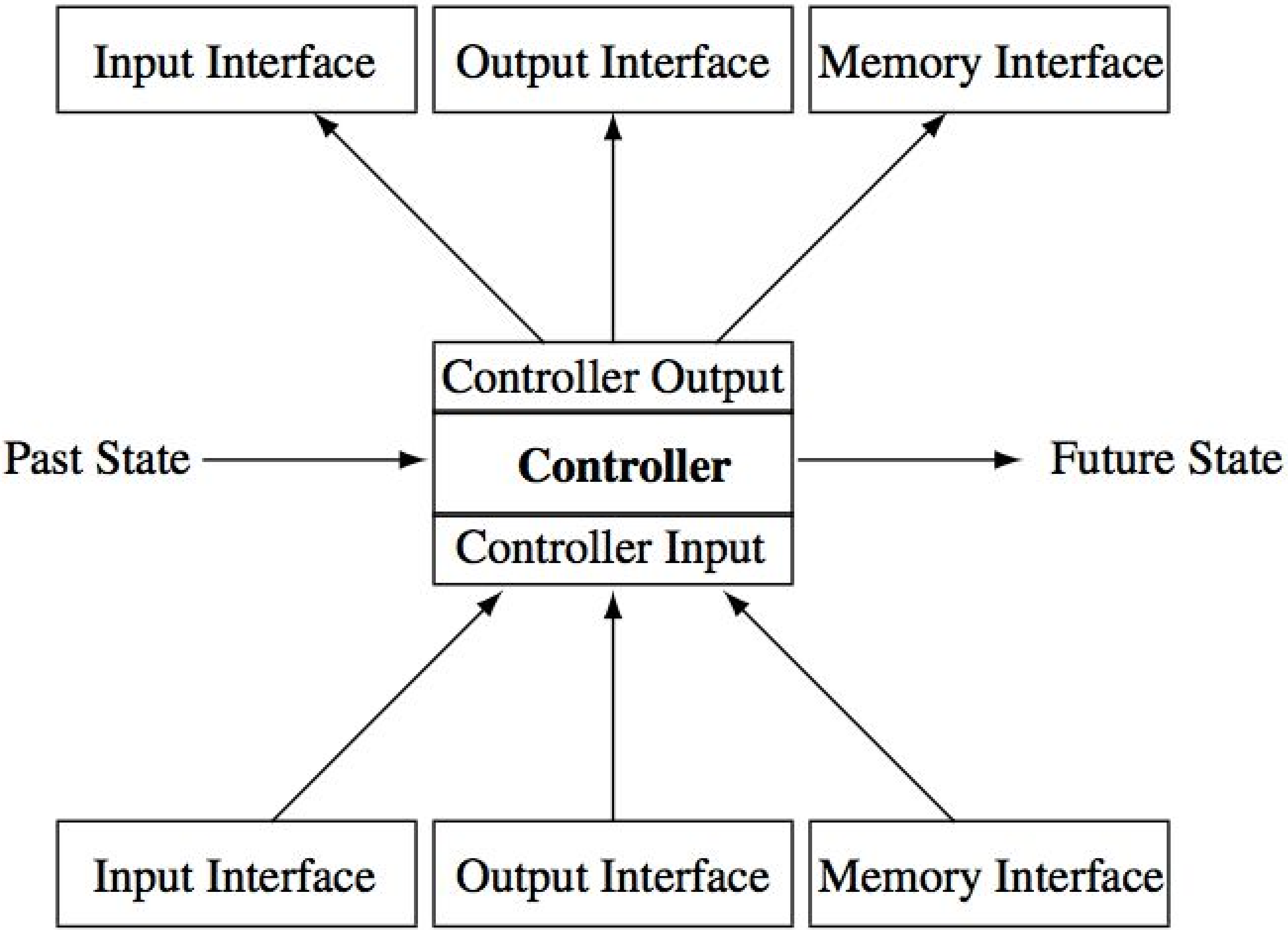
# Discrete
# Neural Turing Machines

by **Wojciech Zaremba**
currently at OpenAI
(work done at Facebook, and Google)

NYU

# Motivation

- Brain in a vat cannot solve many problems
- Need interaction with external tools to solve interesting tasks
  - eyes, hands are an exemplary external interfaces
- Many interesting interfaces are discrete
  - Google search enginee
  - Database etc.


Input Interface
Output Interface
Memory Interface
Controller Output
Past State
Controller
Future State
Controller Input
Input Interface
Output Interface
Memory Interface

# Contemporary models

- RNN, CNN, any feed-forward network
  - have constant running time
  - no external memory

Such models cannot solve problem that requires O(n^2) steps like multiplication

Unlimited memory interface
and arbitrary running time makes model
Turing Complete (not necessary **trainable**)

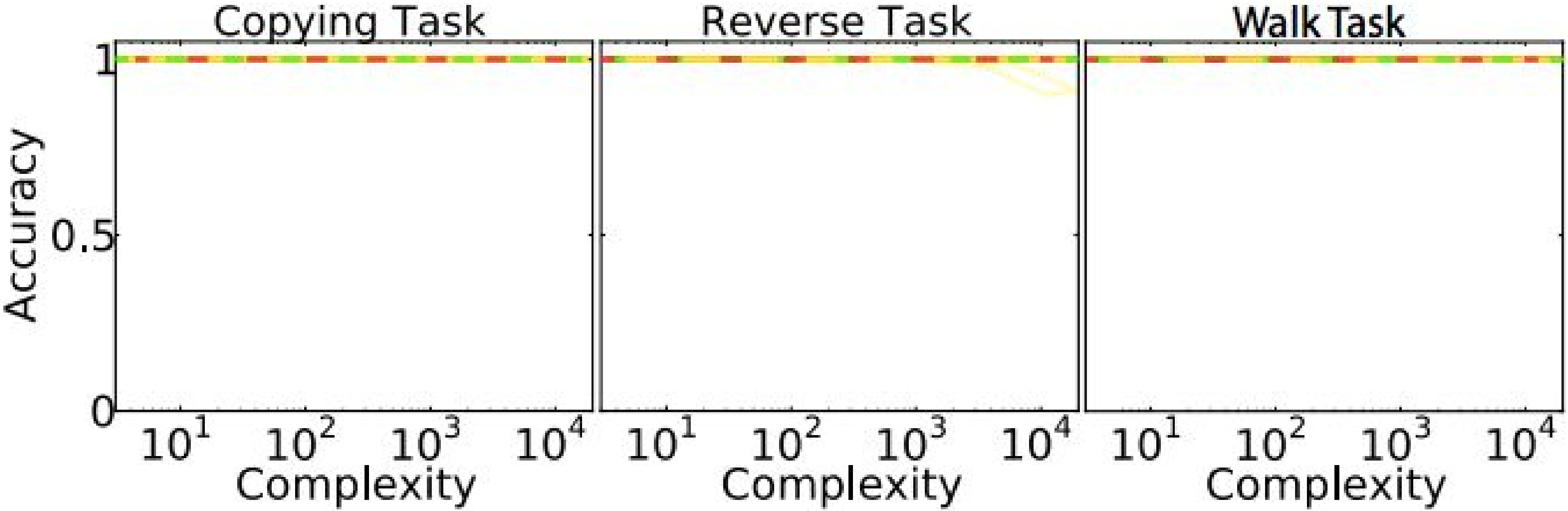
# Controller-Interface paradigm

- NTM
  - continuous memory
- Stack RNN
  - stack memory
- Neural Random-Access Machines
  - addition, subtraction, multiplication gates as an interface
- Memory network
  - attention as the input interface
- RLNTM
  - tapes as interfaces

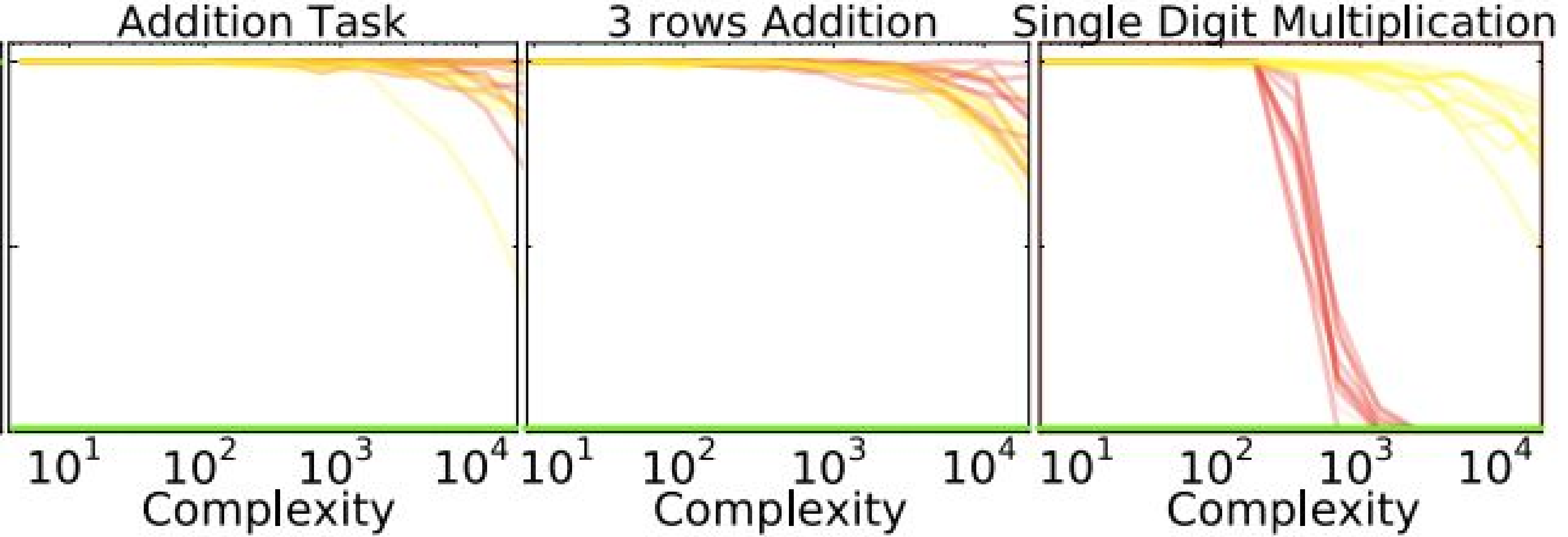
# Trainability

- Tasks that we consider are not-easy
- Can our models learn solution when actions are given
- Later, we train models without providing actions !!!

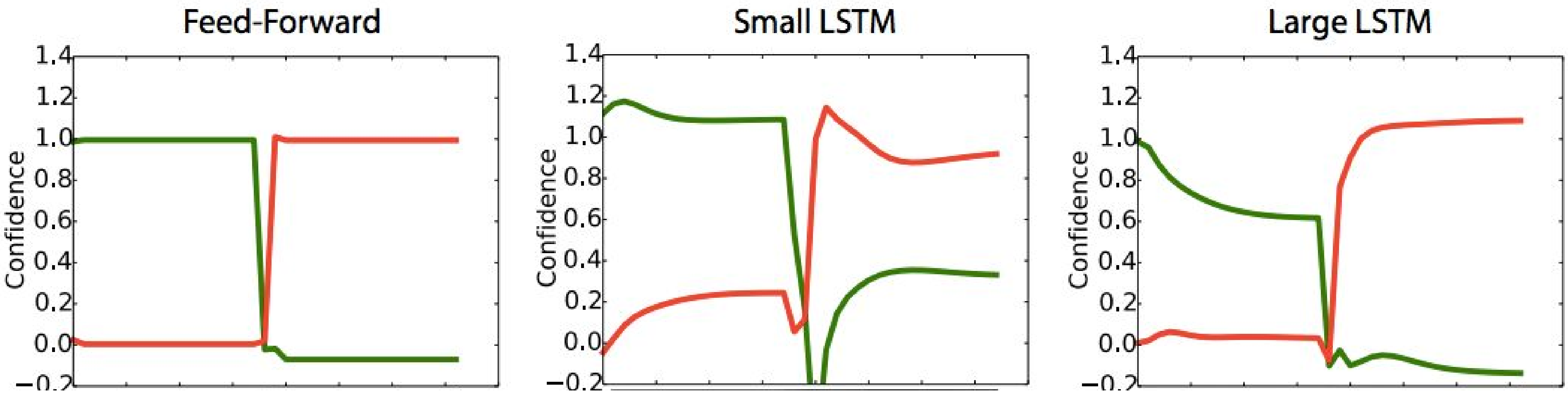
# Training with supervision



Copying Task
Reverse Task
Walk Task
Accuracy
1
0.5
0
$10^1$
$10^2$
$10^3$
$10^4$
Complexity

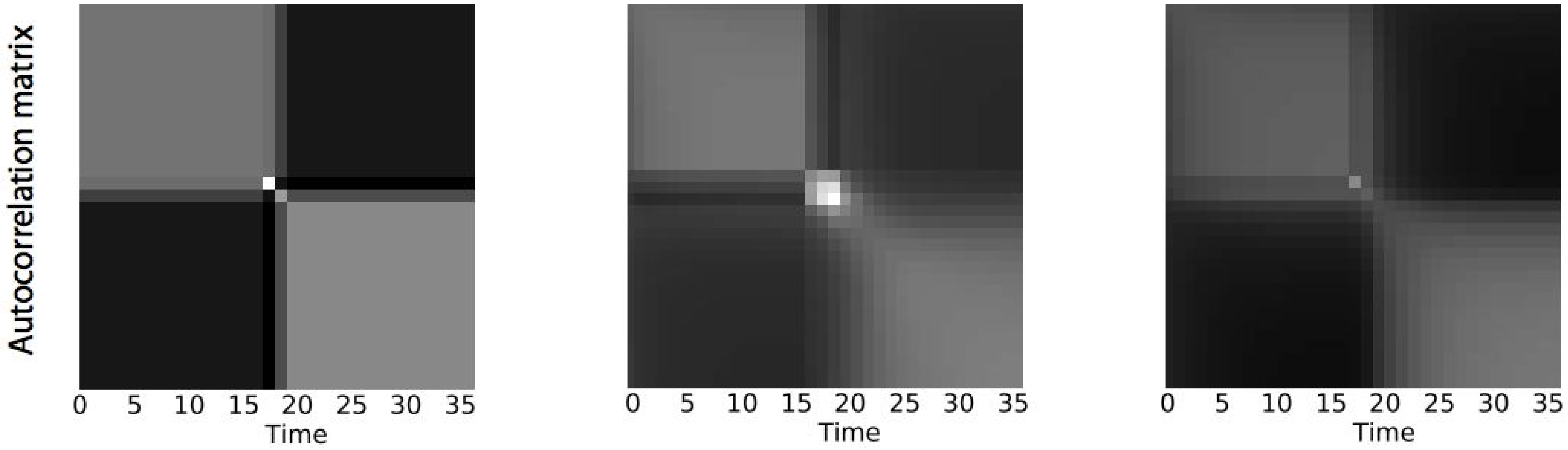


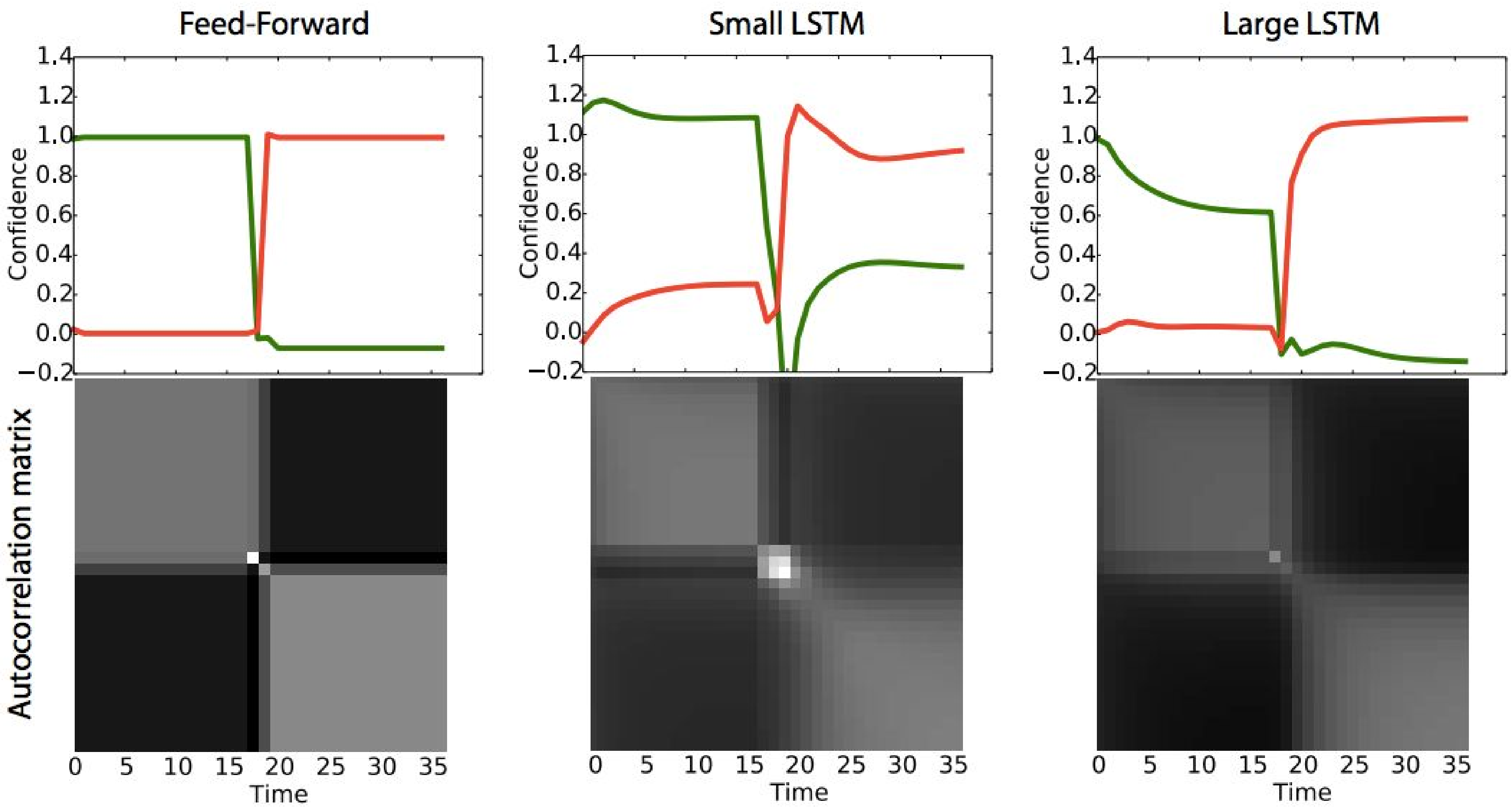
Addition Task
3 rows Addition
Single Digit Multiplication
$10^1$
$10^2$
$10^3$
$10^4$
Complexity
$10^1$
$10^2$
$10^3$
$10^4$
Complexity
$10^1$
$10^2$
$10^3$
$10^4$
Complexity




Confidence in action. (Green) go to the right, (Red) go to the left.

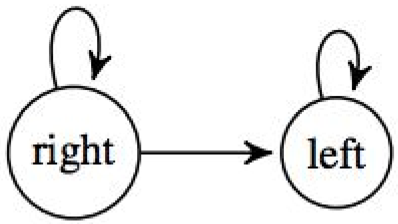
# Autocorrelation of the hidden state : cos(h_i, h_j)

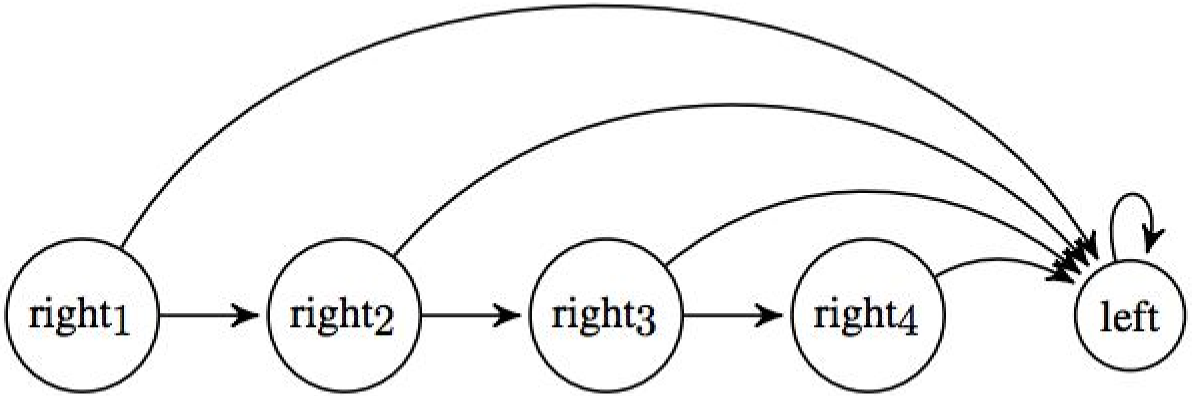




Feed-Forward
Small LSTM
Large LSTM
Confidence
Autocorrelation matrix
Time

# Underlying automata



# Intermediate conclusions

- Models with long memory might have difficult time learning simple algorithms, because there is a mismatch between training, and test states.
- Overfitting might happen with respect to sample length (which is limited for training instances, even though number of samples is huge)

# Intermediate conclusions

- All previous Interface-Controller models were tiny (100 units)
- NTM was not trainable with classic LSTM but with a different unit (possible less powerful).
- future goal: how to train big models for algorithmical tasks (not tackled here).

# Q-learning with NO supervision over actions

# Only input-output pairs

# Q-learning

- Reward of 1 for every correct prediction, and 0 otherwise.
- Model trained with Q-learning
- Q(s, a) estimates sum of the future rewards for an action “a” in a state “s”.
- Q is the off-policy algorithm (remarkable)

$$Q_{t+1}(s,a) = Q_t(s,a) - \alpha\left[Q_t(s,a) - \left(R(s') + \gamma \max_a Q_n(s',a)\right)\right]$$

# Q-learning as off-policy

- Policy inducted by Q is the argmax_a Q(s, a)
- When we follow induced policy, we say that we are on-policy
- When we follow a different policy, we say that we are off-policy
- Q converges to the Q for the optimal policy regardless of policy that we follow (as long as we can visit every state-action pair) !!!

# Watkins Q(lambda)

- Typical policy is a combination of on-policy (95%) with a random uniform policy (5%).
- Most of the time, we are on-policy
- This allows to regress Q on the other estimate:

$$Q^*(s_t, a_t) = \sum_{i=1}^{T} \gamma^{i-1} R(s_{t+i}) + \gamma^T \max_a Q^*(s_{t+n+1}, a)$$

# Dynamic Discount

- In Q-learning, model has to predict sum of future rewards.
- However, length of the episode might vary.
- We reparametrize Q, so it’s estimates sum of future rewards divided by number of predictions left.

$$\hat{Q}(s,a) := \frac{Q(s,a)}{\hat{V}(s)}$$

# Curriculum

- Three rows addition was unsolvable in the original form
- We start with small numbers that do not require carry.

$$\boxed{\begin{array}{c}1\\2\\2\end{array}}\;;\;\boxed{\begin{array}{c}2\\0\\2\end{array}}\;;\;\boxed{\begin{array}{cc}8&3\\3&3\\3&7\end{array}}\;;\;\boxed{\begin{array}{ccccc}3&2&0&6&9\\1&3&1&3&1\\2&8&0&8&3\end{array}}\;;\;\boxed{\begin{array}{ccccccccc}8&0&1&8&5&2&0&2&1\\1&3&1&4&0&7&0&5&4\\3&1&3&2&7&5&0&7&1\end{array}}$$

| Task | Test length | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 1000 | 1000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | #Units | 600 | 400 | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 200 |
| | Discount $\gamma$ | 1 | 1 | 1 | 0.99 | 0.95 | D | D | D | D | D |
| | Watkins $Q(\lambda)$ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ |
| | Penalty | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ |
| Copying | | 30% | 60% | 90% | 50% | 70% | 90% | 100% | 100% | 100% | 100% |
| Reverse | | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| Reverse (FF controller) | | 0% | 0% | 0% | 0% | 0% | 0% | 100% | 90% | 100% | 90% |
| Walk | | 0% | 0% | 0% | 0% | 0% | 0% | 10% | 90% | 10% | 80% |
| Walk (FF controller) | | 0% | 0% | 0% | 0% | 0% | 0% | 100% | 100% | 100% | 100% |
| 2-row Addition | | 10% | 70% | 70% | 70% | 80% | 60% | 60% | 100% | 40% | 100% |
| 3-row Addition | | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| 3-row Addition (extra curriculum) | | 0% | 50% | 80% | 40% | 50% | 50% | 80% | 80% | 10% | 60% |
| Single Digit Multiplication | | 0% | 0% | 0% | 0% | 0% | 100% | 100% | 100% | 0% | 0% |

**Reinforce with NO supervision over actions**

**Only input-output pairs**

# Reinforce

Objective of Reinforce:

$$\sum_{a_1,\ldots,a_n} p(a_1, \ldots, a_n|\theta) \sum_i r_i$$

we access it through sampling:

$$\mathbb{E}_a \sum_i r_i$$

# Reinforce

Derivative:

$$\sum_a p'(a|\theta) \sum_i r_i + \sum_a p(a|\theta) \sum_i r_i'$$

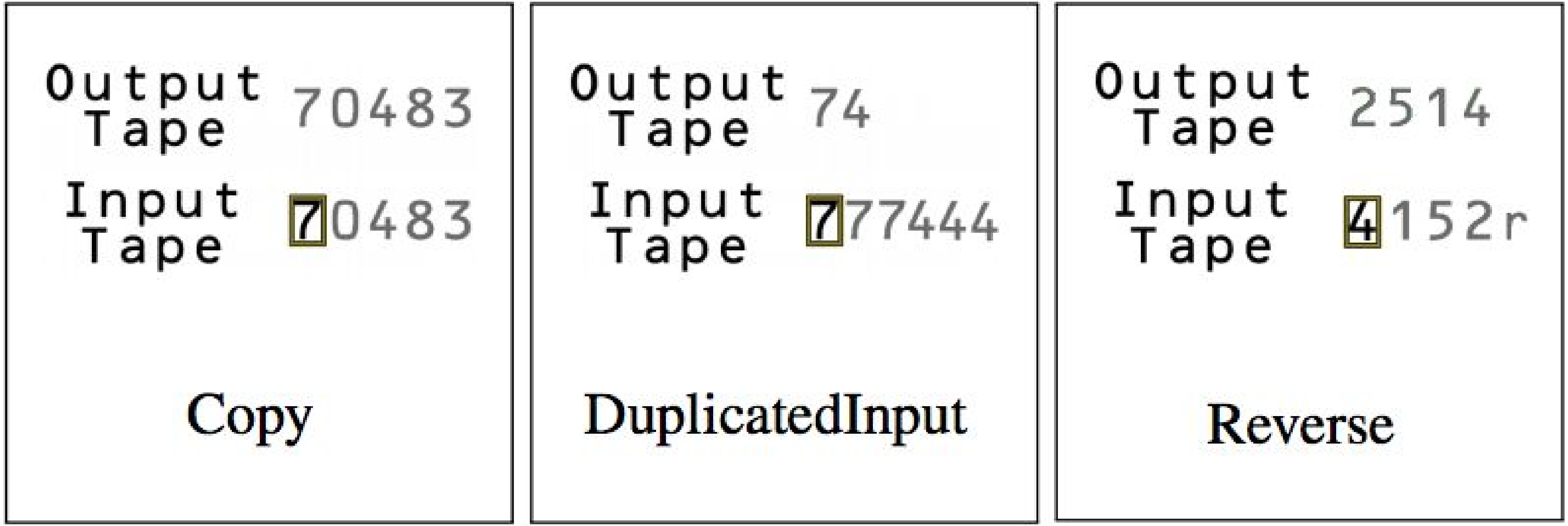
$$p' = p(\log p)'$$

we access it through sampling:

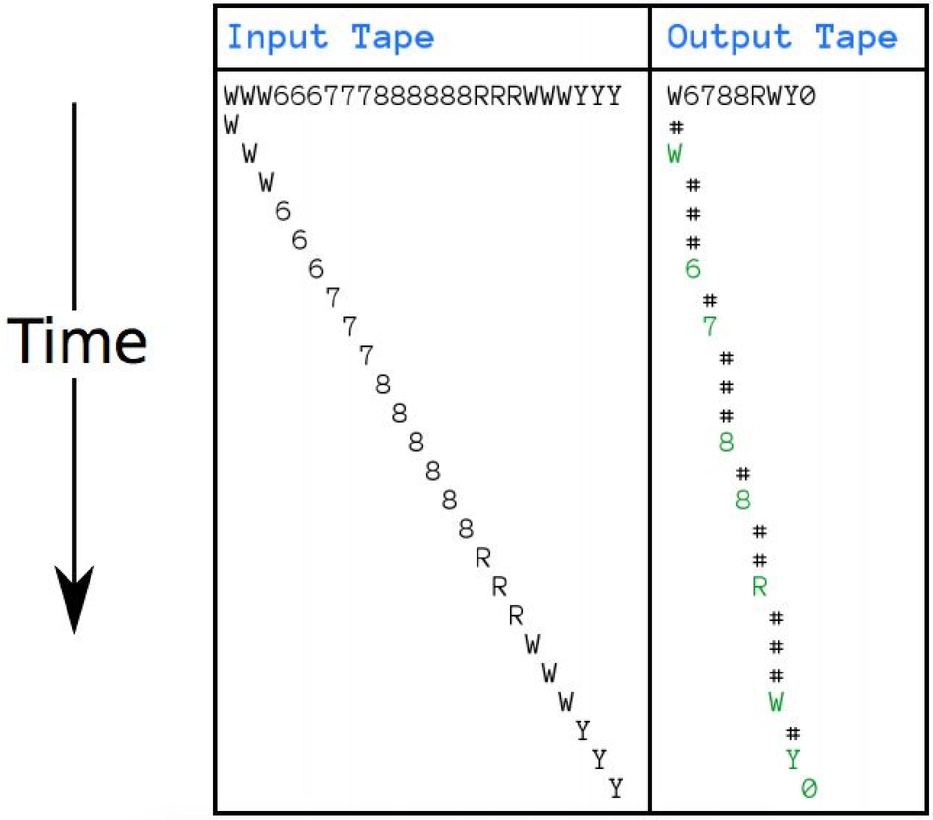
$$\mathbb{E}_a \log p' \sum_i r_i + \sum_i r_i'$$

# Training

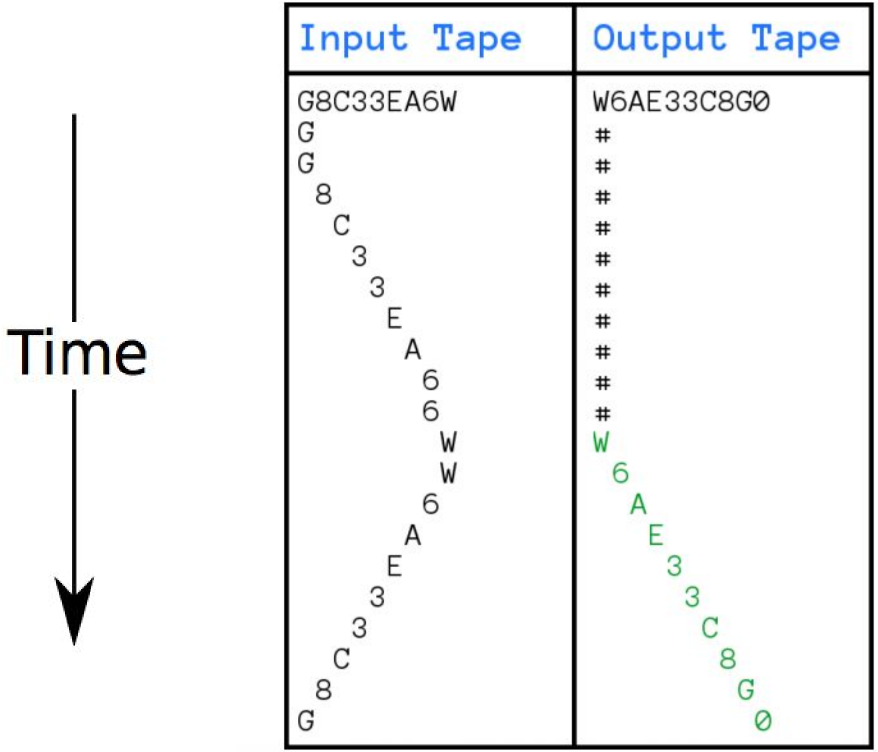
- Trained with SGD
- Curriculum learning is critical
- Not easy to train (due to variance coming from sampling)
  - Various techniques to decrease variance


Output Tape 70483
Input Tape 70483
Copy
Output Tape 74
Input Tape 777444
DuplicatedInput
Output Tape 2514
Input Tape 4152r
Reverse

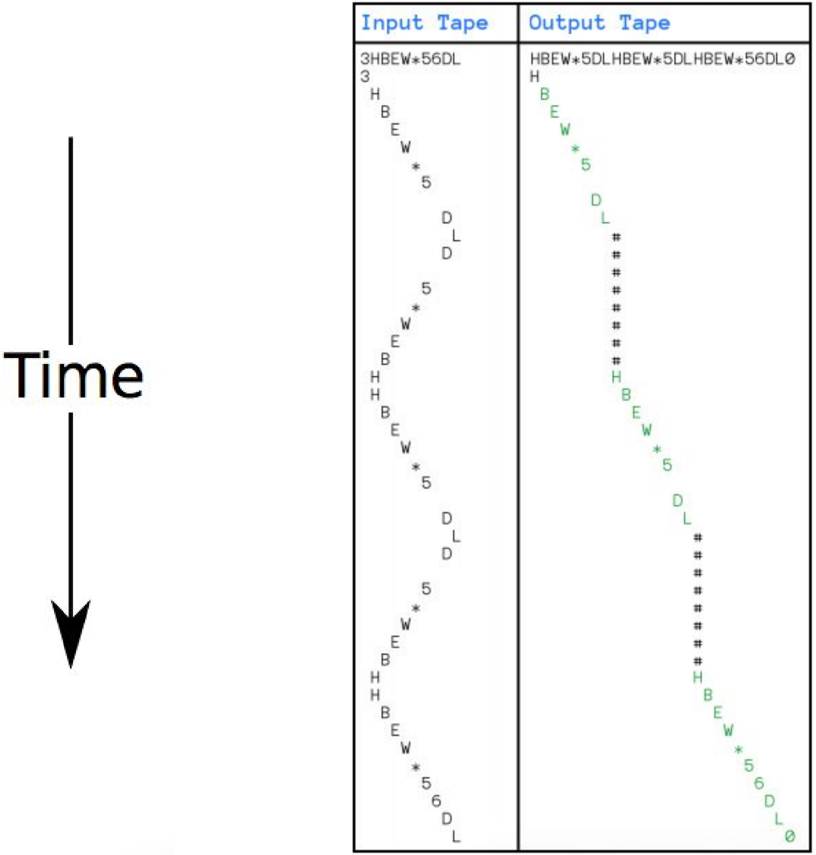
# Task - DuplicatedInput



# Task - Reverse



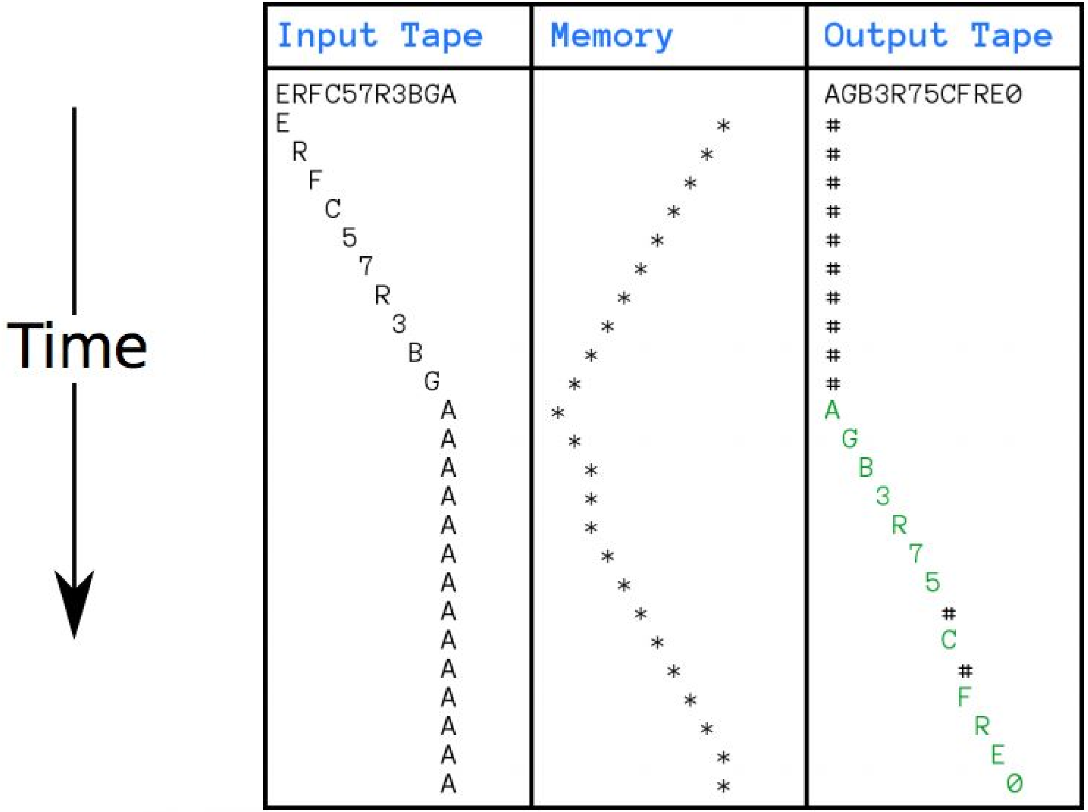
# Task - RepeatCopy



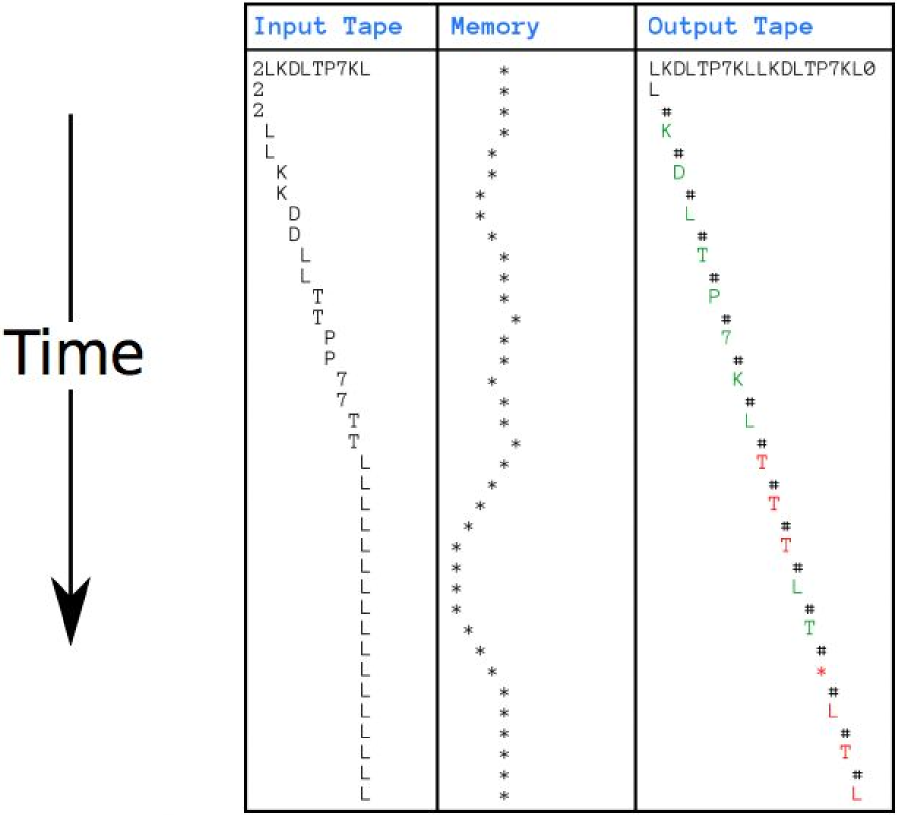
# Memory interface

- Memory is a tape with 3 actions, go to the left, stay, go to the right
- Controller always reads from previous memory location, and always saves to the next memory location
- It stores high dimensional vector through which we backpropagate

# Task - Reverse with memory



# Task. RepeatCopy with memory. Failure



# Gradient Checking - motivation

- Very simple to make a mistake in the implementation

- How to verify stochastic algorithm ?

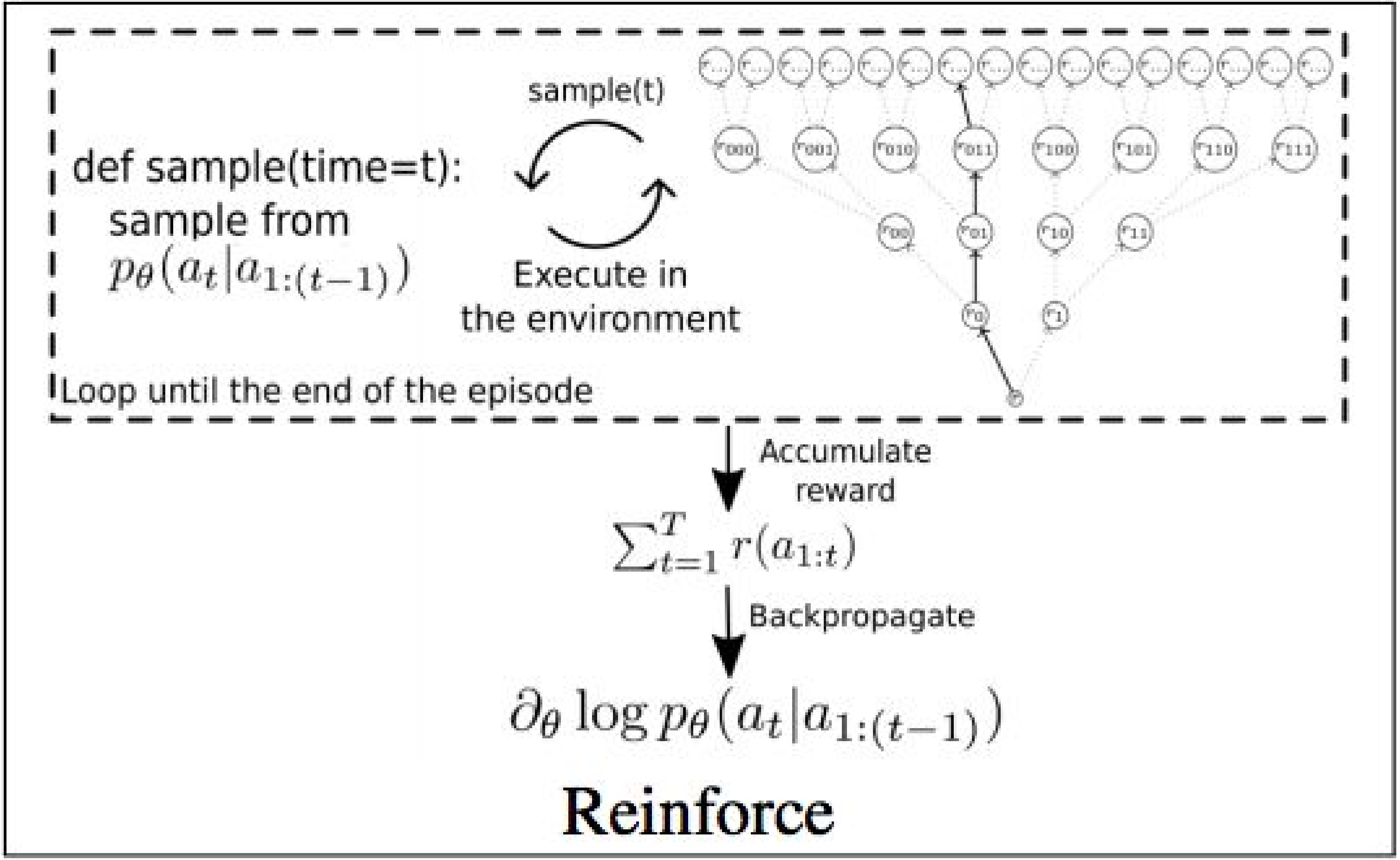
# Gradient Checking for Reinforce

- We could sample actions many times and compare the average gradient to average of the numerical gradient.

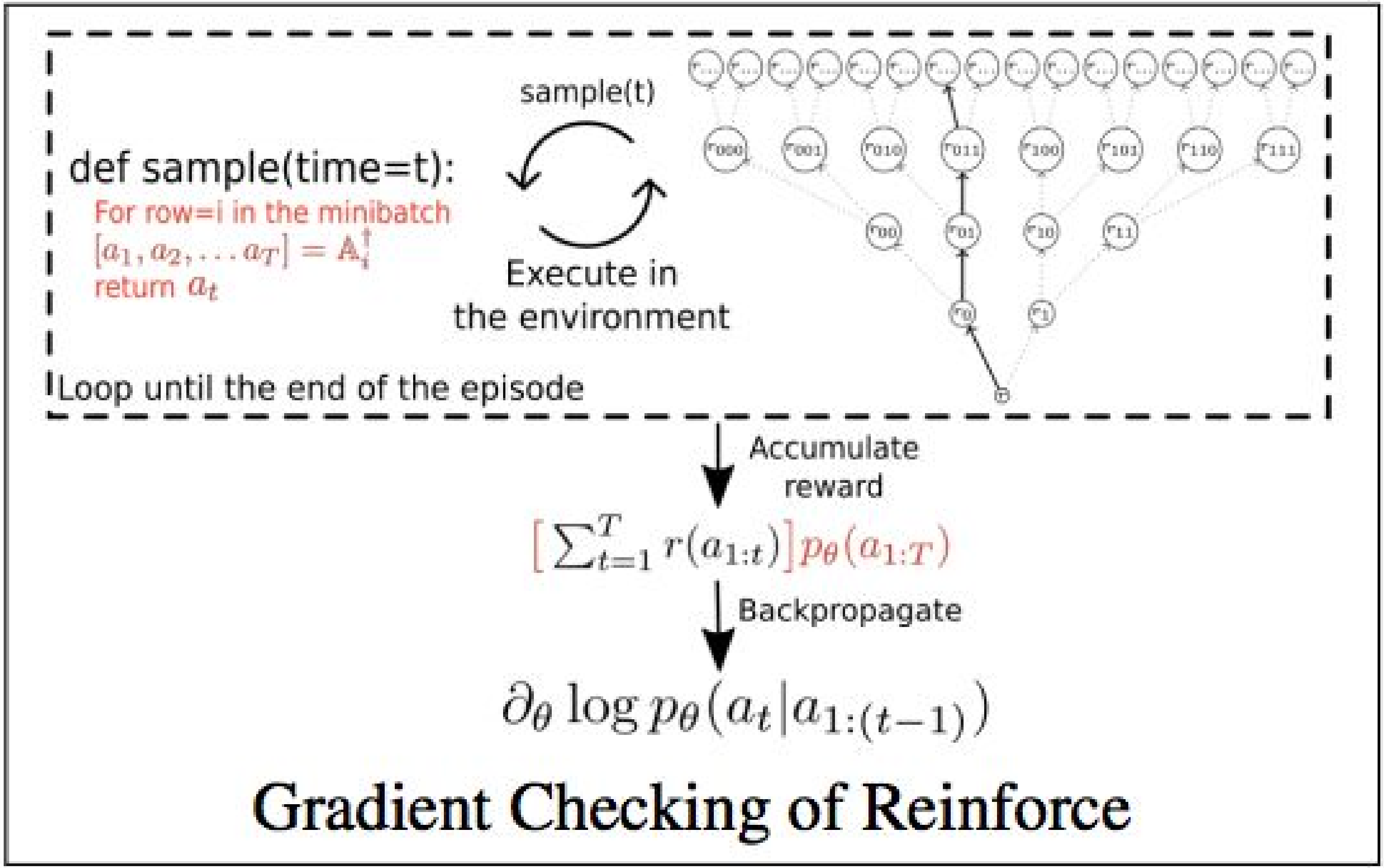
# Gradient Checking for Reinforce

- We could sample actions many times and compare the average gradient to average of the numerical gradient.

- Impractical. To get good precision we would need millions of samples.



def sample(time=t):
sample from
$p_\theta(a_t|a_{1:(t-1)})$
sample(t)
Execute in
the environment
Loop until the end of the episode
Accumulate
reward
$\sum_{t=1}^{T} r(a_{1:t})$
Backpropagate
$\partial_\theta \log p_\theta(a_t|a_{1:(t-1)})$
Reinforce


def sample(time=t):
For row=i in the minibatch
$[a_1, a_2, \ldots a_T] = \mathbb{A}_i^\dagger$
return $a_t$
sample(t)
Execute in the environment
Loop until the end of the episode
Accumulate reward
$\left[\sum_{t=1}^{T} r(a_{1:t})\right] p_\theta(a_{1:T})$
Backpropagate
$\partial_\theta \log p_\theta(a_t | a_{1:(t-1)})$
Gradient Checking of Reinforce

# Gradient Checking for Reinforce

- It was critical to make model work.
- We can limit size of action space during gradient checking
- Gradient checking takes seconds

# Variance of gradients

- Sampling of actions introduces variance into gradient estimate
- We subtract baseline reward to decrease variance

# Baseline reward

$$\sum_a p(a|\theta) = 1$$

$$\sum_a p'(a|\theta) = 0$$

$$\mathbb{E}_a \log p'(\sum_i r_i - v) + \sum_i r_i'$$

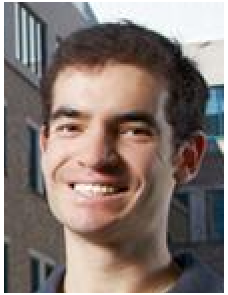
$$||\mathbb{E}_a \sum_i r_i - v||_{L_2}$$

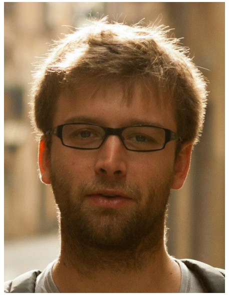
# Future work

- Solve tasks that require more than O(n) steps
- Training with persistent memory (memory that stores entire algorithms)
- Train large models on a family of tasks of increasing complexity (talk by Tomas)

# Thanks to my collaborators

Rob Fergus, Ilya Sutskever, Tomas Mikolov
and Armand Joulin



# Q&A

- Interfaces
- Supervised learning
- Underlying automata
- Q-learning
  - Dynamic discount
  - Watkins Q(lambda)
- Reinforce
- Memory
- Gradient checking
- Variance reduction

http://arxiv.org/pdf/1505.00521.pdf

http://arxiv.org/abs/1511.07275

code: https://github.com/ilyasu123/rlntm

https://github.com/wojzaremba/algorithm-learning