# Beyond Sequence to Sequence with Augmented RNNs

**Ed Grefenstette**

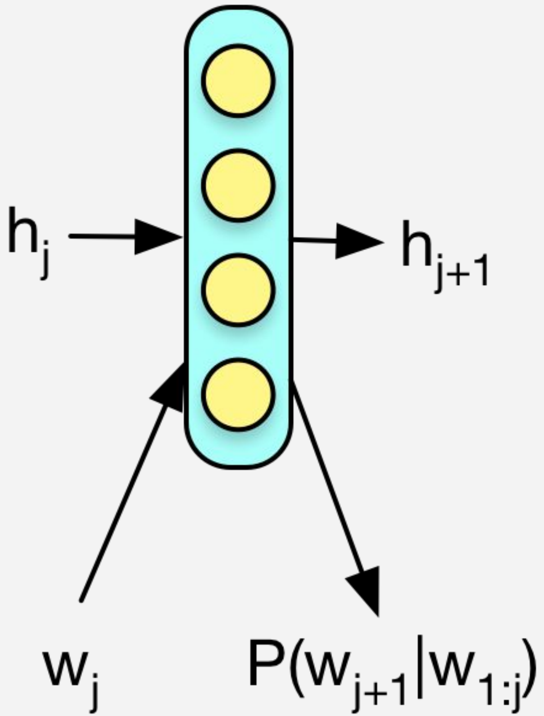
etg@google.com

# The plan

1. The Transduction Bottleneck
2. Limitations of RNNs
3. RNNs Revisited
4. Attention
5. Stacks
6. Register Machines

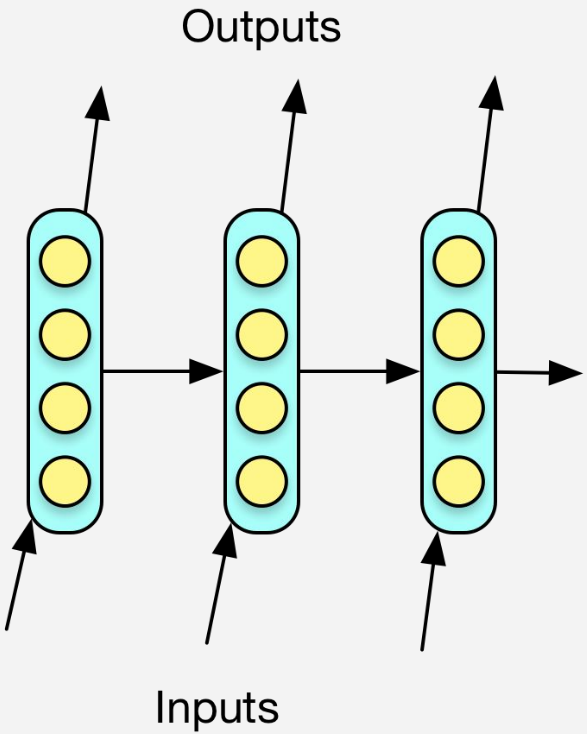
# The Bottleneck

# 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

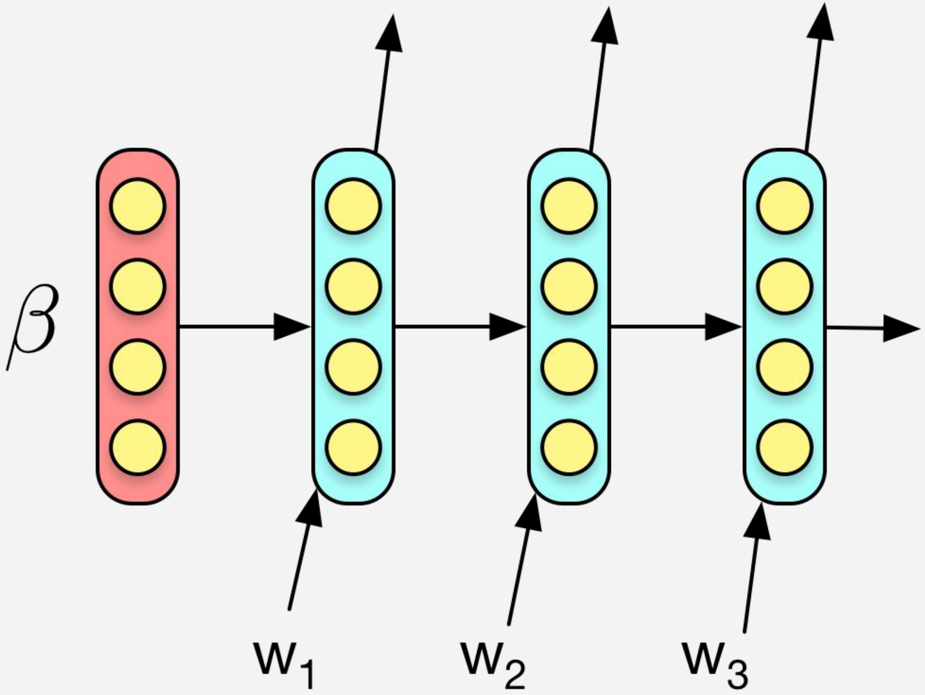
# Some Obvious RNN Applications

- Language modelling: chain rule gives us P(s) from $\prod_{t \in (1,T)} P(s_t|s_1, \ldots, s_{t-1})$.
- Sequence labelling: $P(l_t|s_1, \ldots, s_t)$
- Sentence classification: model P(l|s) from cell state, mean pooling, etc.

**But... simpler/better models exist for most of these.**

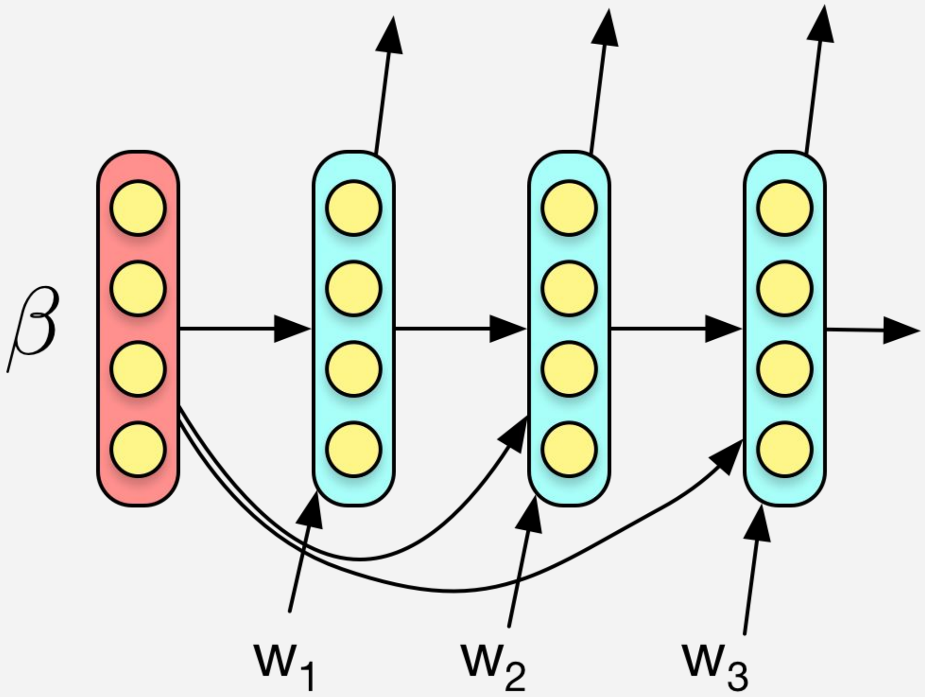
# Transduction with Conditional Models

$$P(\mathbf{w}|\beta)$$



# Transduction with Conditional Models

$$P(\mathbf{w}|\beta)$$



# Sequence to Sequence Mapping with RNNs

Many NLP (and other!) tasks are castable as transduction problems. E.g.:

**Translation:** English to French transduction

**Parsing:** String to tree transduction

**Computation(?!):** Input data to output data transduction

# Sequence to Sequence Mapping with RNNs

Generally, goal is to transform some source sequence

$$\mathbf{s} = s_1, s_2, \ldots, s_m$$

into some target sequence

$$\mathbf{t} = t_1, t_2, \ldots, t_n$$

# Sequence to Sequence Mapping with RNNs

Represent **s** and model $P(t_{i+1}|t_1 \ldots t_n; \mathbf{s})$ 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**.

# A Simple Encoder-Decoder Model

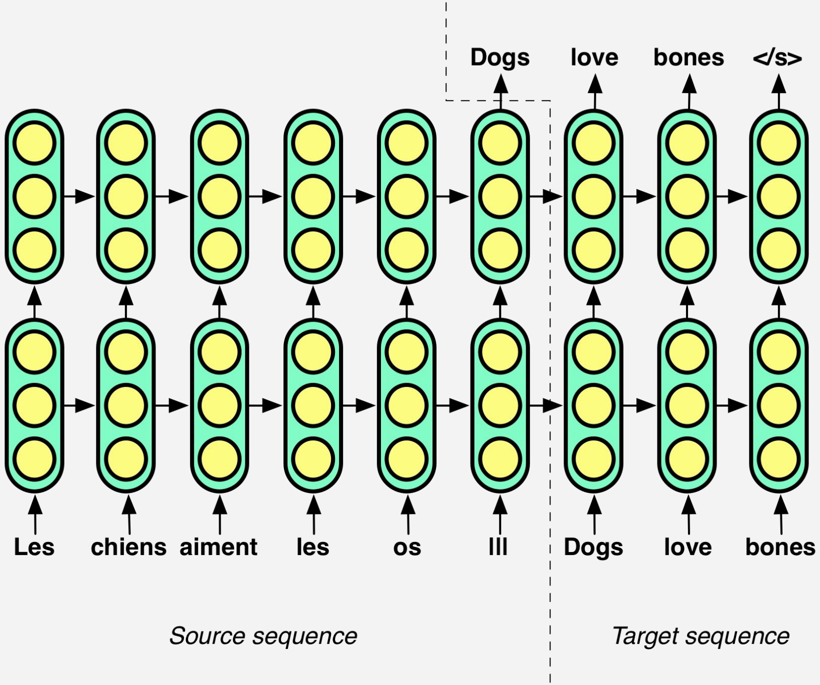
- Concatenate source and target sequences into joint sequences:

$$s_1\ s_2\ \ldots\ s_m\ |||\ t_1\ t_2\ \ldots\ t_n$$

- Train a single RNN or pair of RNNs over joint sequences
- Ignore RNN output until separator symbol (e.g. "|||")
- Jointly learn to compose source and generate target sequences

# Deep LSTMs for Translation

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

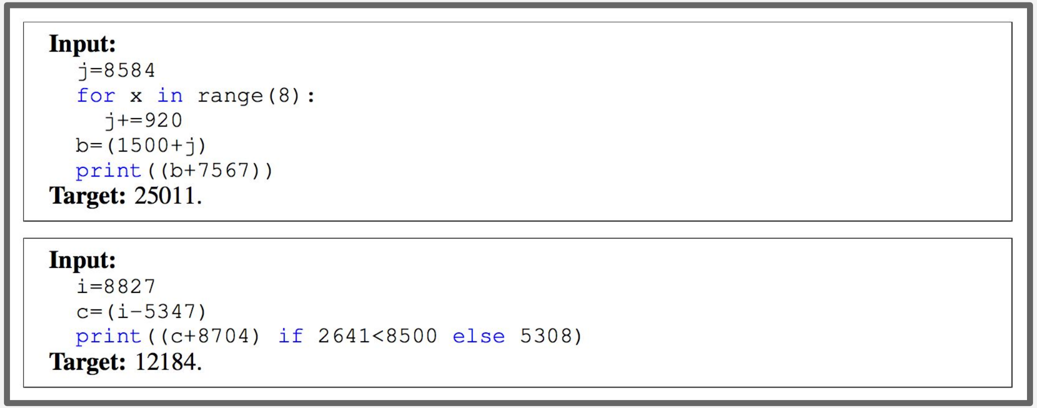


(Sutskever *et al.* NIPS 2014)

# 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))
```

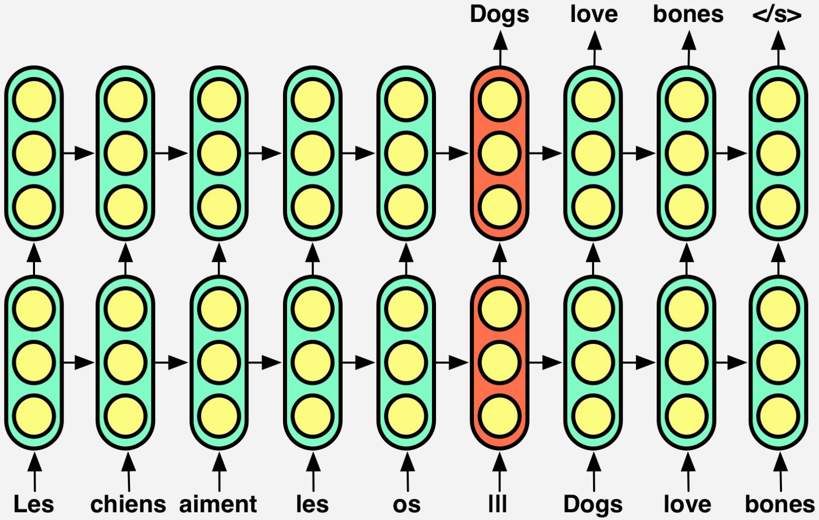
**Target:** 25011.

**Input:**

```
i=8827
c=(i-5347)
print((c+8704) if 2641<8500 else 5308)
```

**Target:** 12184.

# The Bottleneck for Simple RNNs



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

# Limitations of RNNs: A Computational Perspective

# Computational Hierarchy

Turing Machines (computable functions)

⬆ ⬆ ⬆

Pushdown Automata (context free languages)

⬆ ⬆ ⬆

Finite State Machines (regular languages)

# RNNs and Turing Machines

Sieglemann & Sontag (1995)

Any Turing Machine {Q, Γ, δ, ...} can be translated into an RNN:

- One-hot states as hidden layer, size |Q|
- One-hot encoding of symbols of Γ as input
- One-hot encoding of Γ ∪ {L, R} as outputs
- Identity as recurrence matrix, δ as update matrix

By extension, RNNs can express/approximate a set of Turing machines.

**But expressivity ≠ learnability!**

# 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.
- Insane to 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"
- Ultimately, FSMs are pretty basic.

# 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 $a^nb^n$, if in practice n is never more than N.

**Regular language (N+1 rules)**

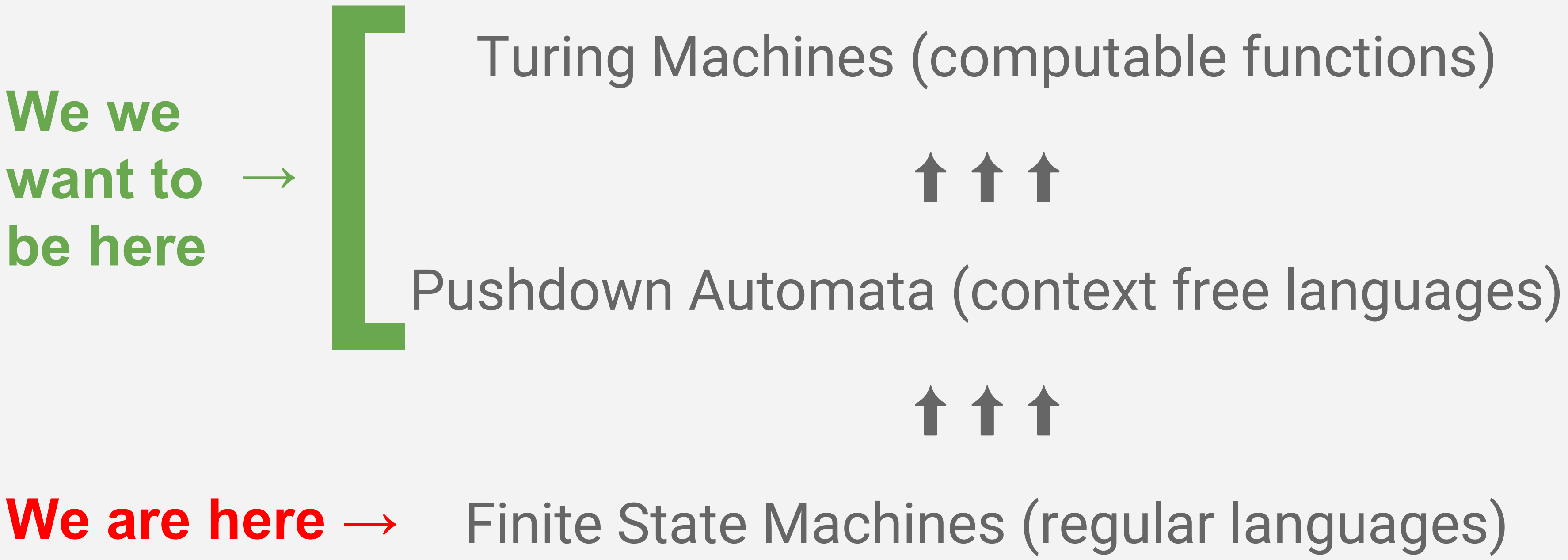
ε|(ab)|(aabb)|(aaabbb)|...

**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)

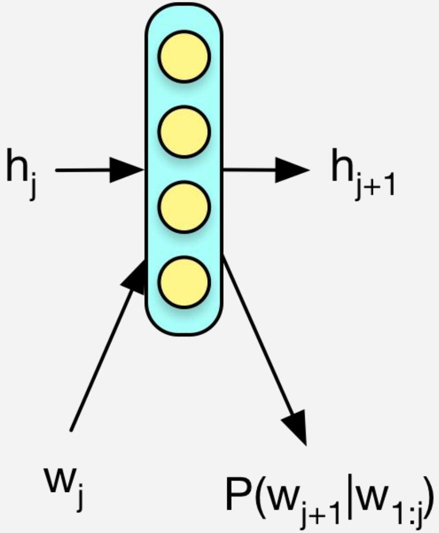
# The plan

1. The Transduction Bottleneck
2. Limitations of RNNs
3. RNNs Revisited
4. Attention
5. Stacks
6. Register Machines

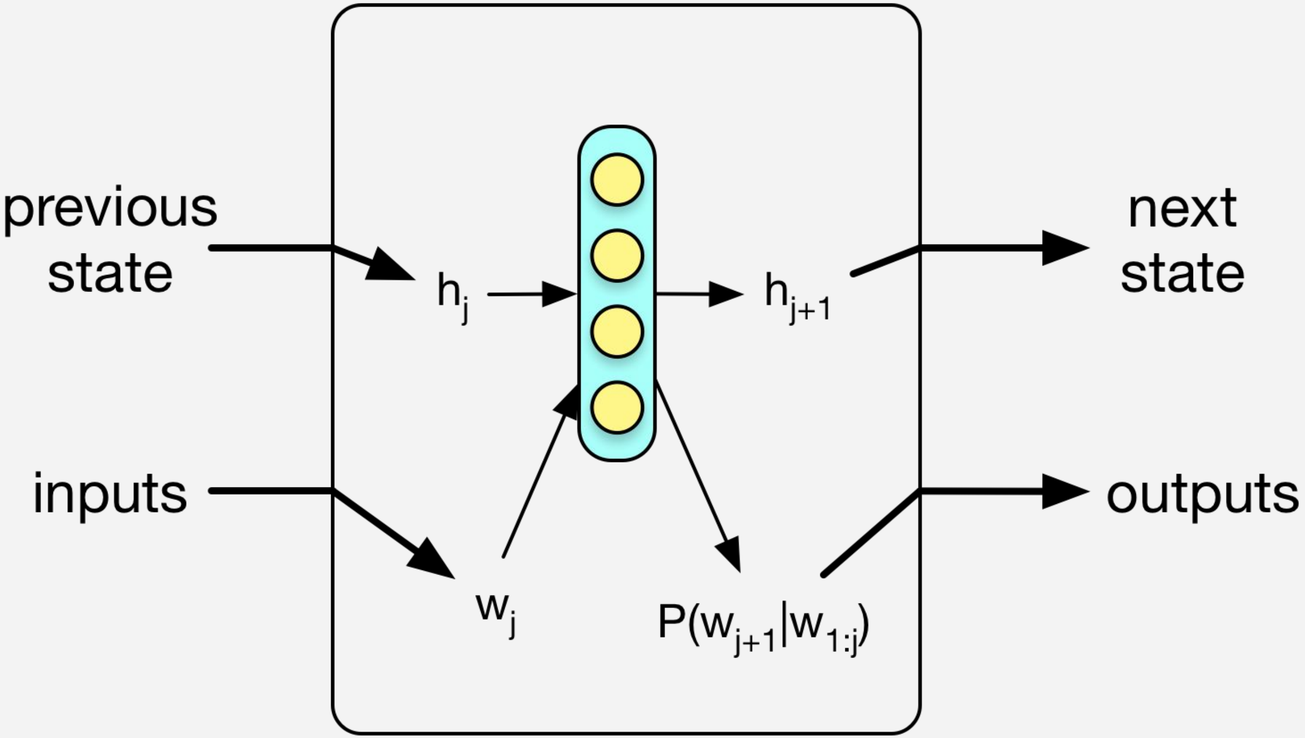
**Questions?**

# RNNs Revisited

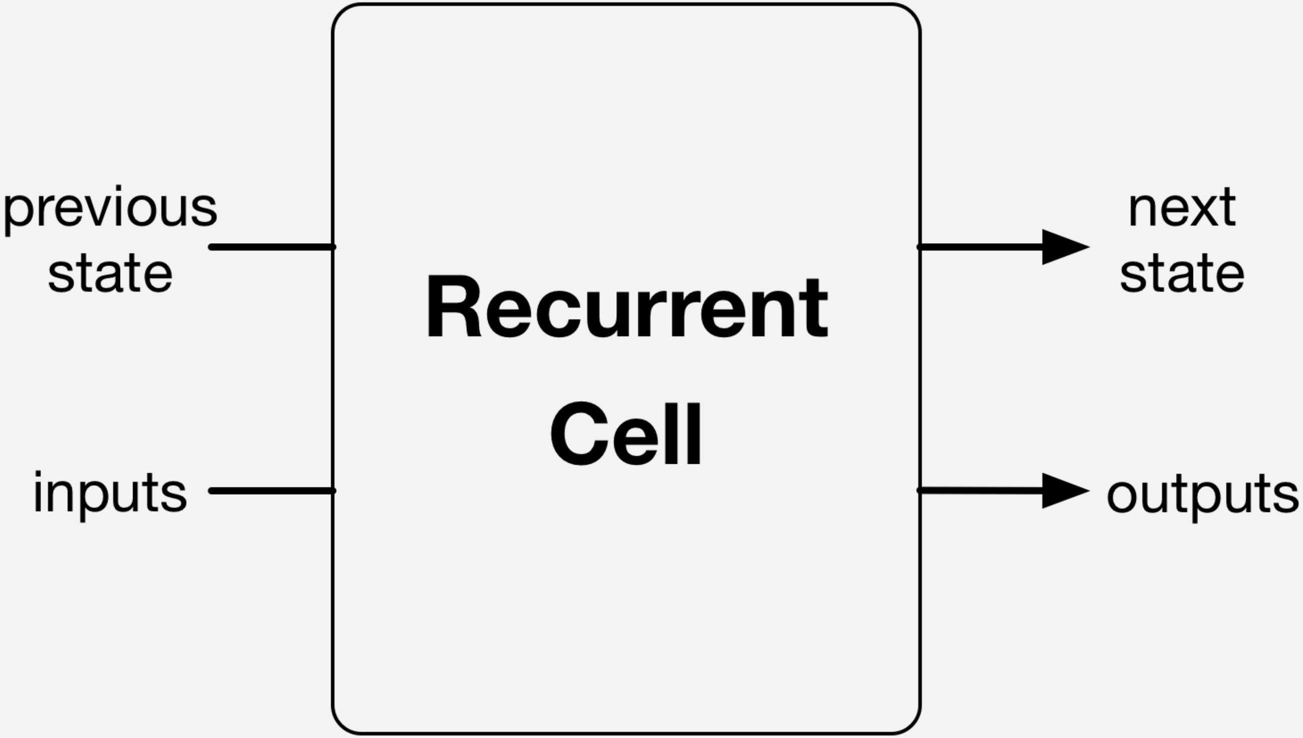
# RNNs: More API than Model



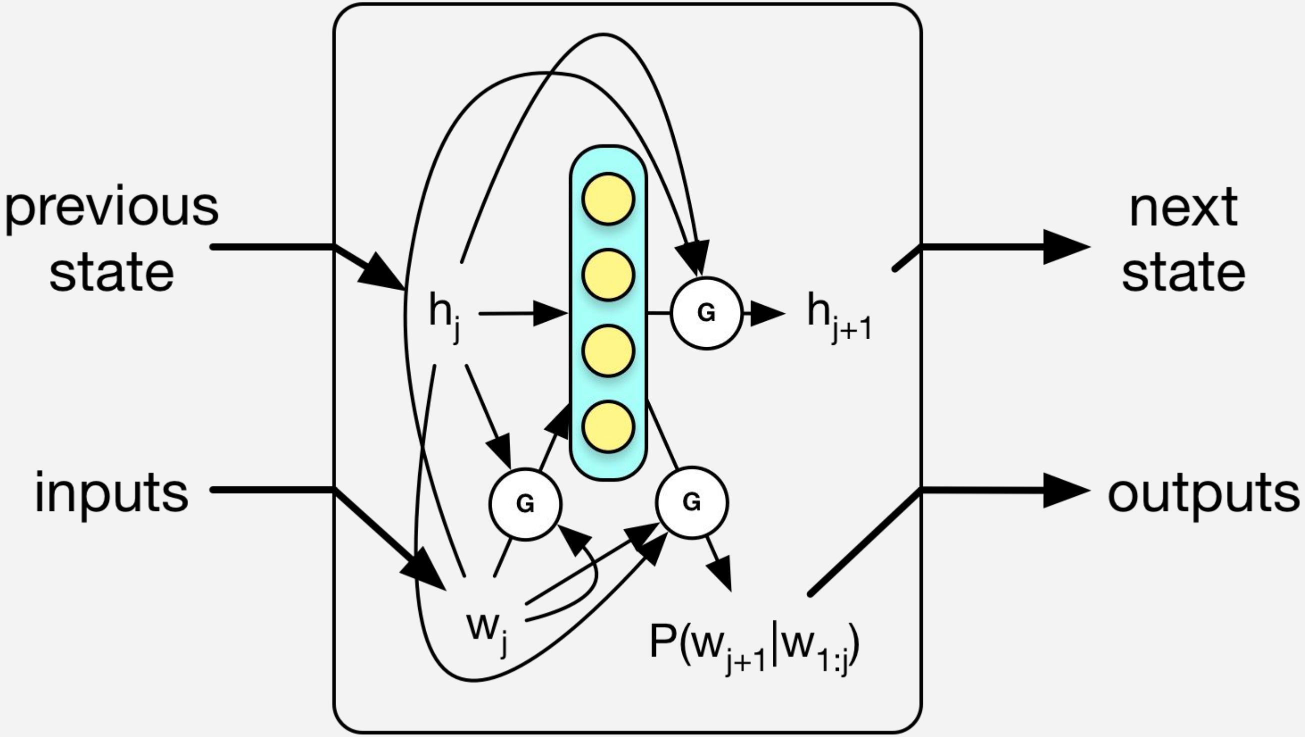
# RNNs: More API than Model



# RNNs: More API than Model



# RNNs: More API than Model



# RNNs: More API than Model

For a recurrent cell, we are modelling a function

$$\text{RNN}: X \times P \rightarrow Y \times N$$

where $x_t \in X$ is an input, $p_t \in P$ is the previous recurrent state, $y_t \in Y$ is an output, and $n_t \in N$ is an updated recurrent state, all possibly nested sets of vectors, scalars, ...

Typically $P = N$, and $p_{t+1} = n_t$ for $p_{t+1}, n_t \in P$.

Sometimes $X=Y$, and $x_{t+1} = f(y_t)$ where f is not necessarily differentiable.

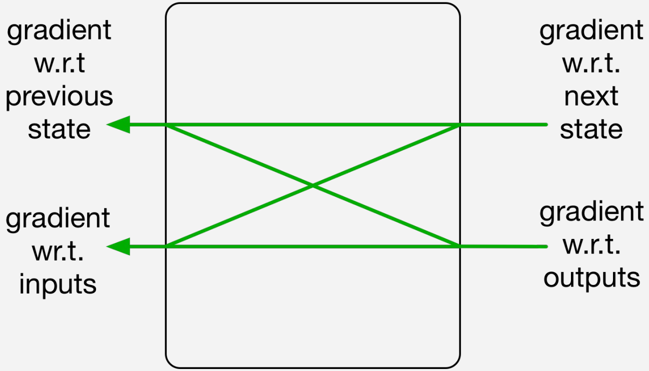
# RNNs: More API than Model

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

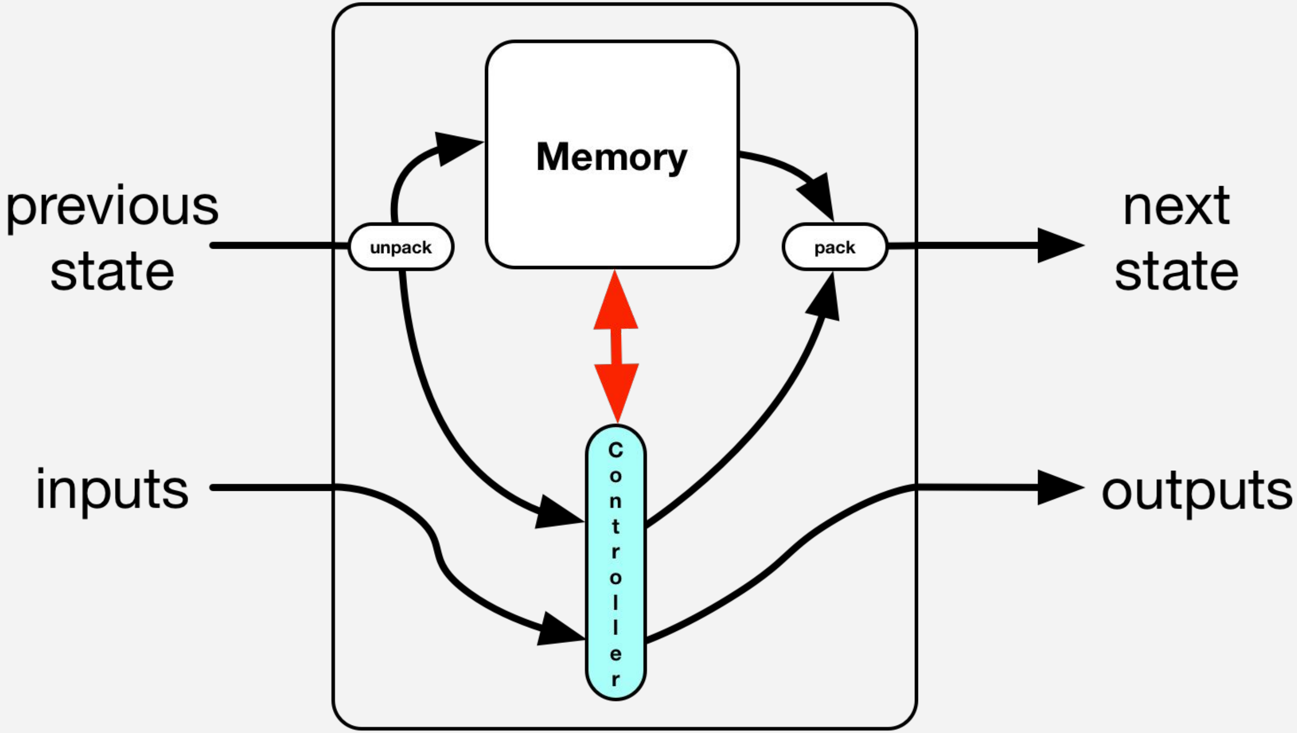
indicates flattened sets.



# The Controller-Memory Split

- Summary: keep things differentiable, use P/N to track state, do whatever you want within the cell.
- Cells can nest/stack/be sequenced, as long as everything "type checks".
- We can address some of the issues discussed before with this API by separating memory and controller.
- That's just the beginning!

# The Controller-Memory Split

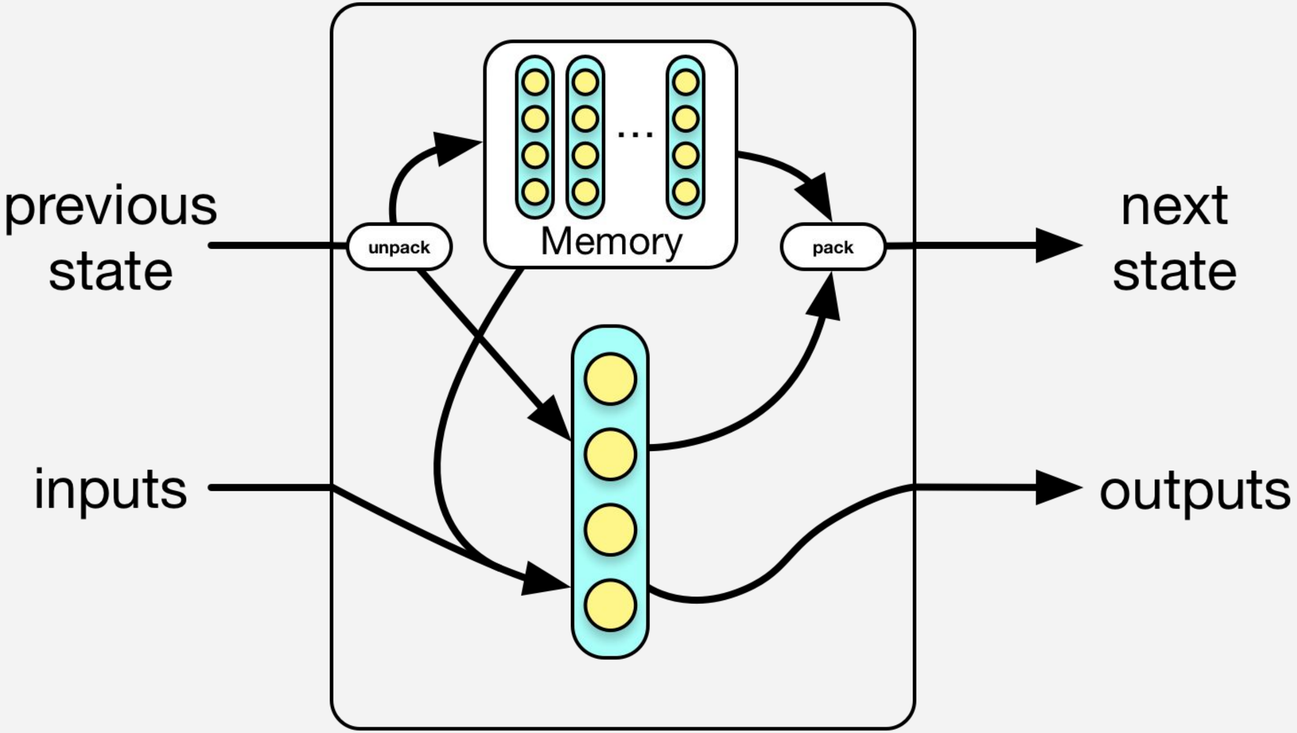


# Attention: ROM

# Attention

- You have an array of vectors representing some data.
- Your controller deals with I/O logic.
- You want to read your data array at each timestep.
- You want to accumulate gradients in your memory.

# Attention (Early Fusion)



# RNN: X × P → Y × N

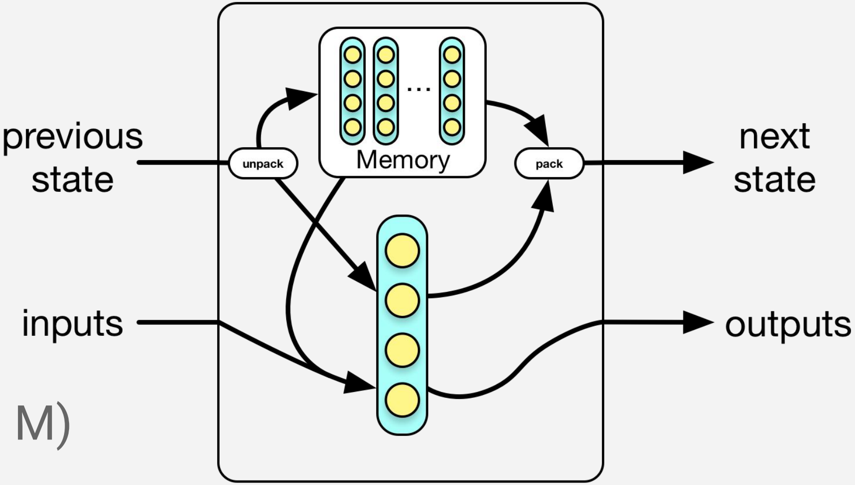
$p_t = (h_{t-1}, M)$

$y_{t\,,}\ h_t = \text{controller}(xs, h_{t-1})$

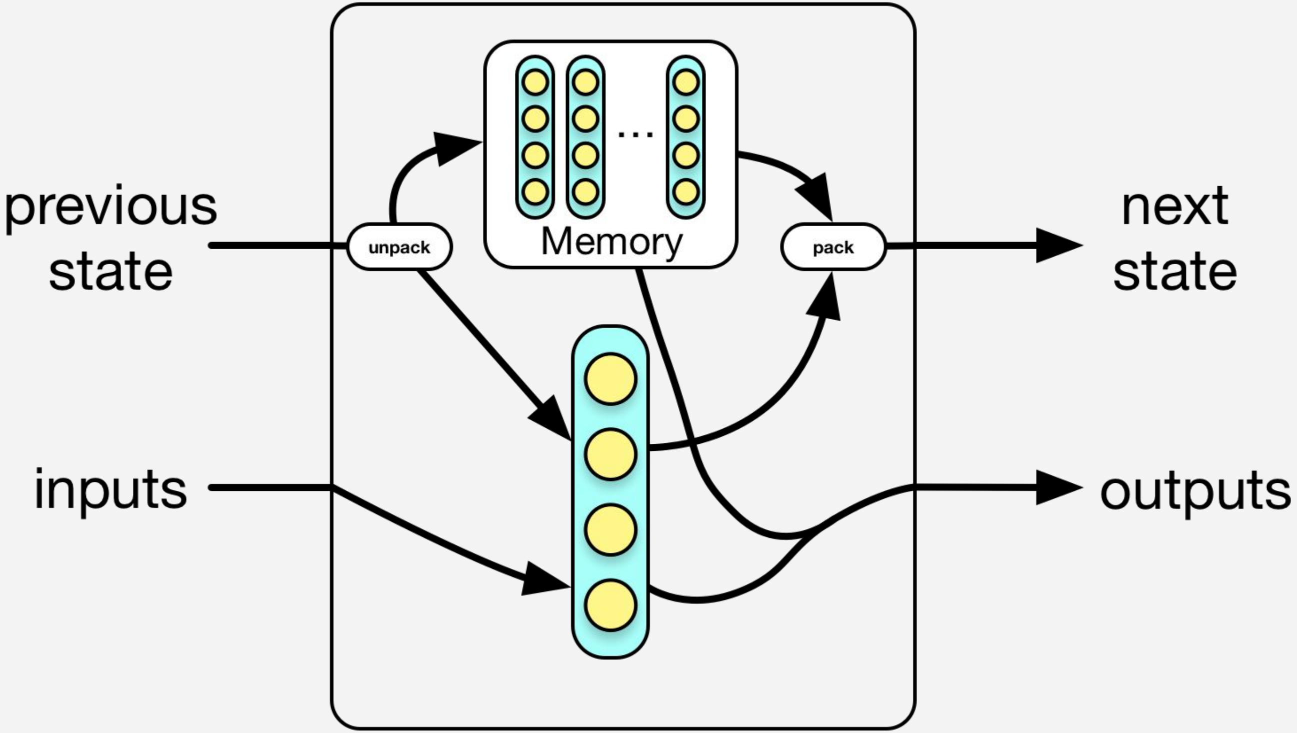
$n_t = (h_t, M)$

$xs = x_t \oplus f_{att}(h_{t-1}, M)$

e.g. $f_{att}(h, M) = M \times \text{softmax}(h \times W_{att} \times M)$



# Attention (Late Fusion)



# RNN: $X \times P \to Y \times N$

$p_t = (h_{t-1}, M)$

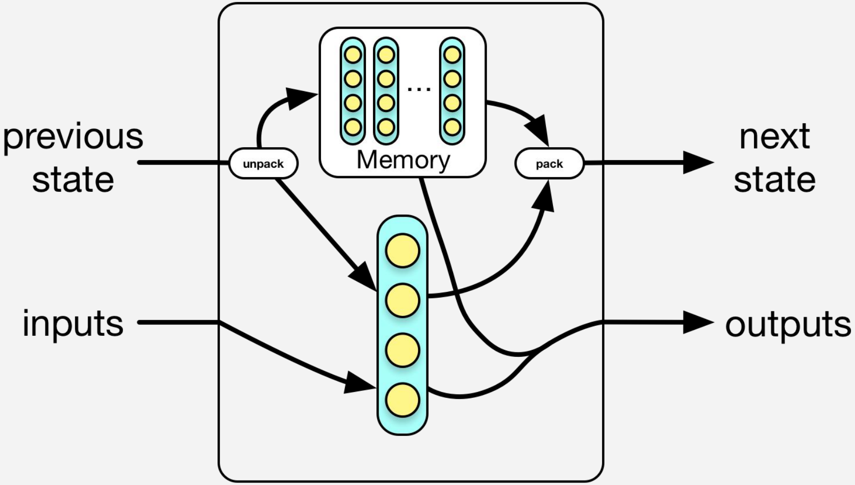
$n_t = (h_t, M)$

$w_{t\,,}\, h_t = \text{controller}(x_t, h_{t-1})$

$y_t = f_{comp}(w_t, f_{att}(h_t, M))$

Alternatively:

$y_t = f_{att}(h_t, M)$ if $f_{att}$ yields a distribution over memory positions.



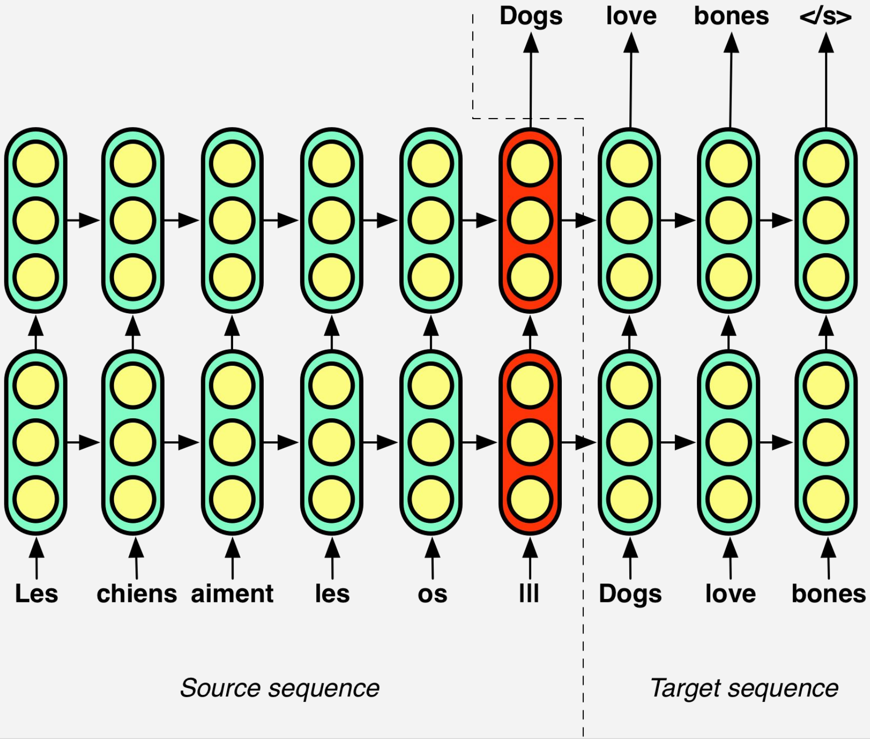
# ROM for Encoder-Decoder Models

- Encoder produces array of representations, e.g. one vector per token.
- Representations are packed into an attention matrix.
- Decoder is a controller + memory model with attention matrix as memory.
- Gradients of error w.r.t. memory provide gradients of error w.r.t. encoder.

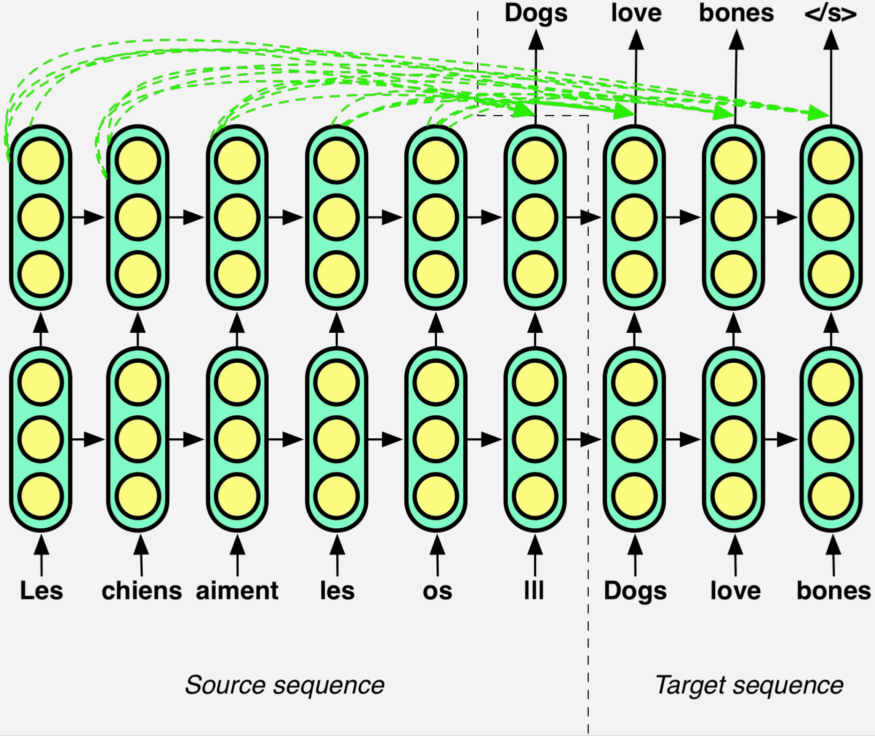
# ROM for Encoder-Decoder Models

- Encoder is gradient starved no more!
- Compute soft alignments between sequences.
- Search for information in larger sequences.
- Memory isn't touched, so operations can be done in place.

# Skipping the bottleneck



# Skipping the bottleneck



# Recognizing Textual Entailment (RTE)

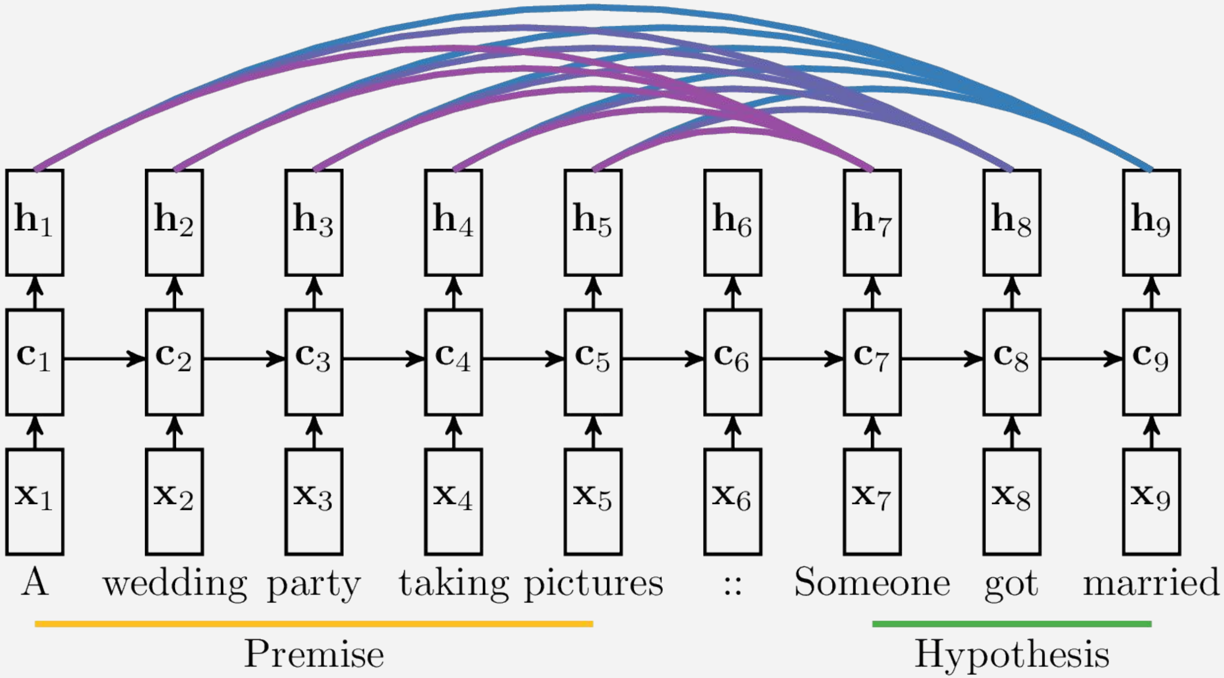
**A man is crowd surfing at a concert**

| | |
|---|---|
| The man is at a football game | **Contradiction** |
| The man is drunk | **Neutral** |
| The man is at a concert | **Entailment** |

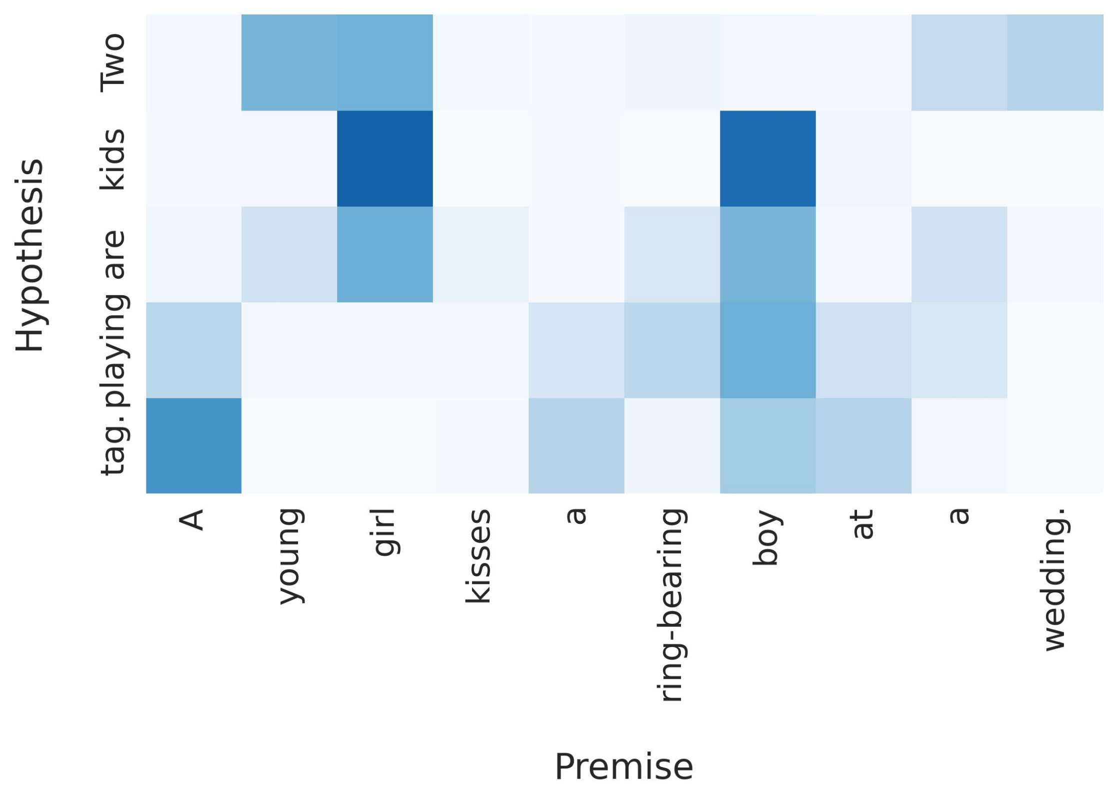
**A wedding party is taking pictures**

| | |
|---|---|
| There is a funeral | **Contradiction** |
| They are outside | **Neutral** |
| Someone got married | **Entailment** |

# Word-by-Word Attention



# Girl + Boy = Kids



# 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:**

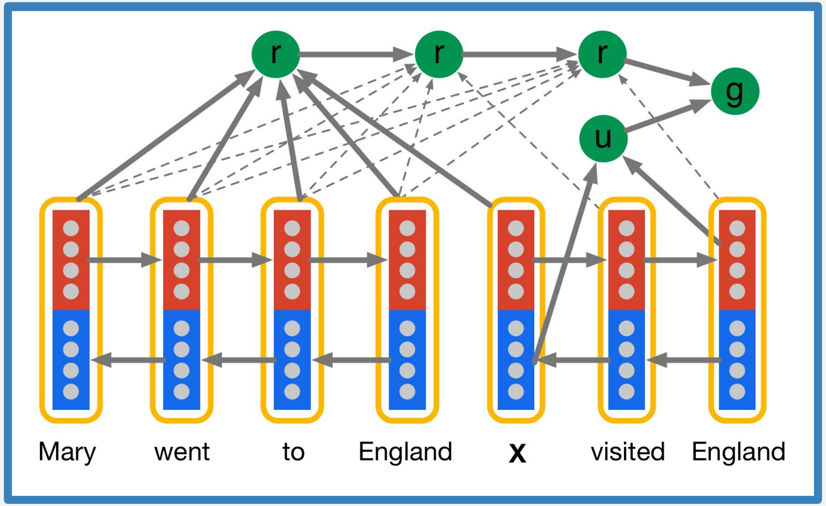
**Query:** Producer **X** will not press charges against Jeremy Clarkson, his lawyer says.

**Answer:** Oisin Tymon

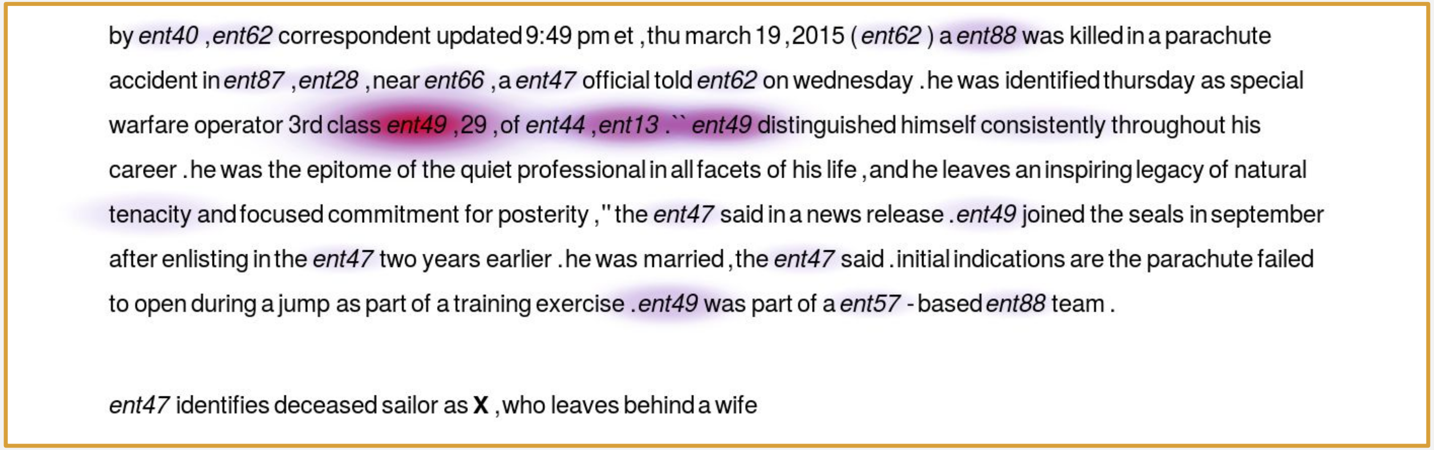
# Machine Reading with Attention

- Create embedding for each token in document
- Read query word by word
- Attend over document at each step through query
- Iteratively combine attention distribution
- Predict answer with increased accuracy

## The Impatient Reader



# Example QA Heatmap



by *ent40* ,*ent62* correspondent updated 9:49 pm et ,thu march 19 ,2015 ( *ent62* ) a *ent88* was killed in a parachute accident in *ent87* ,*ent28* ,near *ent66* ,a *ent47* official told *ent62* on wednesday .he was identified thursday as special warfare operator 3rd class *ent49* ,29 ,of *ent44* ,*ent13* .`` *ent49* distinguished himself consistently throughout his career .he was the epitome of the quiet professional in all facets of his life ,and he leaves an inspiring legacy of natural tenacity and focused commitment for posterity ,'' the *ent47* said in a news release .*ent49* joined the seals in september after enlisting in the *ent47* two years earlier .he was married ,the *ent47* said .initial indications are the parachute failed to open during a jump as part of a training exercise .*ent49* was part of a *ent57* - based *ent88* team .

*ent47* identifies deceased sailor as **X** ,who leaves behind a wife

Correct prediction (***ent49***) - Requires anaphora resolution

# Attention Summary

- Attention successfully mitigates limitations of original seq2seq
- Versatile, adaptable to many problems
- Can be tailored to specific sorts of processes, e.g. pointer networks
- Helpful for learning good source representations, although these are strongly tied to end-task
- Read-only puts strong onus on controller to track what's been read
- Read-only means encoder needs to do a lot of the legwork

# The plan

1. The Transduction Bottleneck
2. Limitations of RNNs
3. RNNs Revisited
4. Attention
5. Stacks
6. Register Machines

**Questions?**

# Stacks: Neural PDAs

# Computational Hierarchy

Turing Machines (computable functions)

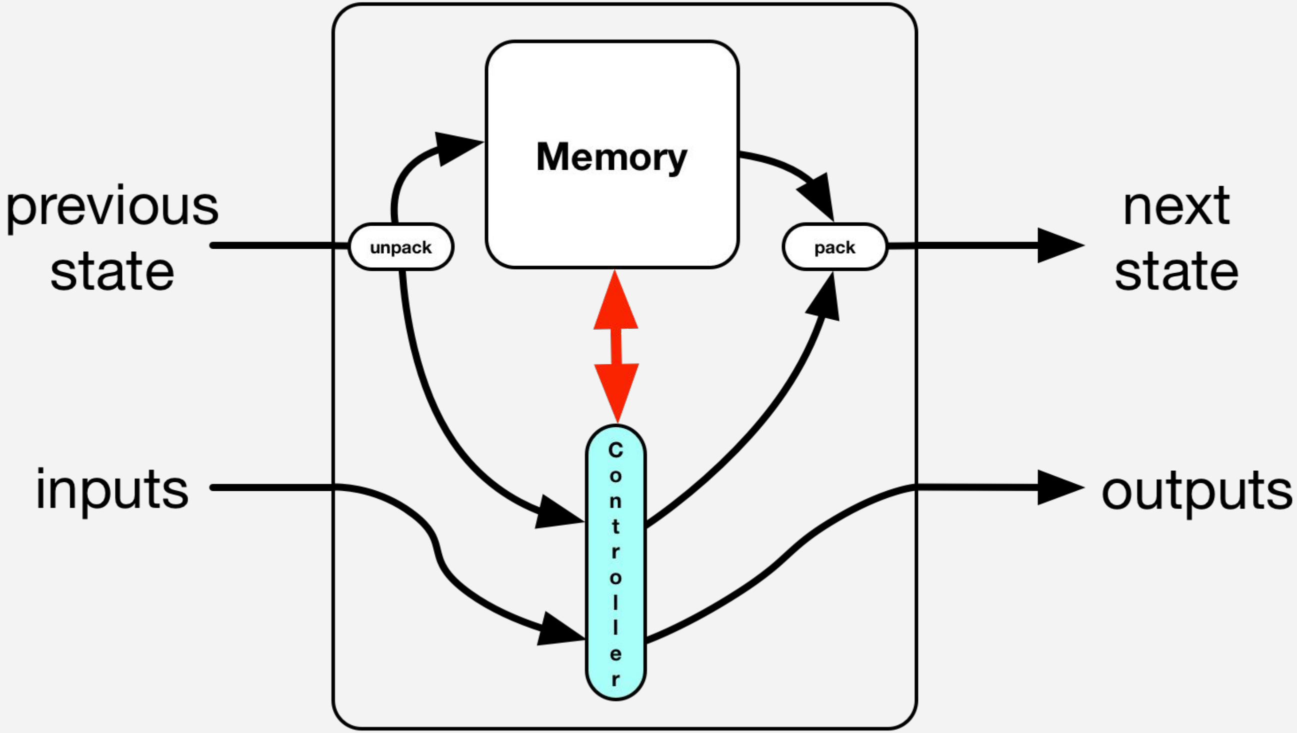
⬆ ⬆ ⬆

**Pushdown Automata (context free languages)**

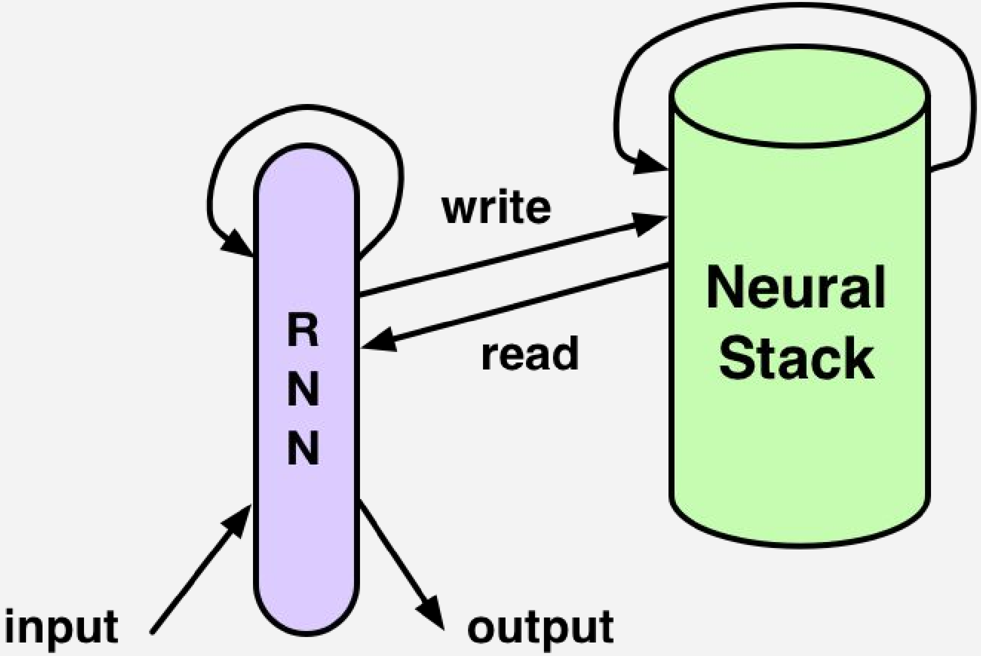
⬆ ⬆ ⬆

Finite State Machines (regular languages)

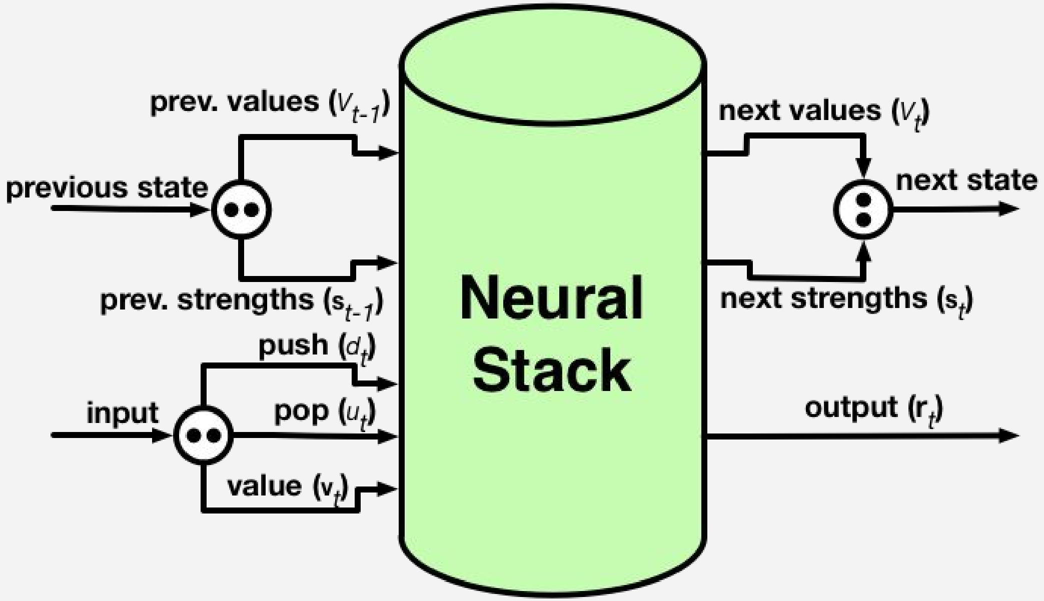
# The Controller-Memory Split



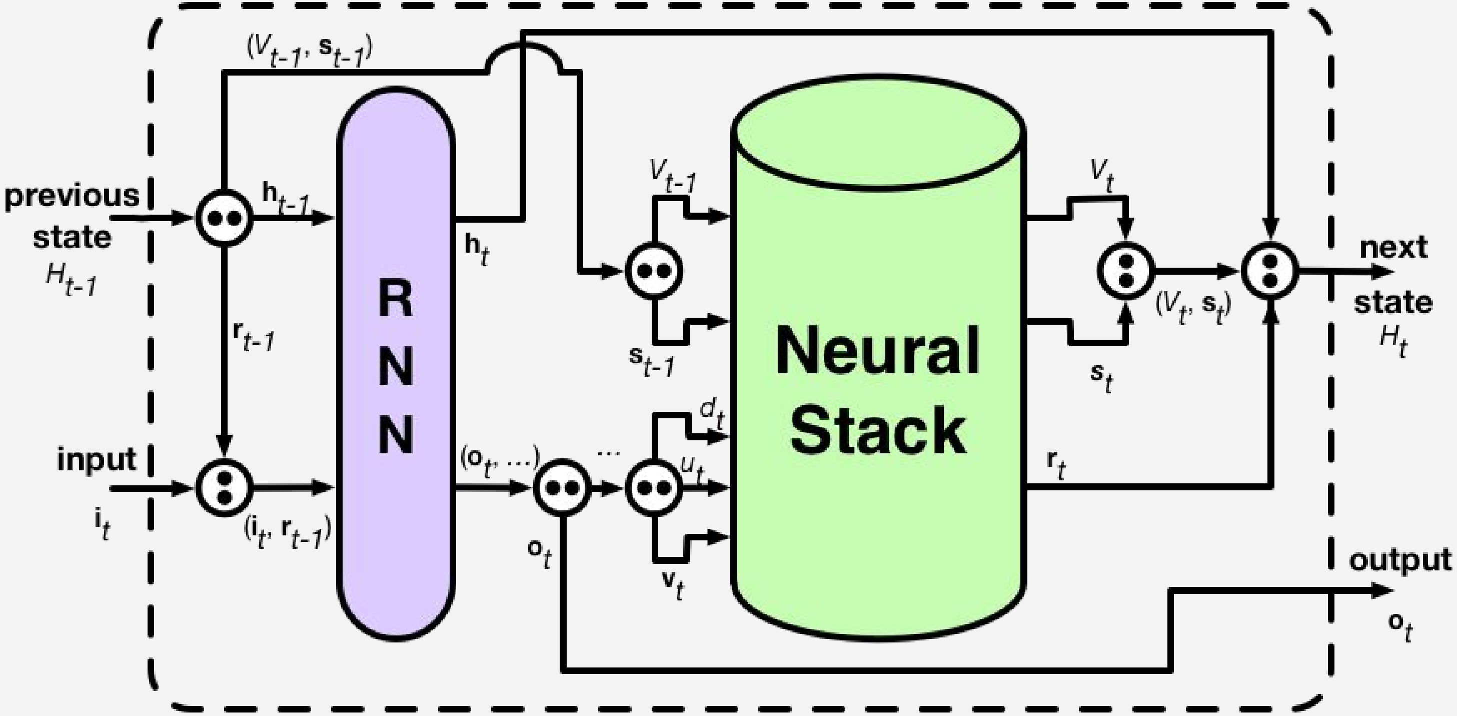
# Controlling a Neural Stack



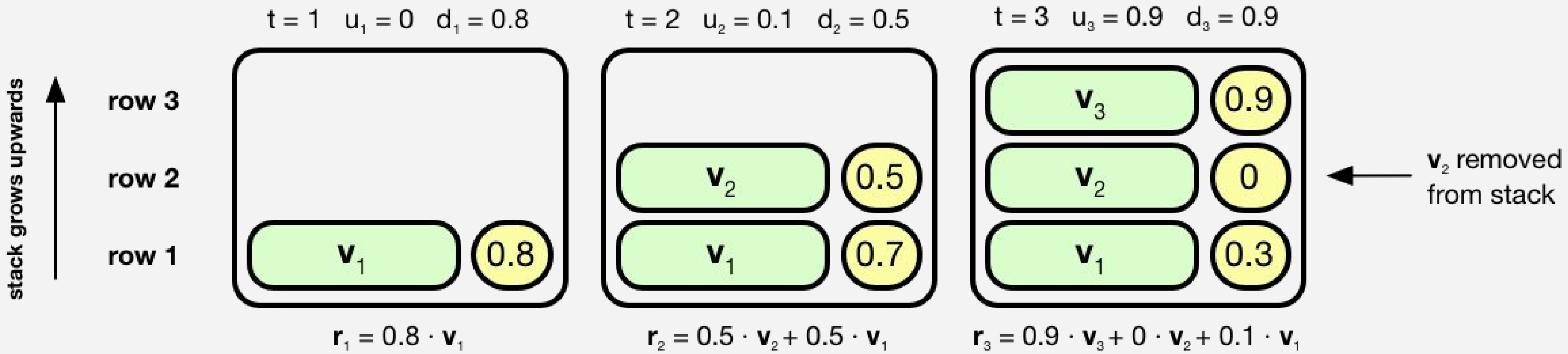
# Controller API



# Controller + Stack Interaction



# Example: A Continuous Stack



stack grows upwards
row 3
row 2
row 1
$t = 1 \quad u_1 = 0 \quad d_1 = 0.8$
$\mathbf{v}_1$
0.8
$\mathbf{r}_1 = 0.8 \cdot \mathbf{v}_1$
$t = 2 \quad u_2 = 0.1 \quad d_2 = 0.5$
$\mathbf{v}_2$
0.5
$\mathbf{v}_1$
0.7
$\mathbf{r}_2 = 0.5 \cdot \mathbf{v}_2 + 0.5 \cdot \mathbf{v}_1$
$t = 3 \quad u_3 = 0.9 \quad d_3 = 0.9$
$\mathbf{v}_3$
0.9
$\mathbf{v}_2$
0
$\mathbf{v}_1$
0.3
$\mathbf{r}_3 = 0.9 \cdot \mathbf{v}_3 + 0 \cdot \mathbf{v}_2 + 0.1 \cdot \mathbf{v}_1$
$\mathbf{v}_2$ removed from stack

# Synthetic Transduction Tasks

## Copy

$$a_1 a_2 a_3 \ldots a_n \rightarrow a_1 a_2 a_3 \ldots a_n$$

## Reversal

$$a_1 a_2 a_3 \ldots a_n \rightarrow a_n \ldots a_3 a_2 a_1$$

# Synthetic ITG Transduction Tasks

## Subject-Verb-Object to Subject-Object-Verb Reordering

si1 vi28 oi5 oi7 si15 rpi si19 vi16 oi10 oi24 → so1 oo5 oo7 so15 rpo so19 vo16 oo10 oo24 vo28

## Genderless to Gendered Grammar

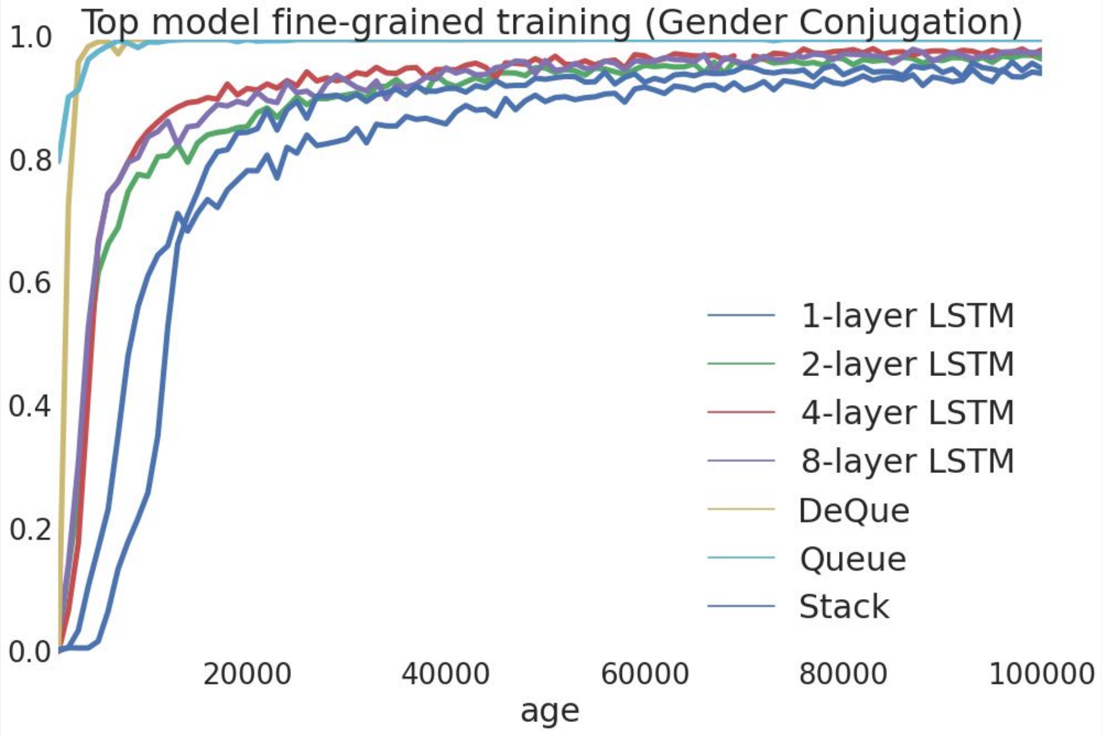
we11 the en19 and the em17 → wg11 das gn19 und der gm17

# Results

| Experiment | Stack | Queue | DeQue | Deep LSTM |
|---|---|---|---|---|
| **Copy** | Poor | **Solved** | **Solved** | Poor |
| **Reversal** | **Solved** | Poor | **Solved** | Poor |
| **SVO-SOV** | **Solved** | **Solved** | **Solved** | Converges |
| **Conjugation** | Converges | **Solved** | **Solved** | Converges |

**Every Neural Stack/Queue/DeQue that solves a problem preserves the solution for longer sequences (tested up to 2x length of training sequences).**

# Rapid Convergence



**Regular language (N+1 rules)**

ε|(ab)|(aabb)|(aaabbb)|...

**CFG (2 rules)**

S → a S b

S → ε

# Differentiable Stacks / Queues / Etc

- Grefenstette *et al.* 2015: Stacks, Queues, Double Ended Queues
  - Presented here
- Joulin and Mikolov 2015: Stacks, Lists
  - More lightweight stack semantics
  - Check out the experiments!
- Dyer *et al.* 2015: Stack LSTMs
  - Supervision needed for push/pop
  - Doesn't fit the controller/memory framework, but worth reading!

# 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
- Does going up in the computational hierarchy help? We need to find out.

# Register Machines: RAM

# Computational Hierarchy

**Turing Machines (computable functions)**

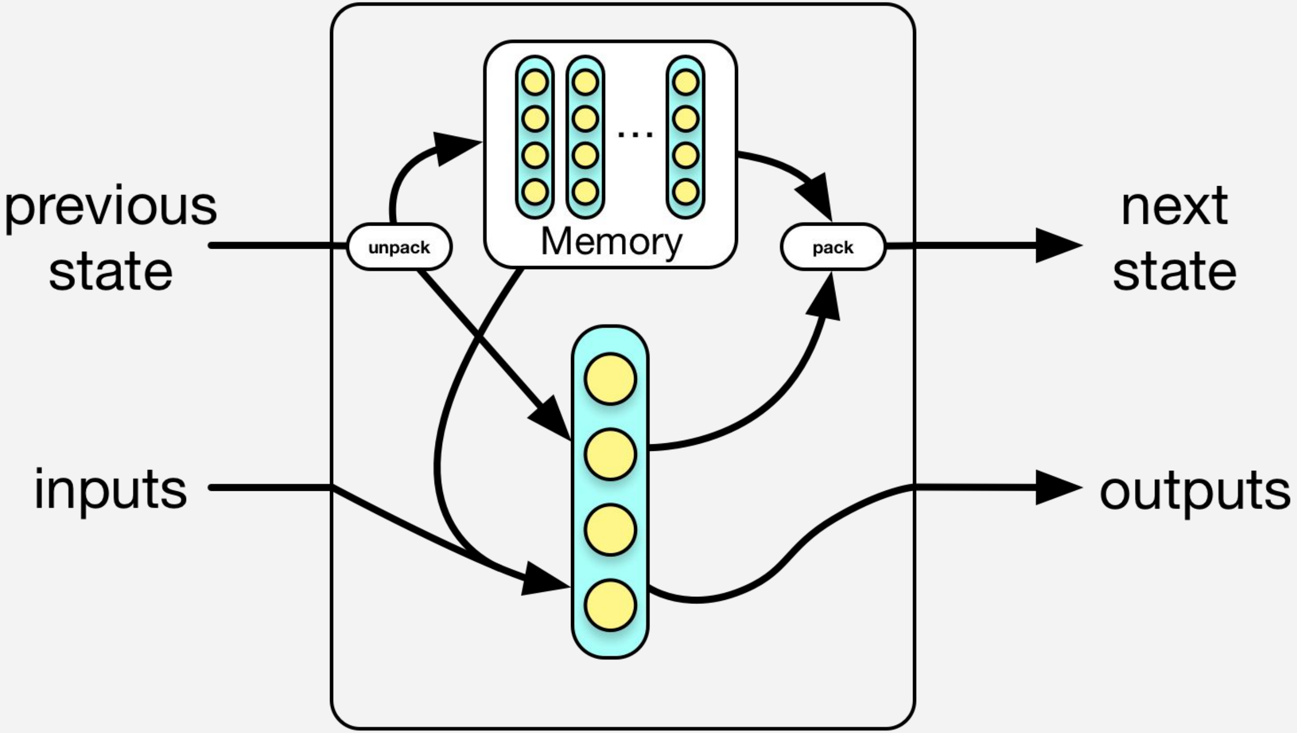
⬆ ⬆ ⬆

Pushdown Automata (context free languages)

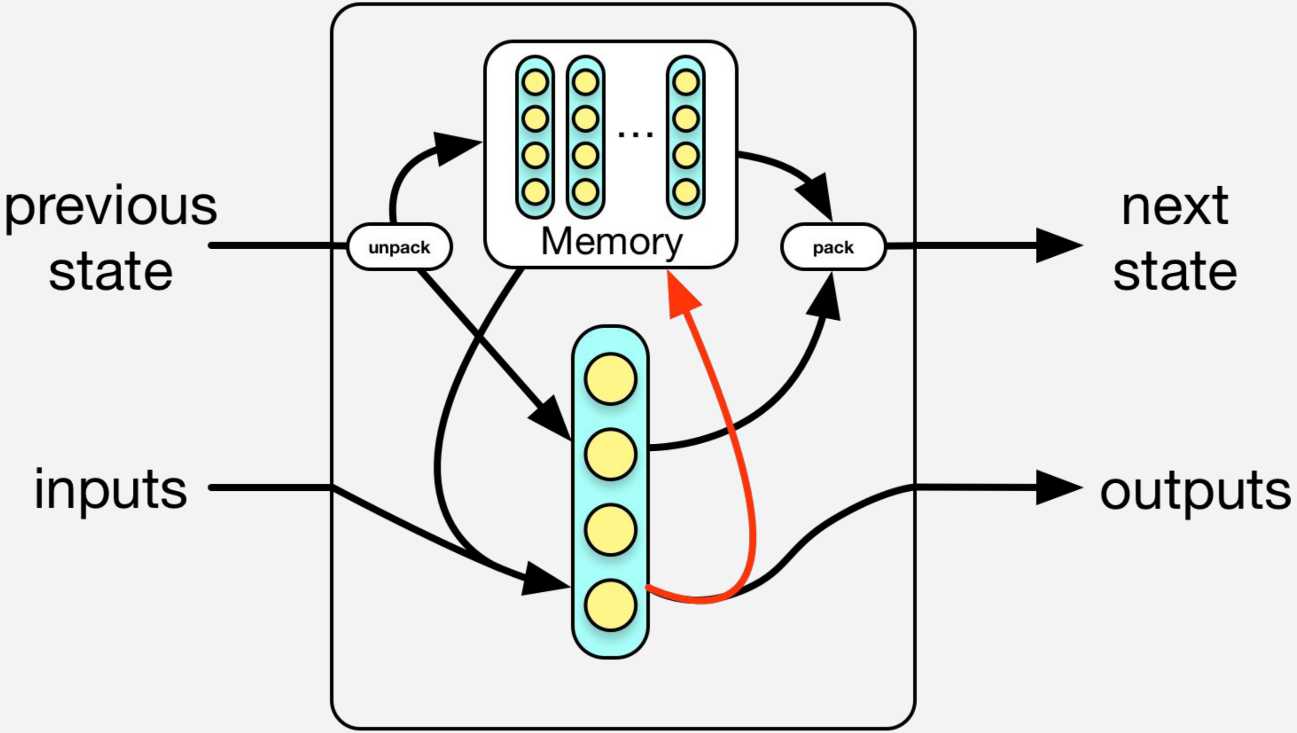
⬆ ⬆ ⬆

Finite State Machines (regular languages)

# Attention as ROM



# Register Memory as RAM



# Neural RAM: General Idea

- Controller deals with I/O
- Controller also produces distributions over memory registers for reading/writing
- Controller decides what to write, how much to erase, etc.
- Controller and memory state are updated, and recurrent cell produces output, next state

# RNN: $X \times P \rightarrow Y \times N$ (an example of)

$p_t = (h_{t-1}, M_{t-1}, r_{t-1})$

$xs = x_t \oplus r_{t-1}$

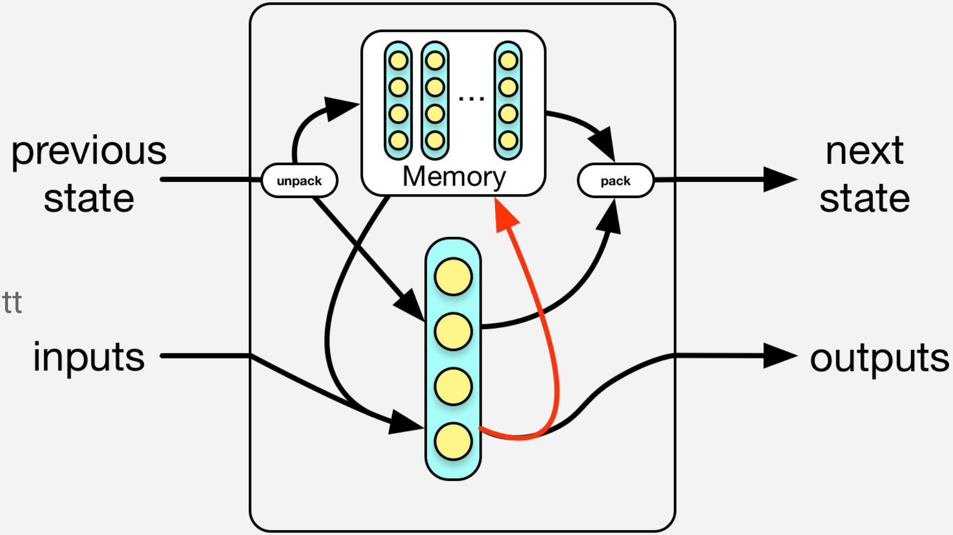
$y_t, k^{read}, k^{write}, v, h_t = \text{controller}(xs, h_{t-1})$

$r_t = f_{read}(k^{read}, M_{t-1})$ where e.g. $f_{read} = f_{att}$

$M_t = f_{write}(v, k^{write}, M_{t-1})$

e.g. $M_t[i] = a[i] \cdot v + (1-a[i]) \cdot M_{t-1}[i]$

where $a = \text{softmax}(k_{write} \cdot W_{write} \cdot M_{t-1})$



# Extensions

- Location based addressing: e.g. state tracks an initially one-hot key which is shifted by the controller.
- Mix of location and content based.
- Hard addressing with REINFORCE
- More esoteric/heuristic addressing mechanisms (better to be kept diff.)
- Memory key/content factorization (e.g. NPI)
- Memory Prefixes

# Relation to actual Turing Machines

- Part of the "tape" is internalised
- Controller can control tape motion via various mechanisms
- RNN could model state transitions
- Weird split between external and internal tape in seq2seq paradigm
- 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.

# Register Machines and NLU

- Complex reasoning probably requires something both more expressive and more structured than RNNs + Attention
- These architectures are complex, hard to train, many design choices
- Not always an immediate mapping of problems to purported capabilities of these architectures.
- Fascinating research to be done here, but don't forget about simpler models!

# Conclusions

- Easy to design an overly complex model. Not always worth it.
- Better to understand the limits of current architectures within the context of a problem.
- By understanding the limitations and their nature, often better solutions pop out of 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.
- Nonetheless, be creative. Plenty of problems to solve in NLU and elsewhere.

# Thanks for listening!

**joinus@deepmind.com**