Adaptive Computation Time

Alex Graves, *DeepMind* (2016)

Presenter: Chris Laver
Facilitators: Alexandre Tomberg, Rohollah Soltani
April 8, 2019

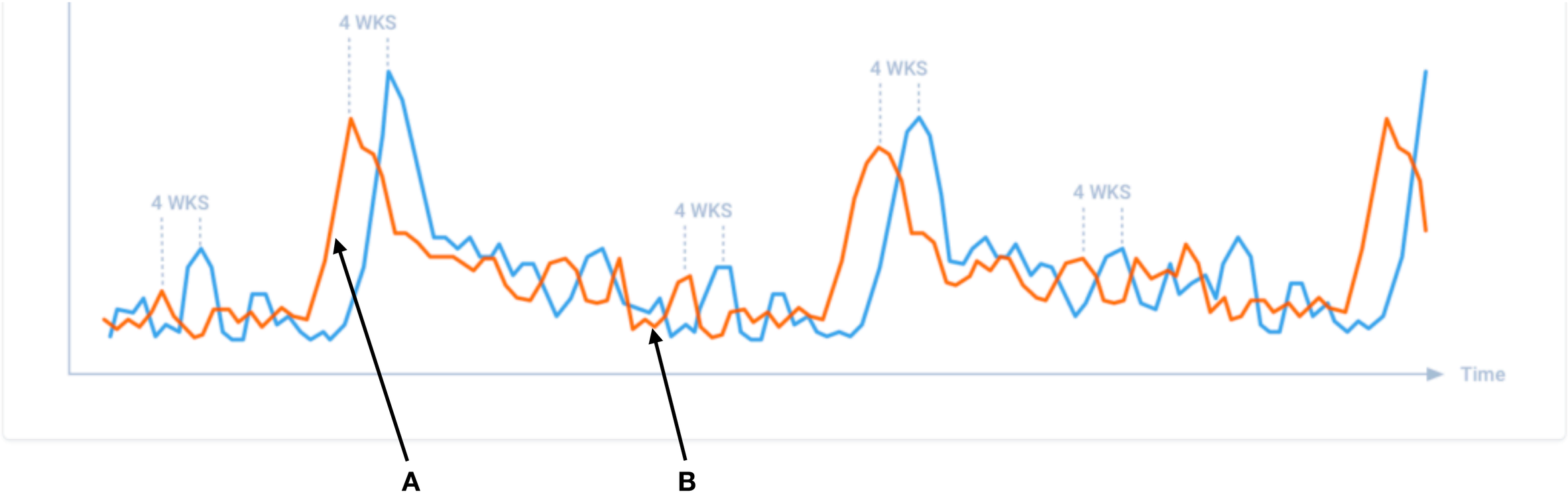
or…

“Pause, and consider”

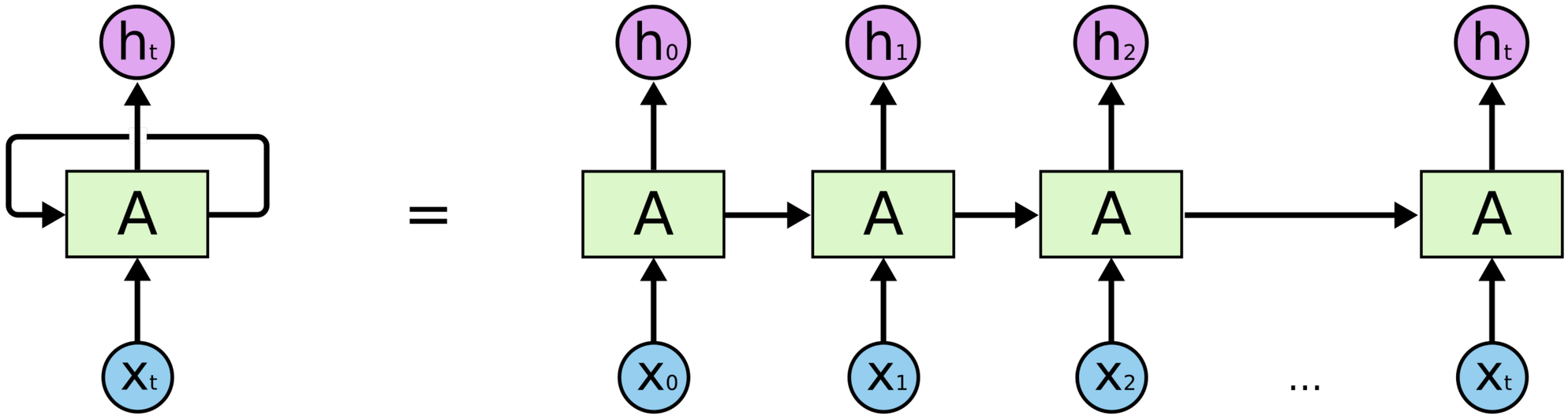
Consider the following stream of tokens

“The president of the United States of America is Donald J. Trump”

Not every token in this stream carries the same amount of information…
but a standard RNN model (LSTM/GRU/etc,.) applies the same amount of computation to every token.


4 WKS
4 WKS
4 WKS
4 WKS
4 WKS
Time
A
B

A standard RNN processes every token in a sequence one at a time, collecting context and emitting intermediate output at every time step



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Some sequences are just plain hard…

Pronoun resolution

“Jeremy picked up Matthew from work, then he drove to the park.”

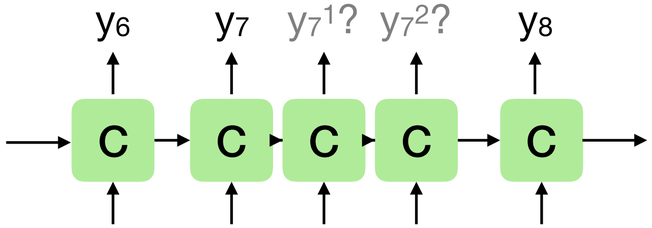
Winograd schema

“The city councilmen refused the demonstrators a permit because they [feared/advocated] violence.”

Prepositional phrase attachment

“The man saw the boy on the hill with the telescope.”

How might we let an RNN pause and consider difficult tokens?



“Jeremy picked up Matthew from work, then he he he drove to the park.”

Adaptive Computation Time
Alex Graves, 2016
https://arxiv.org/abs/1603.08983

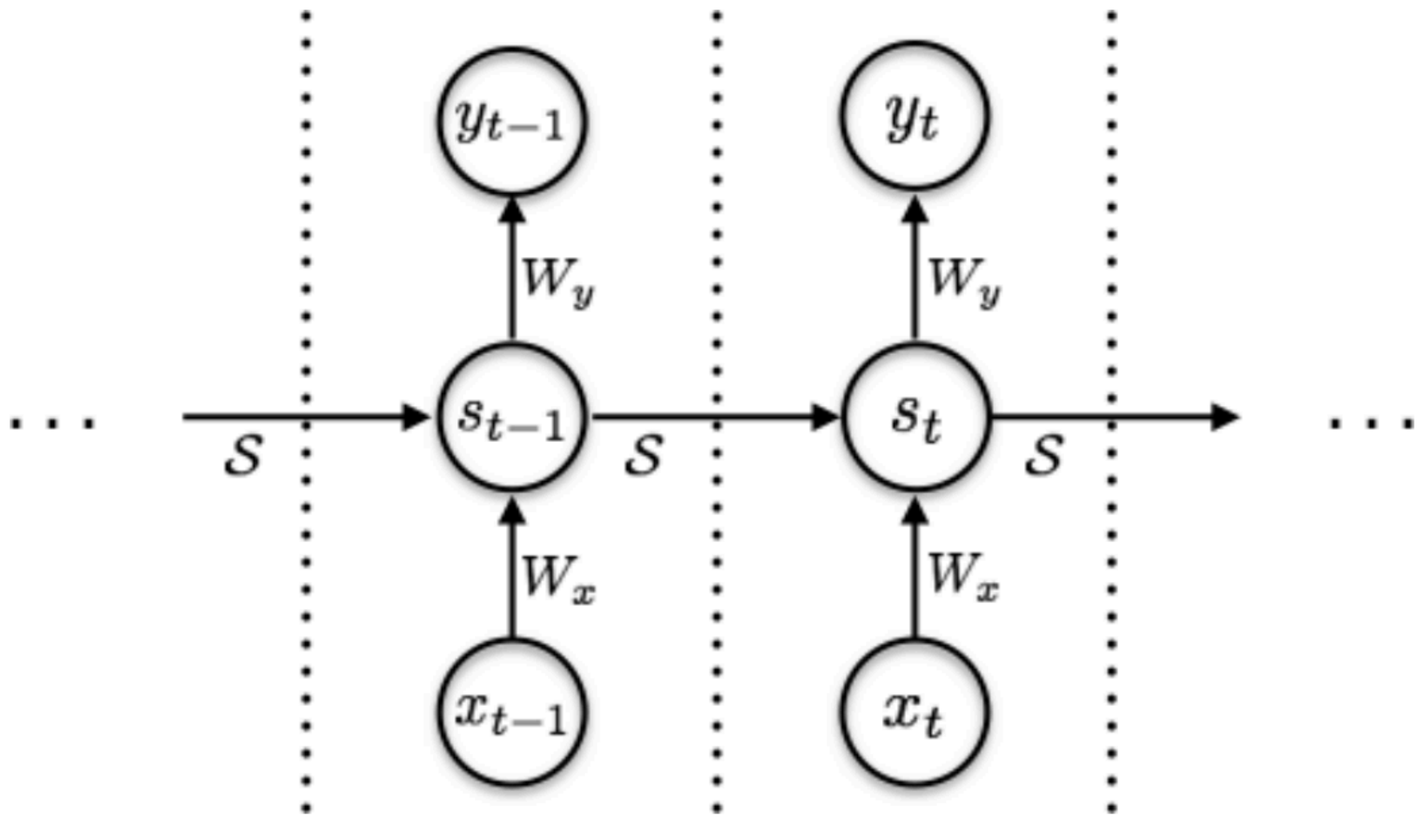
- Proposes a mechanism to “pause, and consider”, spending additional computation on tokens with more information.

- Additional computation budget is adaptively applied to tokens, no prior knowledge or expert guidance is required.

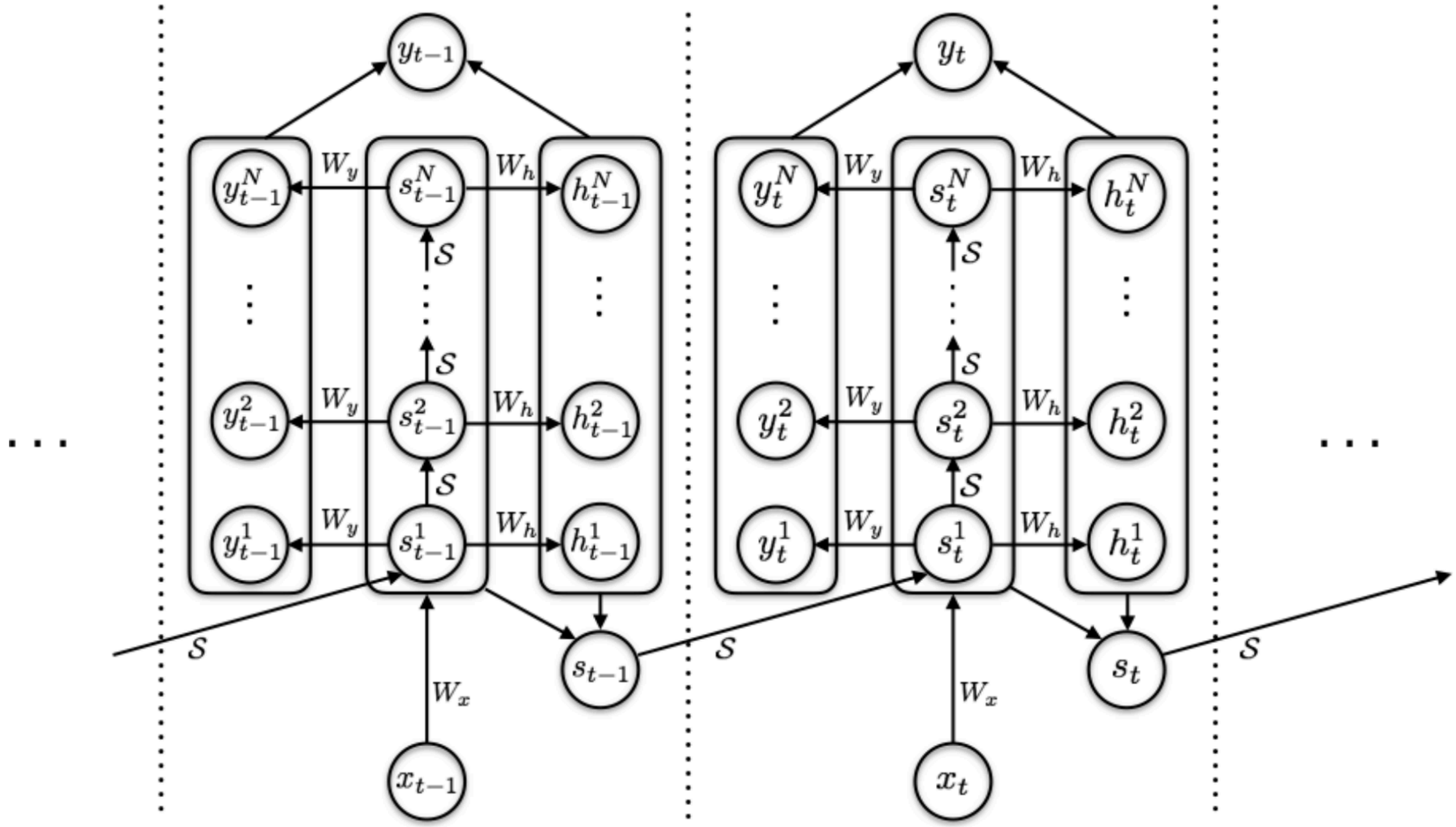
- An additional neuron is added to the recurrent unit which will give instruction to the network

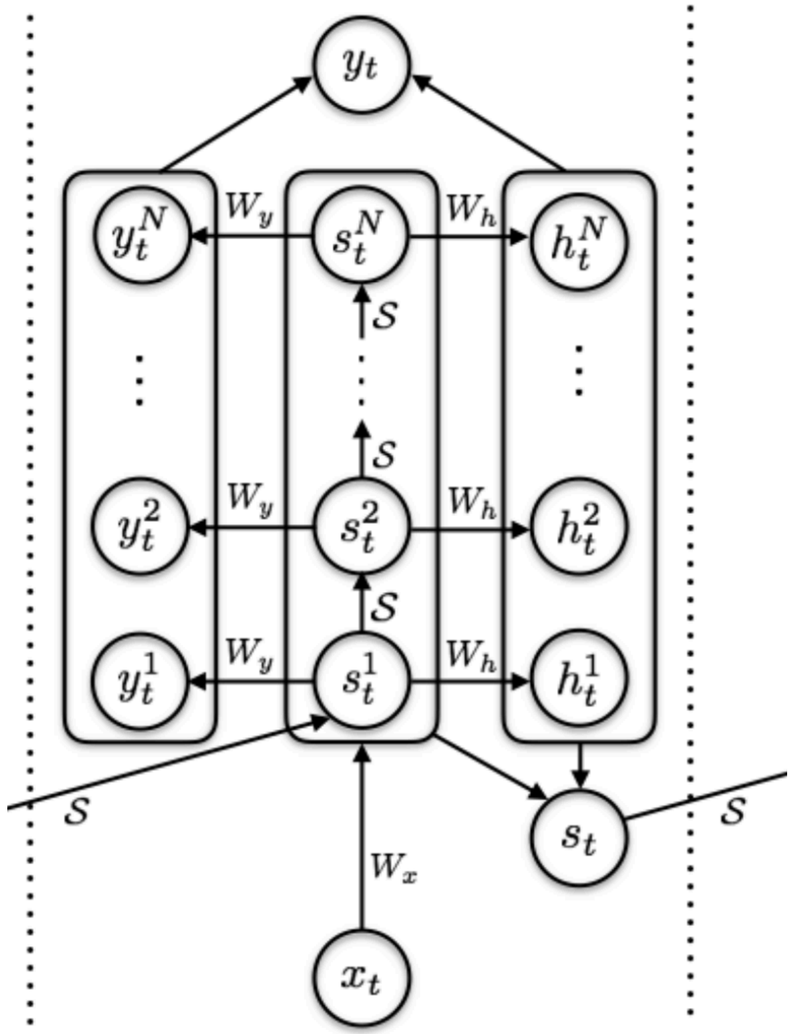
Advantages & Disadvantages of ACT	
Advantages	**Disadvantages**
Spend more time on tokens with high information	Additional cost for a technique that's already expensive
Adaptively determine where additional computation is beneficial	Seriously, LSTMs are already very expensive…
Generic technique, can be applied to any standard RNN	Adds a hyperparameter which is difficult to set in a principled manner

Reminder: An unrolled standard RNN



At each time step, ACT performs a variable number of state update steps



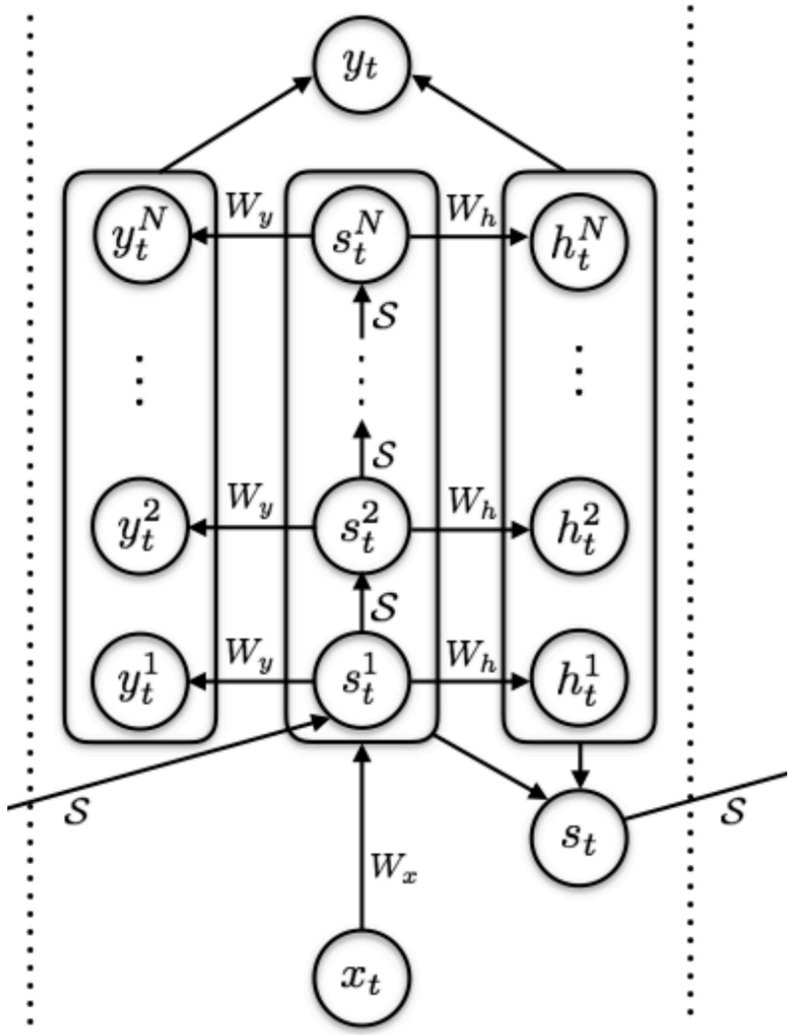


$$s_t^n = \begin{cases} \mathcal{S}(s_{t-1}, x_t^1) \text{ if } n = 1 \\ \mathcal{S}(s_t^{n-1}, x_t^n) \text{ otherwise} \end{cases}$$

$$s_t = \sum_{n=1}^{N(t)} p_t^n s_t^n$$

We distinguish between intermediate states and final state at a given time step.

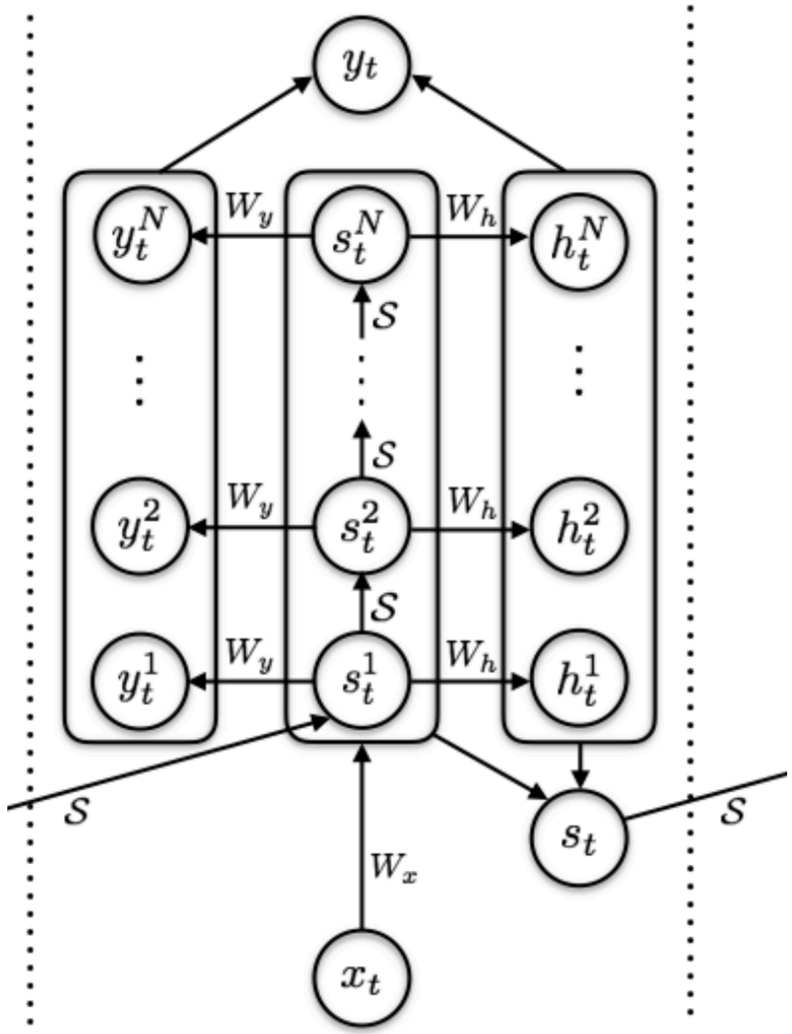
Note: The input in the above formula is augmented with a binary flag indicating if this is the first intermediate state at this time step.



$$y_t^n = W_y s_t^n + b_y$$

$$y_t = \sum_{n=1}^{N(t)} p_t^n y_t^n$$

As with state, we distinguish between intermediate and final outputs for each time step.



$$h_t^n = \sigma\left(W_h s_t^n + b_h\right)$$

$$p_t^n = \begin{cases} R(t) \text{ if } n = N(t) \\ h_t^n \text{ otherwise} \end{cases}$$

$$N(t) = \min\{n' : \sum_{n=1}^{n'} h_t^n >= 1 - \epsilon\}$$

$$R(t) = 1 - \sum_{n=1}^{N(t)-1} h_t^n$$

At each intermediate step, calculate the contribution of this immediate step to the whole time step, the remaining “budget” at this time step, and whether it’s time to stop this time step

To summarize:

Expand each time step to include N intermediate steps

a) At each time step, compute an intermediate state *s*, intermediate output *y*, and halting probability *h*.

b) Continue computing intermediate values until computation budget is consumed.

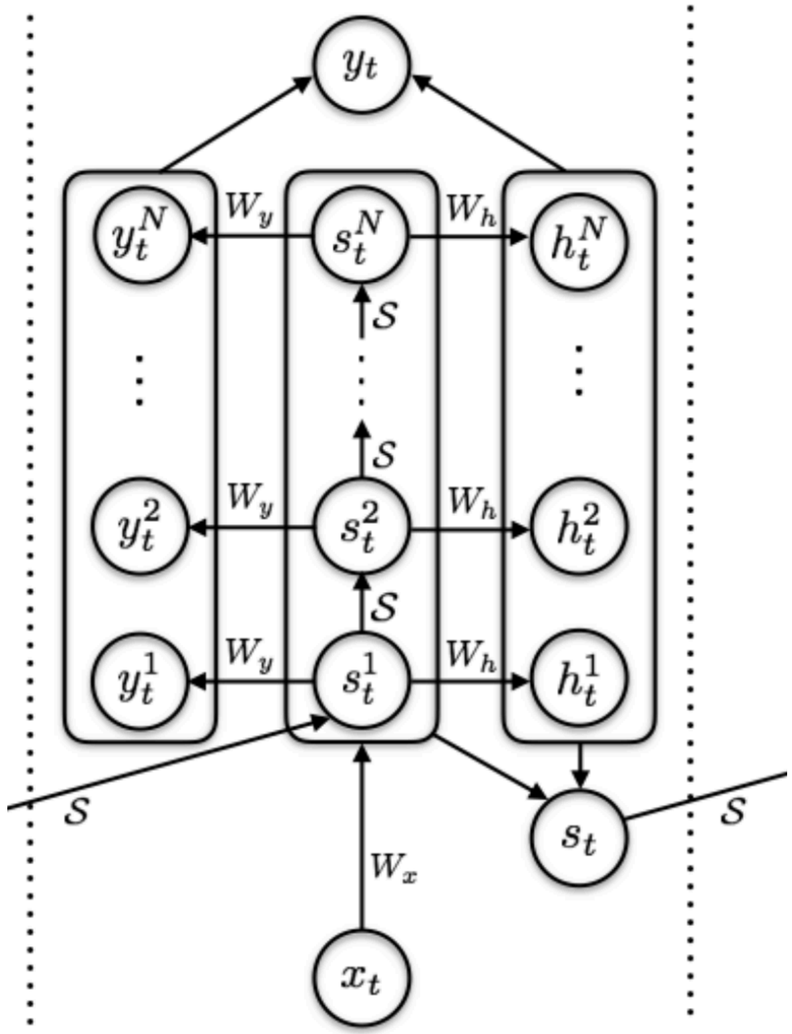
c) Compute final state and output values as a weighted sum (of halting probabilities) over the intermediate values.

Now we have a mechanism to ponder for a variable number of intermediate steps at each time step… at what cost?

For a standard RNN, total cost is O(C + ||S|| * R) where C is the cost of the non-recurrent network components, ||S|| is the sequence length, and R is cost of the recurrent network components. C is usually dominated by the recurrent term, and can be effectively ignored.

For an RNN with ACT, total cost is O(C + ||S|| * mean(N) * R) where mean(N) is the average number of intermediate steps per time step.

But how many steps is that…?



$$h_t^n = \sigma\left(W_h s_t^n + b_h\right)$$

$$p_t^n = \begin{cases} R(t) \text{ if } n = N(t) \\ h_t^n \text{ otherwise} \end{cases}$$

$$N(t) = \min\{n' : \sum_{n=1}^{n'} h_t^n >= 1 - \epsilon\}$$

$$R(t) = 1 - \sum_{n=1}^{N(t)-1} h_t^n$$

The network is incentivized to make the halting unit contribution approach 0 at each time step, to provide effectively infinite “ponder” computation at each time step.

There is a need to penalize the network for long ponder sequences (but not completely rule them out when lots of ponder is necessary).

$$(\rho_1, \ldots, \rho_T)$$
$$\rho_t = N(t) + R(t)$$

Ponder sequence

$$\mathcal{P}(\mathbf{x}) = \sum_{t=1}^{T} \rho_t$$

Ponder cost

$$\hat{\mathcal{L}}(\mathbf{x}, \mathbf{y}) = \mathcal{L}(\mathbf{x}, \mathbf{y}) + \tau \mathcal{P}(\mathbf{x})$$

New loss function

Tau is a penalty term on time that weights the relative cost of error vs. computation time. This paper explores its impact on quality of results, but leaves a principled mechanism for setting tau as future work.

Early in the training regime, while the network is still learning how to use the halting neuron, it is advantageous to penalize the ponder cost more heavily.

It's also beneficial to put a hard cap on the number of intermediate steps at each time step, just to avoid degenerate edge-case scenarios.

a) Apply a decreasing schedule for tau during training

b) Modify N(t) to put a hard cap on the number of intermediate steps

$$N(t) = \min\{M, \min\{n' : \sum_{n=1}^{n'} h_t^n >= 1 - \epsilon\}\}$$

The paper takes a mean-field approach to the halting probability, and the value of final state and output values for a time step. It's also possible to stochastically sample to determine if we should halt intermediate computation.

$\tilde{n} \sim Bernoulli(p_t^n)$

$s_t = s_t^{\tilde{n}}$
$y_t = y_t^{\tilde{n}}$

See the paper for a discussion of why the author considers this unnecessary.

Break

This paper evaluates ACT on four synthetic tasks and one real-world task, comparing against an RNN without ACT, as well as different parameter settings for ACT.

1. Binary sequence parity

2. Sequential application of logic functions

3. Variable-length sequential addition

4. Sorting

5. Wikipedia next-character prediction

The synthetic tasks all show broadly the same outcome, which is that a basic RNN which solves the problem poorly will solve the problem well with ACT.

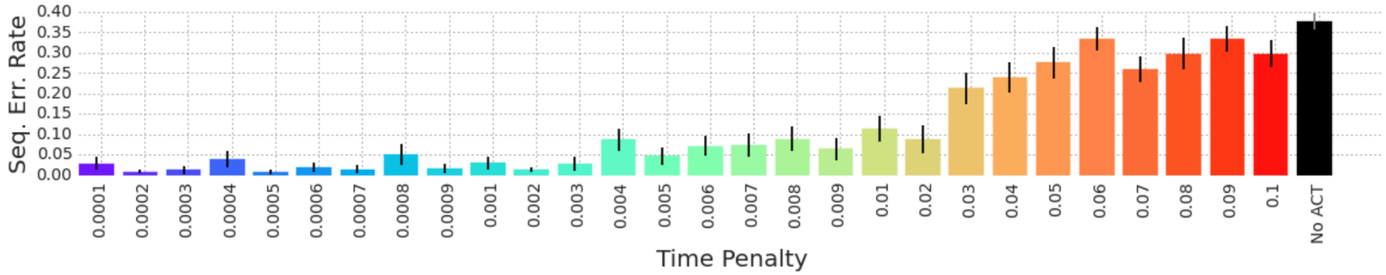
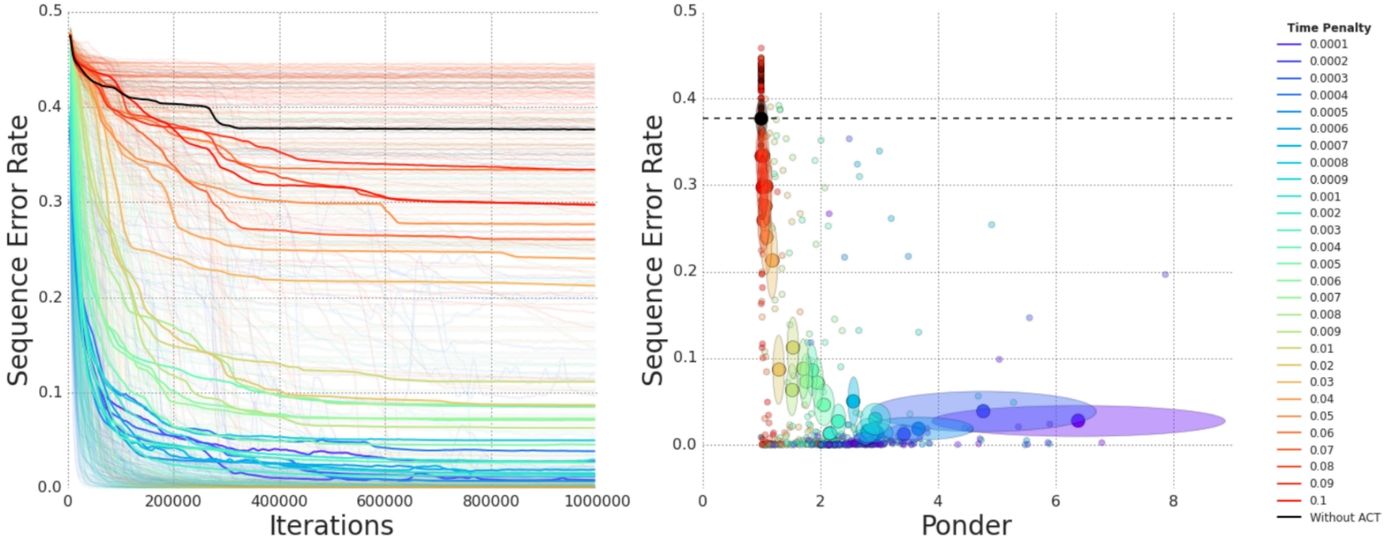


Figure 4: Parity Error Rates. Bar heights show the mean error rates for different time penalties at the end of training. The error bars show the standard error in the mean.



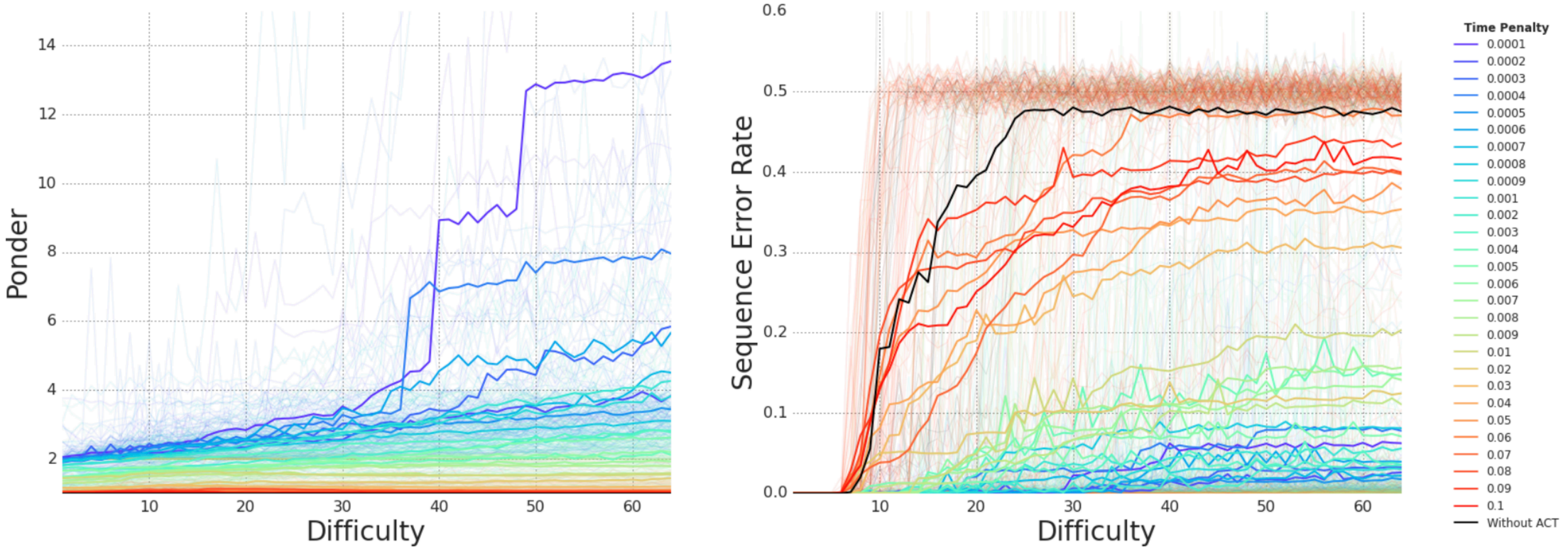
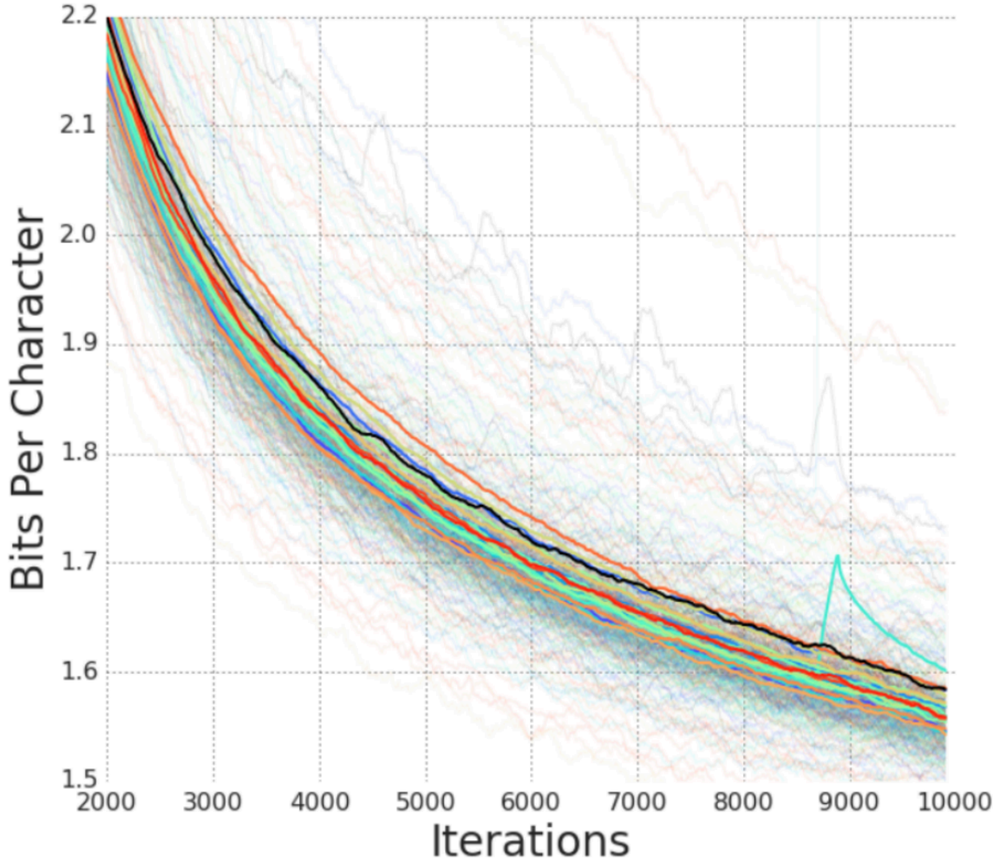
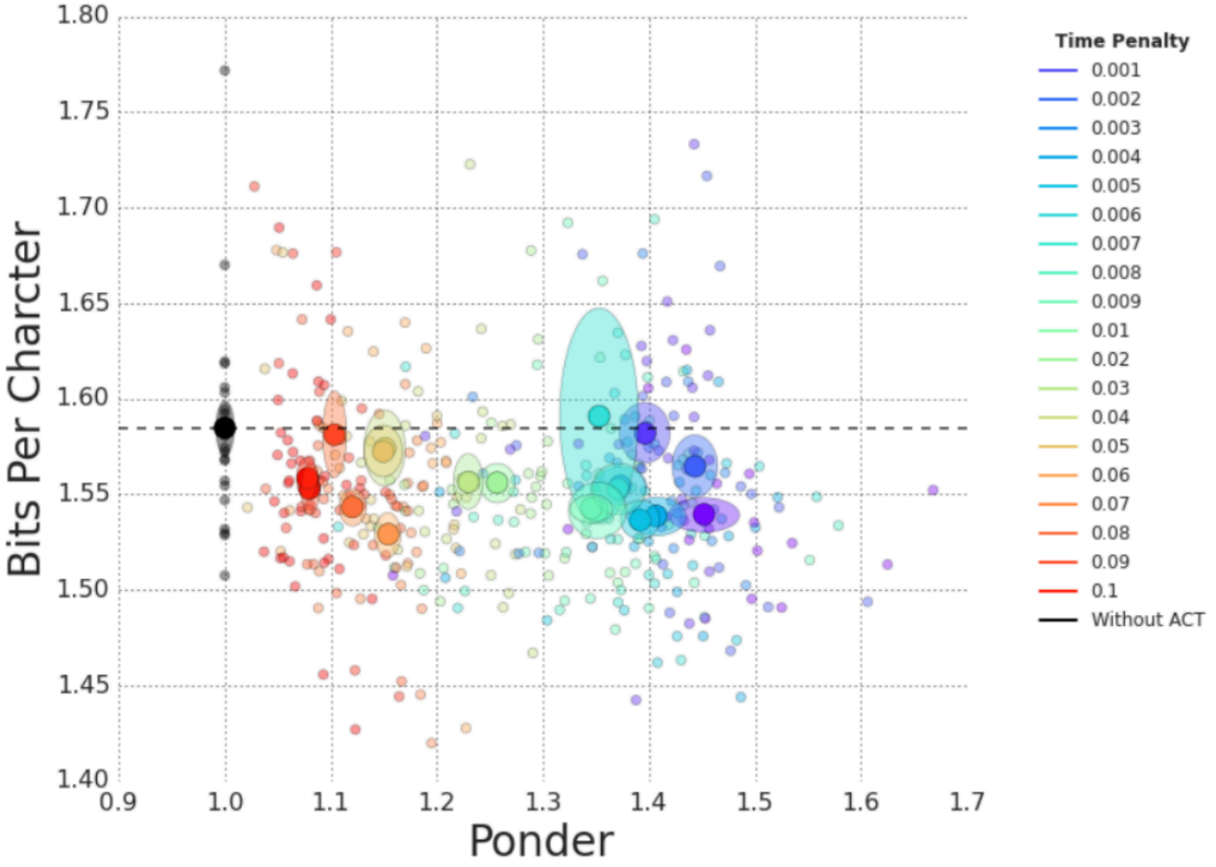


Figure 6: Parity Ponder Time and Error Rate Versus Input Difficulty. Faint lines are individual runs, bold lines are means over 20 networks. 'Difficulty' is the number of bits in the parity vectors, with a mean over 1,000 random vectors used for each data-point.

The wikipedia task character prediction task didn’t show a large improvement in loss with ACT applied, but did show some intriguing results relating to when the network chose to apply longer ponder time.





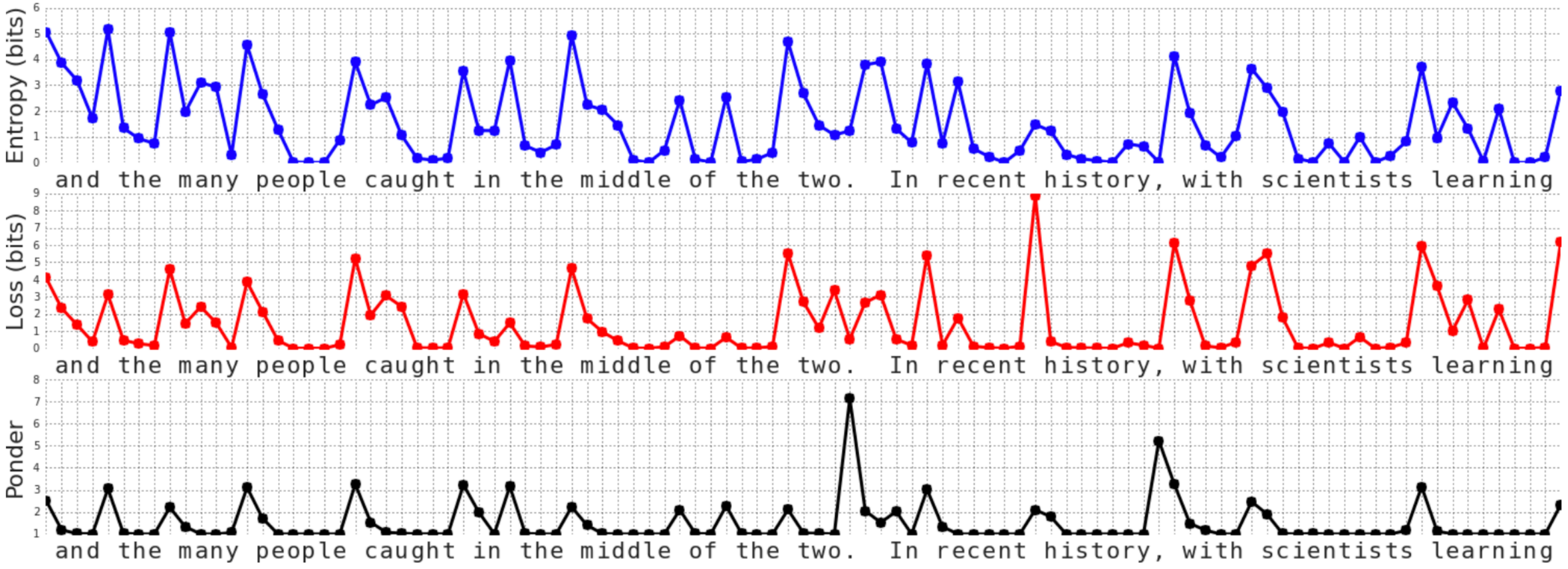


Figure 23: Ponder Time, Prediction loss and Prediction Entropy During a Wikipedia Text Sequence. Plot created using a network trained with $\tau = 6e^{-3}$

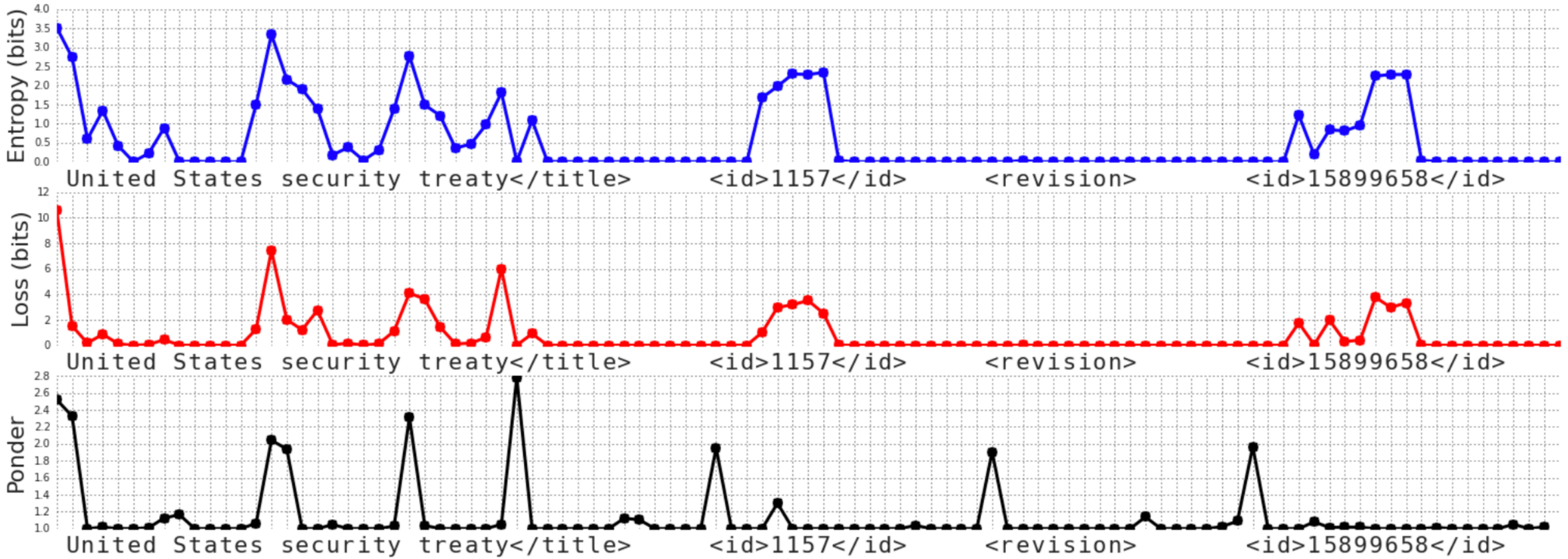
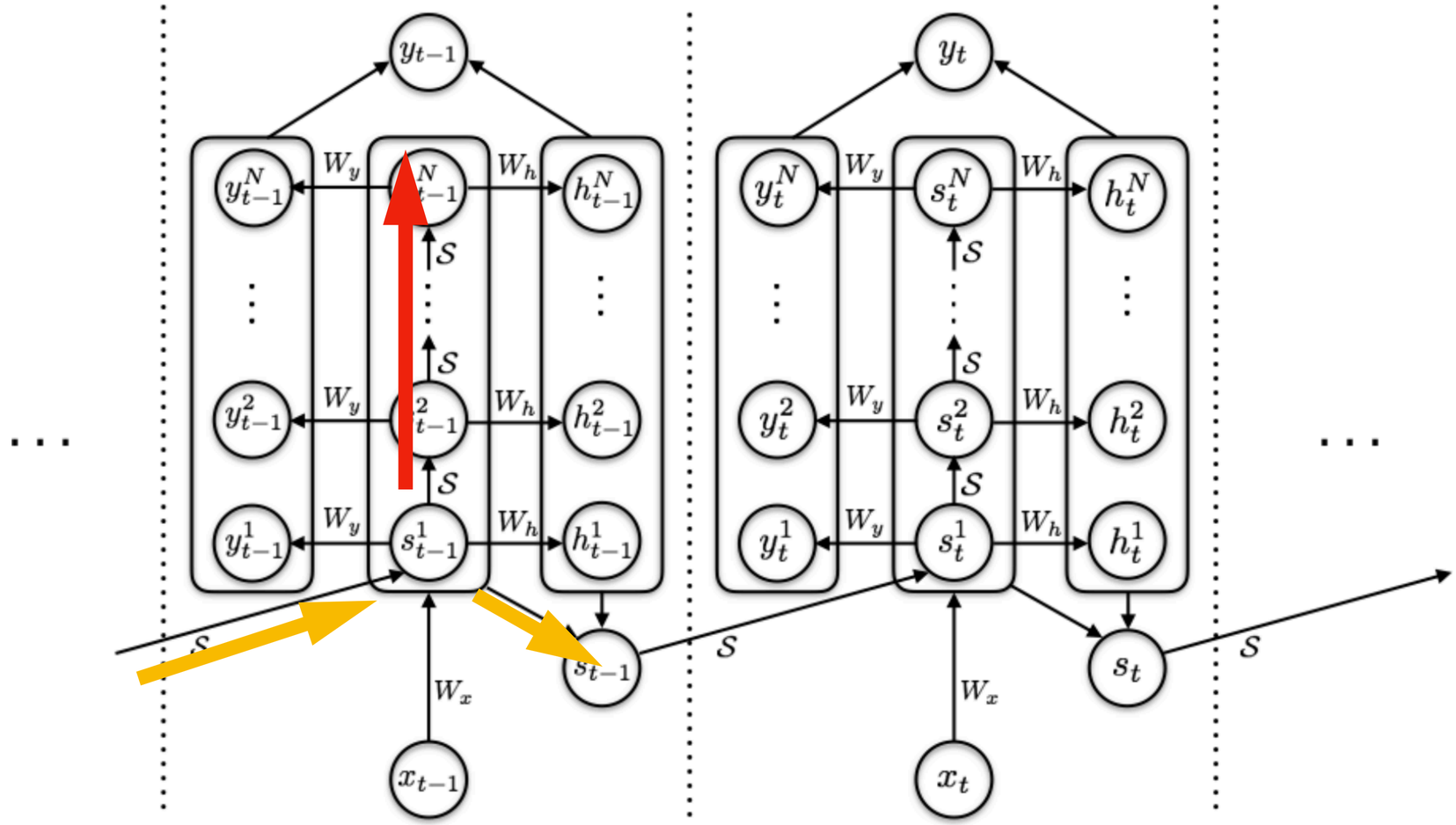
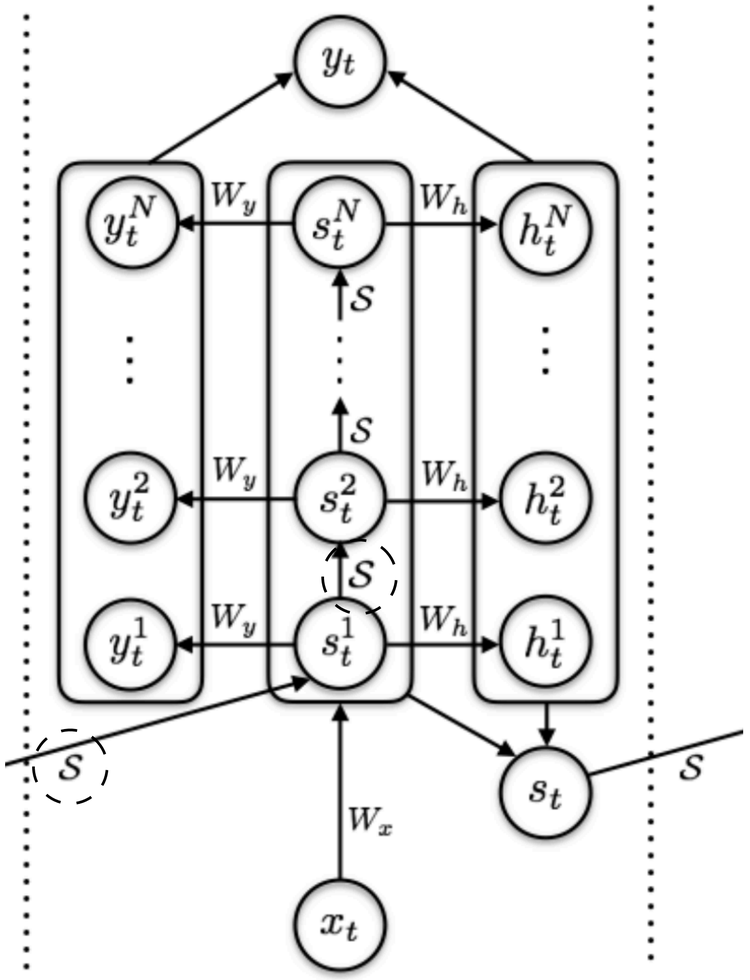


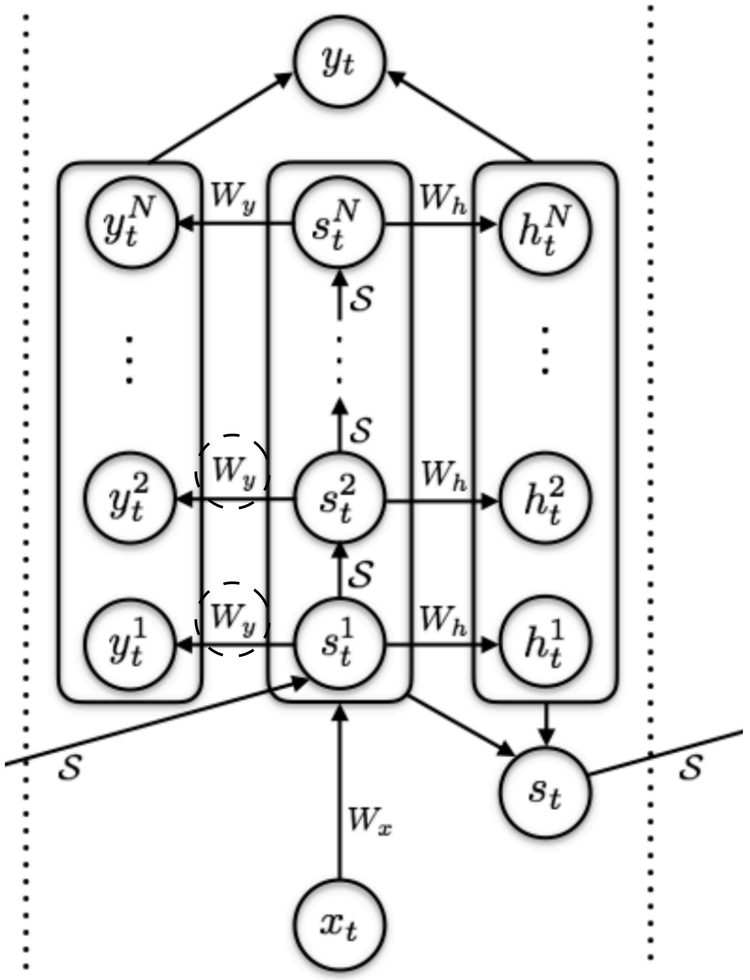
Figure 24: Ponder Time, Prediction loss and Prediction Entropy During a Wikipedia Sequence Containing XML Tags. Created using the same network as Figure 23.

Have we exacerbated the long-term dependency problem?



What if we don’t share parameters?





Given the marginal improvement on a real-world task, is the additional cost of ACT “worth it”?

- Multiple sequential queries to external memory? (e.g. memory net, dynamic memory networks, *neural differentiable computer/NTM*, etc,.)

- “hard” sequences?

Thanks for listening!