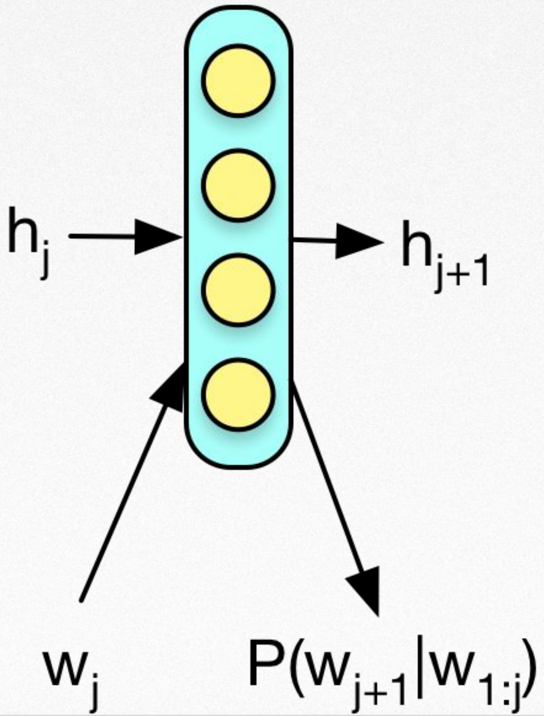
Recurrent Neural Networks and Models of Computations

Edward Grefenstette
etg@google.com

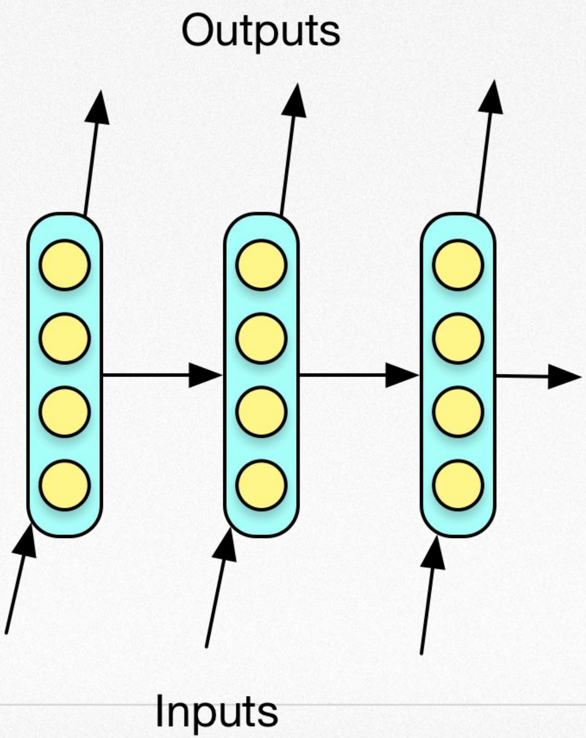
DeepMind

Some Preliminaries: RNNs



- Recurrent hidden layer outputs distribution over next symbol/label/nil
- Connects "back to itself"
- Conceptually: hidden layer models history of the sequence.

Some Preliminaries: RNNs



- RNNs fit variable width problems well
- Unfold to feedforward nets with shared weights
- Can capture long(ish) range dependencies

The Ubiquity of RNNs

RNNs: an established class of architectures for dealing with sequence data.

Turning point: Long Short Term Memory (Hochreiter and Schmidhuber, 1997; Gers and Schmidhuber, 2000)

A (relatively) **simple** architecture which adapts well across domains.

What do its **failure modes** tell us? What should research focus on?

Let's review some notable successes first...

Language Modelling

Task: Model the joint probability of a sequence of tokens $P(t_1, \ldots, t_n)$.

Factorise it as $\prod_{i \in [1,n]} P(t_i | t_1, \ldots, t_{i-1})$.

n-gram models rely on order-n markov assumption to do this...

RNN cells model, in their activations, $P(t_i | t_1, \ldots, t_{i-1})$.

No explicit bound to the history conditioning prediction at any time step.

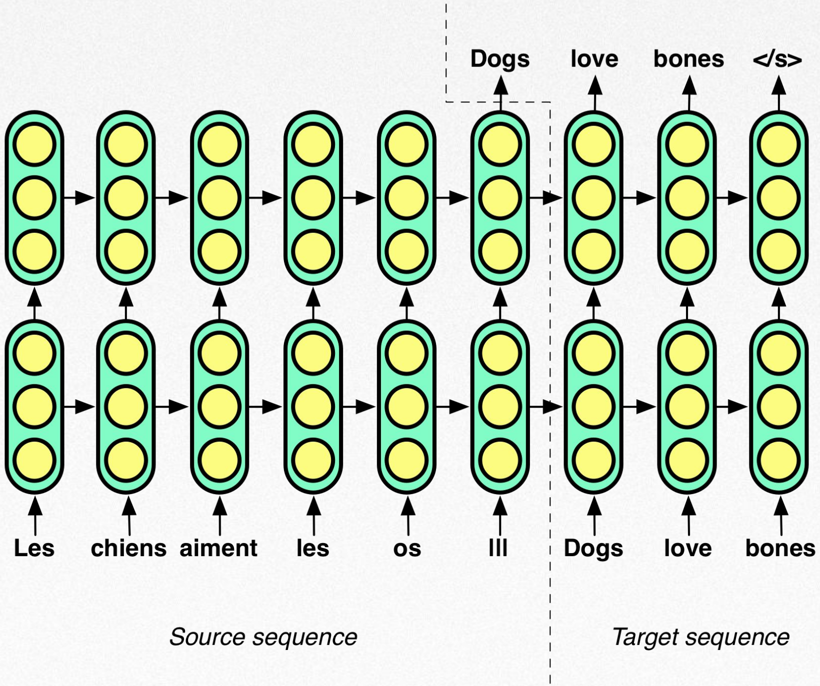
Sequence to Sequence Mapping with RNNs

Represent source sequence **s** and model probability of target sequence **t** via the conditional language modelling factorisation $P(t_{i+1}|t_1 \ldots t_n; \mathbf{s})$ with RNNs:

1. Read in source sequence to produce **s**.
2. Train model to maximise the likelihood of **t** given **s**.
3. Test time: Generate target sequence **t** (greedily, beam search, etc) from **s**.

Neural Machine Translation

$$P(\text{some english}|\text{du français})$$



(Sutskever *et al.* NIPS 2014)

Learning to Execute

Task (Zaremba and Sutskever, 2014):

- Read simple python scripts character-by-character
- Output numerical result character-by-character.

Input:
```
j=8584
for x in range(8):
  j+=920
b=(1500+j)
print((b+7567))
```
Target: 25011.

Input:
```
i=8827
c=(i-5347)
print((c+8704) if 2641<8500 else 5308)
```
Target: 12184.

Large-scale Supervised Reading Comprehension

The BBC producer allegedly struck by Jeremy Clarkson will not press charges against the "Top Gear" host, his lawyer said Friday. Clarkson, who hosted one of the most-watched television shows in the world, was dropped by the BBC Wednesday after an internal investigation by the British broadcaster found he had subjected producer Oisin Tymon "to an unprovoked physical and verbal attack." …

Cloze-style question:

Query: Producer **X** will not press charges against Jeremy Clarkson, his lawyer says.

Answer: Oisin Tymon

(Hermann *et al.* NIPS 2015)

Failure Modes of LSTM-RNNs: Language Modelling

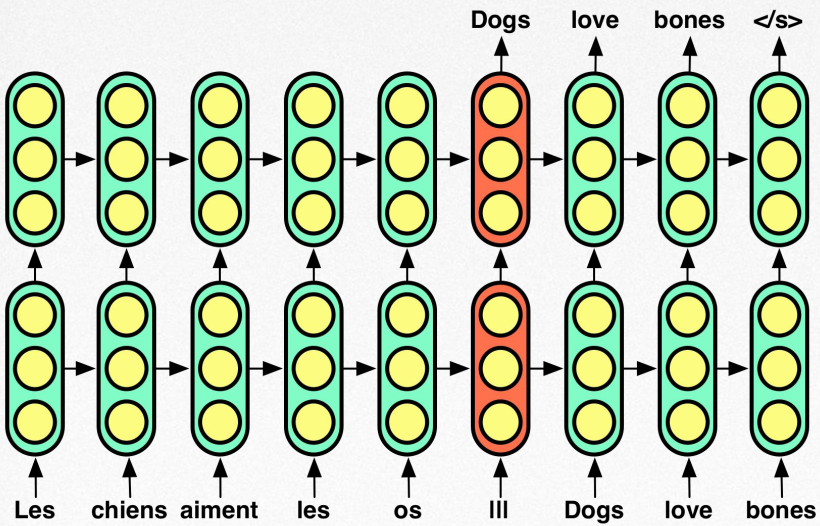
LSTMs make for good **local language models**, but bad at document-level context.

The **LAMBADA** dataset (Paperno *et al.* 2016)

1. Get some n-sentence long paragraphs from books, news, etc. (n≅3 here)
2. Get annotators to predict the (unseen) last word. Remove paragraphs with annotator disagreement.
3. Train LMs, remove paragraphs where they score above a likelihood threshold.
4. Get annotators to predict the last (unseen) word, observing the last sentence only. Remove paragraphs where they succeed.

That's your **test set**. Good luck!

Failure Modes of LSTM-RNNs: Sequence-to-Sequence



There's a **transduction bottleneck**:

- Non-adaptive capacity
- Target sequence modelling dominates training
- Gradient-starved encoder
- Fixed size considered harmful?

Failure Modes of LSTM-RNNs: Copy/Reverse

Randomly generated data:

1. Sample a length l from e.g. 8 to 64.
2. Sample l integers from 1 to N to form a sequence.
3. Target: copy/reverse sequence after reading it.

LSTM seq2seq can do this quite well (it takes a while).

It will "generalise" to **unseen sequences** in the [8, 64] token range.

Immediate failure on sequences in range [65, ...].

More parameters does not help.

Computational Hierarchy

Are RNNs here? → Turing Machines (computable functions)
Sieglemann & Sontag (1995)

⬆⬆⬆

Pushdown Automata (context free languages)

⬆⬆⬆

Finite State Machines (regular languages)

RNNs and Turing Machines

Simple RNNs (basic, GRU, LSTM) **cannot*** learn Turing Machines:

- RNNs do not **control** the "tape". Sequence exposed in forced order.
- **Maximum likelihood** objective ($p(x|\theta)$, $p(x,y|\theta)$, ...) produces model close to training data distribution.
- Can we reasonably expect regularisation to yield **structured computational model** as an out-of-sample generalisation mechanism?

* Through "normal" sequence-based maximum likelihood training.

RNNs and Finite State Machines

Not a proof, but think of simple RNNs as approximations of FSMs:

- Effectively order-N Markov chains, but N need not be specified
- **Memoryless** in theory, but can simulate memory through dependencies:

  E.g. "**.*a...a**" → p(X="a"|"a" was seen four symbols ago)
- Very limited, **bounded** form of memory
- **No incentive** under ML objectives to learn dependencies beyond the sort and range observed during training

RNNs and Finite State Machines

Some problems:

- RNN state acts as both controller and "memory"
- Longer dependencies require more "memory"
- Tracking more dependencies requires more "memory"
- More complex/structured dependencies require more "memory"

Why more than FSM?

Natural Language is arguably at least Context Free (need at least a PDA)

Even if it's not, rule parsimony matters!

E.g. model $\mathbf{a}^n\mathbf{b}^n$, if in practice n is never more than N.

Regular language (N+1 rules)

$\varepsilon|(ab)|(aabb)|(aaabbb)|\ldots$

CFG (2 rules)

$S \rightarrow a\,S\,b$

$S \rightarrow \varepsilon$

Computational Hierarchy

We we want to be here →

Turing Machines (computable functions)

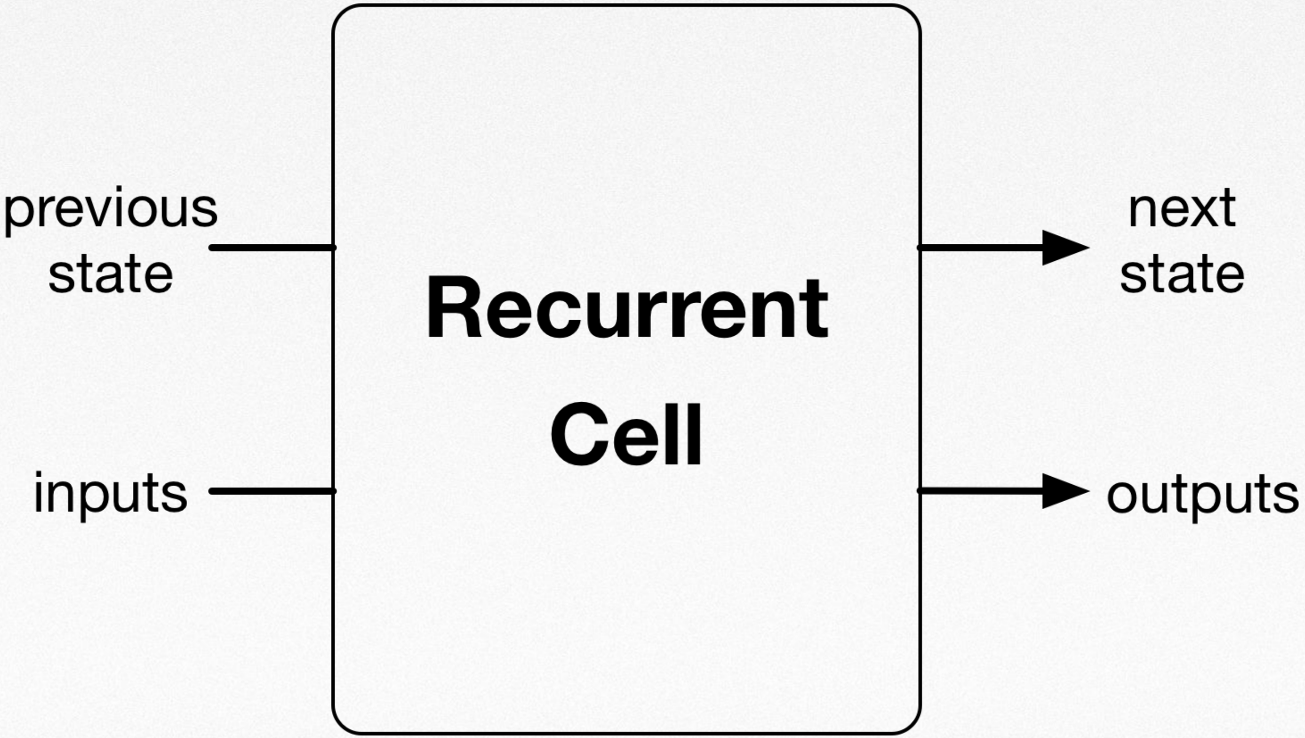
⬆ ⬆ ⬆

Pushdown Automata (context free languages)

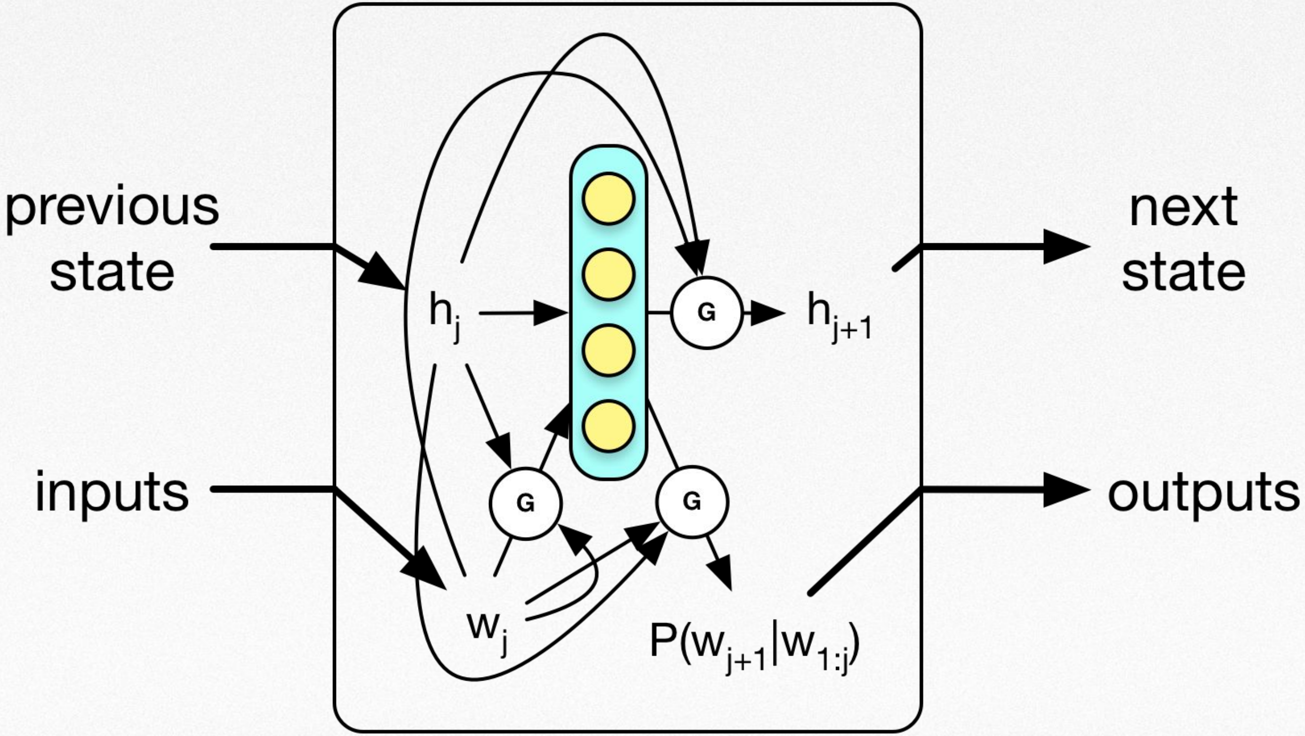
⬆ ⬆ ⬆

We are here → Finite State Machines (regular languages)

RNNs: More API than Model



RNNs: More API than Model

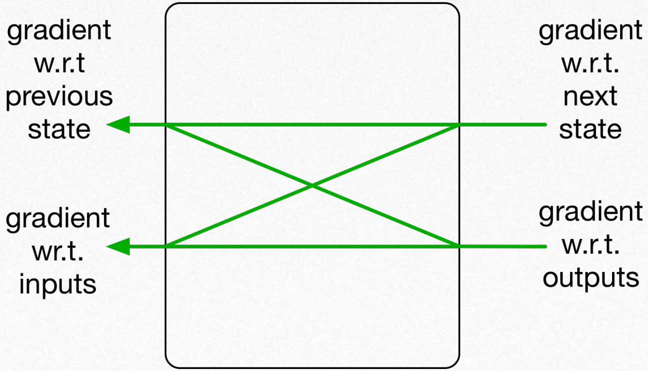


RNNs: More API than Model

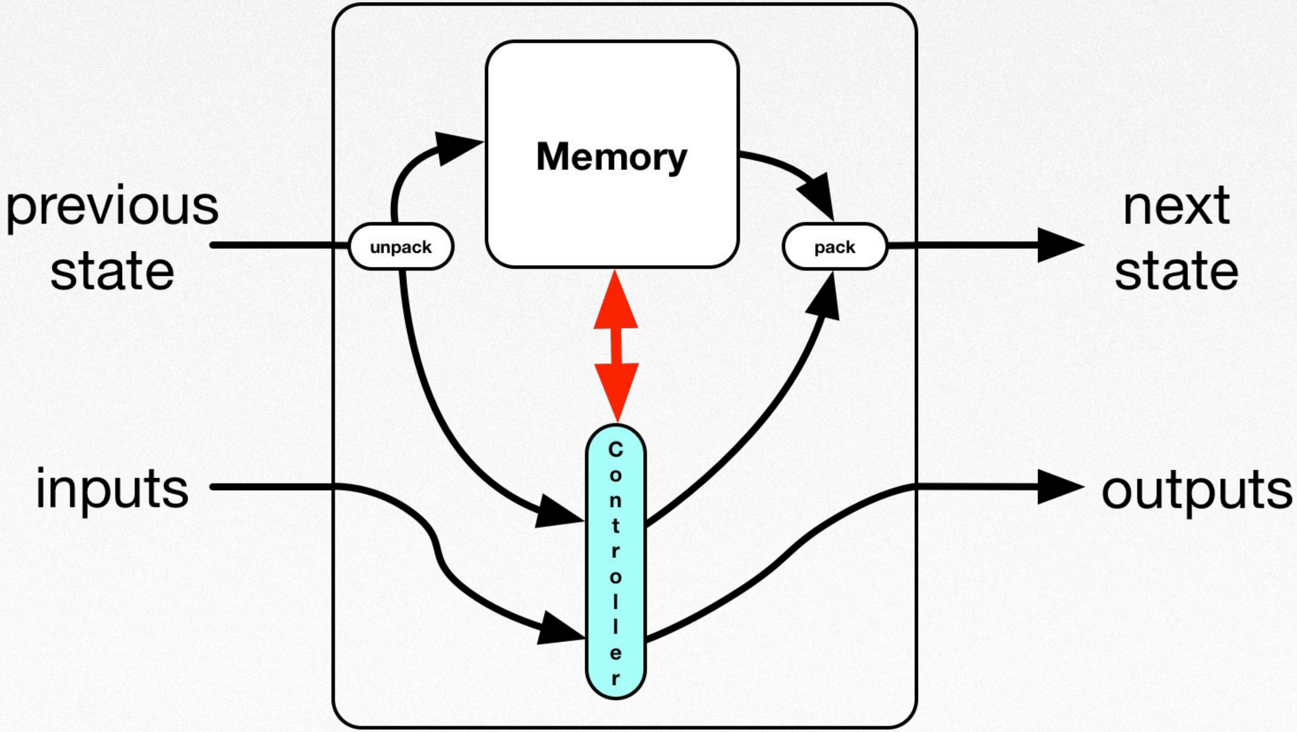
We aim to satisfy the following constraint (with some exceptions):

$$\forall x_t \in \bar{X}, p_t \in \bar{P}, y_t \in \bar{Y}, n_t \in \bar{N} \quad \frac{\partial y_t}{\partial x_t}, \frac{\partial y_t}{\partial p_t}, \frac{\partial n_t}{\partial x_t}, \frac{\partial n_t}{\partial p_t} \quad \text{are defined.}$$

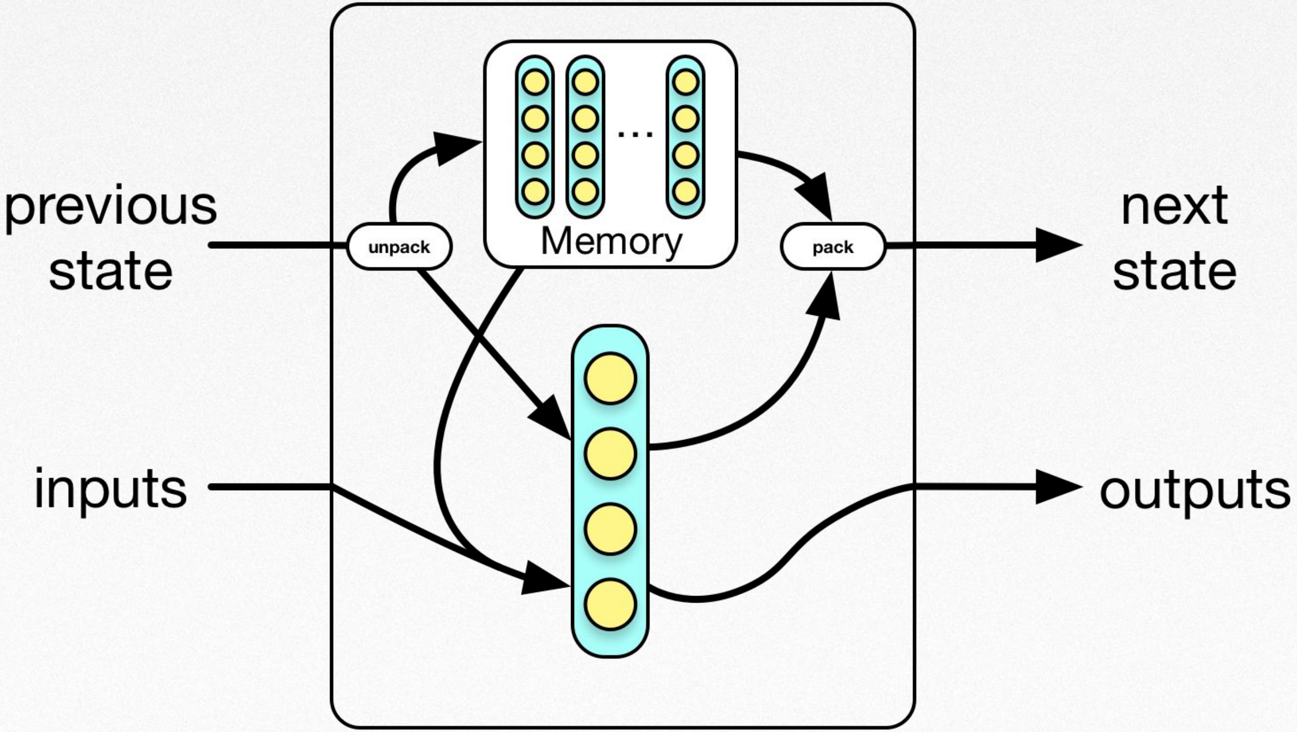
where the bar operator indicates flattened sets.



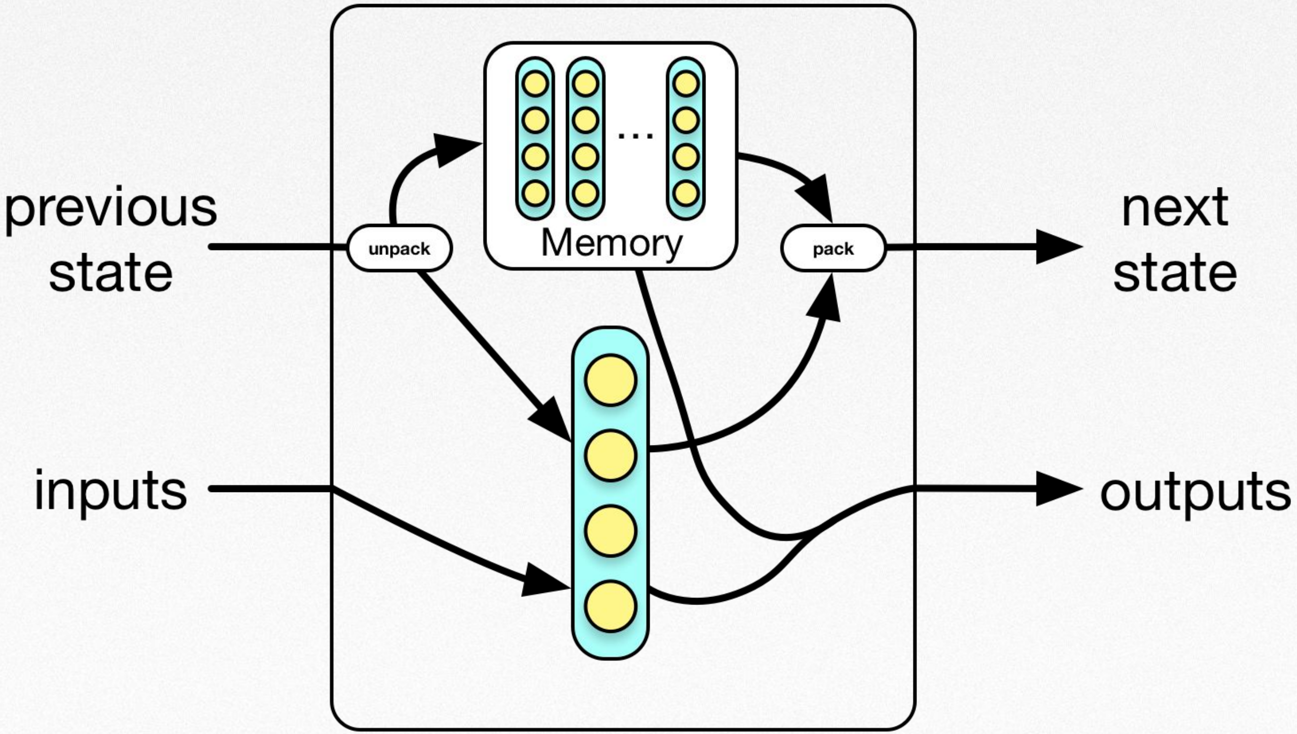
The Controller-Memory Split



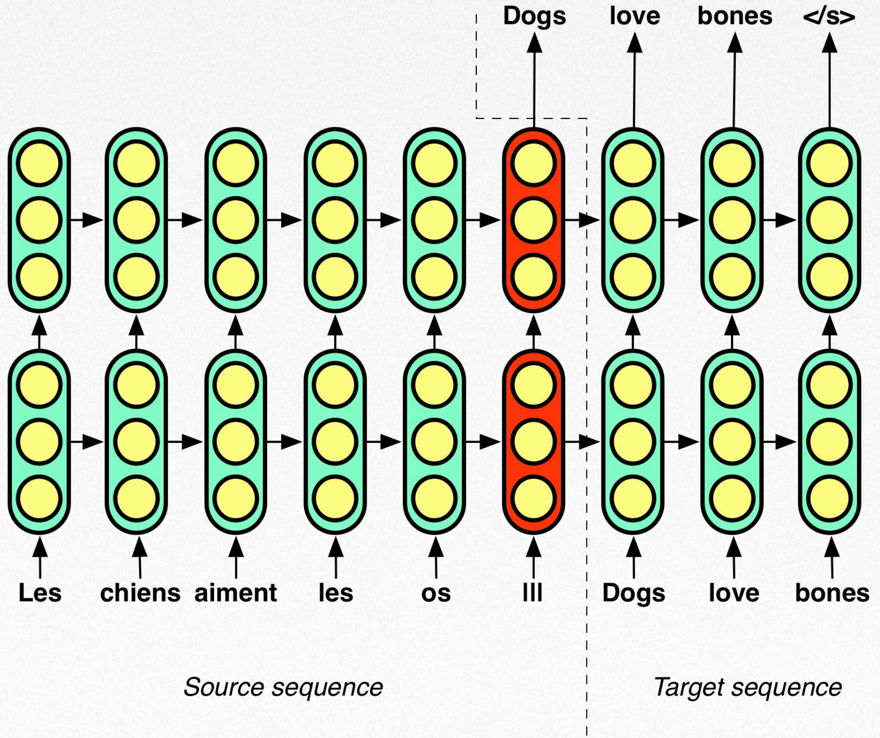
Attention (Early Fusion)



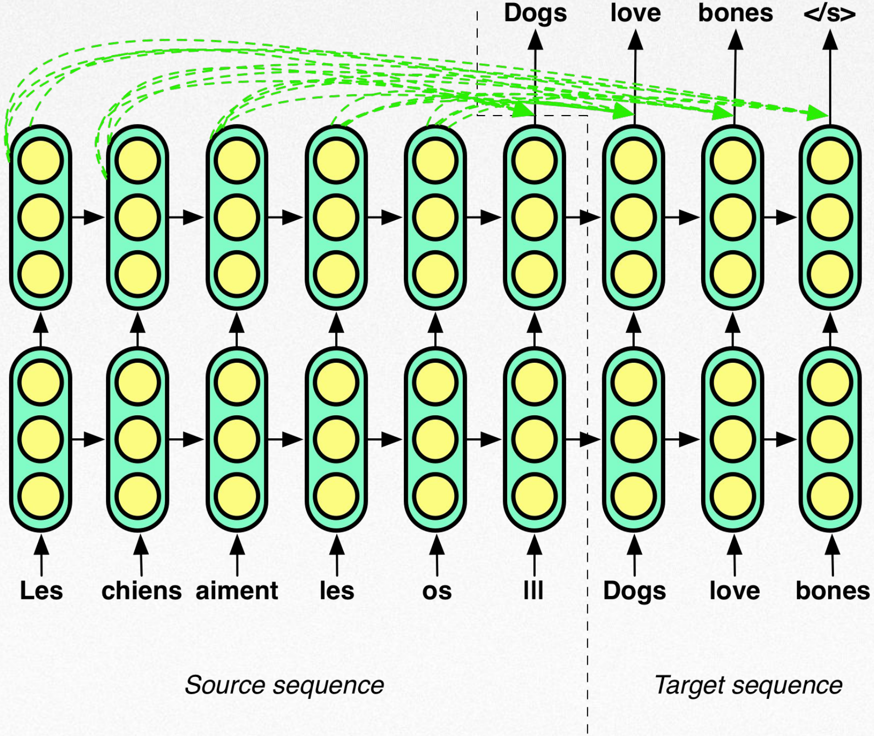
Attention (Late Fusion)



Skipping the bottleneck



Skipping the bottleneck



Limitations of ROM + RNN

Constrained to **one-to-one** or **one-to-many** alignments.

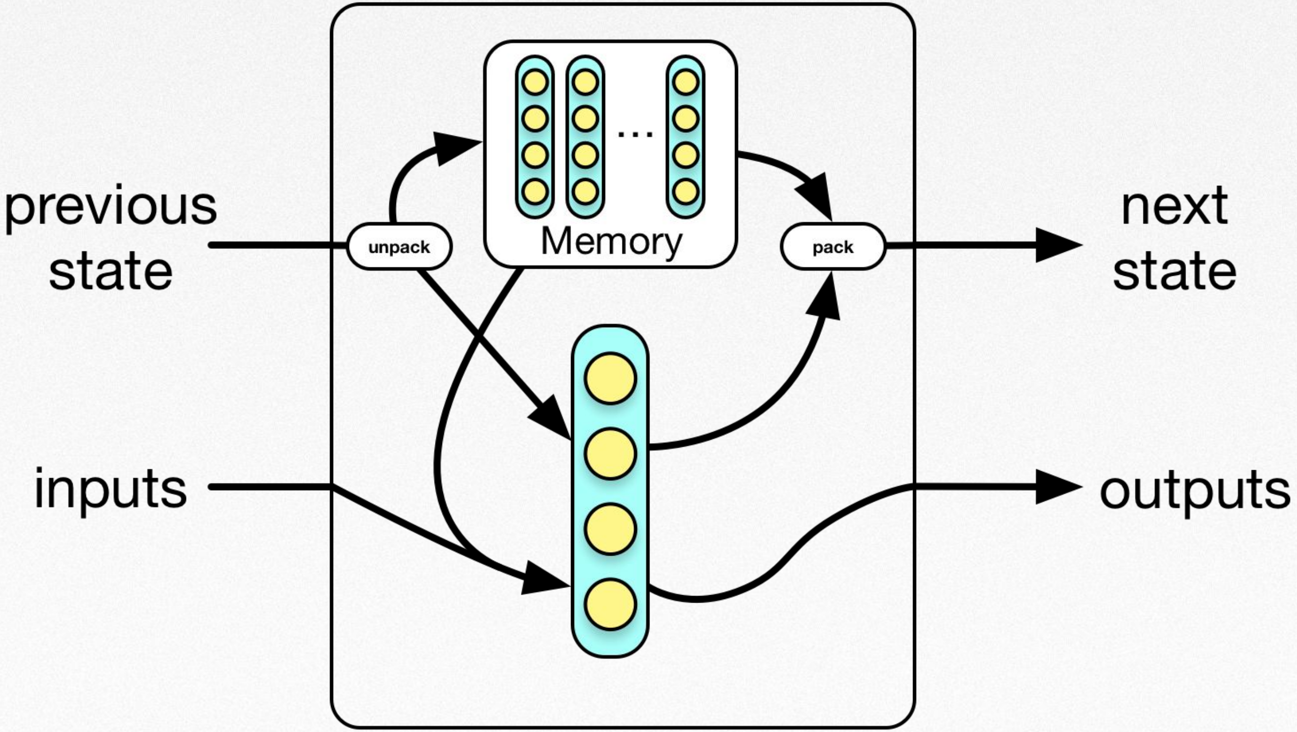
Representations must be **updated** across documents with model changes.

Multi-hop attention is difficult without changing ROM.

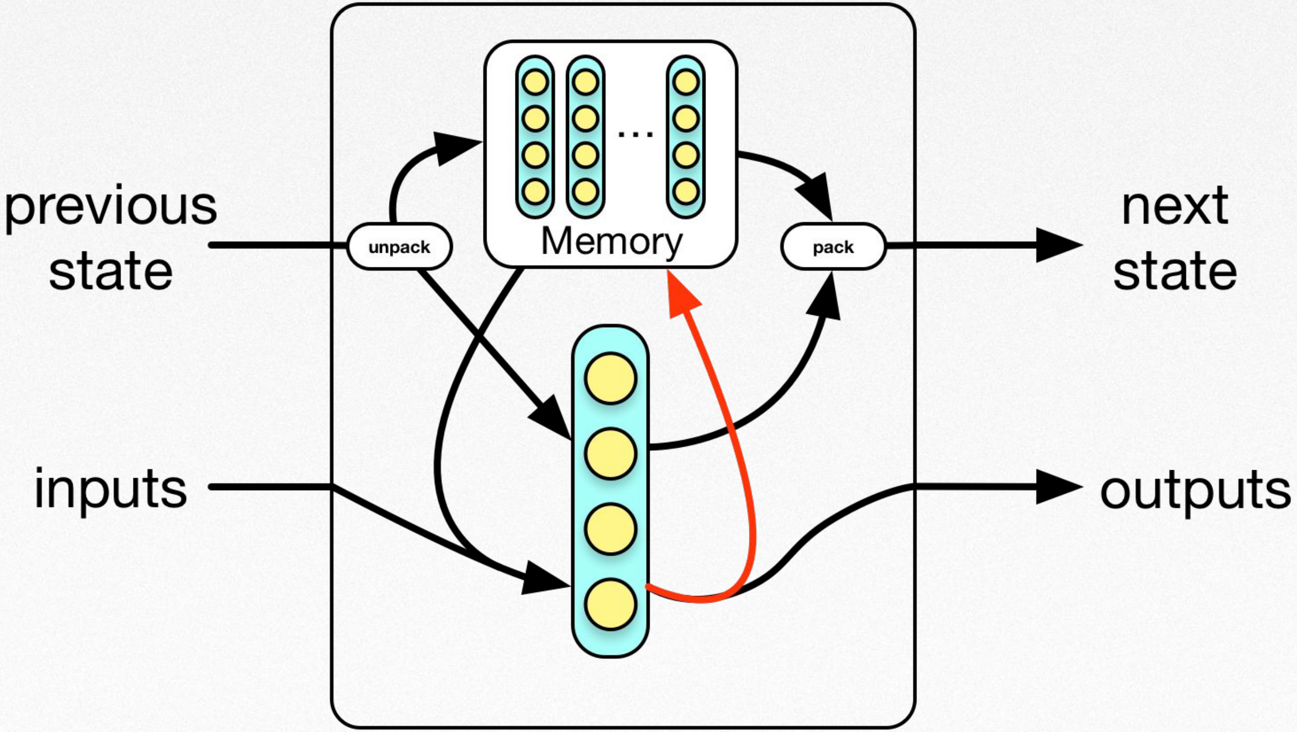
Risk of **information overload**. No explicit sense of saliency.

Scalability is an issue.

Attention as ROM



Register Memory as RAM



Relation to actual Turing Machines

Part of the "tape" is **internalised**

Controller can **control tape motion** via various mechanisms

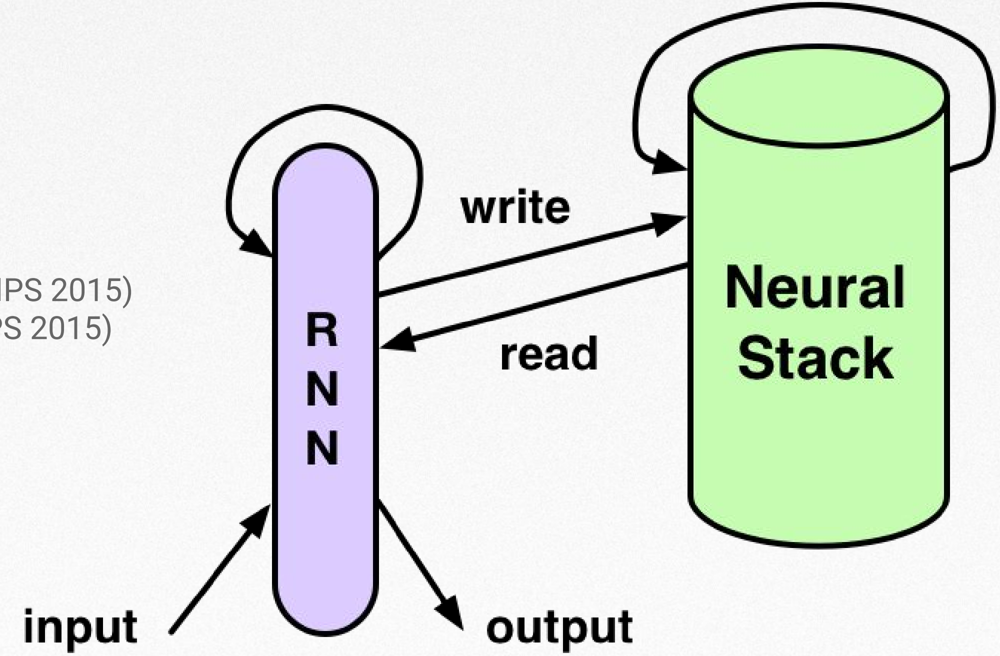
RNN could model **state transitions**

In ML-based training, **number of computational steps is tied to data**

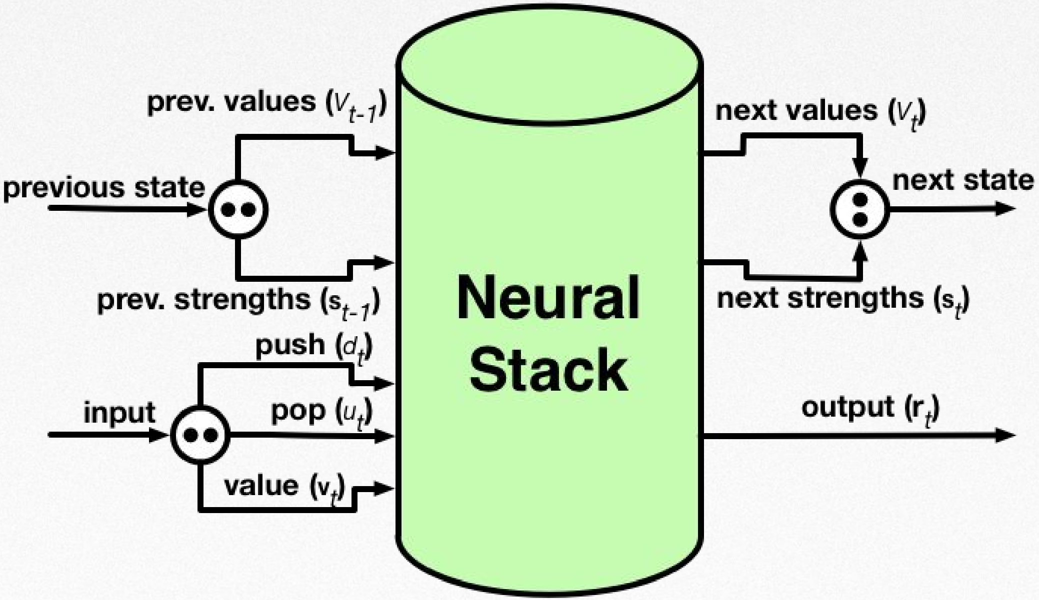
Unlikely(?) to learn a general algorithm, but experiments (e.g. Graves *et al.* 2014) show better **generalisation on symbolic tasks**.

Controlling a Neural Stack

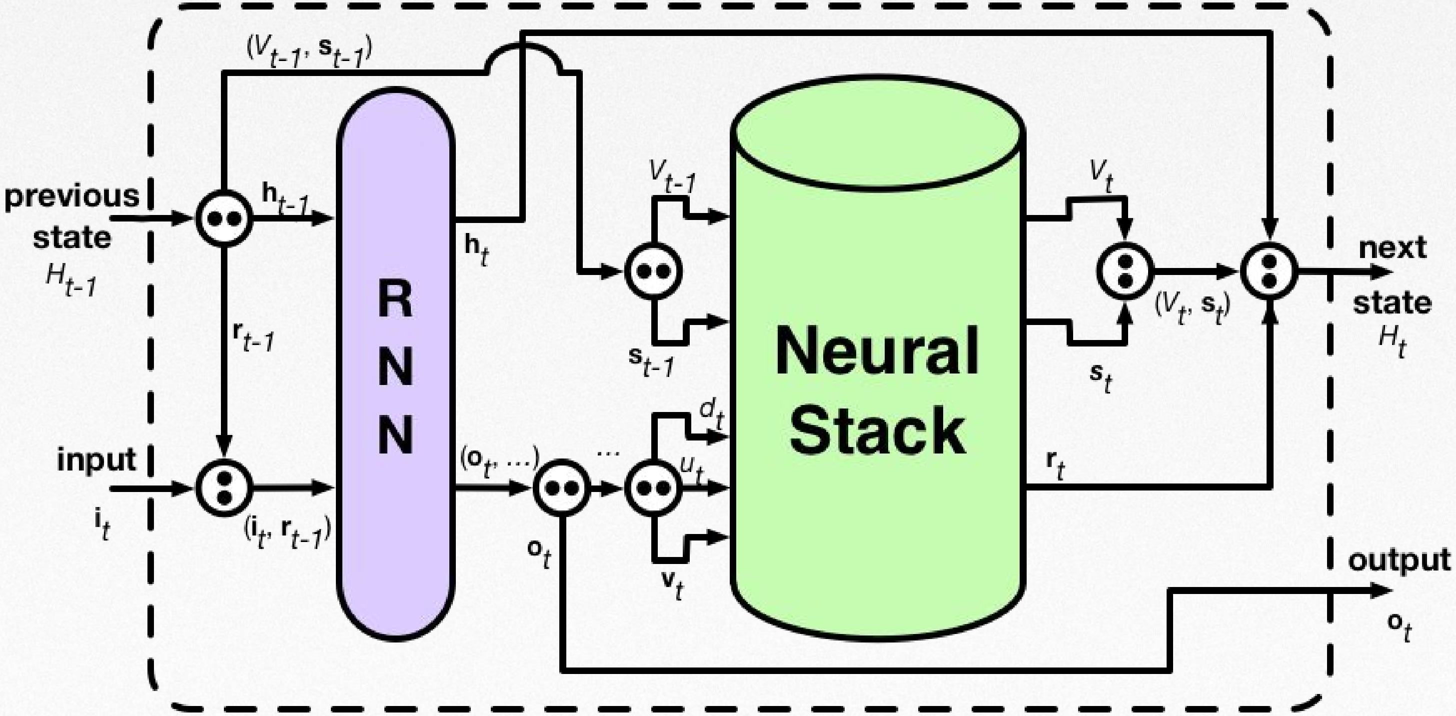
(Joulin and Mikolov, NIPS 2015)
(Grefenstette *et al.* NIPS 2015)



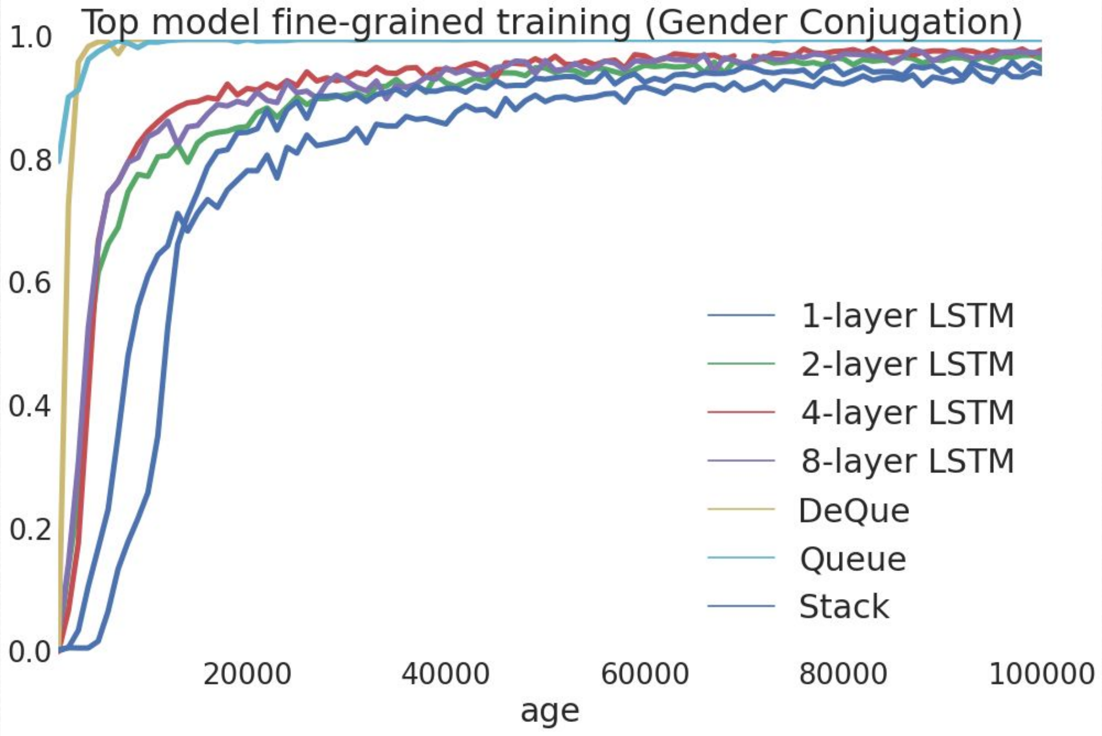
Stack API


prev. values (V_{t-1})
previous state
prev. strengths (s_{t-1})
push (d_t)
input
pop (u_t)
value (v_t)
Neural Stack
next values (V_t)
next state
next strengths (s_t)
output (r_t)

Controller + Stack Interaction



Rapid Convergence



Regular language (N+1 rules)

ε|(ab)|(aabb)|(aaabbb)|...

CFG (2 rules)

S → a S b

S → ε

Neural PDA Summary

- Decent **approximations of classical PDA**
- **Architectural bias** towards recursive/nested dependencies
- Should be useful for **syntactically rich** natural language
  - Parsing
  - Compositionality
  - But little work on applying these architectures
- **Limitation**: memory operations operate in lock-step with input-output.

Conclusions

Complexity needed, but it's easy to design an **overly complex** model.

Better to **understand limits of existing models** w.r.t. a problem.

By understanding the limitations and their nature, often better solutions **pop out by analysis**. Best example: Chapters 1-3 of Felix Gers' thesis (2001).

Think not just about the model, but about the **complexity of the problem** you want to solve.

THANK YOU

Credits

DeepMind Team

Additional Credits

Montreal Deep Learning Summer School 2016 attendees for their insightful comments.

https://deepmind.com/careers/