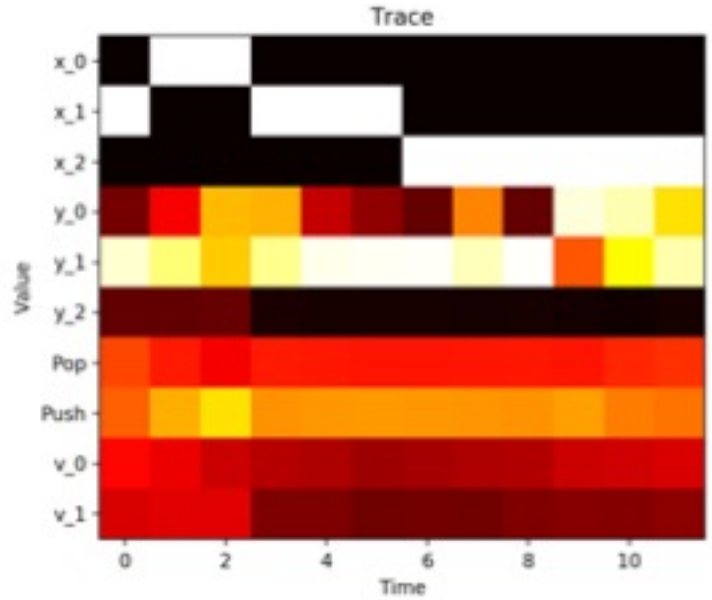
Columns indicate the pop strengths, push strengths, and pushed vectors throughout the course of the computation, along with the input and predicted output in one-hot notation.
Lighter colors indicate higher values.



2 symbols, linear controller

Input: 100111# . . . #

Output: . . . 111001

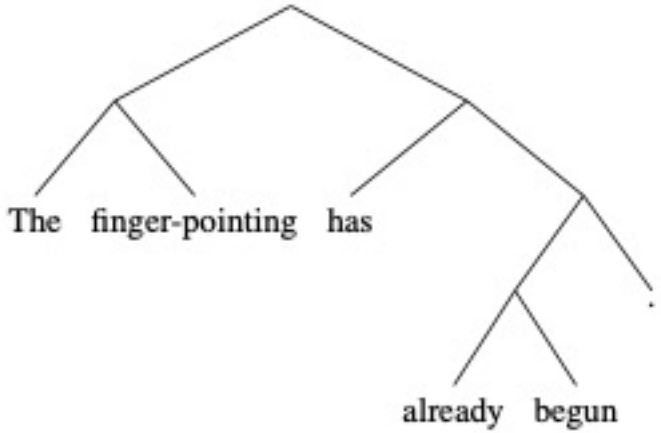


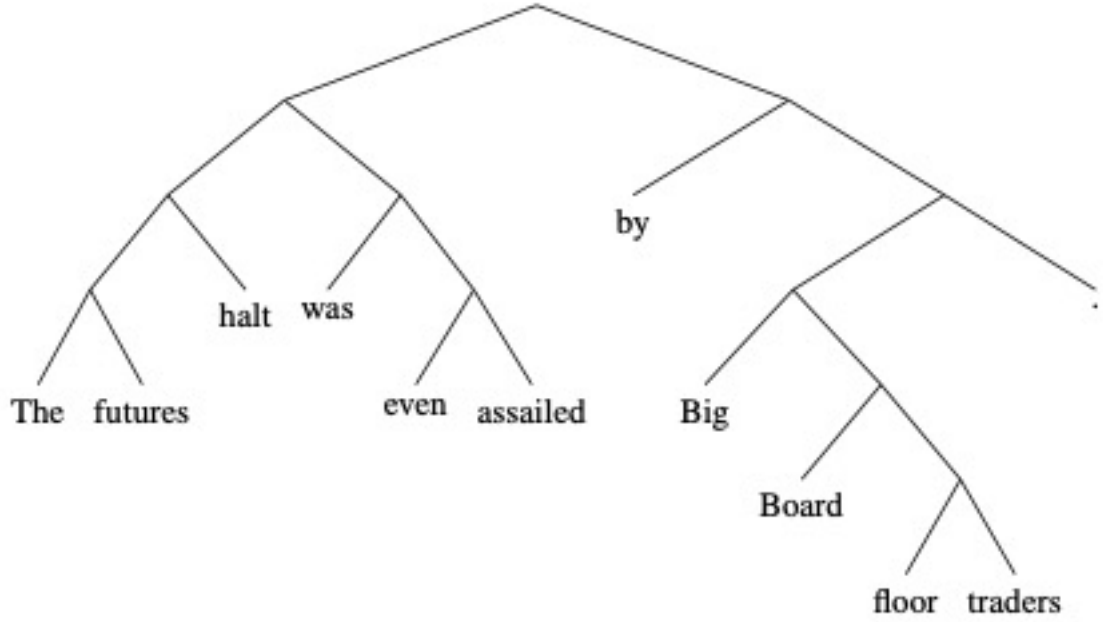
2 symbols, LSTM controller

Input: 100111# . . . #

Output: . . . 111001

Trees from a stack-based language model (Merrill et al., 2019)





LSTM controller