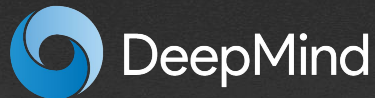


# Structure and Grounding in Natural Language

Phil Blunsom

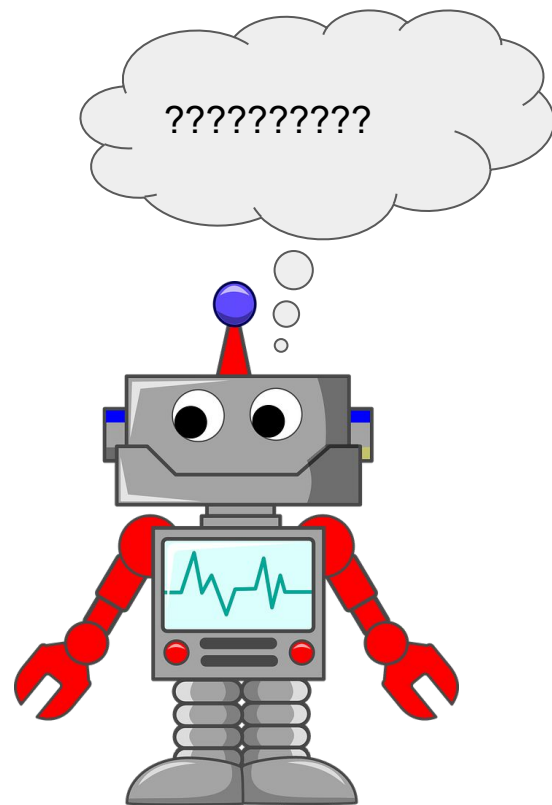
In collaboration with Chris Dyer, Dani Yogotama, Felix Hill, Karl Moritz Hermann and the  
DeepMind Natural Language Group



# Challenges for language and Artificial Intelligence

**Syntax:** We need models that can induce and exploit the hierarchical structure of language to learn rapidly.

**Semantics:** Our agents must be able to ground their understanding of linguistic symbols to concepts in their world.



# This Lecture

1. Modelling hierarchical structure in language with recurrent networks
2. Inducing hierarchical structure from distant reward
3. Learning compositional language in simulated 3D environment

# This Lecture

1. Modelling hierarchical structure in language with recurrent networks
2. Inducing hierarchical structure from distant reward
3. Learning compositional language in simulated 3D environment

# Inductive biases in Recurrent Neural Networks

RNNs struggle to capture non-sequential dependencies:

- gradients shrink across time,
- we often need to employ hacks like reversing sequences in seq2seq learning,
- or more directly encode structural assumptions with enhancements like attention.

LSTMs and other gated architectures reduce, but do not negate these issues.

Generative Linguistics (Chomsky) argues that sequential recency is not the right bias for effective (i.e. fast) learning of human language.

# Why do we believe that language has syntax?

Syntax exists separate from semantics. We can assign semantics to ungrammatical utterances, and syntax to semantically incoherent ones:

Colourless green oranges sleep furiously .

Syntax mediates between the hearing/speech (sensory-motor interface) and meaning (conceptual-intentional interface).

Why do we believe that language has syntax?

The students are enjoying the lecture

Are the students enjoying the lecture



Simple Rule: to convert a statement into a question move the first auxiliary (are) to the front (Auxiliary Fronting)

# Why do we believe that language has syntax?

The students who are sleeping are enjoying the lecture

Are the students who sleeping are enjoying the lecture



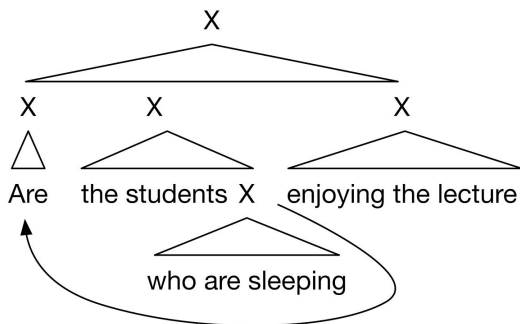
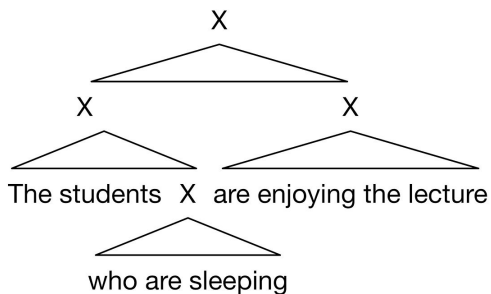
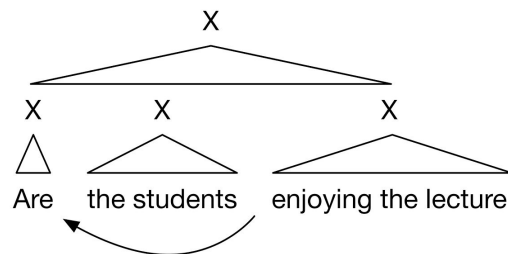
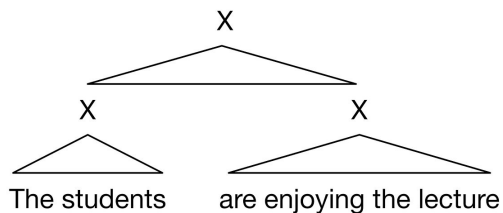
Are the students who are sleeping enjoying the lecture



~~Simple Rule: to convert a statement into a question move the first auxiliary (are) to the front (Auxiliary Fronting)~~

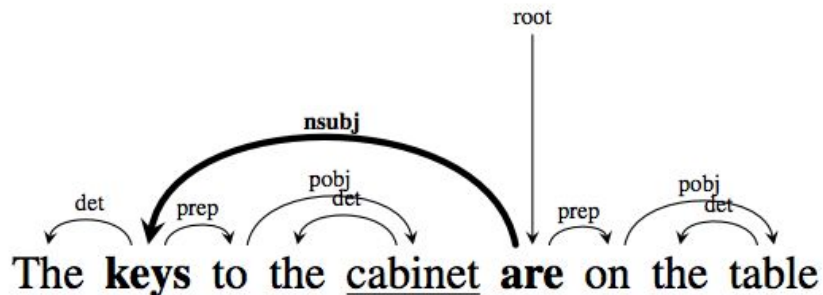


# Why do we believe that language has syntax?



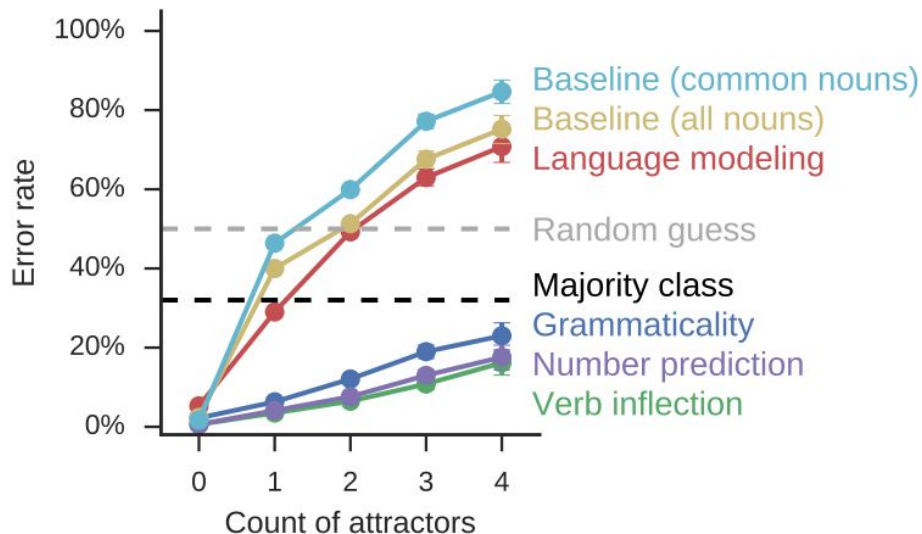
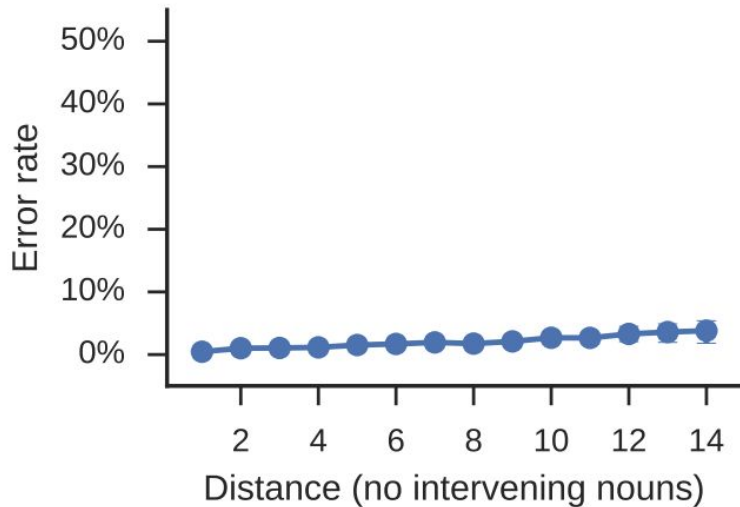
Better Rule: to convert a statement into a question move the auxiliary (are) that follows the **subject** (a syntactic concept) to the front.

# Do RNN language models learn recursive structure?



Training objective	Sample input	Training signal	Prediction task	Correct answer
Number prediction	<i>The keys to the cabinet</i>	PLURAL	SINGULAR/PLURAL?	PLURAL
Verb inflection	<i>The keys to the cabinet [is/are]</i>	PLURAL	SINGULAR/PLURAL?	PLURAL
Grammaticality	<i>The keys to the cabinet are here.</i>	GRAMMATICAL	GRAMMATICAL/UNGRAMMATICAL?	GRAMMATICAL
Language model	<i>The keys to the cabinet</i>	are	$P(\text{are}) > P(\text{is})?$	True

# Do RNN language models learn recursive structure?



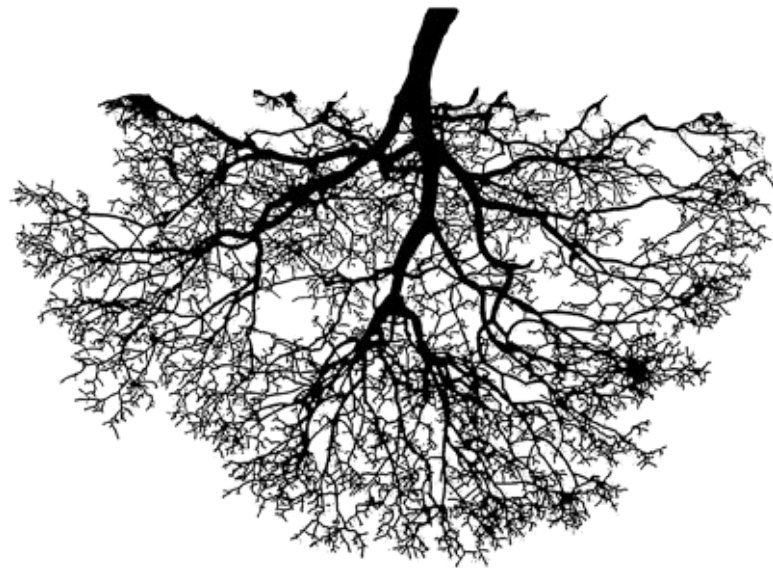
Training objective	Sample input	Training signal	Prediction task	Correct answer
Number prediction	<i>The keys to the cabinet</i>	PLURAL	SINGULAR/PLURAL?	PLURAL
Verb inflection	<i>The keys to the cabinet [is/are]</i>	PLURAL	SINGULAR/PLURAL?	PLURAL
Grammaticality	<i>The keys to the cabinet are here.</i>	GRAMMATICAL	GRAMMATICAL/UNGRAMMATICAL?	GRAMMATICAL
Language model	<i>The keys to the cabinet</i>	are	$P(are) > P(is)$ ?	True

# A Recurrent Network for Generating Trees

Given enough data RNNs will learn the distribution of language, but then the same can be said of simple Maximum Likelihood Estimators.

For AI we are primarily interested in efficient learning, i.e. **sample complexity**.

Our models must have the right biases in order to learn fast.



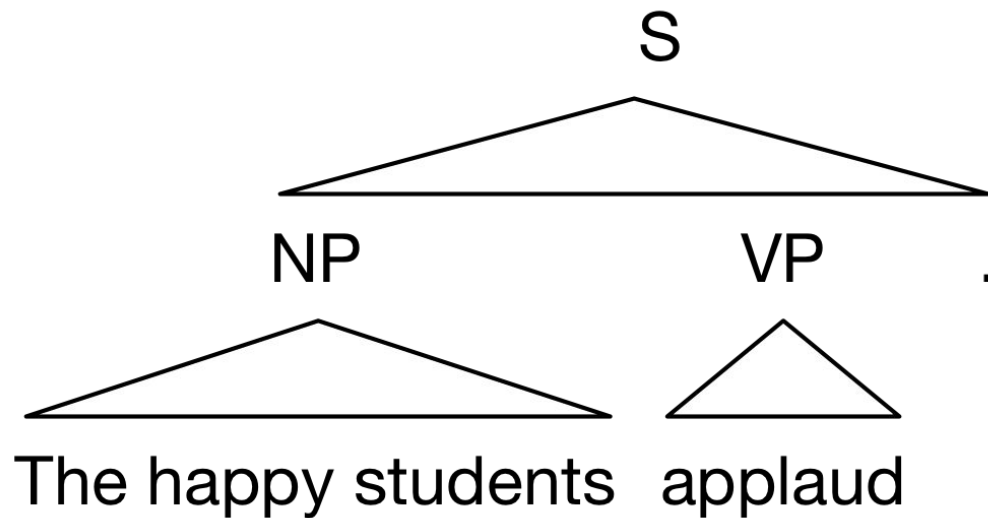
# Recurrent Neural Network Grammars

RNNGs define a **top-down, left-to-right generative process** for trees such that they can be **generated sequentially from an RNN**.

Latent control symbols are added that rewrite the history (hidden state) occasionally:

- Hierarchically **compress subsequences** into a single hidden state (constituents)
- The RNN predicts the next terminal/control symbol based on the history of compressed elements (**constituents**) and non-compressed terminals (**words**)

# Example Derivation



# Example Derivation

Stack	Action	Probability
	NT(S)	$p(\text{NT(S)} \mid \text{TOP})$

# Example Derivation

Stack	Action	Probability
(S	NT(S)	p( NT(S)   TOP )



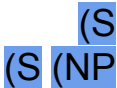
# Example Derivation

Stack	Action	Probability
(S	NT(S) NT(NP)	$p(\text{NT(S)} \mid \text{TOP})$ $p(\text{NT(NP)} \mid (\text{S}))$

# Example Derivation

Stack	Action	Probability
(S (NP (S	NT(S) NT(NP)	$p(\text{NT(S)} \mid \text{TOP})$ $p(\text{NT(NP)} \mid (\text{S}))$

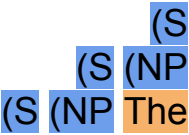
# Example Derivation

Stack	Action	Probability
 <p>(S (S (NP</p>	<p>NT(S) NT(NP) GEN(The)</p>	<p><math>p(\text{NT(S)} \mid \text{TOP})</math> <math>p(\text{NT(NP)} \mid (\text{S}))</math> <math>p(\text{GEN(The)} \mid (\text{S (NP)})</math></p>

# Example Derivation

Stack	Action	Probability
<p>(S (NP The</p> <p>(S (NP</p> <p>(S</p>	<p>NT(S)</p> <p>NT(NP)</p> <p>GEN(The)</p>	<p><math>p(\text{NT(S)} \mid \text{TOP})</math></p> <p><math>p(\text{NT(NP)} \mid (\text{S}))</math></p> <p><math>p(\text{GEN(The)} \mid (\text{S (NP)})</math></p>

# Example Derivation

Stack	Action	Probability
 <p>(S (NP The</p>	<p>NT(S) NT(NP) GEN(The) GEN(happy)</p>	<p><math>p(\text{NT(S)} \mid \text{TOP})</math> <math>p(\text{NT(NP)} \mid (\text{S}))</math> <math>p(\text{GEN(The)} \mid (\text{S (NP)})</math> <math>p(\text{GEN(happy)} \mid (\text{S (NP The)})</math></p>

# Example Derivation

Stack	Action	Probability
<p>(S (NP The happy</p> <p>(S (NP The</p> <p>(S (NP</p> <p>(S</p>	<p>NT(S)</p> <p>NT(NP)</p> <p>GEN(The)</p> <p>GEN(happy)</p>	<p><math>p(\text{NT(S)} \mid \text{TOP})</math></p> <p><math>p(\text{NT(NP)} \mid (\text{S}))</math></p> <p><math>p(\text{GEN(The)} \mid (\text{S (NP)})</math></p> <p><math>p(\text{GEN(happy)} \mid (\text{S (NP The)})</math></p>

# Example Derivation

Stack	Action	Probability
<p>(S (NP The happy</p> <p>(S (NP The</p> <p>(S (NP</p> <p>(S</p>	<p>NT(S)</p> <p>NT(NP)</p> <p>GEN(The)</p> <p>GEN(happy)</p> <p>GEN(students)</p>	<p><math>p(\text{NT(S)} \mid \text{TOP})</math></p> <p><math>p(\text{NT(NP)} \mid (\text{S}))</math></p> <p><math>p(\text{GEN(The)} \mid (\text{S (NP)})</math></p> <p><math>p(\text{GEN(happy)} \mid (\text{S (NP The)})</math></p> <p><math>p(\text{GEN(students)} \mid \dots)</math></p>

# Example Derivation

Stack	Action	Probability
<p>(S (NP The happy students</p> <p>(S (NP The happy</p> <p>(S (NP The</p> <p>(S (NP</p> <p>(S</p>	<p>NT(S)</p> <p>NT(NP)</p> <p>GEN(The)</p> <p>GEN(happy)</p> <p>GEN(students)</p>	<p><math>p(\text{NT(S)} \mid \text{TOP})</math></p> <p><math>p(\text{NT(NP)} \mid (\text{S}))</math></p> <p><math>p(\text{GEN(The)} \mid (\text{S (NP)})</math></p> <p><math>p(\text{GEN(happy)} \mid (\text{S (NP The)})</math></p> <p><math>p(\text{GEN(students)} \mid \dots)</math></p>



# Example Derivation

Stack	Action	Probability
<p>(S (NP The happy students</p> <p>(S (NP The happy</p> <p>(S (NP The</p> <p>(S (NP</p> <p>(S</p>	<p>NT(S)</p> <p>NT(NP)</p> <p>GEN(The)</p> <p>GEN(happy)</p> <p>GEN(students)</p> <p>REDUCE</p>	<p><math>p(\text{NT(S)} \mid \text{TOP})</math></p> <p><math>p(\text{NT(NP)} \mid (\text{S}))</math></p> <p><math>p(\text{GEN(The)} \mid (\text{S (NP)})</math></p> <p><math>p(\text{GEN(happy)} \mid (\text{S (NP The)})</math></p> <p><math>p(\text{GEN(students)} \mid \dots)</math></p> <p><math>p(\text{REDUCE} \mid \dots)</math></p>



# Example Derivation

Stack	Action	Probability
<p>(S (NP The happy students)</p> <p>(S (NP The happy)</p> <p>(S (NP The)</p> <p>(S (NP</p> <p>(S</p>	<p><b>NT</b>(S)</p> <p><b>NT</b>(NP)</p> <p><b>GEN</b>(The)</p> <p><b>GEN</b>(happy)</p> <p><b>GEN</b>(students)</p> <p><b>REDUCE</b></p> <p><b>NT</b>(VP)</p>	<p><math>p(\text{NT}(\text{S}) \mid \text{TOP})</math></p> <p><math>p(\text{NT}(\text{NP}) \mid (\text{S}))</math></p> <p><math>p(\text{GEN}(\text{The}) \mid (\text{S} (\text{NP}))</math></p> <p><math>p(\text{GEN}(\text{happy}) \mid (\text{S} (\text{NP} \text{The}))</math></p> <p><math>p(\text{GEN}(\text{students}) \mid \dots)</math></p> <p><math>p(\text{REDUCE} \mid \dots)</math></p> <p><math>p(\text{NT}(\text{VP}) \mid (\text{S} (\text{NP} \dots))</math></p>

# Example Derivation

Stack	Action	Probability
<p>(S (NP The happy students) (VP</p> <p>(S (NP The happy students)</p> <p>(S (NP The happy</p> <p>(S (NP The</p> <p>(S (NP</p> <p>(S</p>	<p>NT(S)</p> <p>NT(NP)</p> <p>GEN(The)</p> <p>GEN(happy)</p> <p>GEN(students)</p> <p>REDUCE</p> <p>NT(VP)</p>	<p><math>p(\text{NT(S)} \mid \text{TOP})</math></p> <p><math>p(\text{NT(NP)} \mid (\text{S}))</math></p> <p><math>p(\text{GEN(The)} \mid (\text{S (NP)})</math></p> <p><math>p(\text{GEN(happy)} \mid (\text{S (NP The)})</math></p> <p><math>p(\text{GEN(students)} \mid \dots)</math></p> <p><math>p(\text{REDUCE} \mid \dots)</math></p> <p><math>p(\text{NT(VP)} \mid (\text{S (NP ...)})</math></p>

# Example Derivation

Stack	Action	Probability
<p>(S  (S (NP  (S (NP The  (S (NP The happy  (S (NP The happy students  (S (NP The happy students)  (S (NP The happy students) (VP</p>	<p>NT(S)  NT(NP)  GEN(The)  GEN(happy)  GEN(students)  REDUCE  NT(VP)  GEN(applaud)</p>	<p><math>p(\text{NT(S)} \mid \text{TOP})</math>  <math>p(\text{NT(NP)} \mid (\text{S}))</math>  <math>p(\text{GEN(The)} \mid (\text{S (NP)})</math>  <math>p(\text{GEN(happy)} \mid (\text{S (NP The)})</math>  <math>p(\text{GEN(students)} \mid \dots)</math>  <math>p(\text{REDUCE} \mid \dots)</math>  <math>p(\text{NT(VP)} \mid (\text{S (NP ...)})</math></p>

# Example Derivation

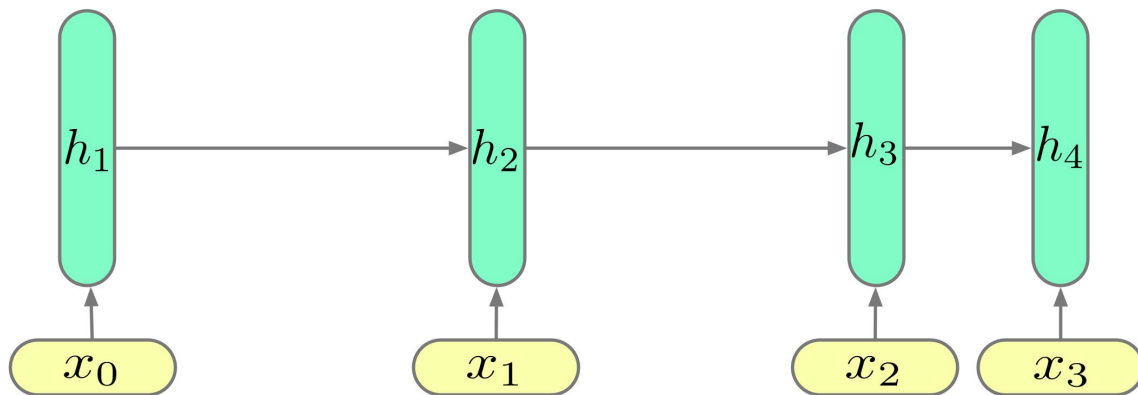
Stack	Action	Probability
(S	NT(S)	$p(\text{NT(S)} \mid \text{TOP})$
(S	NT(NP)	$p(\text{NT(NP)} \mid (\text{S}))$
(S (NP	GEN(The)	$p(\text{GEN(The)} \mid (\text{S (NP)})$
(S (NP The	GEN(happy)	$p(\text{GEN(happy)} \mid (\text{S (NP The)})$
(S (NP The happy	GEN(students)	$p(\text{GEN(students)} \mid \dots)$
(S (NP The happy students	REDUCE	$p(\text{REDUCE} \mid \dots)$
(S (NP The happy students)	NT(VP)	$p(\text{NT(VP)} \mid (\text{S (NP ...)})$
(S (NP The happy students) (VP	GEN(applaud)	
(S (NP The happy students) (VP applaud	REDUCE	
(S (NP The happy students) (VP applaud)	GEN(.)	
(S (NP The happy students) (VP applaud) .	REDUCE	
(S (NP The happy students) (VP applaud) .)		

# Is this a coherent probabilistic model of trees?

- For every (tree, string) pair there is exactly one valid sequence of actions (specifically, the depth first, left-to-right traversal of the trees)
- Every stack configuration exactly encodes the complete history of actions.
- Therefore, the probability of a (tree,string) pair decomposes by the chain rule:

$$\begin{aligned} p(x, y) &= p(\text{actions}(x, y)) \\ p(\text{actions}(x, y)) &= \prod_i p(a_i | \mathbf{a}_{<I}) \\ &= \prod_i p(a_i | \text{stack}(a_{<I})) \end{aligned}$$

# Modeling the action conditional probabilities



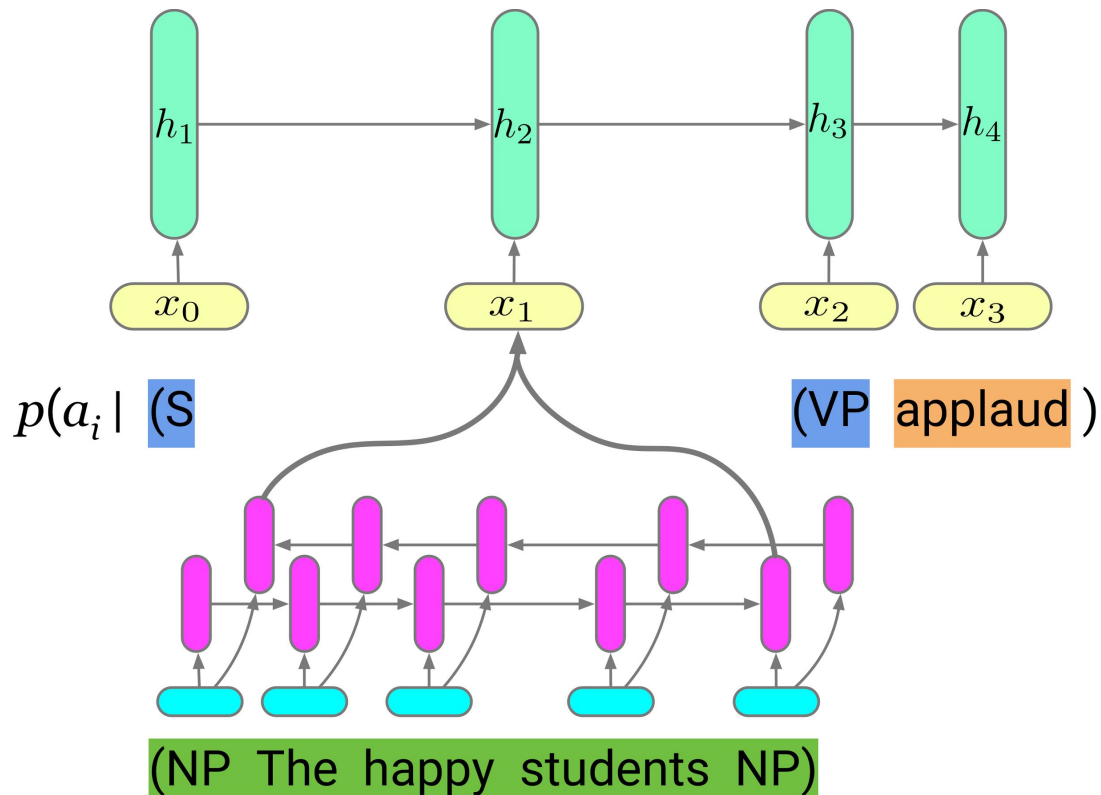
$p(a_i |$  (S (NP The happy students) (VP applaud) )



Model the history with an RNN



# Syntactic composition



# Training

## Generative

- Jointly model sentence  $\mathbf{x}$  and its tree  $\mathbf{y}$
- Trained using gold standard trees (from a tree bank) to minimize cross-entropy
- We call this joint distribution  $p(\mathbf{x}, \mathbf{y})$

## Discriminative

- Given a sentence  $\mathbf{x}$ , predict the sequence of actions  $\mathbf{y}$  necessary to build its parse tree - *the full sentence  $\mathbf{x}$  is observable*
- Instead of GEN, use SHIFT
- We call this conditional distribution  $q(\mathbf{y} | \mathbf{x})$

To parse: simply use beam search to find the best sequence.

# English PTB (Parsing)

	Type	F1
Petrov and Klein (2007)	Gen	90.1
Shindo et al (2012) Single model	Gen	91.1
Vinyals et al (2015) PTB only	Disc	90.5
Shindo et al (2012) Ensemble	Gen	92.4
Vinyals et al (2015) Semisupervised	Disc+SemiS up	92.8
<i>Discriminative</i> PTB only	Disc	91.7
<i>Generative</i> PTB only	Gen	<b>93.6</b>

# English Language Modeling

	Perplexity
<b>5-gram IKN</b>	169.3
<b>LSTM + Dropout</b>	113.4
<b>Generative (approx.)</b>	<b>102.4</b>

$$p(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{Y}_{\mathbf{x}}} p(\mathbf{x}, \mathbf{y})$$

# RNNGs Summary

Language has hierarchical structure.

Recurrent Neural Network Grammars allow us to incorporate this bias into sequential RNNs and provide a strong generative model for parsing and language modelling.

**However**, introducing discrete latent variables (Control Actions) creates hard inference problems.

For an excellent analysis see: Kuncoro et al. What Do Recurrent Neural Network Grammars Learn About Syntax? EACL 2017.

# This Lecture

1. Modelling hierarchical structure in language with recurrent networks
2. Inducing hierarchical structure from distant reward
3. Learning compositional language in simulated 3D environment

# Inducing hierarchical structure from distant reward

We do not observe the syntactic structure of language in raw input data.

- In supervised settings we assume we have a dataset of utterances annotated with syntactic structures.
- In unsupervised settings we treat the structure as latent and search for the structure that optimises an objective function. The most common objective is likelihood of the utterance (i.e. language modelling).

Here we consider an alternative: using Reinforcement Learning to select the structure that maximises a task based reward.

# Inducing hierarchical structure from distant reward

Advances in deep learning have led to three predominant approaches for constructing representations of sentences

- Convolutional neural networks

[Kim, 2014](#); [Kalchbrenner et al., 2014](#); [Ma et al., 2015](#)

- Recurrent neural networks

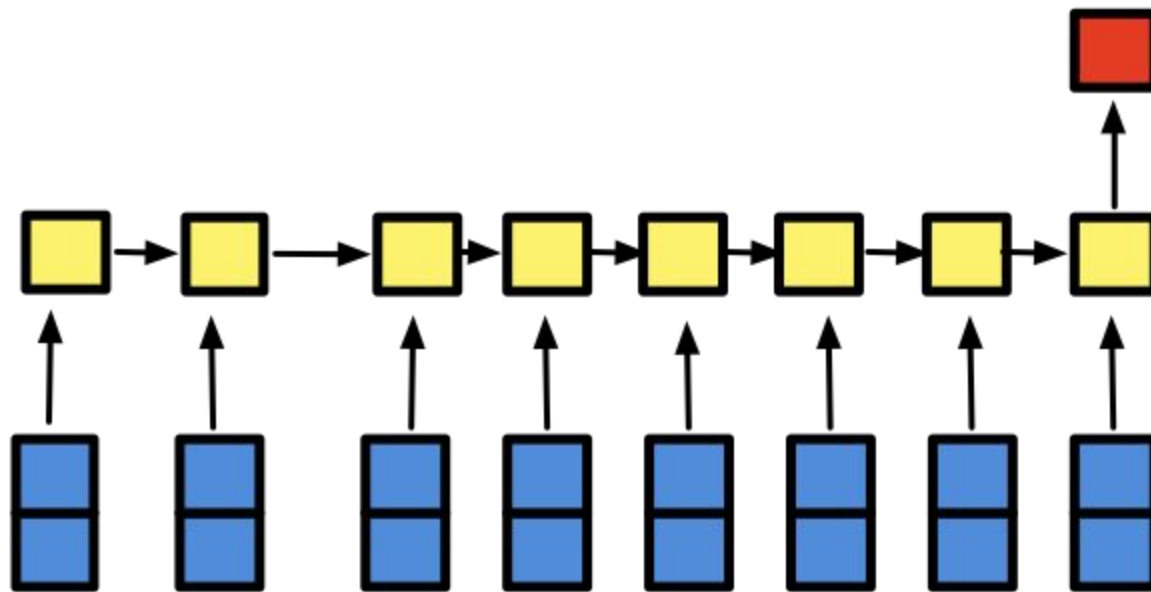
[Cho et al., 2014](#); [Sutskever et al., 2014](#); [Bahdanau et al., 2014](#)

- Recursive neural networks

[Socher et al., 2011](#); [Socher et al., 2013](#); [Tai et al., 2015](#); [Bowman et al., 2016](#)



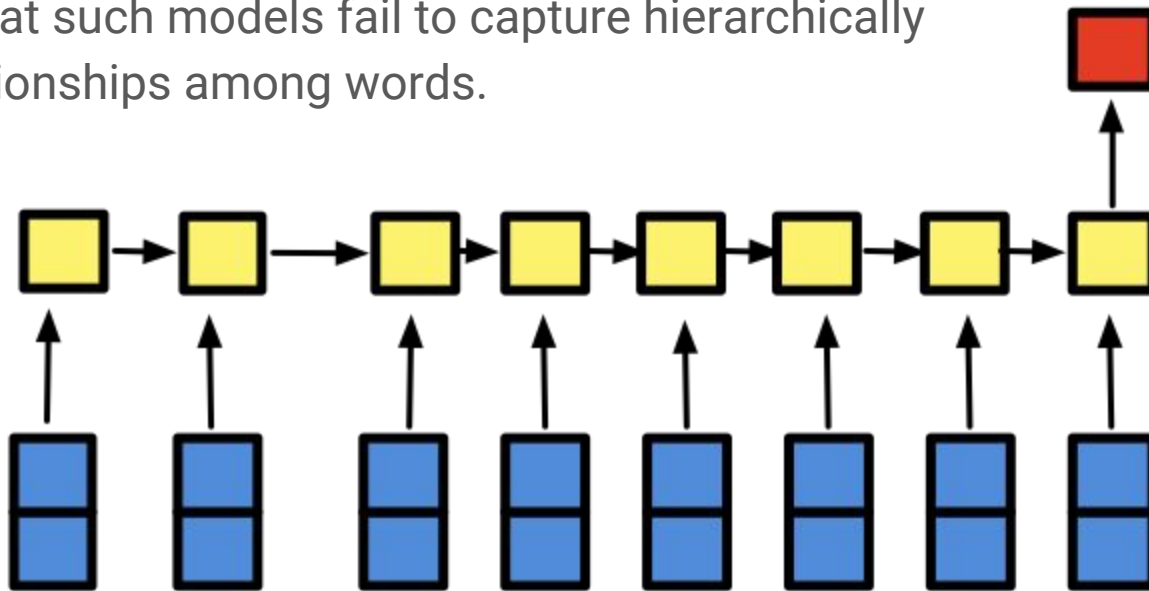
# Recurrent Neural Network Encoder



A boy drags his sleds through the snow

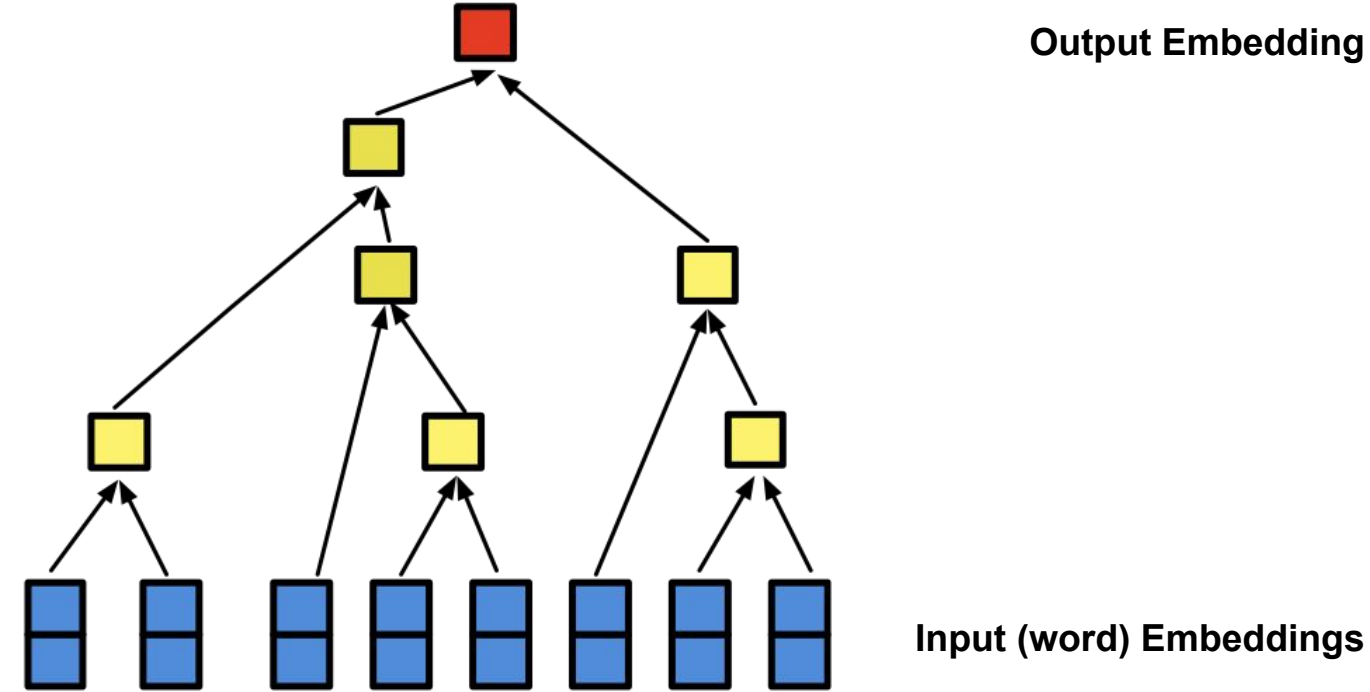
# Learning language structure from distant rewards

But we know that such models fail to capture hierarchically organised relationships among words.



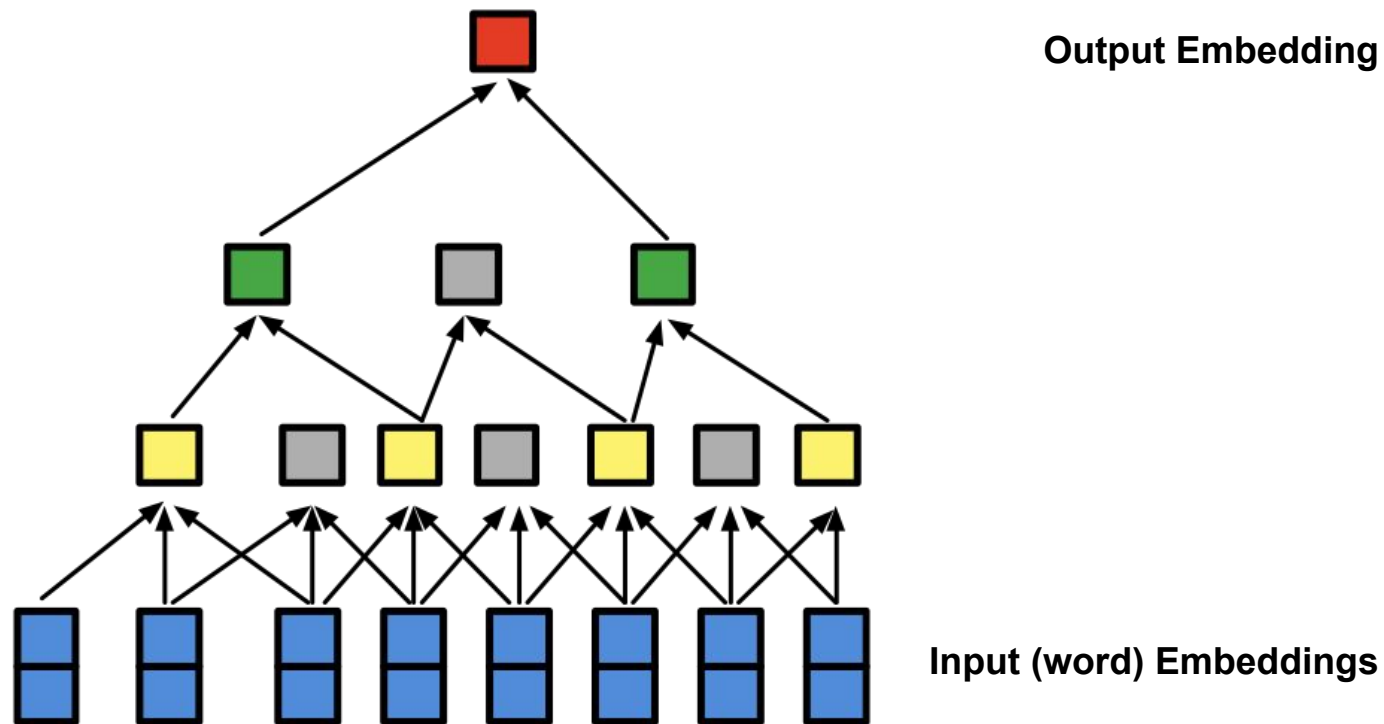
A boy drags his sleds through the snow

# Recursive Neural Network Encoder



A boy drags his sleds through the snow

# Convolutional Encoder



A boy drags his sleds through the snow

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
		A boy drags his sled

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
	<b>SHIFT</b>	A boy drags his sled

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
A	SHIFT	A boy drags his sled boy drags his sled

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
A	SHIFT SHIFT	A boy drags his sled boy drags his sled




# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
A A boy	SHIFT SHIFT	A boy drags his sled boy drags his sled drags his sled


# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
A A boy	SHIFT SHIFT REDUCE	A boy drags his sled boy drags his sled drags his sled

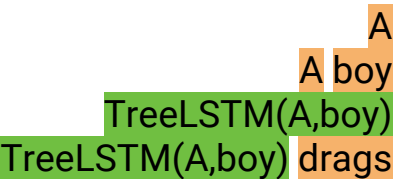
# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
 <p>A boy TreeLSTM(A,boy) A</p>	<p>SHIFT SHIFT REDUCE</p>	<p>A boy drags his sled boy drags his sled drags his sled drags his sled</p>

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
 <p>A A boy TreeLSTM(A,boy)</p>	<p>SHIFT SHIFT REDUCE SHIFT</p>	<p>A boy drags his sled boy drags his sled drags his sled drags his sled</p>

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
	<b>SHIFT</b> <b>SHIFT</b> <b>REDUCE</b> <b>SHIFT</b>	A boy drags his sled boy drags his sled drags his sled drags his sled his sled

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
A	SHIFT	A boy drags his sled
A boy	SHIFT	boy drags his sled
TreeLSTM(A,boy)	REDUCE	drags his sled
TreeLSTM(A,boy) drags	SHIFT	drags his sled
	SHIFT	his sled

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
	<b>SHIFT</b>	A boy drags his sled
	<b>SHIFT</b>	boy drags his sled
	<b>REDUCE</b>	drags his sled
TreeLSTM(A,boy)	<b>SHIFT</b>	drags his sled
TreeLSTM(A,boy) drags	<b>SHIFT</b>	his sled
TreeLSTM(A,boy) drags his		sled

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
	<b>SHIFT</b>	A boy drags his sled
	<b>SHIFT</b>	boy drags his sled
	<b>REDUCE</b>	drags his sled
TreeLSTM(A,boy)	<b>SHIFT</b>	drags his sled
TreeLSTM(A,boy) drags	<b>SHIFT</b>	his sled
TreeLSTM(A,boy) drags his	<b>SHIFT</b>	sled



# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
	<b>SHIFT</b>	A boy drags his sled
	<b>SHIFT</b>	boy drags his sled
	<b>REDUCE</b>	drags his sled
TreeLSTM(A,boy)	<b>SHIFT</b>	drags his sled
TreeLSTM(A,boy) drags	<b>SHIFT</b>	his sled
TreeLSTM(A,boy) drags his	<b>SHIFT</b>	sled
TreeLSTM(A,boy) drags his sled		

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
	<b>SHIFT</b>	A boy drags his sled
	<b>SHIFT</b>	boy drags his sled
	<b>REDUCE</b>	drags his sled
TreeLSTM(A,boy)	<b>SHIFT</b>	drags his sled
TreeLSTM(A,boy) drags	<b>SHIFT</b>	his sled
TreeLSTM(A,boy) drags his	<b>SHIFT</b>	sled
TreeLSTM(A,boy) drags his sled	<b>REDUCE</b>	

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
	<b>SHIFT</b>	A boy drags his sled
	<b>SHIFT</b>	boy drags his sled
	<b>REDUCE</b>	drags his sled
TreeLSTM(A,boy)	<b>SHIFT</b>	drags his sled
TreeLSTM(A,boy) drags	<b>SHIFT</b>	his sled
TreeLSTM(A,boy) drags his	<b>SHIFT</b>	sled
TreeLSTM(A,boy) drags his sled	<b>REDUCE</b>	
TreeLSTM(A,boy) drags TreeLSTM(his,sled)		

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
	SHIFT	A boy drags his sled
A	SHIFT	boy drags his sled
A boy	REDUCE	drags his sled
TreeLSTM(A,boy)	SHIFT	drags his sled
TreeLSTM(A,boy) drags	SHIFT	his sled
TreeLSTM(A,boy) drags his	SHIFT	sled
TreeLSTM(A,boy) drags his sled	REDUCE	
TreeLSTM(A,boy) drags TreeLSTM(his,sled)	REDUCE	

# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
	SHIFT	A boy drags his sled
A	SHIFT	boy drags his sled
A boy	REDUCE	drags his sled
TreeLSTM(A,boy)	SHIFT	drags his sled
TreeLSTM(A,boy) drags	SHIFT	his sled
TreeLSTM(A,boy) drags his	SHIFT	sled
TreeLSTM(A,boy) drags his sled	REDUCE	
TreeLSTM(A,boy) drags TreeLSTM(his,sled)	REDUCE	
TreeLSTM(A,boy) TreeLSTM(drags,TreeLSTM(his,sled))		

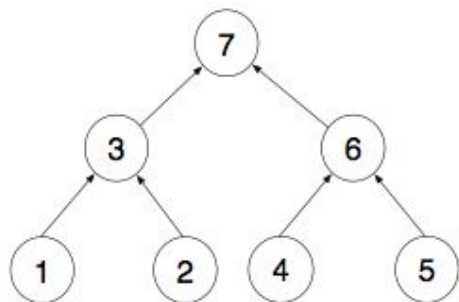
# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
	SHIFT	A boy drags his sled
	SHIFT	boy drags his sled
	REDUCE	drags his sled
TreeLSTM(A,boy)	SHIFT	drags his sled
TreeLSTM(A,boy) drags	SHIFT	his sled
TreeLSTM(A,boy) drags his	SHIFT	sled
TreeLSTM(A,boy) drags his sled	REDUCE	
TreeLSTM(A,boy) drags TreeLSTM(his,sled)	REDUCE	
TreeLSTM(A,boy) TreeLSTM(drags,TreeLSTM(his,sled))	REDUCE	

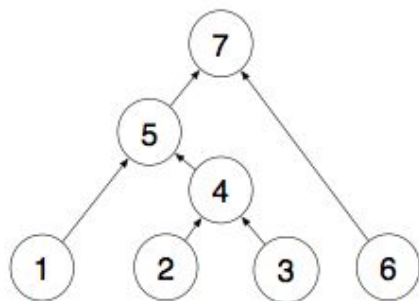
# Shift-Reduce TreeLSTM: Example Derivation

Stack	Action	Buffer
	SHIFT	A boy drags his sled
A	SHIFT	boy drags his sled
A boy	REDUCE	drags his sled
TreeLSTM(A,boy)	SHIFT	drags his sled
TreeLSTM(A,boy) drags	SHIFT	his sled
TreeLSTM(A,boy) drags his	SHIFT	sled
TreeLSTM(A,boy) drags his sled	REDUCE	
TreeLSTM(A,boy) drags TreeLSTM(his,sled)	REDUCE	
TreeLSTM(A,boy) TreeLSTM(drags,TreeLSTM(his,sled))	REDUCE	
TreeLSTM(TreeLSTM(A,boy),TreeLSTM(drags,TreeLSTM(his,sled)))		

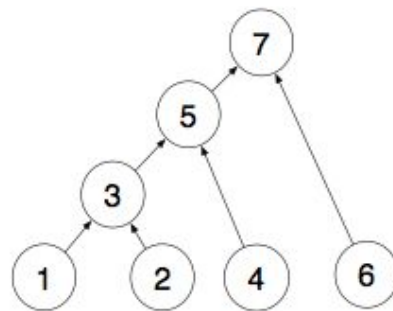
# Shift Reduce Parsing (Aho and Ullman, 1972)



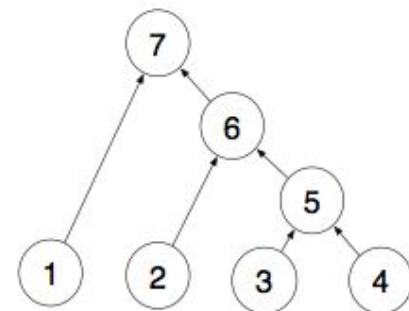
*S, S, R, S, S, R, R*



*S, S, S, R, R, S, R*



*S, S, R, S, R, S, R*



*S, S, S, S, R, R, R*

Each unique Shift Reduce sequence maps to a single tree structure



# Reinforcement Learning

- How do we learn an optimal sequence of shift reduce in the absence of supervised data?
- Given a state, and a set of possible actions, an agent needs to decide what is the best possible action to take
- There is no supervisor, only reward which can be not observed until after several actions are taken

# Reinforcement Learning

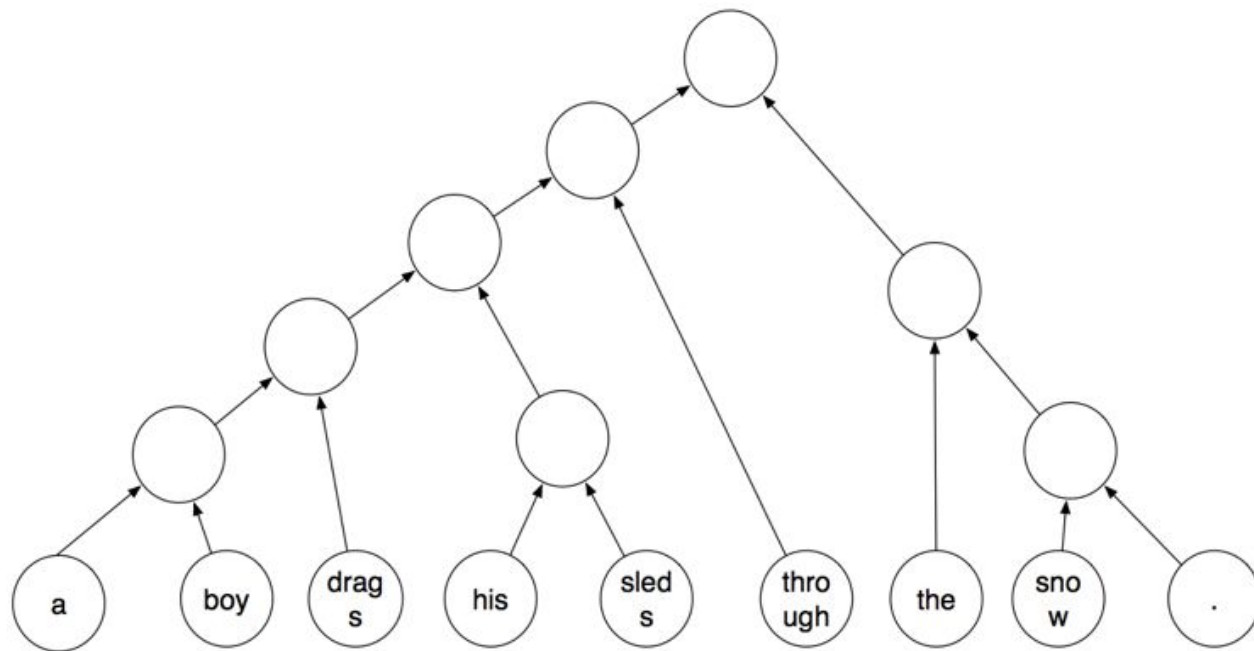
- Given a state, and a set of possible actions, an agent needs to decide what is the best possible action to take.
- There is no supervisor, only reward which can be not observed until after several actions are taken.
- A Shift-Reduce agent
  - a. State: embeddings of top two elements of the stack, embedding of head of the queue
  - b. Actions: shift, reduce
  - c. Reward: log likelihood on a downstream task given the produced representation
- Here we use the REINFORCE algorithm.

# Sentiment Analysis Results

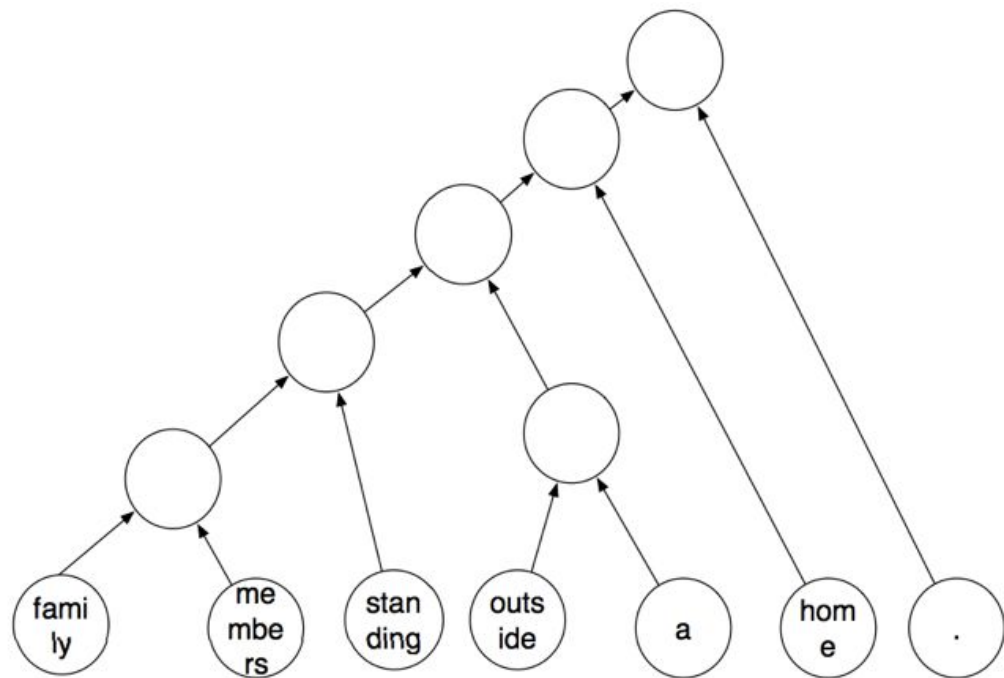
Method	Accuracy
Naive Bayes (from Socher et al., 2013)	81.8
SVM (from Socher et al., 2013)	79.4
Average of Word Embeddings (from Socher et al., 2013)	80.1
Bayesian Optimization (Yogatama et al., 2015)	82.4
Weighted Average of Word Embeddings (Arora et al., 2017)	82.4
Left-to-Right LSTM	84.7
Right-to-Left LSTM	83.9
Bidirectional LSTM	84.7
Supervised Syntax	85.3
Semi-supervised Syntax	86.1
Latent Syntax	86.5

CNN-based methods range from 82.7 to 88.1

# Learned Example



# Learned Example



# This Lecture

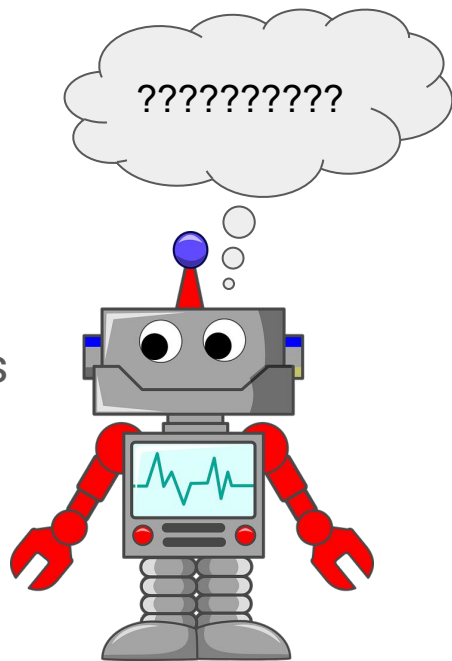
1. Modelling hierarchical structure in language with recurrent networks
2. Inducing hierarchical structure from distant reward
3. Learning compositional language in simulated 3D environment

# The Paradigm Problem

What is a minimally **adequate** training paradigm for an intelligent agent to learn to comprehend language?

Linguistic symbols must be **grounded** in an environment in order for an agent to interpret the meaning of an utterance.

Traditionally we ground language data with supervised labels (e.g. sentiment analysis, document classification), or in terms of other language data (e.g. word embeddings, MT, etc.).



# The Paradigm Problem

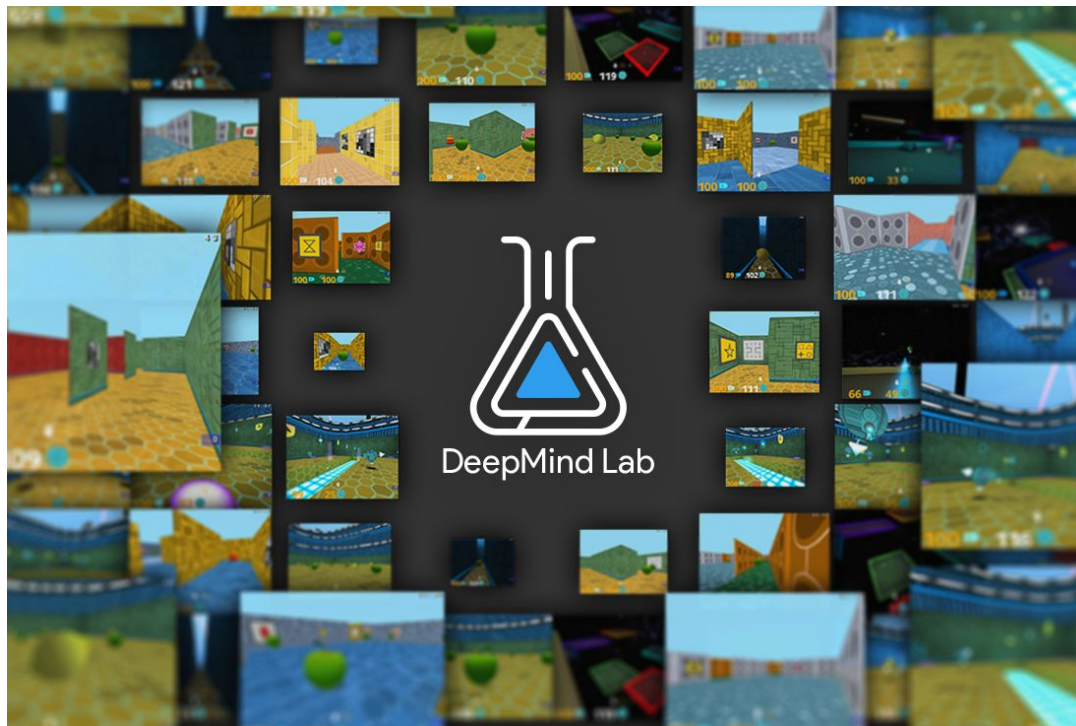
Question: What is the (data) environment within which humans learn language?

- Children learn language with minimal direct teaching and with incredible variations in data,
- they learn amazingly quickly from sparse and ambiguous data,
- children learn language in adverse circumstances, despite blindness, brain injury, or the inability to move or speak.

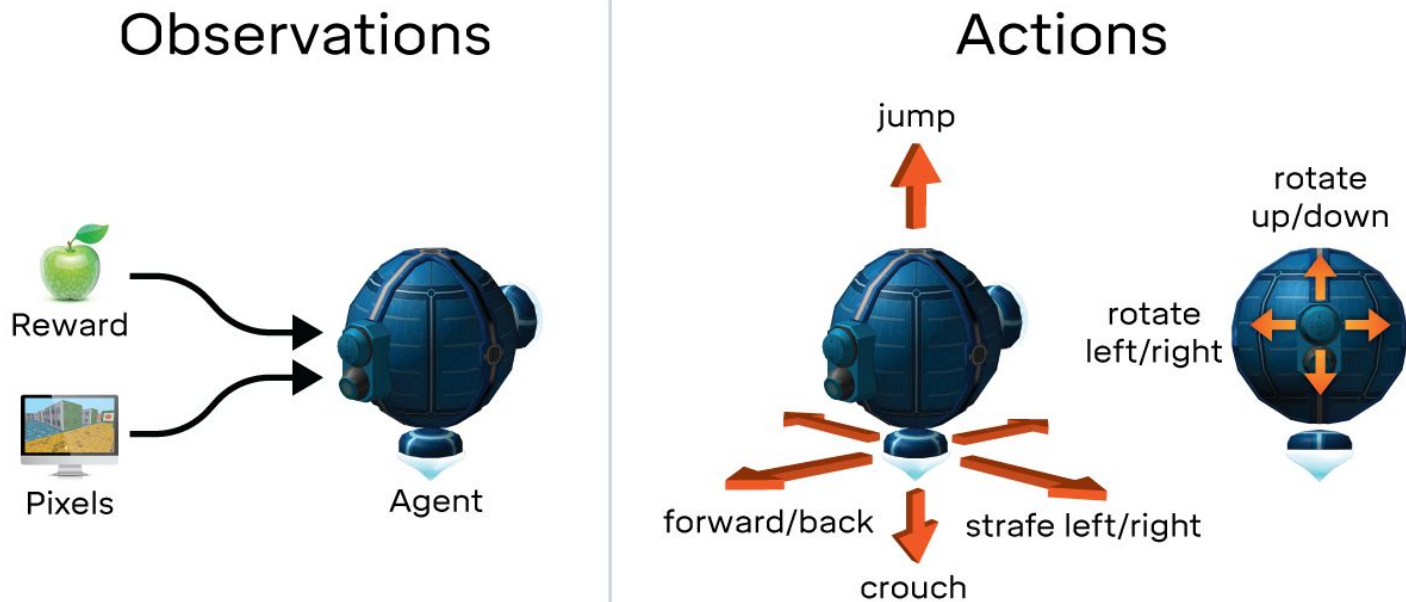
There is no one answer to this question. Here I will discuss a minimal starting point of situating an agent in a continuous 3D environment in which it can act and perform tasks specified by a teacher.



# DeepMind Lab



# DeepMind Lab



# DeepMind Lab



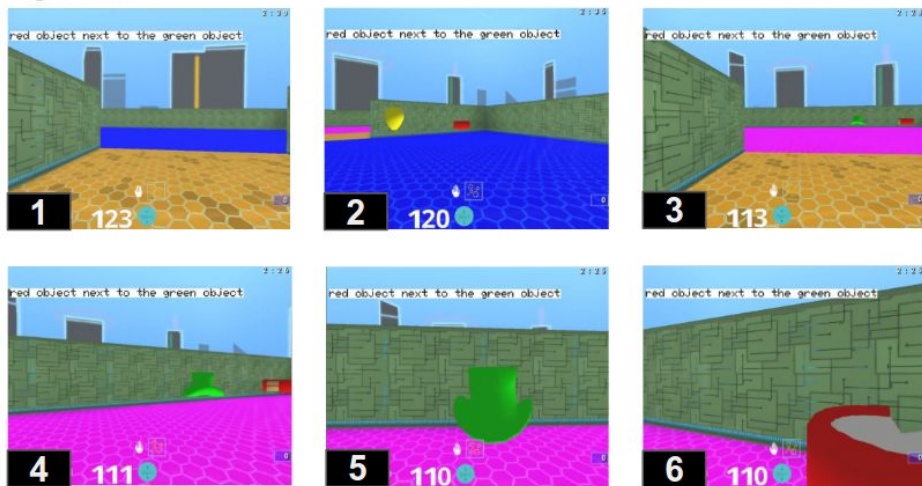
Beattie et al. DeepMind Lab. arXiv 2016. (<https://github.com/deepmind/lab>)

# Language in DeepMind Lab

We wish to train situated language agents, s.t.:

- the agent perceives its surroundings via a constant stream of visual input and a textual instruction,
- it perceives the world actively, controlling what it sees via movement of its visual field and exploration of its surroundings.

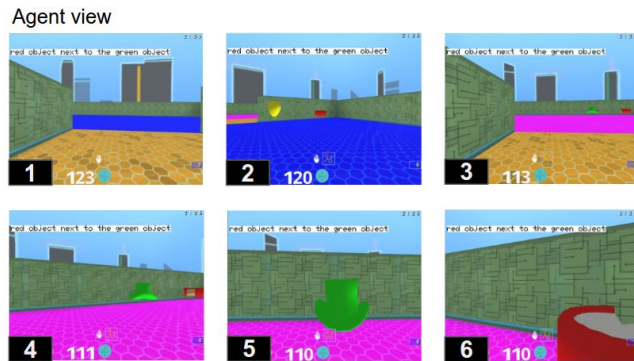
Agent view



# Language in DeepMind Lab

We specify the general layouts, possible objects and the form of language instructions that describe how the agent can obtain rewards.

The precise world experienced by the agent is chosen at random from billions of possibilities, corresponding to different instantiations of objects, their colours, surface patterns, relative positions and the overall layout of the 3D world.



# Language in DeepMind Lab: The Lexicon

**Shapes (40)** tv, ball, balloon, cake, can, cassette, chair, guitar, hairbrush, hat, ice lolly, ladder, mug, pencil, suitcase, toothbrush, key, bottle, car, cherries, fork, fridge, hammer, knife, spoon, apple, banana, cow, flower, jug, pig, pincer, plant, saxophone, shoe, tennis racket, tomato, tree, wine glass, zebra.

**Colours (13)** red, blue, white, grey, cyan, pink, orange, black, green, magenta, brown, purple, yellow.

**Patterns (9)** plain, chequered, crosses, stripes, discs, hex, pinstripe, spots, swirls.

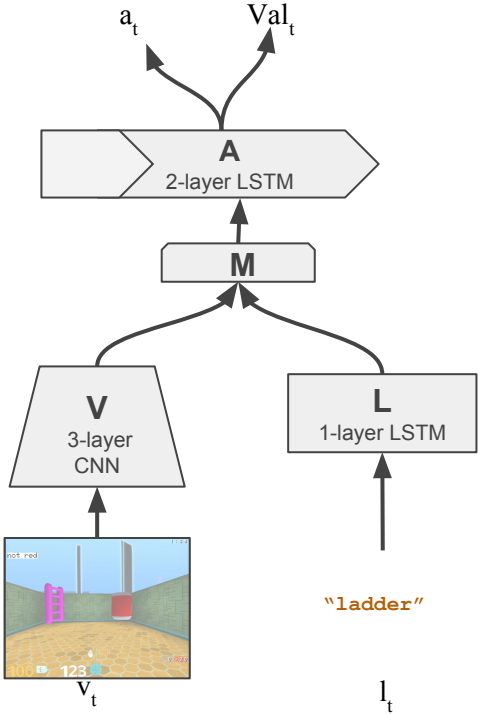
**Shades (3)** light, dark, neutral.

**Sizes (3)** small, large, medium.

# A trained agent following instructions

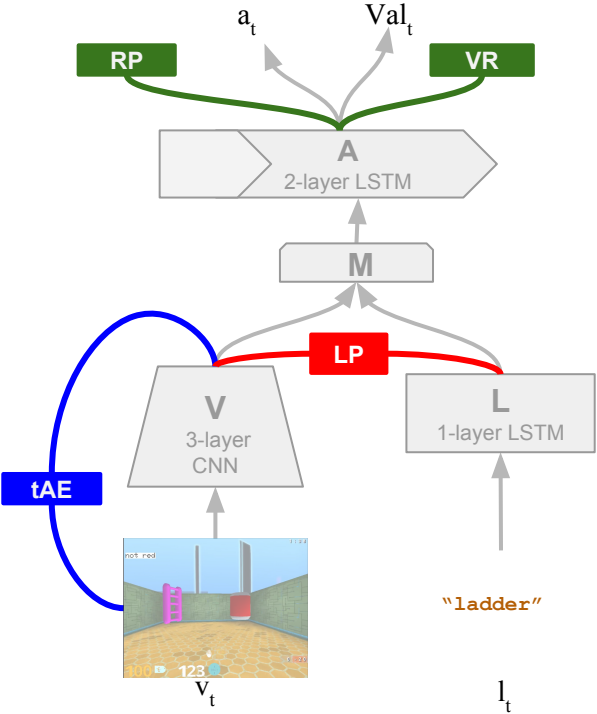


# A basic agent based on the A3C algorithm

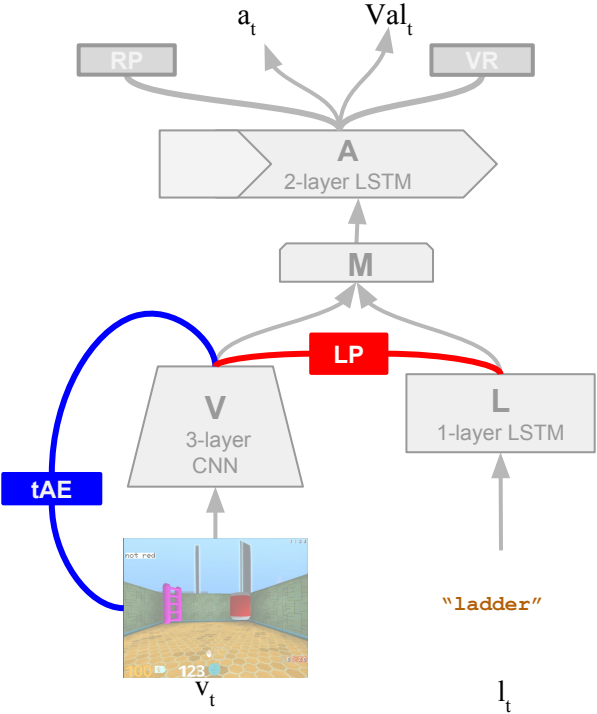




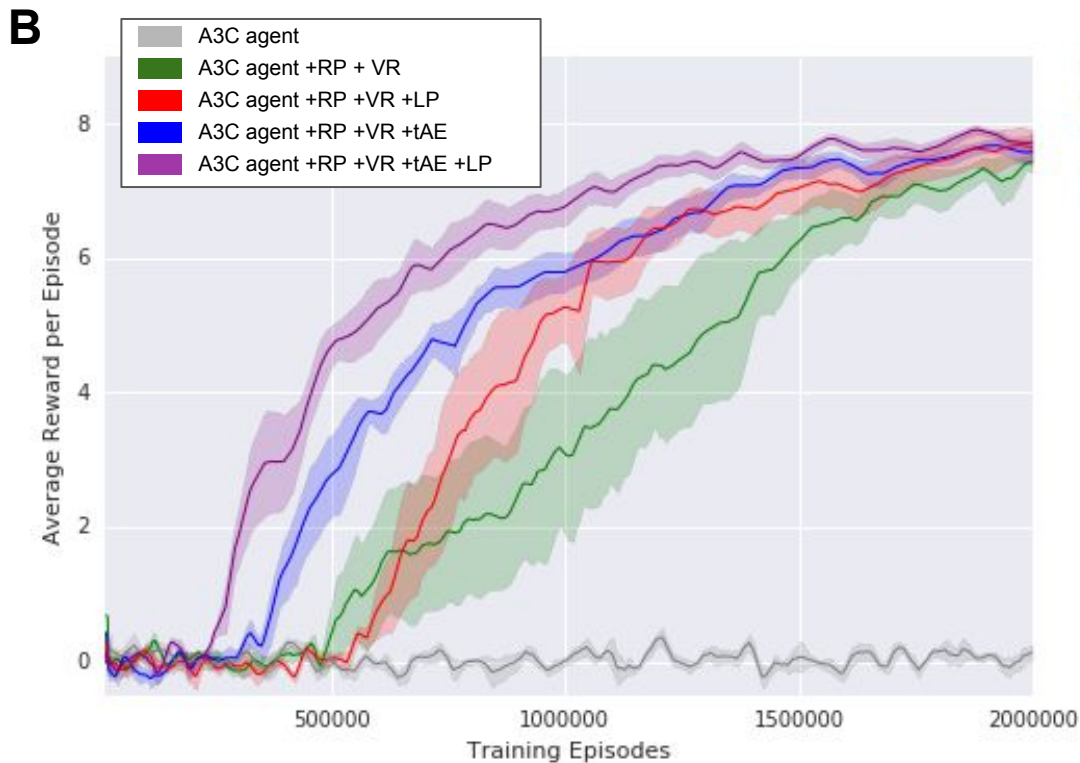
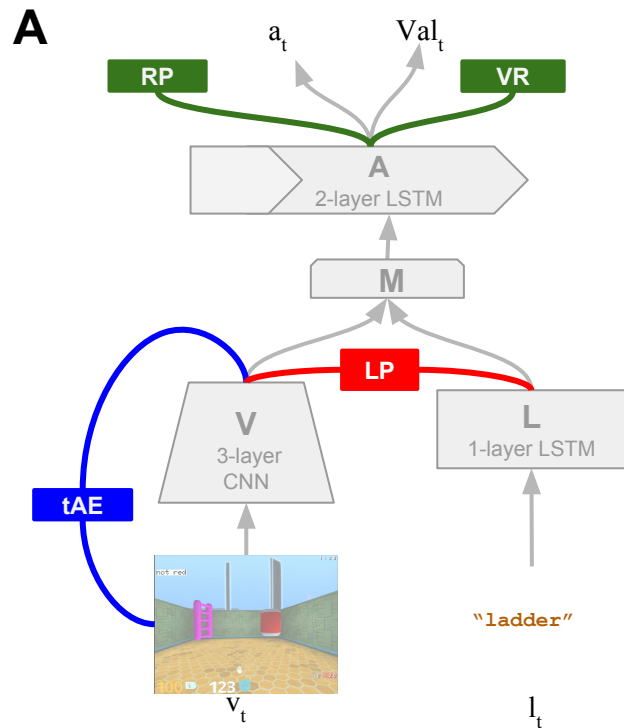
# Additional (similar to UNREAL) auxiliary objectives



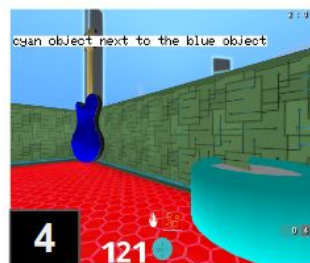
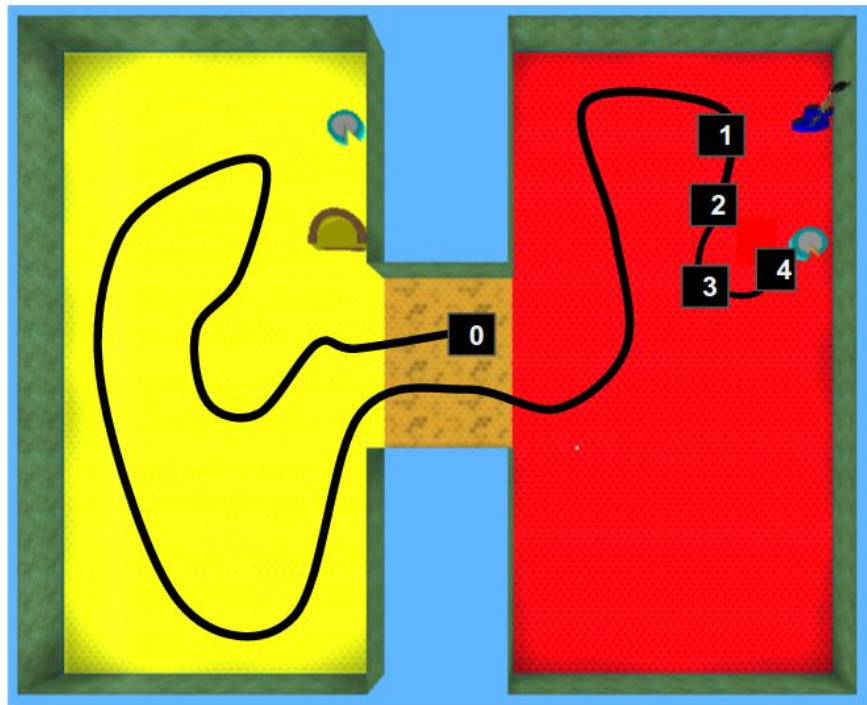
# Additional (similar to UNREAL) auxiliary objectives



# Unsupervised learning makes word learning possible



# DeepMind Lab



And provides insight into agents' 'thoughts'....



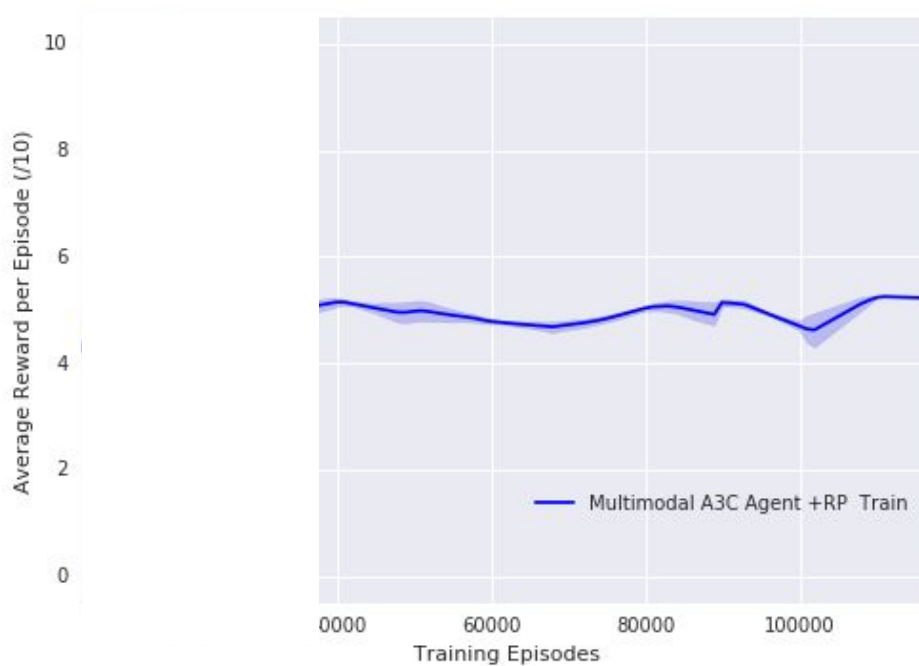
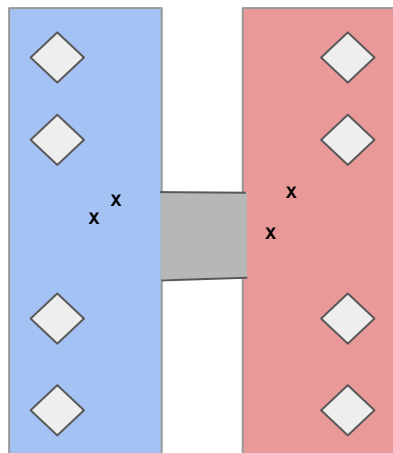
# DeepMind Lab



Hermann and Hill et al. Grounded Language Learning in a Simulated 3D World. arXiv 2017.

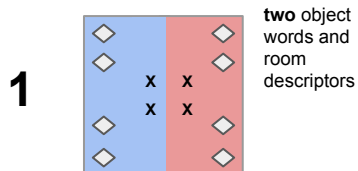
# Curriculum Learning for Complex Tasks

Top-down view of the level

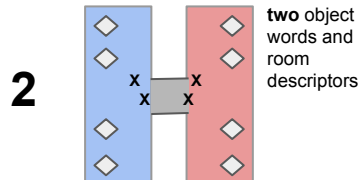


# Curriculum Learning for Complex Tasks

single-room layout



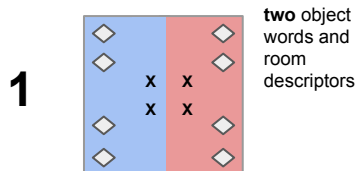
two room layout



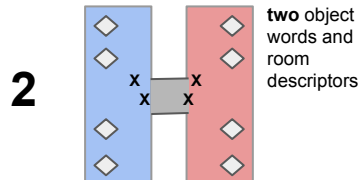


# Curriculum Learning for Complex Tasks

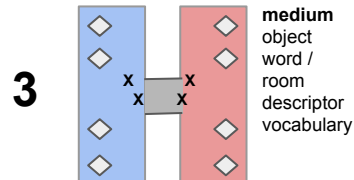
single-room layout



two room layout

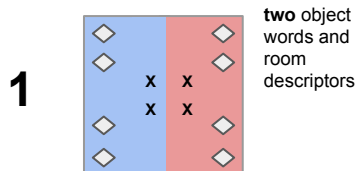


two room layout

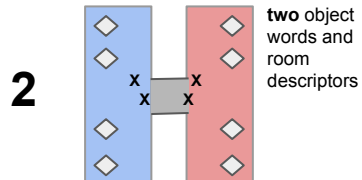


# Curriculum Learning for Complex Tasks

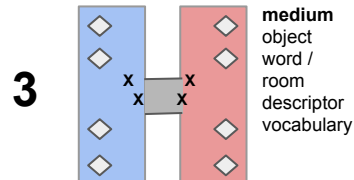
single-room layout



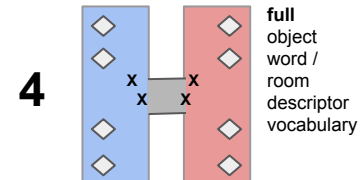
two room layout



two room layout



two room layout



# Agents learn to generalise word composition



## Training

"ladder"  
"mug"  
"pencil"  
"suitcase"  
"toothbrush"

x 40

"red"  
"green"  
"blue"  
"pink"

x 13

"red ladder"  
"green mug"  
"blue pencil"

x 400

## Test

"pink ladder"  
"yellow mug"  
"green pencil"

x 120

# Decompose before re-compose



Training

"red ladder"  
"green mug"  
"blue pencil"

x 400

Test

"pink ladder"  
"yellow mug"  
"green pencil"

x 120

# Apply modifiers and predicates to novel objects



## Training

"larger" (ball)  
"smaller" (ball)

x 30

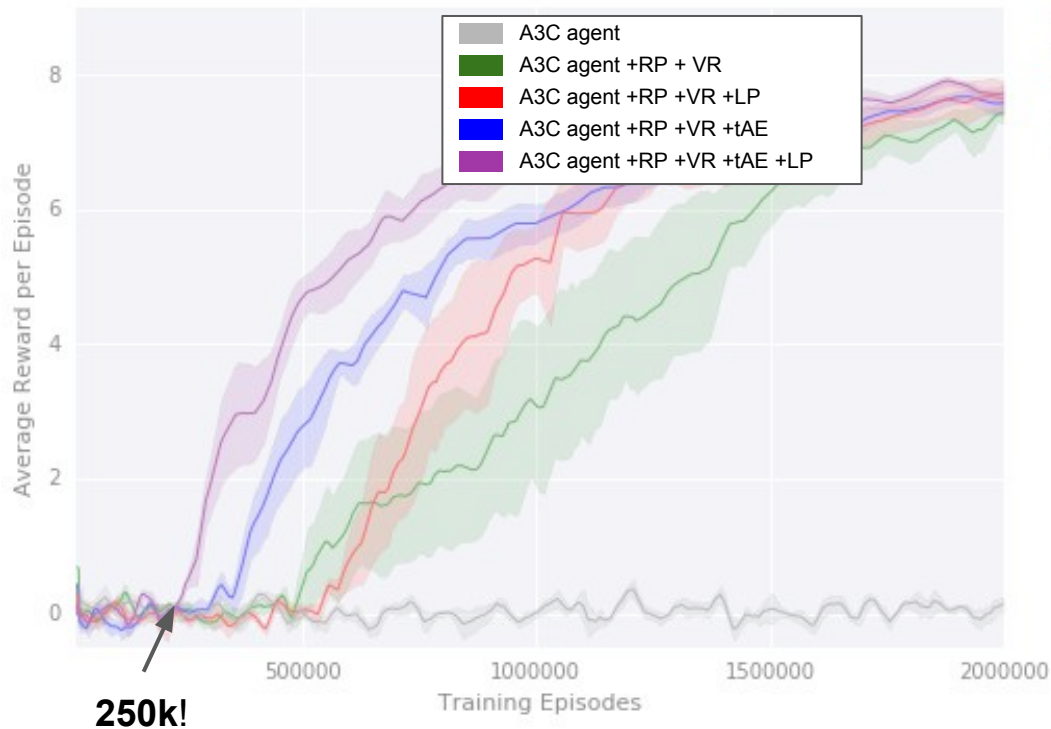
"larger" (mug)  
"smaller" (mug)

## Test

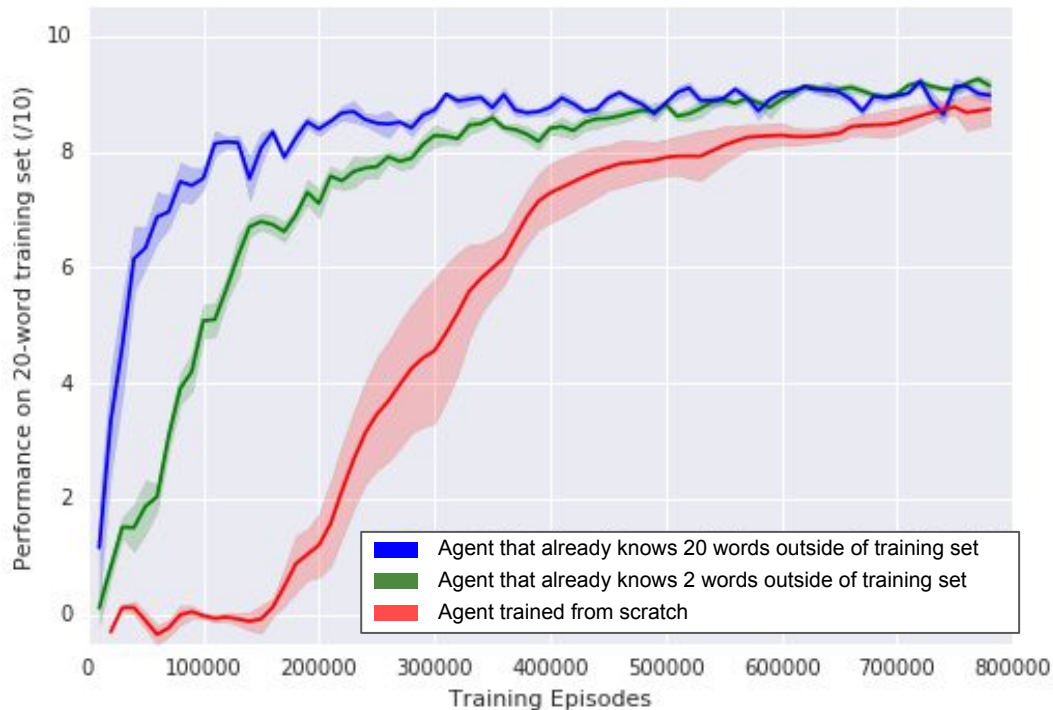
"larger" (pencil)  
"smaller" (pencil)

x 10

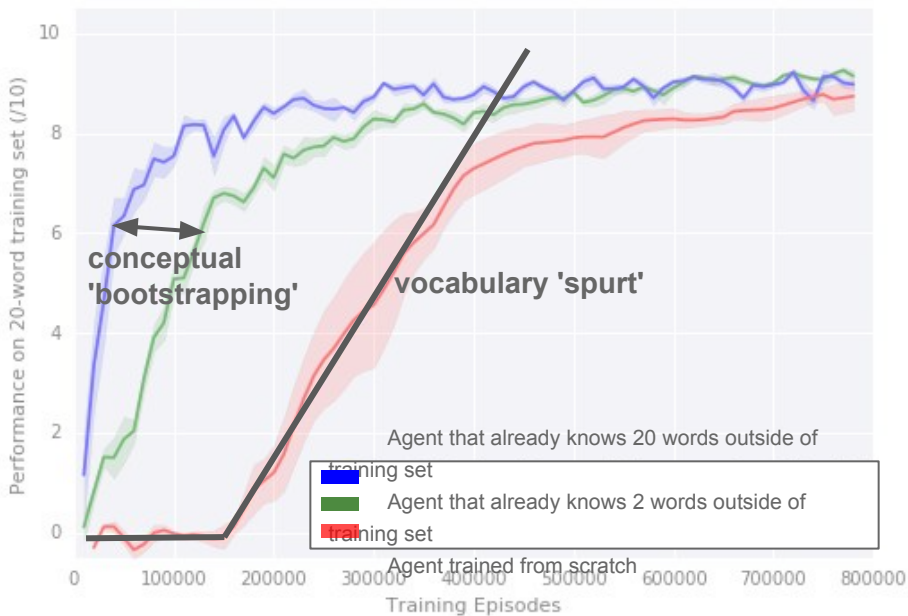
# But sample complexity remains an issue



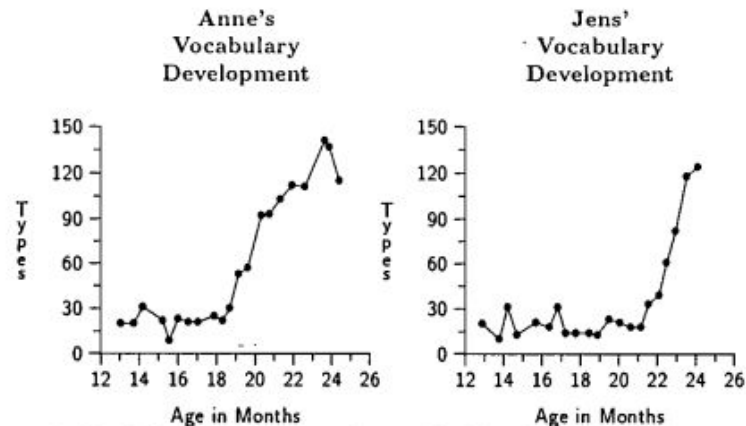
# Knowing some words makes learning more fast



# This mirrors the 'Vocabulary Spurt' observed in infants



296 K. Plunkett et al.



**Figure 1.** Vocabulary development in two Danish children: the plateau period is followed by a period of accelerated growth referred to as the vocabulary spurt.



# Summary

Natural Language Understanding requires models that can exploit the syntactic structure of language to produce ground representations of meaning.

We have a number of plausible models to achieve this, but we still require richer training environments in order to fully utilise them.

