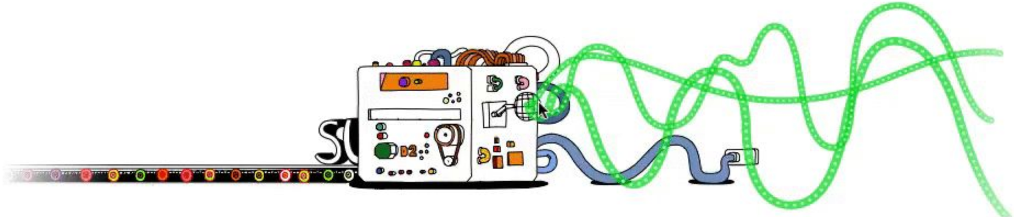
New Directions For Recurrent Neural Networks

Alex Graves

DeepMind

RNNs Work!

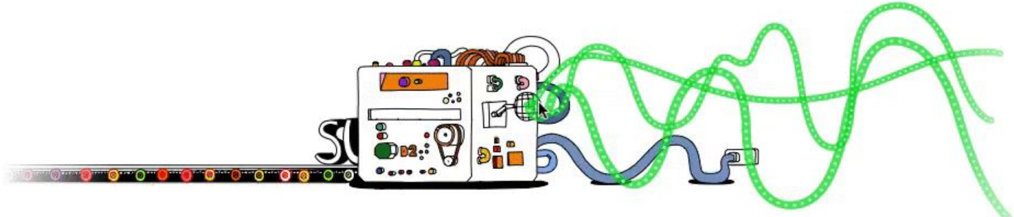
RNNs — especially LSTM / GRU variants — are now ubiquitous in ML research and routinely used for large-scale commercial tasks, including speech and handwriting recognition, machine translation, text-to-speech and many others.



Increasingly trained **end-to-end**: feed the input sequence in, get the desired output sequence out

RNNs Work!

RNNs — especially LSTM / GRU variants — are now ubiquitous in ML research and routinely used for large-scale commercial tasks, including speech and handwriting recognition, machine translation, text-to-speech and many others.



Increasingly trained **end-to-end**: feed the input sequence in, get the desired output sequence out

So what can't they do, and what can we do about it?

Extension 1: External Memory

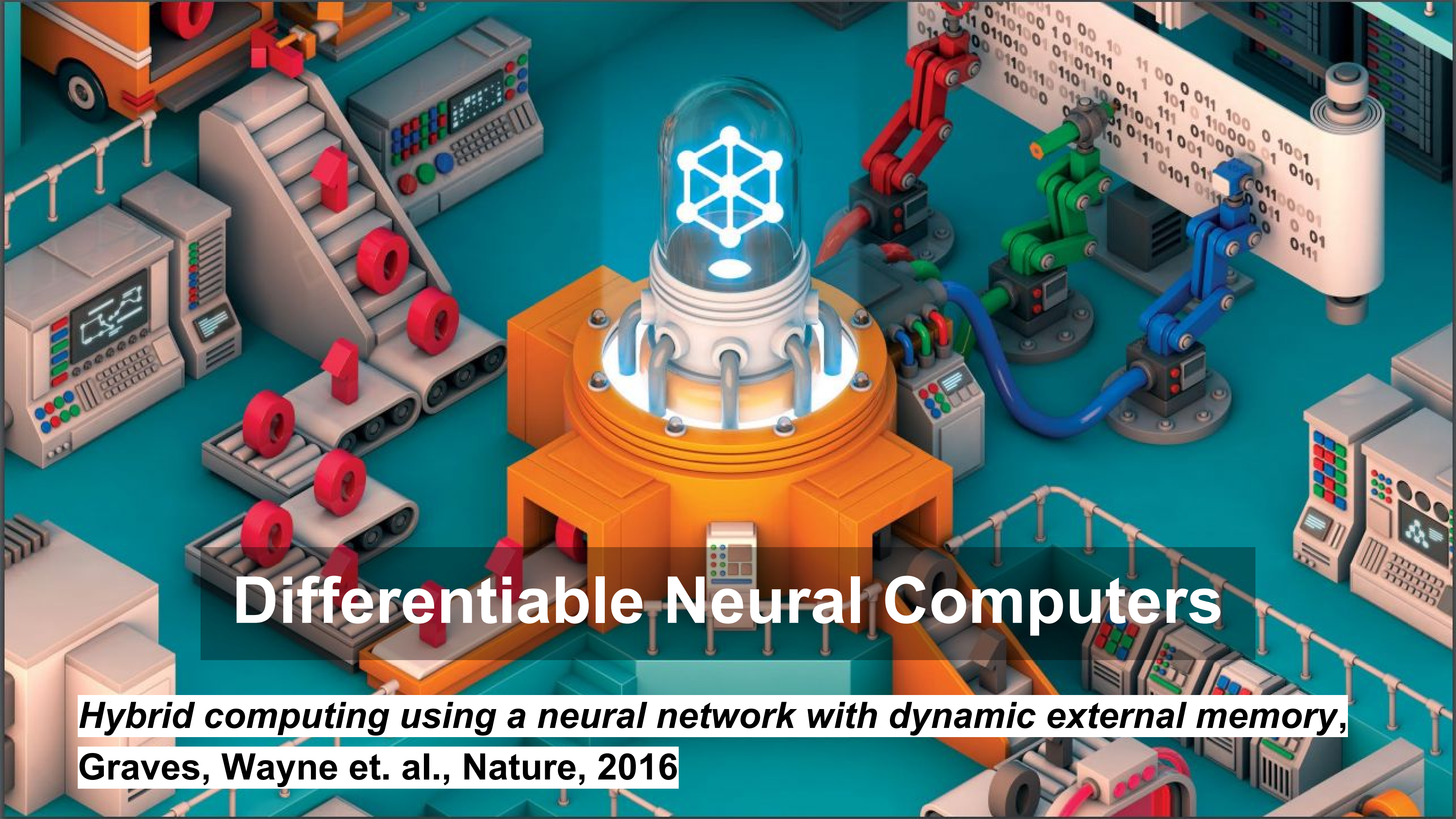
Problem: **RNN memory is stored in the vector of hidden activations**

- Activation memory is ‘fragile’: tends to be overwritten by new information
- No. of weights and hence computational cost grows with memory size (can’t put a whole book in memory)
- ‘Hard-coded’ memory locations make indirection (and hence variables) hard

Solution: **Give the net access to external memory**

- Less fragile: only some memory is ‘touched’ at each step
- Indirection is possible because memory *content* is independent of *location*
- Separates ***computation*** from ***memory***

Neural Machine Translation by Jointly Learning to Align and Translate, Bahdanau et. al. (2014)
Memory Networks, Weston et. al. (2014)
Neural Turing Machines, Graves, Wayne, Danihelka (2014)



Differentiable Neural Computers

***Hybrid computing using a neural network with dynamic external memory*,**

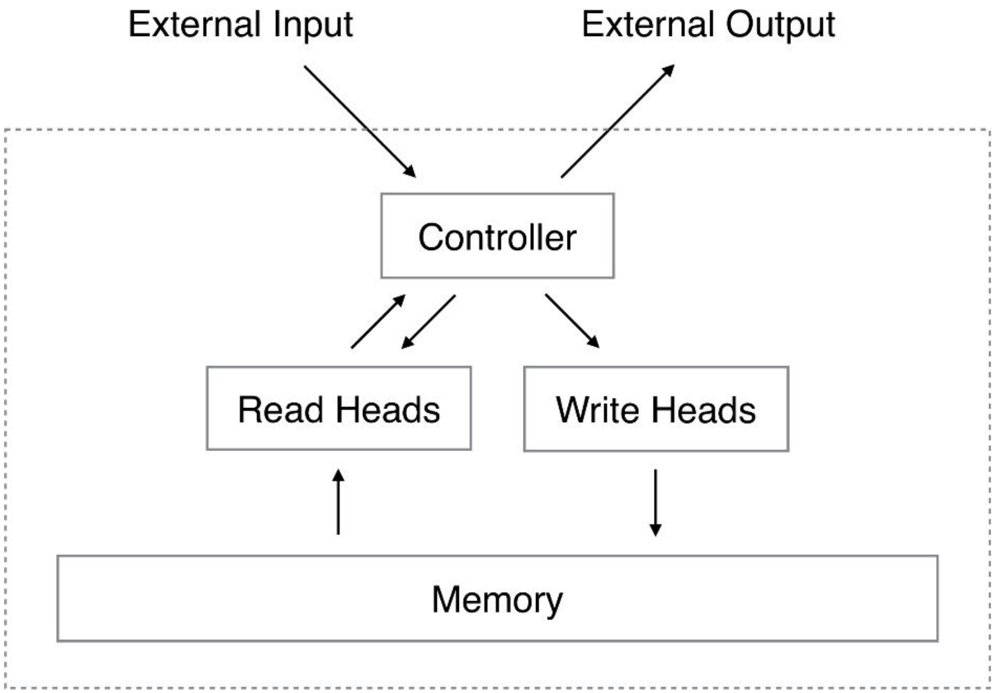
Graves, Wayne et. al., Nature, 2016

Basic Read/Write Architecture

The **Controller** is a **neural network** (recurrent or feedforward)

The **Heads** **select** portions of the memory and **read** or **write** to them

The **Memory** is a real-valued **matrix**



Memory Access

Most networks with external memory (RNNs with attention, Memory Nets, NTM, DNC…) use some form of content-based memory access: find the memory *closest* (e.g. cosine similarity) to some key vector emitted by the network, return either the memory contents or an associated value vector

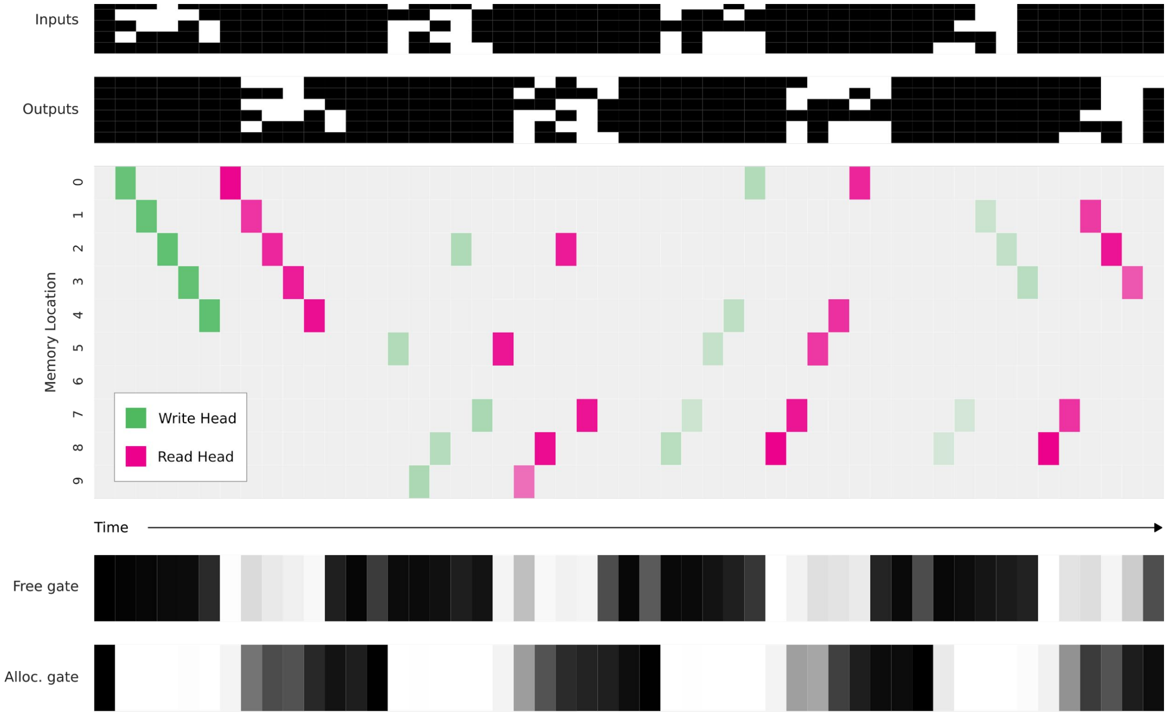
A universal access mechanism (*c.f.* associative computers)

But maybe not the most convenient for all tasks: e.g. we search real computers using **text strings, directory trees, read/write time, user-defined titles or tags**… many more mechanisms to be tried

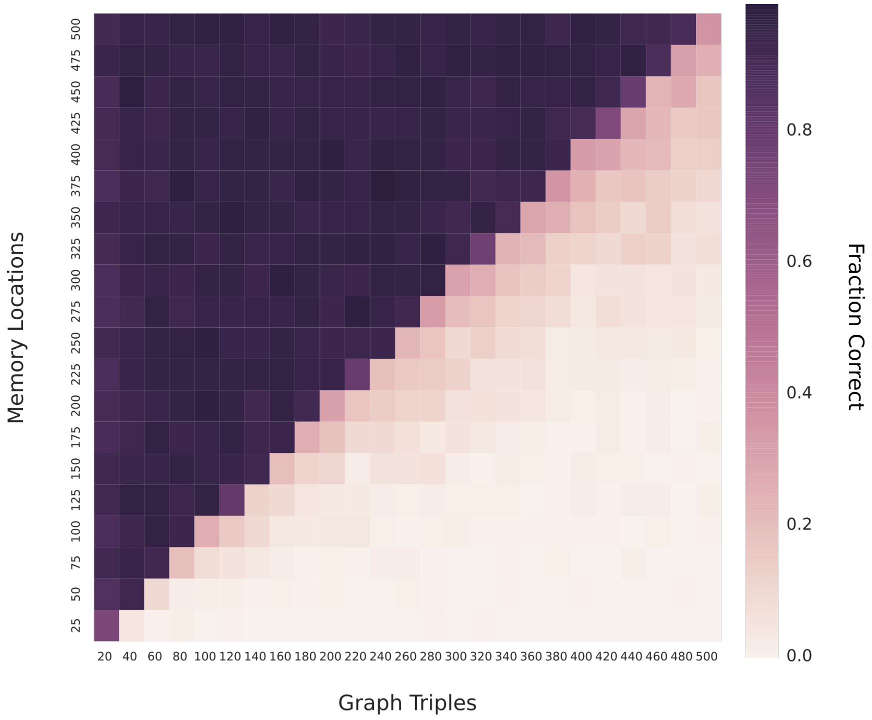
Dynamic Memory Allocation

- NTM could only ‘allocate’ memory in contiguous blocks, leading to memory management problems
- DNC defines a differentiable **free list** tracking the **usage** of each memory location
- Usage is automatically *increased* after each write and optionally *decreased* after each read
- The network can then choose to write to the **most free** location in memory, rather than searching by content

Memory Allocation Test



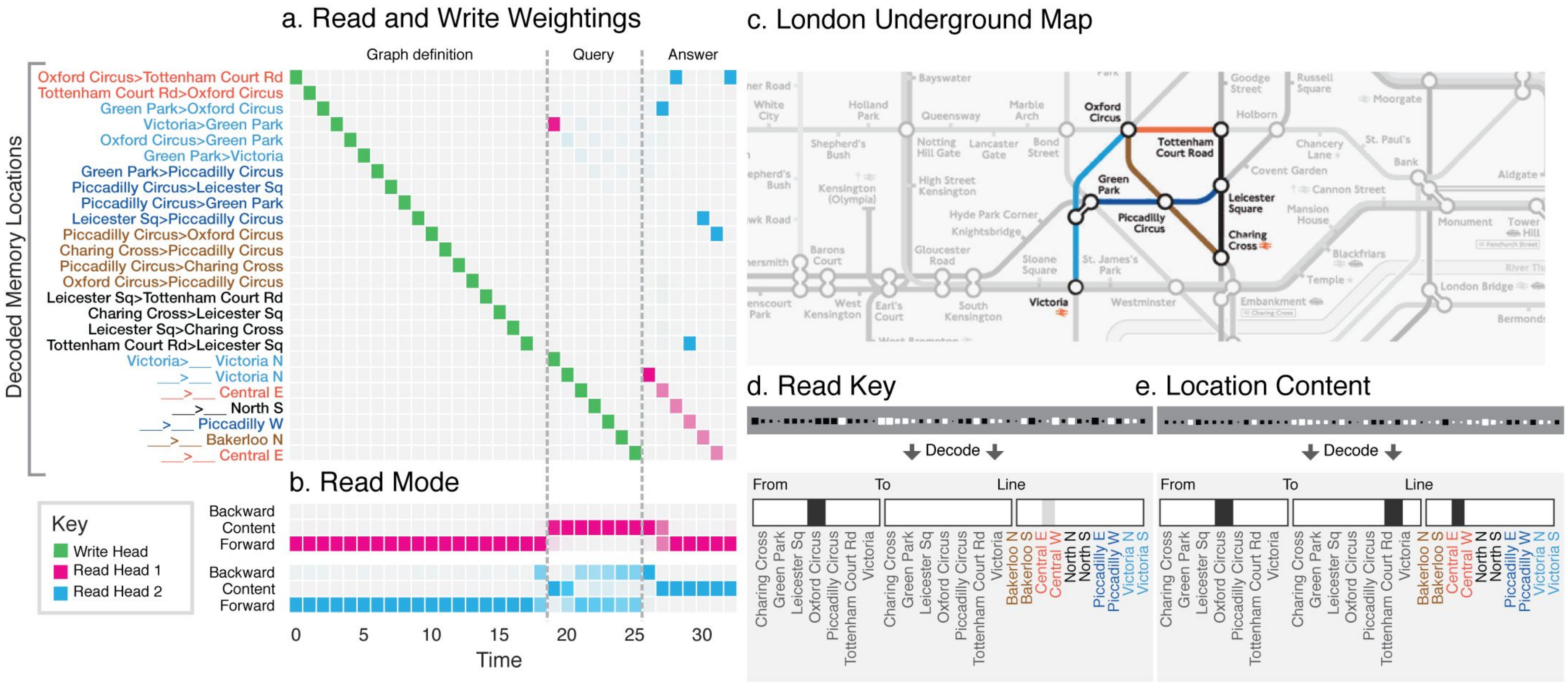
Memory Resizing Test



Searching By Time

- We wanted DNC to be able to iterate through memories in chronological order
- To do this it maintains a **temporal link matrix** L_t whose i,j^{th} element is interpreted as the probability that memory location *i* was **written to** immediately before location *j*
- When reading from memory, DNC can choose to follow these links instead of searching by content.
- Unlike *location-based* access this facilitates two cognitively important functions:
  - **Sequence chunking** (don't write at every step)
  - **Recoding** (iteratively reprocess a sequence, chunking each time)

London Underground with DNC



bAbI Results

Task	bAbI Best Results						
	LSTM (Joint)	NTM (Joint)	DNC1 (Joint)	DNC2 (Joint)	MemN2N (Joint) [21]	MemN2N (Single) [21]	DMN (Single) [20]
1: 1 supporting fact	24.5	31.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
2: 2 supporting facts	53.2	54.5	1.3	0.4	1.0	**0.3**	1.8
3: 3 supporting facts	48.3	43.9	2.4	**1.8**	6.8	2.1	4.8
4: 2 argument rels.	0.4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
5: 3 argument rels.	3.5	0.8	**0.5**	0.8	6.1	0.8	0.7
6: yes/no questions	11.5	17.1	**0.0**	**0.0**	0.1	0.1	**0.0**
7: counting	15.0	17.8	**0.2**	0.6	6.6	2.0	3.1
8: lists/sets	16.5	13.8	**0.1**	0.3	2.7	0.9	3.5
9: simple negation	10.5	16.4	**0.0**	0.2	**0.0**	0.3	**0.0**
10: indefinite knowl.	22.9	16.6	0.2	0.2	0.5	**0.0**	**0.0**
11: basic coreference	6.1	15.2	**0.0**	**0.0**	**0.0**	0.1	0.1
12: conjunction	3.8	8.9	0.1	**0.0**	0.1	**0.0**	**0.0**
13: compound coref.	0.5	7.4	**0.0**	0.1	**0.0**	**0.0**	0.2
14: time reasoning	55.3	24.2	0.3	0.4	**0.0**	0.1	**0.0**
15: basic deduction	44.7	47.0	**0.0**	**0.0**	0.2	**0.0**	**0.0**
16: basic induction	52.6	53.6	52.4	55.1	**0.2**	51.8	0.6
17: positional reas.	39.2	25.5	24.1	**12.0**	41.8	18.6	40.4
18: size reasoning	4.8	2.2	4.0	**0.8**	8.0	5.3	4.7
19: path finding	89.5	4.3	**0.1**	3.9	75.7	2.3	65.5
20: agent motiv.	1.3	1.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
Mean Err. (%)	25.2	20.1	4.3	**3.8**	7.5	4.2	6.4
Failed (err. > 5%)	15	16	**2**	**2**	6	3	**2**

Ask me anything: dynamic memory networks for natural language processing, Kumar et. al. (2015)

End-to-end memory networks, Sukhbaatar et. al. (2015)

Sparse Memory Access

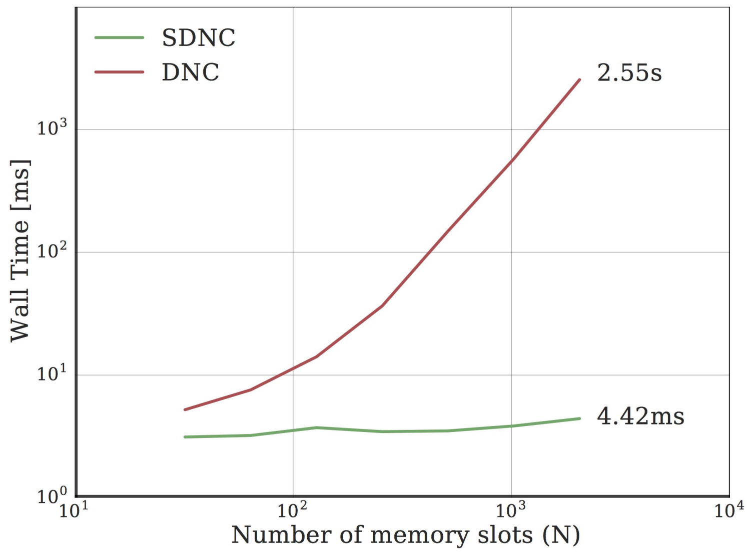
	Dense	Sparse
Content-based addressing	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
Temporal addressing	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
Read	$\mathcal{O}(n)$	$\mathcal{O}(1)$
Erase	$\mathcal{O}(n)$	$\mathcal{O}(1)$
Add	$\mathcal{O}(n)$	$\mathcal{O}(1)$

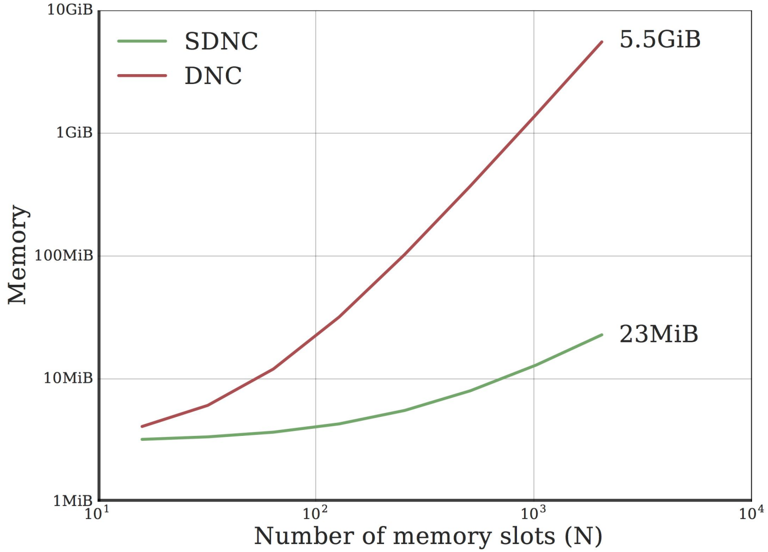
Using a KNN

By restricting reads and writes to 8 (say) locations per step.

Scaling Memory-Augmented Neural Networks with Sparse Reads and Writes, Rae, Hunt et. al. (2016)

Sparse DNC Efficiency





Extension 2: Learning When to Halt

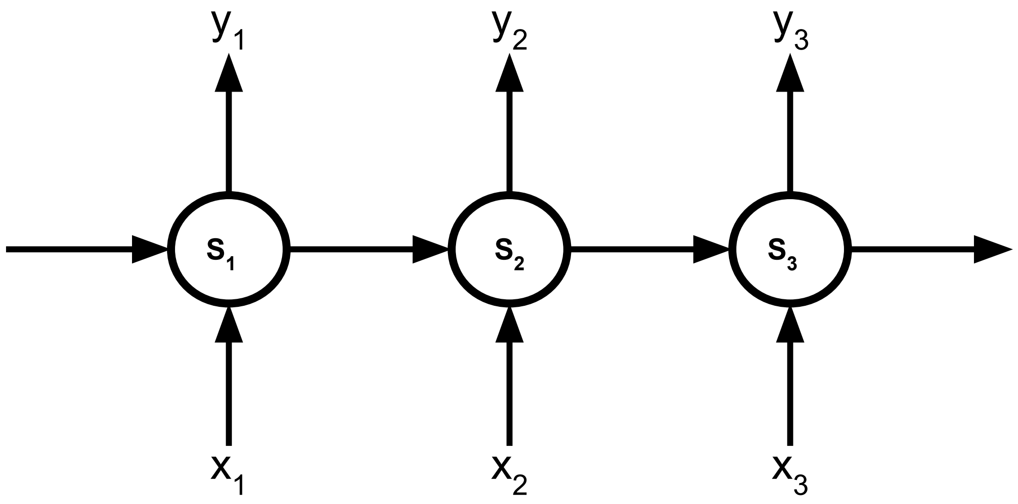
Problem: The number of steps of computation an RNN gets before emitting an output is determined by the length of the input sequence, not the difficulty of the task.

- *Do*$_1$ *any*$_2$ *three*$_3$ *positive*$_4$ *integers*$_5$ *a,b,c*$_6$ *satisfy*$_7$ $a^n+b^n=c^n{}_8$ *for*$_9$ *any*$_{10}$ *integer*$_{11}$ *n*$_{12}$ *greater*$_{13}$ *than*$_{14}$ *two?*$_{15}$

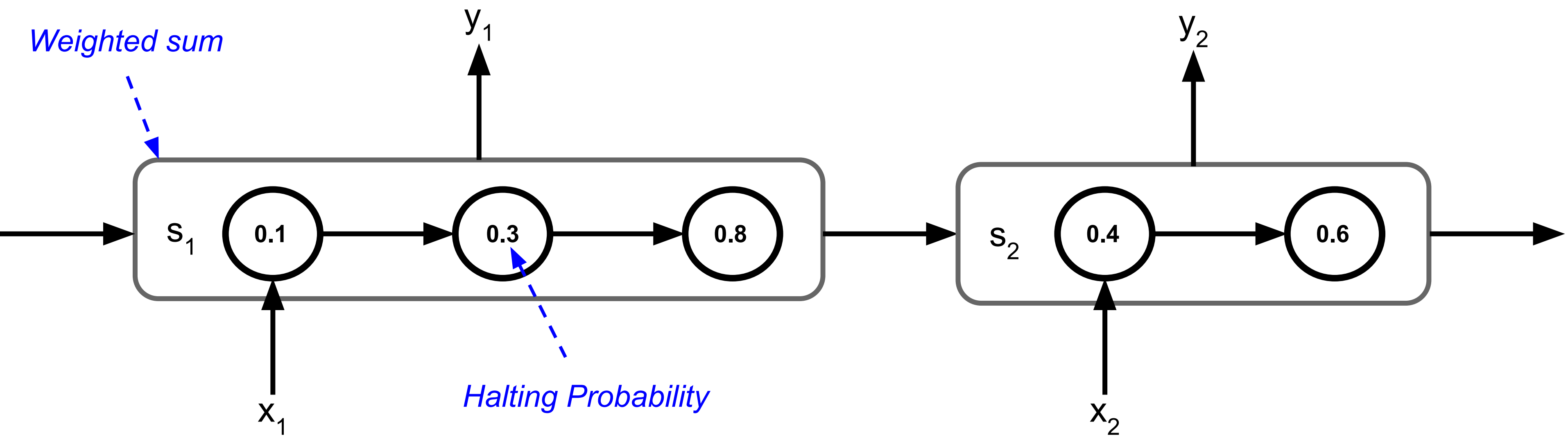
Solution: Train the network to learn how long to ‘think’ before it ‘acts’

- separate ***computation time*** from ***data time***

RNN Computation Graph



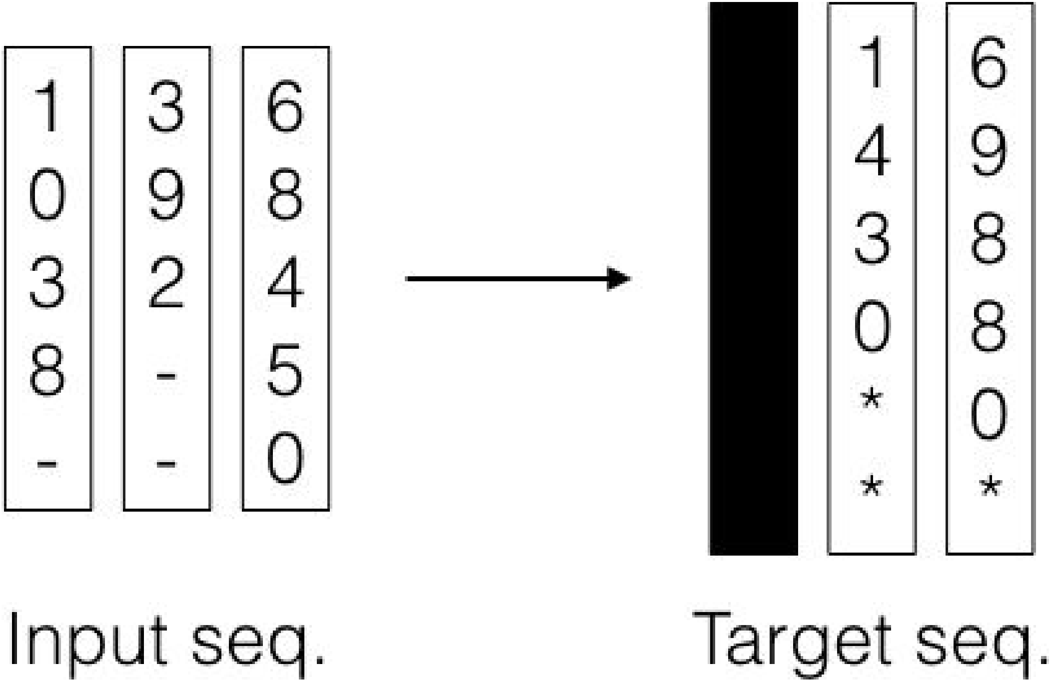
Adaptive computation Time (ACT)



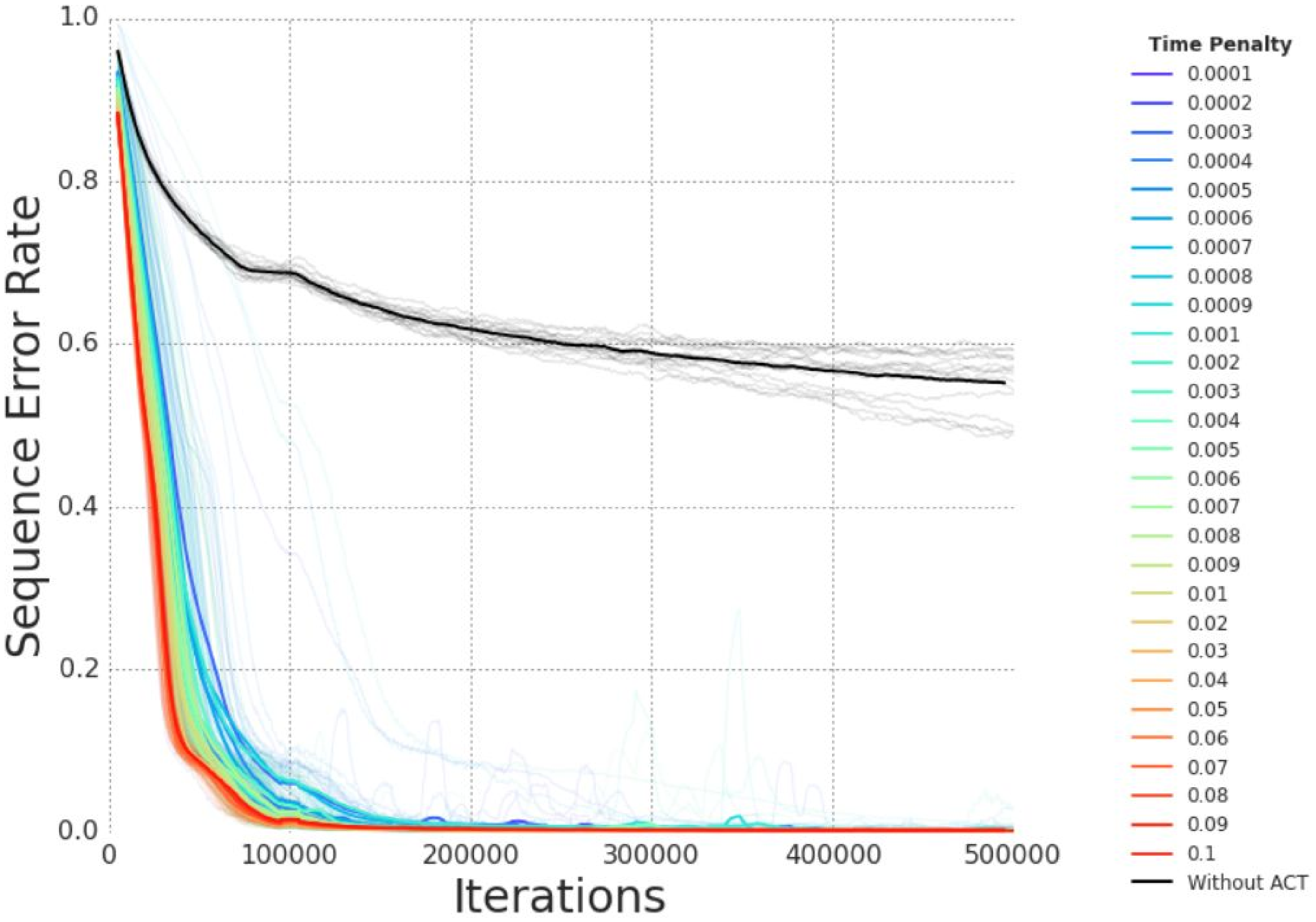
A **time penalty** acts to reduce the total number of **'ponder'** steps

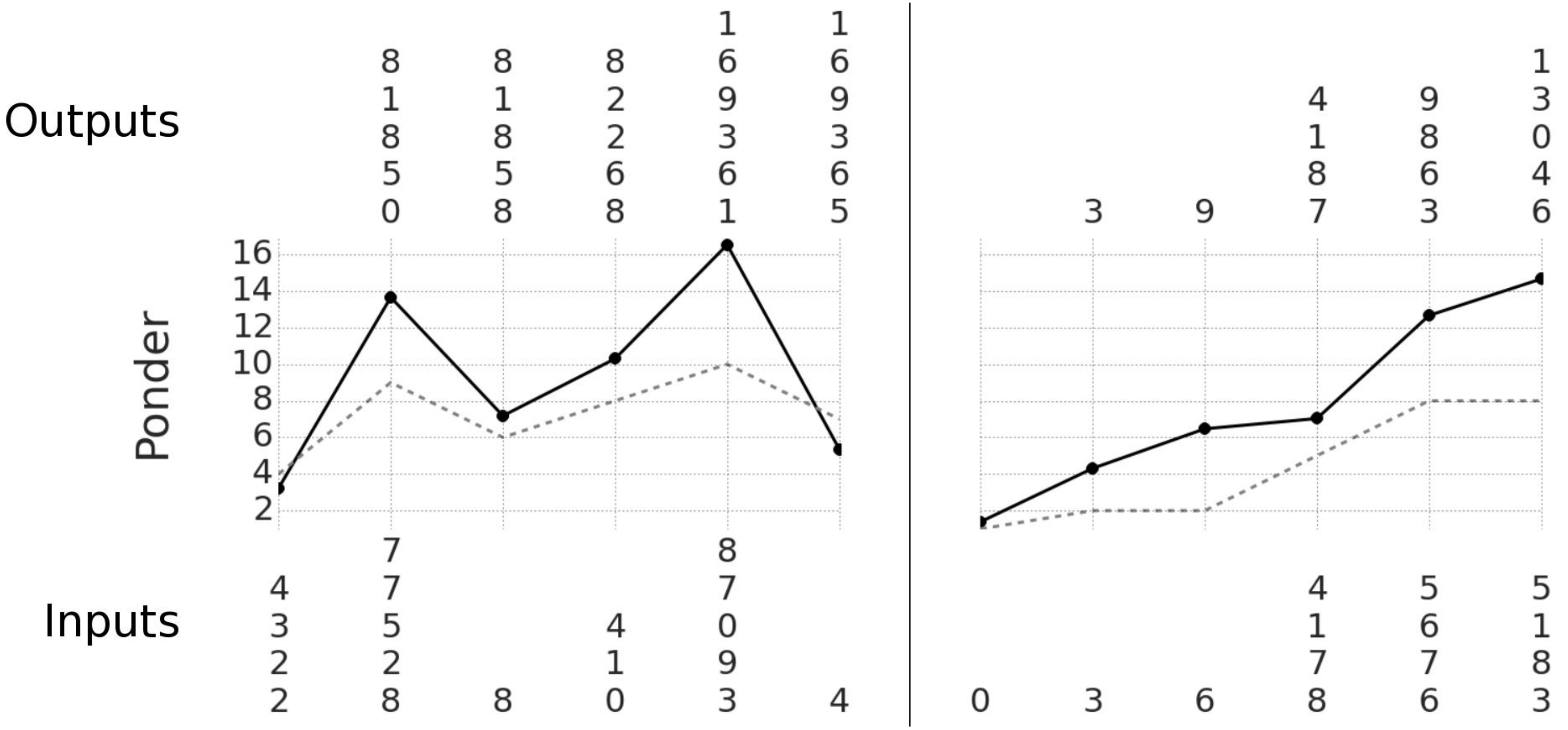
Adaptive Computation Time With Recurrent Neural Networks, Graves (2016)

Addition with ACT



Addition Results




Outputs
81850
81858
82268
169361
169365
3
9
4187
9863
13046
Ponder
16
14
12
10
8
6
4
2
Inputs
43222
77528
8
410
87093
4
0
3
6
4178
5673
5183

Machine Translation

Dataset: WMT14 test set, English to French

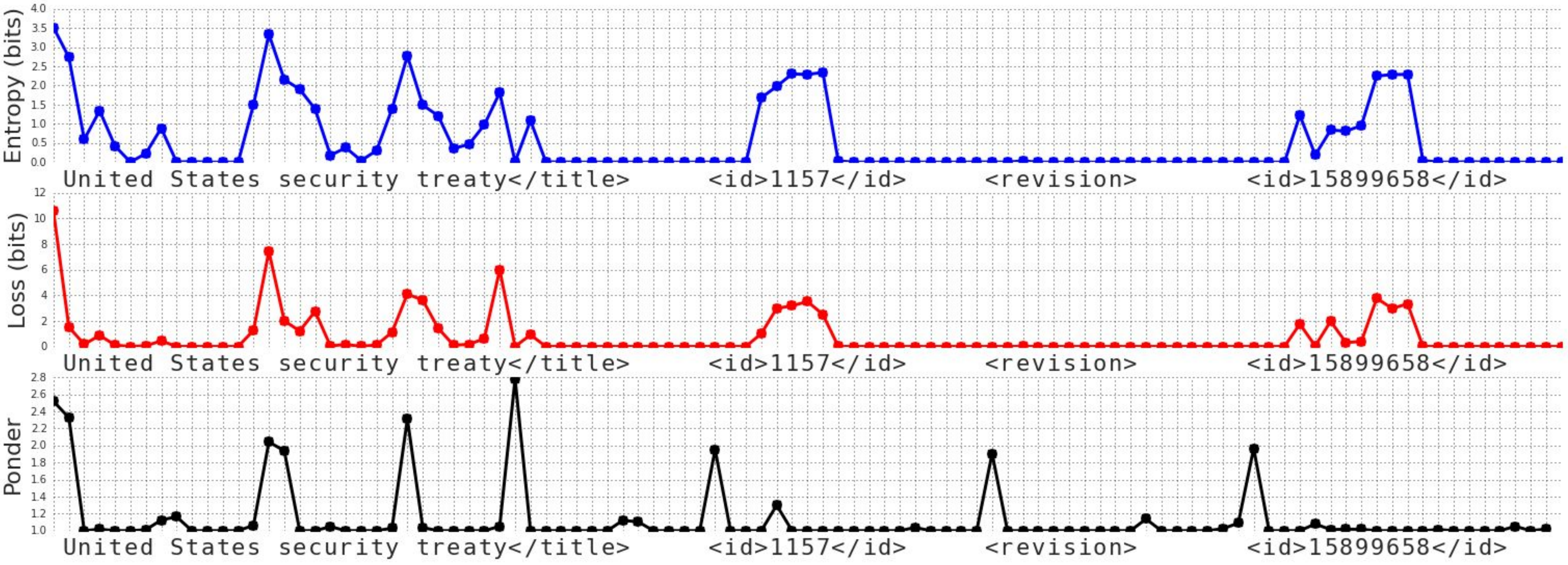
(SMT): 37.0 BLEU

Baseline AttLSTM: 3.4 PPL, 37.5 BLEU

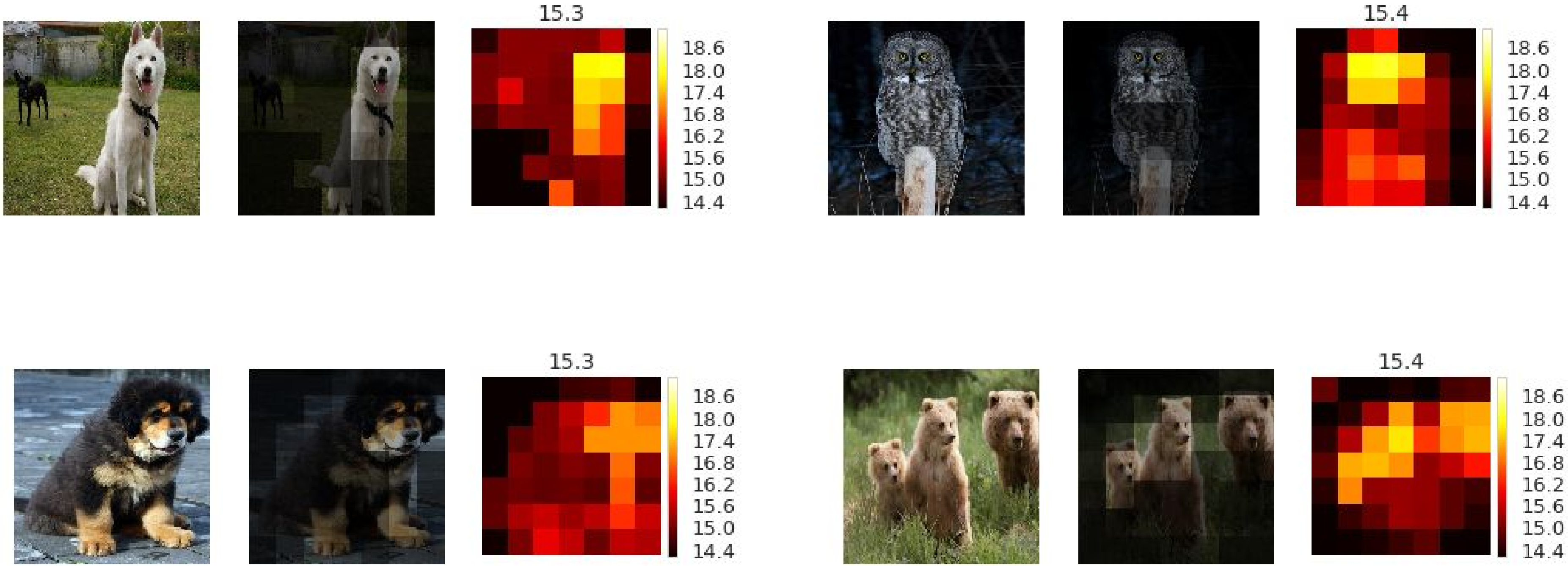
AttLSTM + ACT: **3.1** PPL, **38.3** BLEU

Vinyals, Jozefowicz - unpublished (yet)

Pondering Wikipedia (character level)

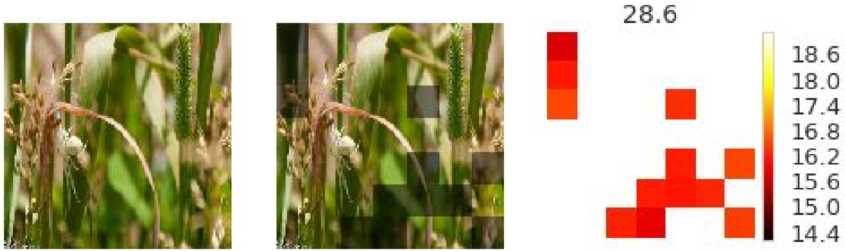


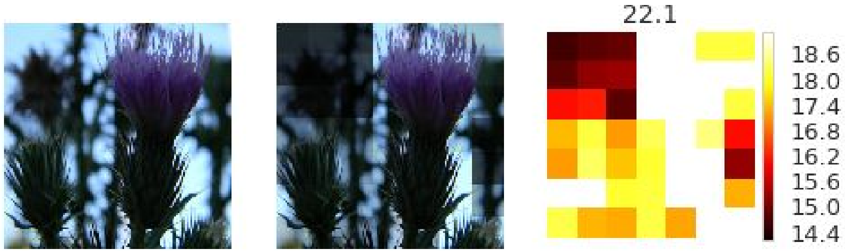
ACT for Feedforward Nets

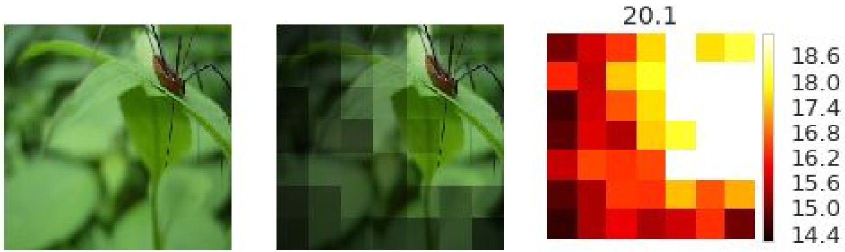


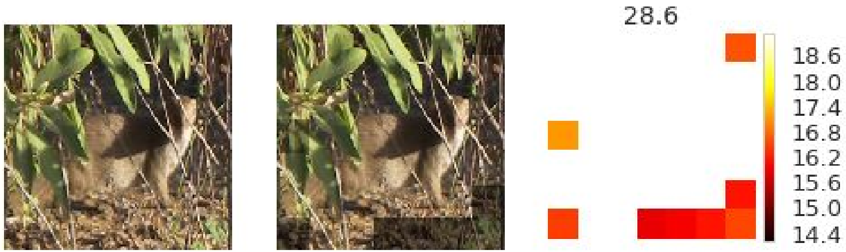
Spatially Adaptive Computation Time for Residual Networks, Figurnov et. al, 2016

ImageNet high ponder cost examples









Extension 3: Beyond BPTT

Problem: Most RNNs are trained with Backpropagation Through Time (BPTT)

- Memory cost increases with sequence length
- Weight update frequency decreases
- The better RNNs get, the longer the sequences we train them on

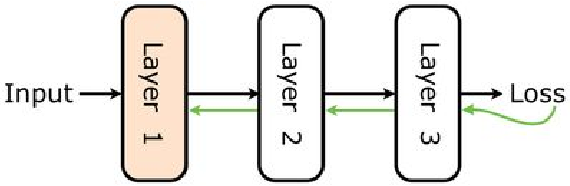
Solutions:

1. Truncated backprop (misses long range interactions)
2. RTRL (too expensive)
3. Approximate/local RTRL (promising)
4. **Synthetic Gradients (drastic)**

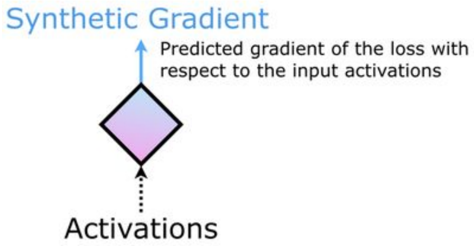
Training recurrent net-works online without backtracking. Ollivier et. al. (2015)
Long Short-Term Memory. Hochreiter and Schmidhuber (1997)

DECOUPLED NEURAL INTERFACES

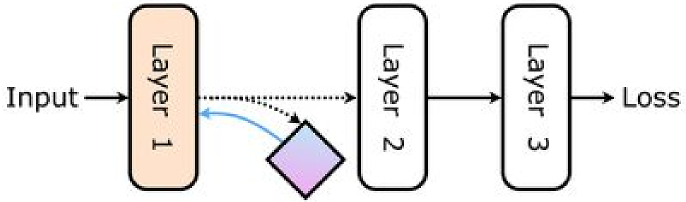
Consider a regular feed-forward network



We can create a **model of error gradients** using local information



The result is Layer 1 can now update **before the execution of Layer 2**.

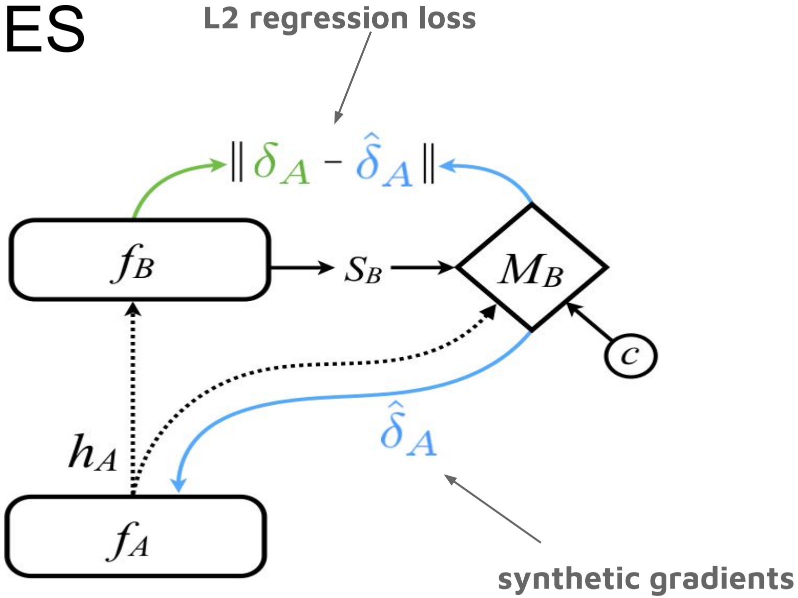


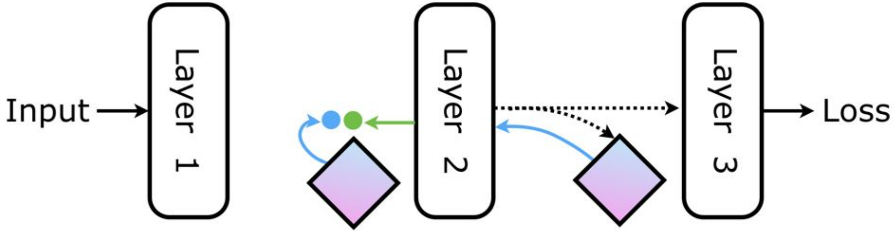
Decoupled Neural Interfaces using Synthetic Gradients. Jaderberg et. al. (2016)

Decoupled Neural Interfaces

The **synthetic gradient model** is trained to predict target gradients.

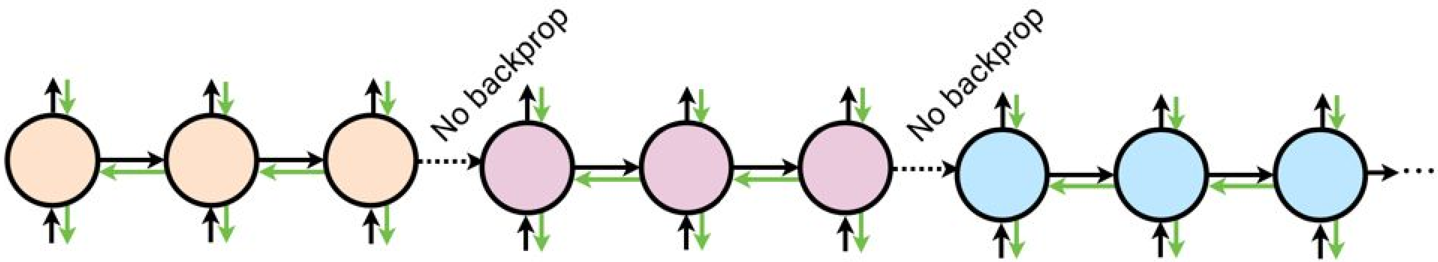
The target gradients could themselves be bootstrapped from other downstream synthetic gradient models.



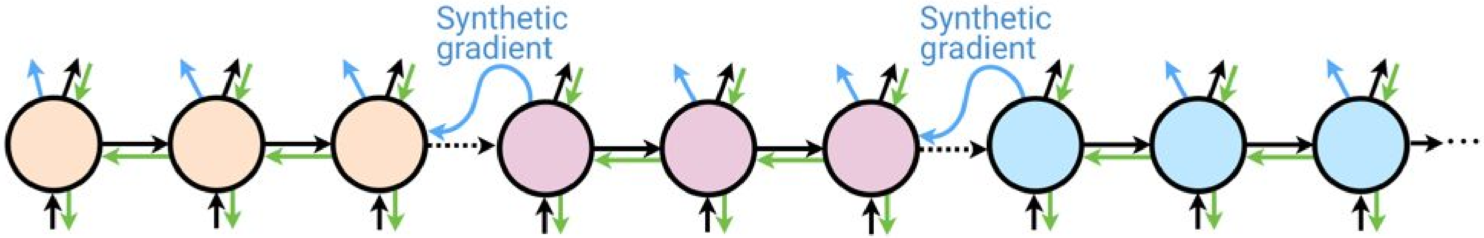


Analogous to return prediction bootstrapping in RL: 'Learn a guess from a guess'

Truncated BPTT



BPTT with Synthetic Gradients

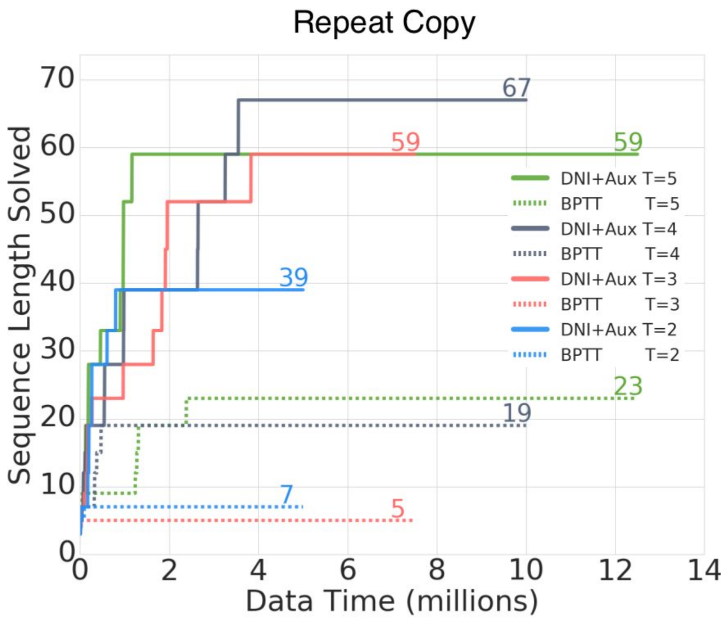


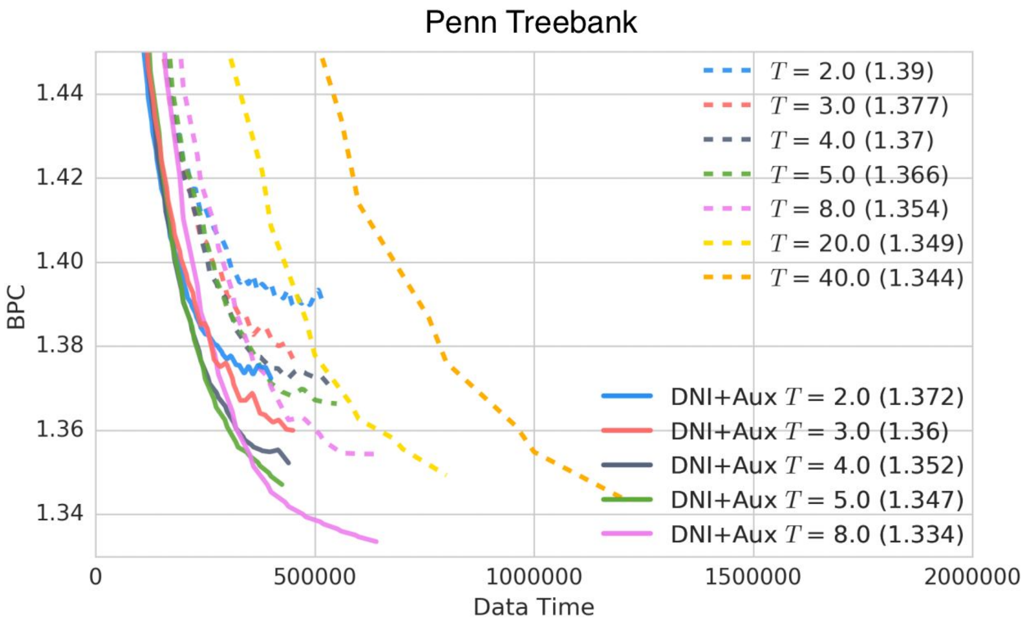
RNN learns to predict the gradients returned by its future self

RECURRENT MODELS

DNI extends the time over which a truncated BPTT model can learn.

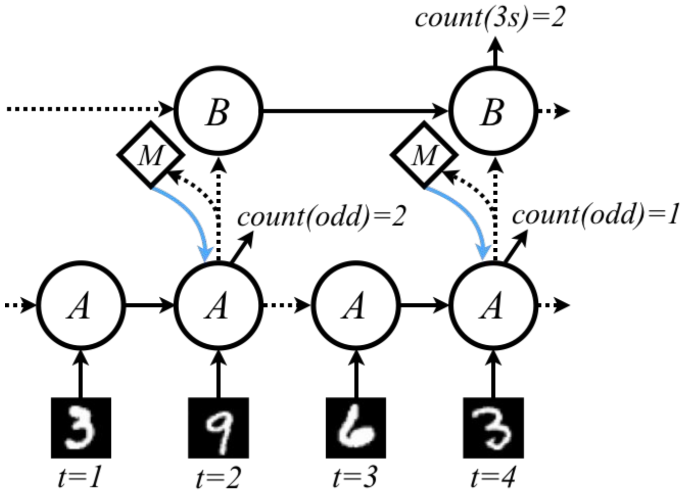
+ Convergence speed + Data efficiency

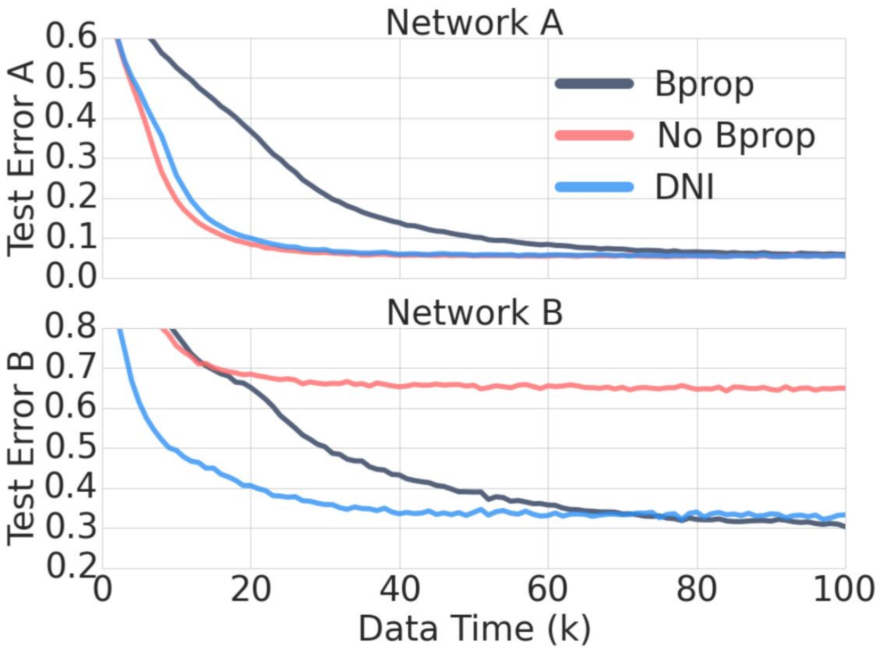




Multi Network

Two RNNs. Tick at different clock speeds. Must communicate to solve task.





Overall Architecture