Recurrent Neural Networks

Zhirong Wu
Jan 9th 2015

Recurrent Neural Networks

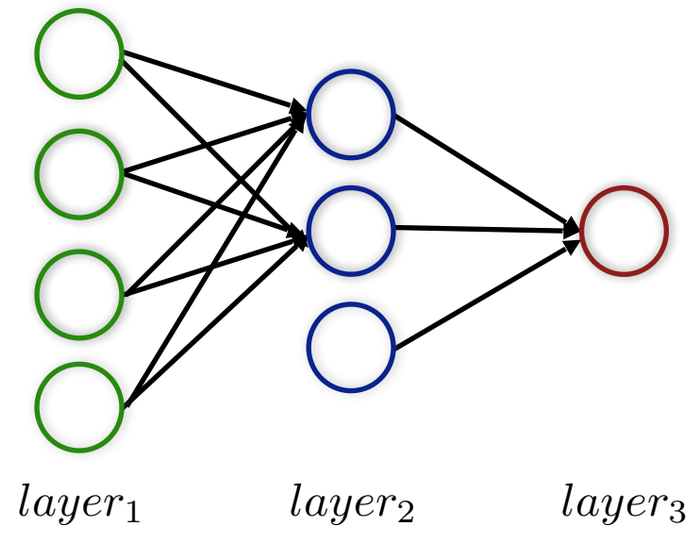
Wikipedia:

a recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle.

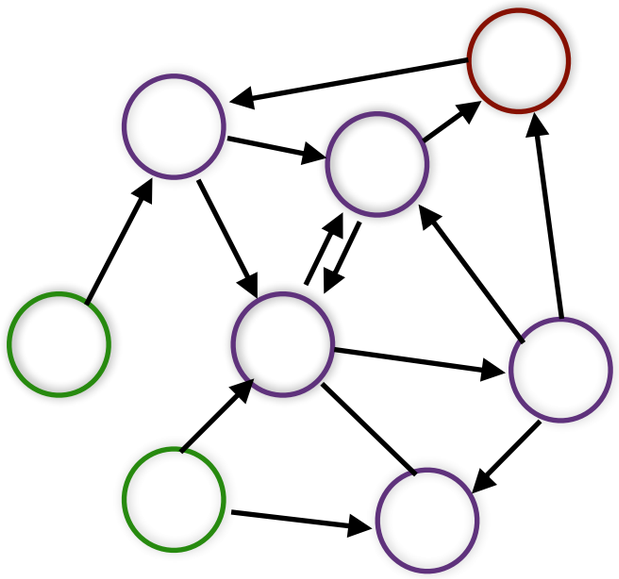
Recurrent neural networks constitute a broad class of machines with dynamic state; that is, they have state whose evolution depends both on the input to the system and on the current state

Artificial Neural Networks

- FeedForward
  - Model static input-output functions
  - exist a single forward direction
  - proven useful in many pattern classification tasks



- Recurrent
  - model dynamic state transition system
  - feedbacks connections included
  - practical difficulties for applications
  - can be useful for pattern classification,stochastic sequence modeling, associative memory

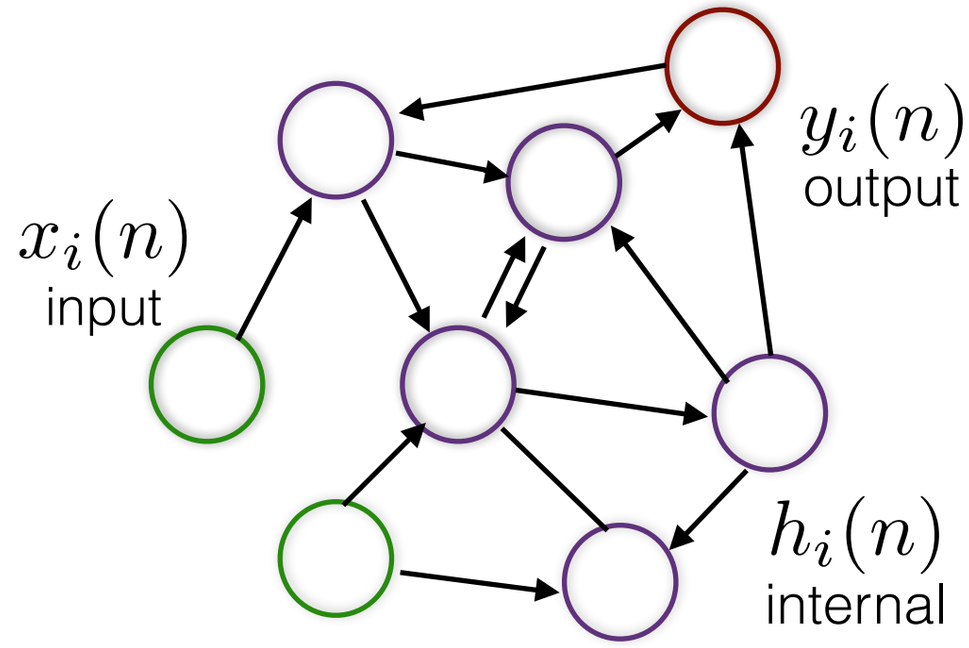


Recurrent Neural Networks

Computations (discrete time):

given the input sequence $x_i(n)$, the model updates its internal states and output value as follows:

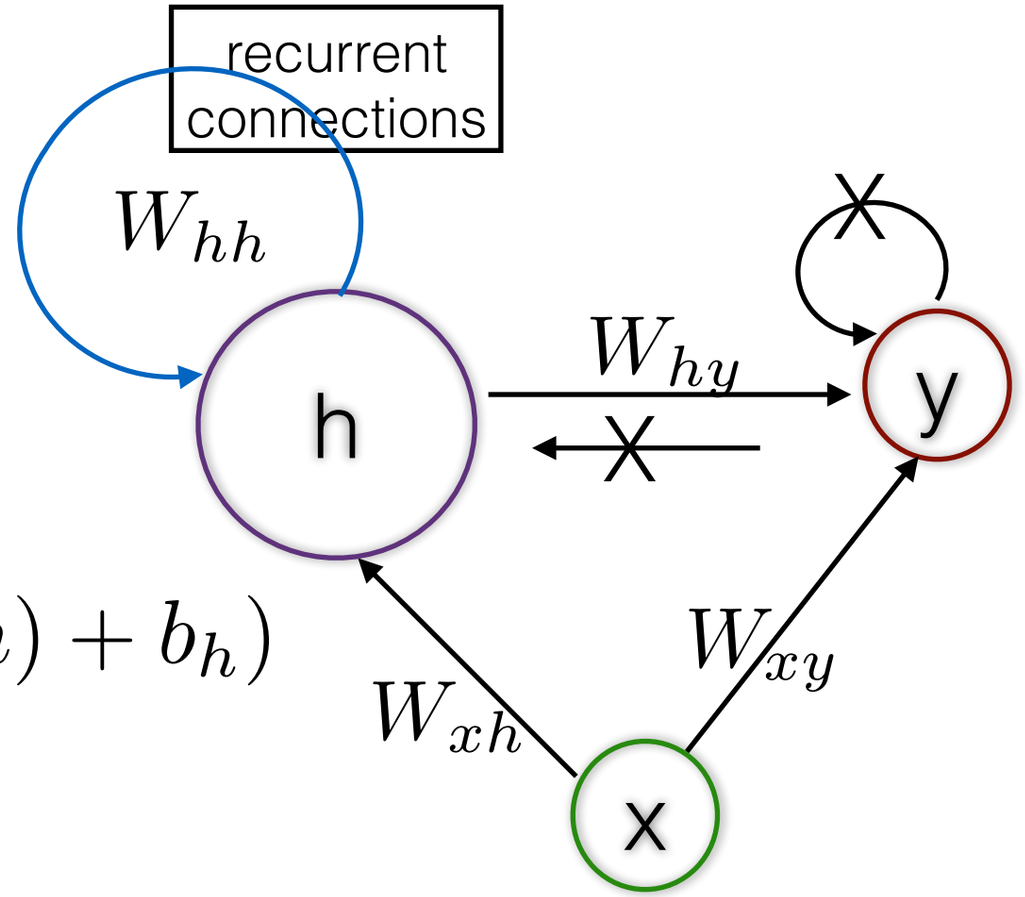
Computations (continuous time): describes in differential equations



$$h_i(n+1) = f(\text{weighted input})$$
$$= f(\sum_j w_j^{(in)} x_j(n+1) + \sum_j w_j h_j(n) + \sum_j w_j^{(out)} y_j(n))$$
$$y_i(n+1) = f(\text{weighted input})$$
$$= f(\sum_j w_j^{(in)} x_j(n+1) + \sum_j w_j h_j(n+1) + \sum_j w_j^{(out)} y_j(n))$$

Recurrent Neural Networks

A typical widely used formulation:



$$h(n+1) = f(W_{xh}x(n+1) + W_{hh}h(n) + b_h)$$

no feedbacks from output to internal units

$$y(n+1) = f(W_{xy}x(n+1) + W_{hy}h(n+1) + b_y)$$

no recurrent connections in the output units

Recurrent Neural Networks

Learning:

given m sequences of input $x^{(i)}(n)$ and output samples $d^{(i)}(n)$ of length T, minimise:

$$E = \frac{1}{2} \sum_{i=1}^{m} \sum_{n=1}^{T} ||d^{(i)}(n) - y^{(i)}(n)||^2$$

here $y^{(i)}(n)$ is the output from the RNN given the input $x^{(i)}(n)$

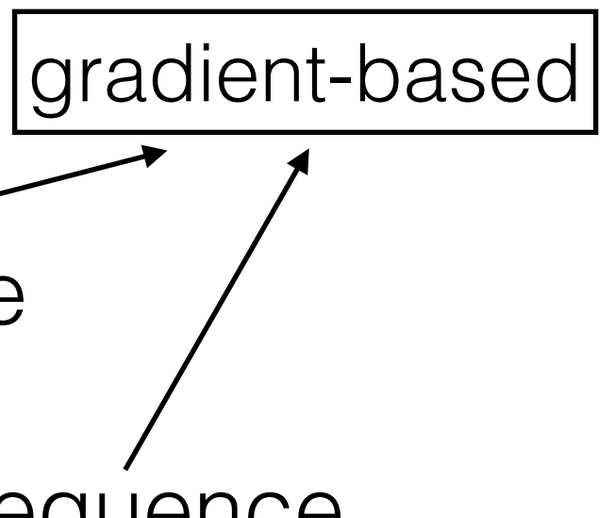
Backpropagation doesn't work with cycles.

How?

Learning Algorithms

3 popular methods:

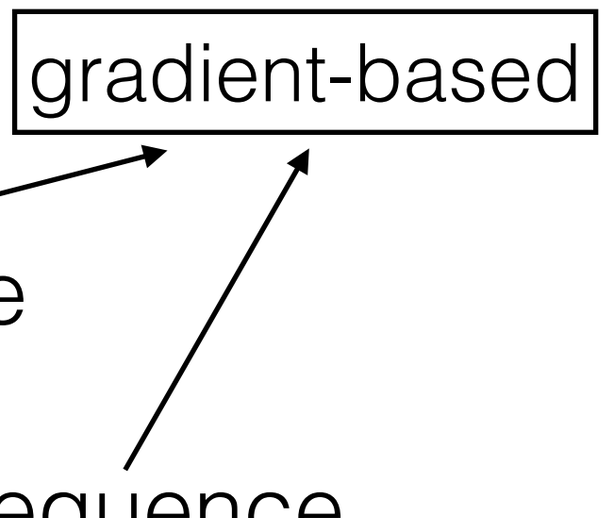
- BackPropagation Through Time
  - gradients update for a whole sequence
- Real Time Recurrent Learning
  - gradients update for each frame in a sequence
- Extended Kalman Filtering



Learning Algorithms

3 popular methods:

- BackPropagation Through Time
  - gradients update for a whole sequence
- Real Time Recurrent Learning
  - gradients update for each frame in a sequence
- Extended Kalman Filtering

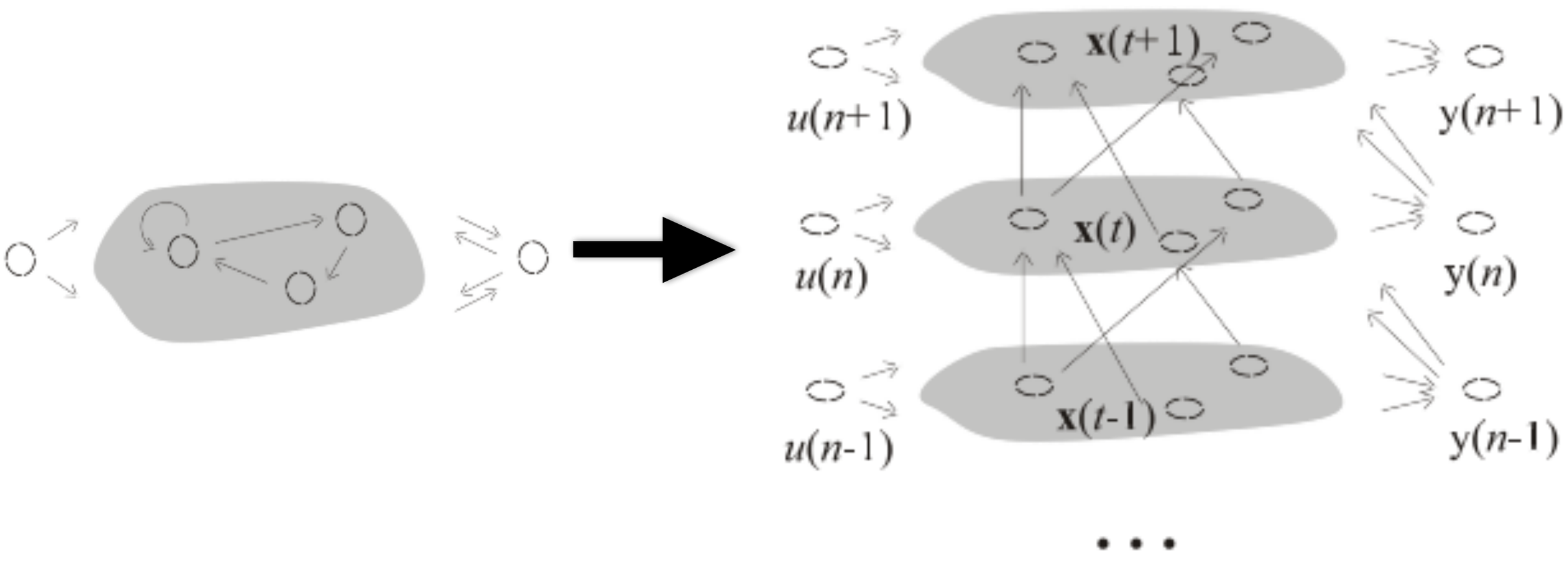


Learning Algorithms

Backpropagation Through Time (BPTT):

Williams, Ronald J., and David Zipser. "Gradient-based learning algorithms for recurrent networks and their computational complexity." Back-propagation: Theory, architectures and applications (1995): 433-486.

Given the whole sequence of inputs and outputs, unfold the RNN through time. The RNN thus becomes a standard feedforward neural network with each layer corresponding to a time step in the RNN. Gradients can be computed using standard backpropagation.



Learning Algorithms

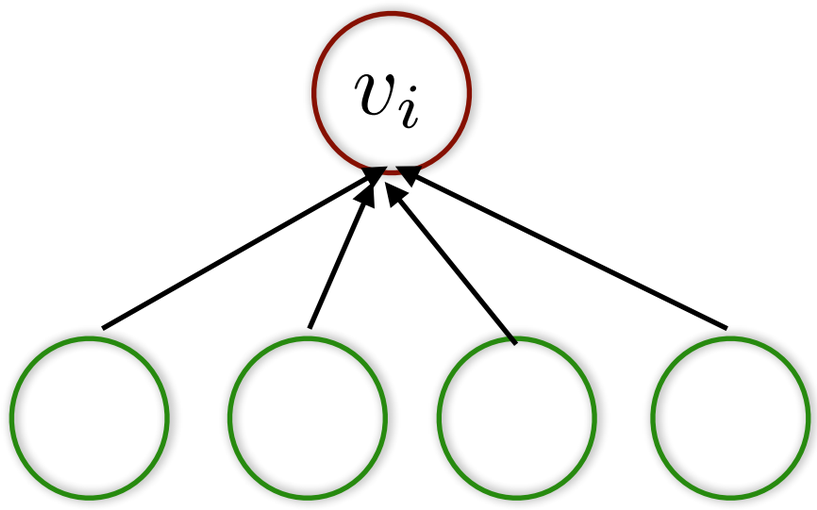
Real Time Recurrent Learning (RTRL):

Williams, Ronald J., and David Zipser. "A learning algorithm for continually running fully recurrent neural networks." Neural computation 1.2 (1989): 270-280.

For a particular frame in a sequence, get gradients of model parameters W.

Now think about a single unit v_i and its K input units $v_j, j = 1...K$

$$v_i(n+1) = f(\sum_{j=1}^{K} w_{ij} v_j(n))$$



Then differentiate it

Learning Algorithms

Real Time Recurrent Learning (RTRL):

differentiate this unit to a weight parameter w_{pq}

$$v_i(n+1) = f(\sum_{j=1}^{K} w_{ij} v_j(n))$$

we get,

$$\frac{\partial v_i(n+1)}{\partial w_{pq}} = f'(z_i) \left[(\sum_{j=1}^{K} w_{ij} \frac{\partial v_j(n)}{\partial w_{pq}}) + \delta_{ip} v_q(n) \right]$$

where $z_i = \sum_{j=1}^{K} w_{ij} v_j(n)$

$\delta_{ip} = 1, \text{if } i = p$

$\delta_{ip} = 0, \text{otherwise}$

Learning Algorithms

Real Time Recurrent Learning

compute gradients of time n+1 using gradients and activations of time n.

$$\frac{\partial v_i(n+1)}{\partial w_{pq}} = f'(z_i)\left[(\sum_{j=1}^{K} w_{ij}\frac{\partial v_j(n)}{\partial w_{pq}}) + \delta_{ip}v_q(n)\right]$$

initialize time 0, $\frac{\partial v_j(0)}{\partial w_{pq}} = 0, j = 1...K$

accumulate gradients from $n = 0 \rightarrow T_0$

$$\frac{\partial E}{\partial w_{pq}} = \sum_{i=1}^{L}\frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial w_{pq}}$$

finally, gradient descent.

Learning Algorithms

update rule:

1. batch update. Use m full sequences.
2. pattern update. Use one full sequence.
3. online update. Use each time step in a sequence.

(only applied to RTRL)

BPTT is a lot time efficient than RTRL, and it is most widely used for RNN training.

Learning Algorithms

Vanishing and Exploding Gradients:

for detailed analysis, see

Hochreiter, Sepp, and Jürgen Schmidhuber.
"Long short-term memory." Neural computation

Intuition:
the recurrent connection W_{hh} is applied on the error signal recursively backward through time.

- If the eigenvalue is bigger than 1, the gradients tend to explode. Learning will never converge.

- If the eigenvalue is smaller than 1, the gradients tend to vanish. Error signals can only affect small time lags leading to a short-term memory.

Long Short-Term Memory

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

breakthrough paper for RNN

LSTM architecture fixes the vanishing gradient problem,
it allows the error signal to affect steps long time before by introducing gates.

original recurrent update:

$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

LSTM replaces the function f by a memory cell:

input gate $$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)$$

forget gate $$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$

memory core $$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)$$

output gate $$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)$$

output $$h_t = o_t \tanh(c_t)$$

Long Short-Term Memory

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

breakthrough paper for RNN

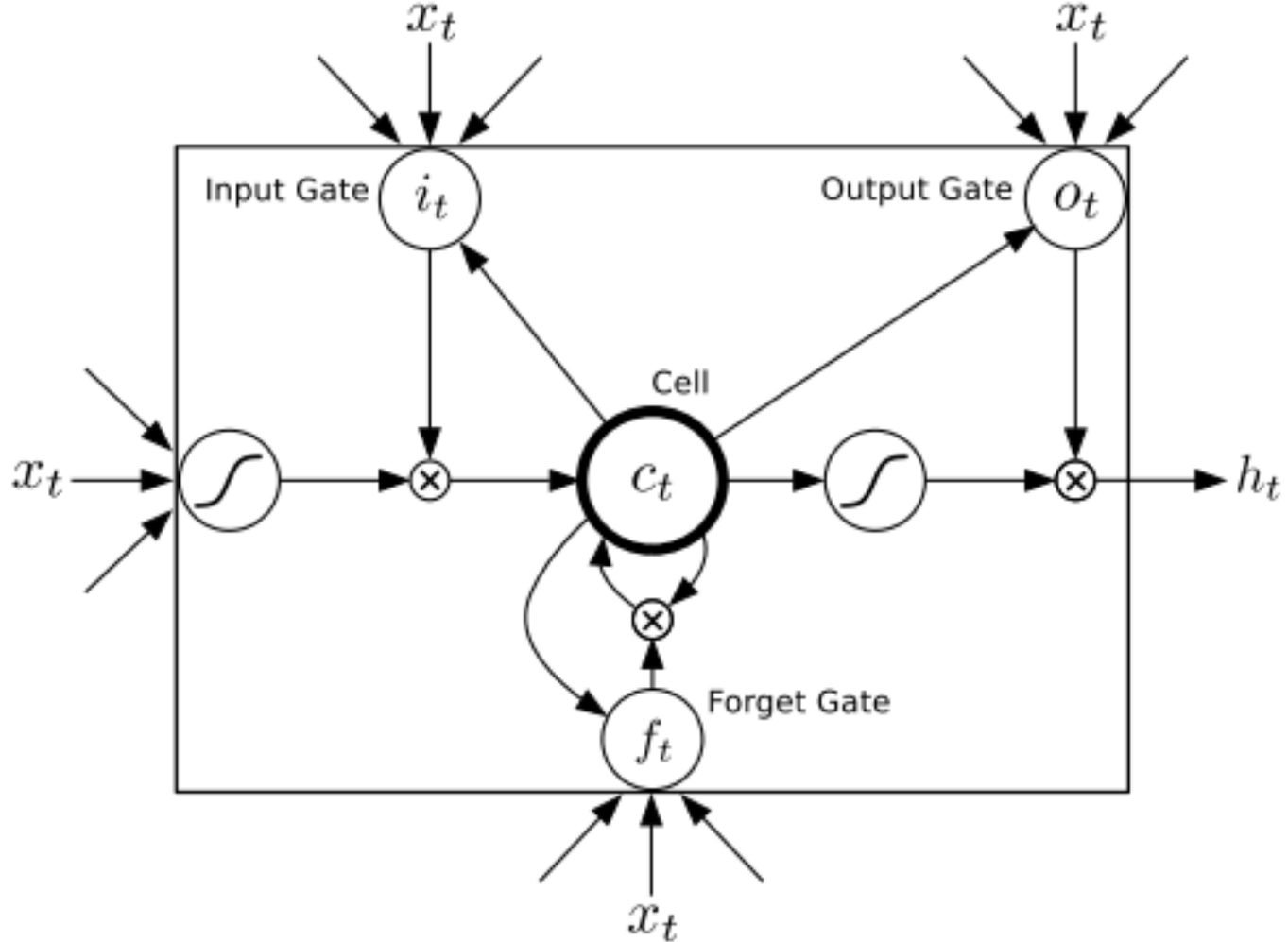
input gate $$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)$$

forget gate $$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$

memory core $$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)$$

output gate $$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)$$

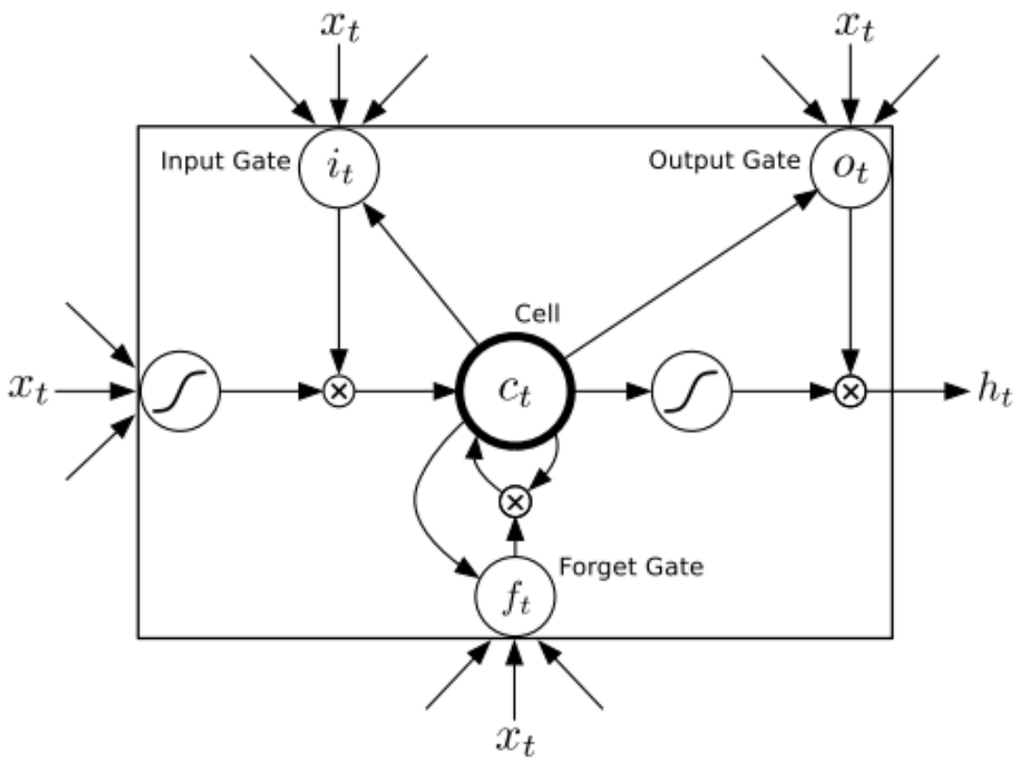
output $$h_t = o_t \tanh(c_t)$$



Long Short-Term Memory

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

breakthrough paper for RNN



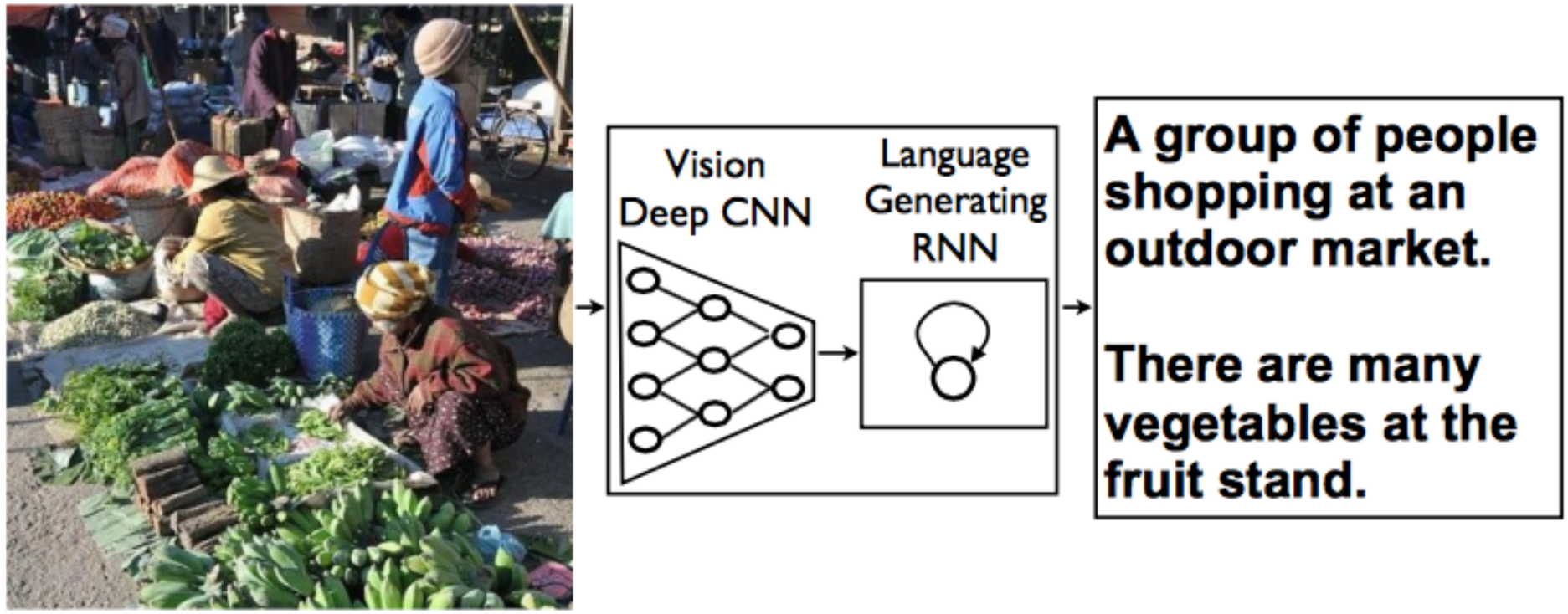
- input gate controls when the memory gets perturbed.
- output gate controls when the memory perturbs others.
- forget gate controls when the memory gets flushed.

- during forward computation, information trapped in the memory could directly interact with inputs long time afterwards.
- likewise, during backpropagation, error signals trapped in the memory could directly propagate to units long time before.
- LSTM retains information over 1000 time steps than 10.
- the parameters are learned altogether. Learning is a bit complicated using a combination of BPTT and RTRL to deal with gates.

Application: image caption generator

Vinyals, Oriol, et al. "Show and tell: A neural image caption generator." arXiv preprint arXiv:1411.4555 (2014).

- Given an image, output possible sentences to describe the image. The sentence could have varying length.
- Use CNN for image modelling and RNN for language generation.

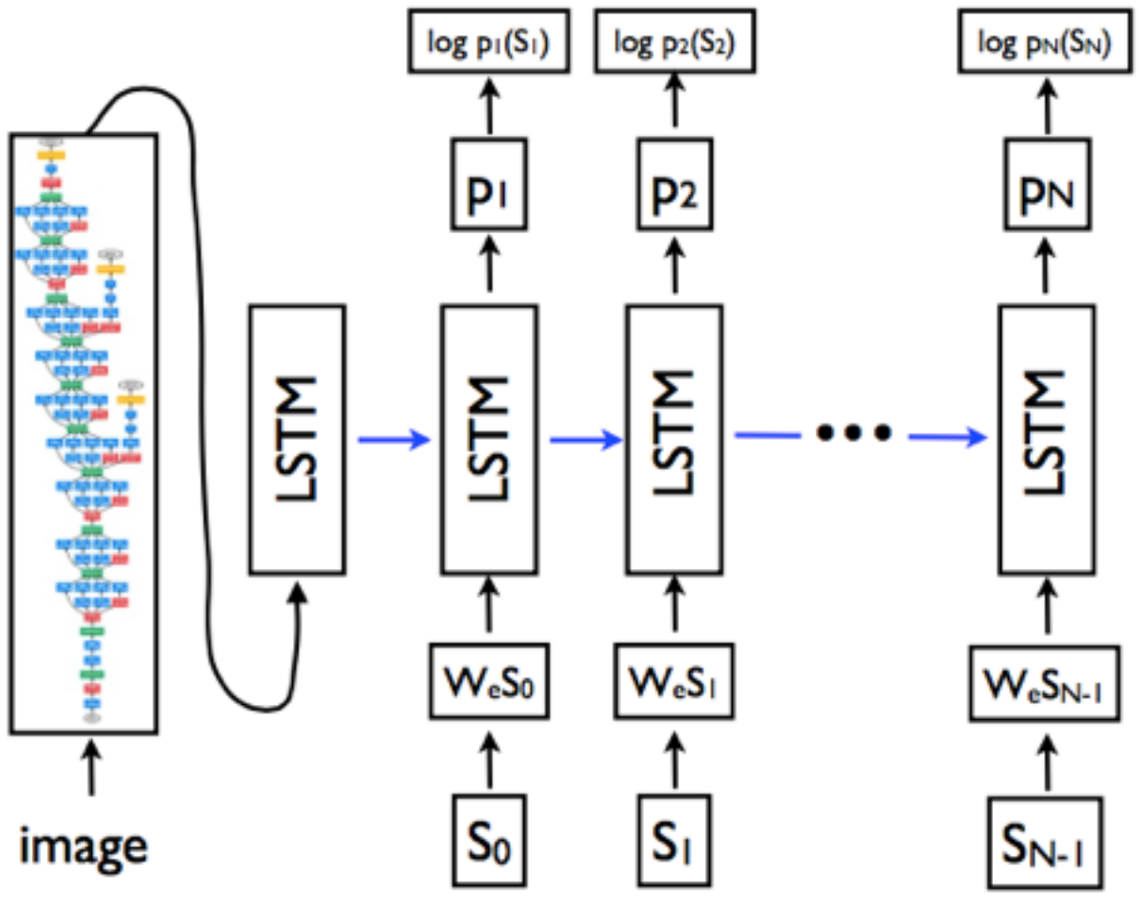


Application: image caption generator

Vinyals, Oriol, et al. "Show and tell: A neural image caption generator." arXiv preprint arXiv:1411.4555 (2014).

Training: BPTT $$log\ p(S|I) = \sum_{t=1}^{N} log\ p(S_t|I, S_0 \dots S_{t-1})$$

$$loss(S, I) = -\sum_{t=1}^{N} p_t(S_t)$$

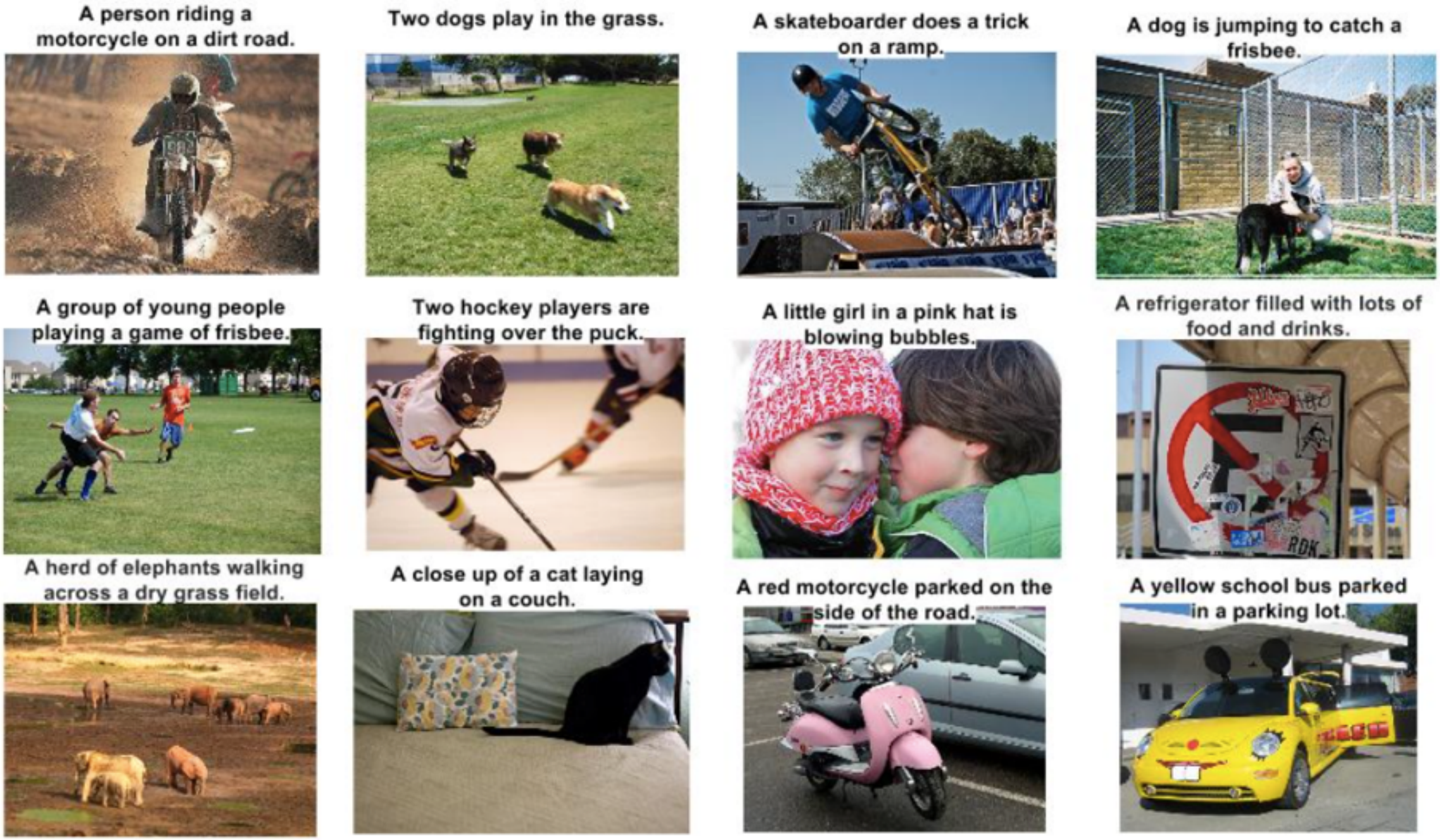


W_e
word embedding

Application: image caption generator

Vinyals, Oriol, et al. "Show and tell: A neural image caption generator." arXiv preprint arXiv:1411.4555 (2014).

results:



Application: visual attention

Mnih, Volodymyr, Nicolas Heess, and Alex Graves. "Recurrent models of visual attention."

in a reinforcement framework:

> The agent only has limited “bandwidth” to observe the environment. Each time, it makes some actions to maximize the total reward based on a history of observations.

environment: translated and cluttered MNIST
bandwidth: location and resolution
action: recognition prediction and moving action

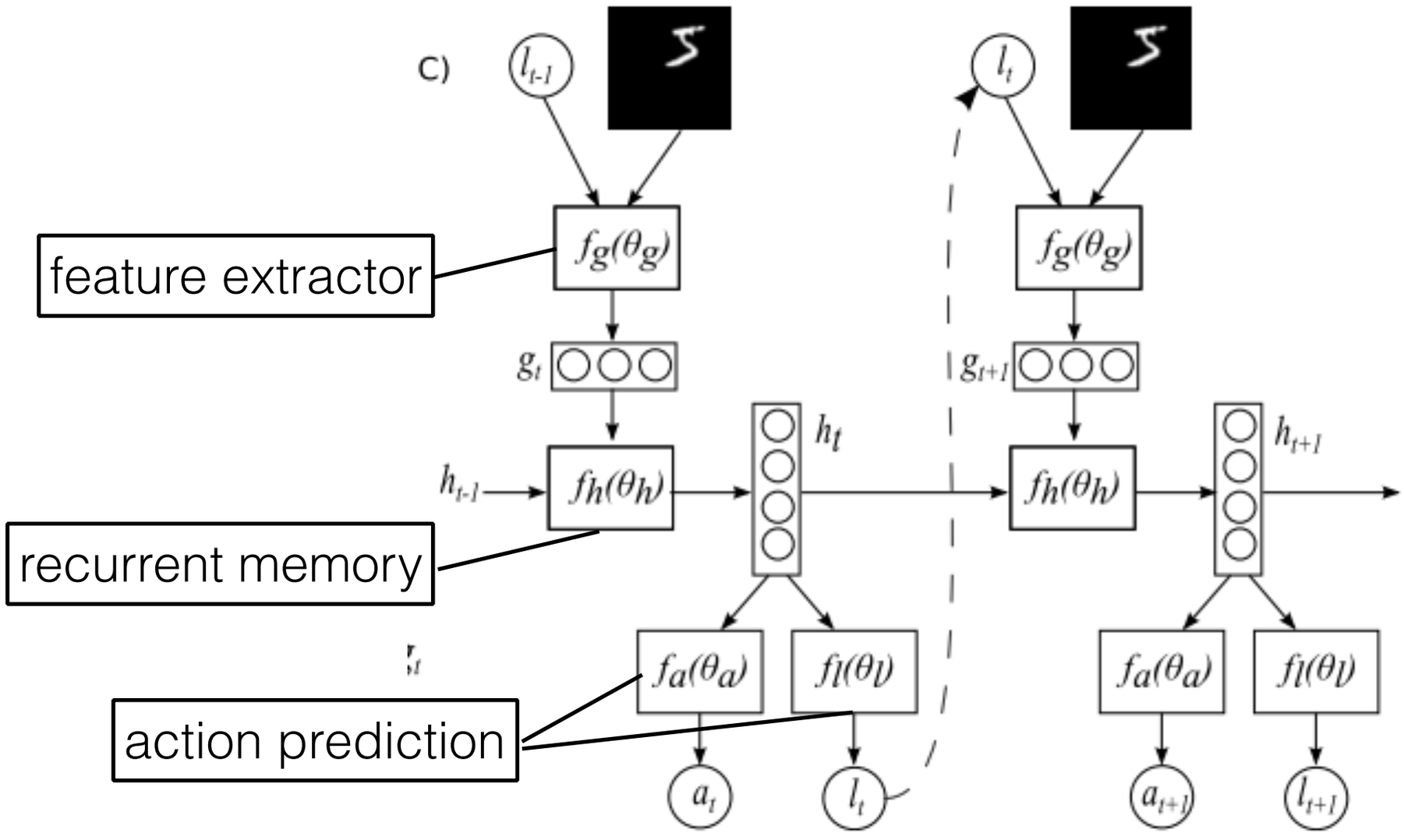
unlike CNN, this framework could recognize an image of arbitrary input size without scaling.

Application: visual attention

Mnih, Volodymyr, Nicolas Heess, and Alex Graves. "Recurrent models of visual attention."

Training:

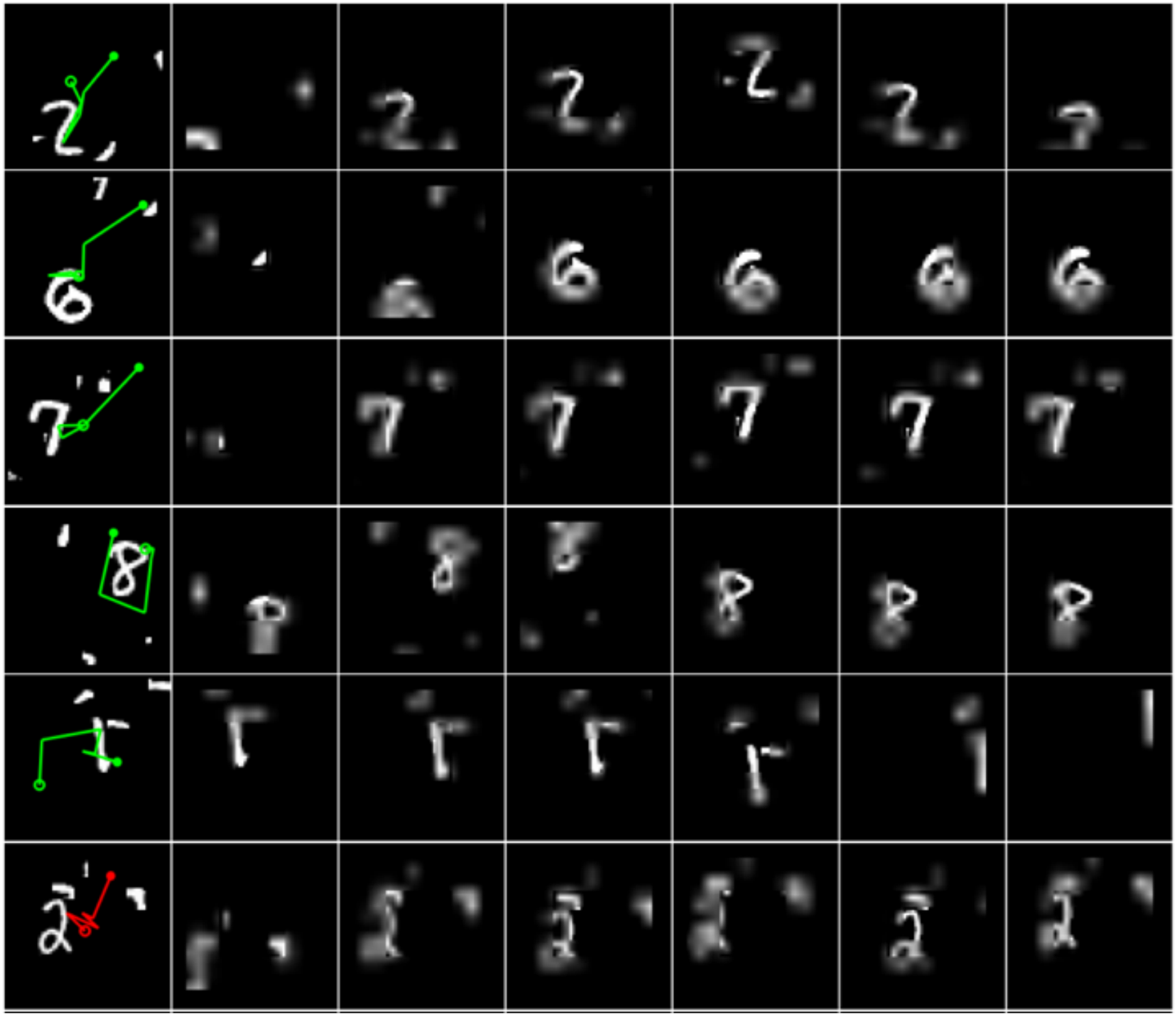
- the number of steps is fixed, say 6 steps.
- the reward is defined as the classification result of the last step.



Application: visual attention

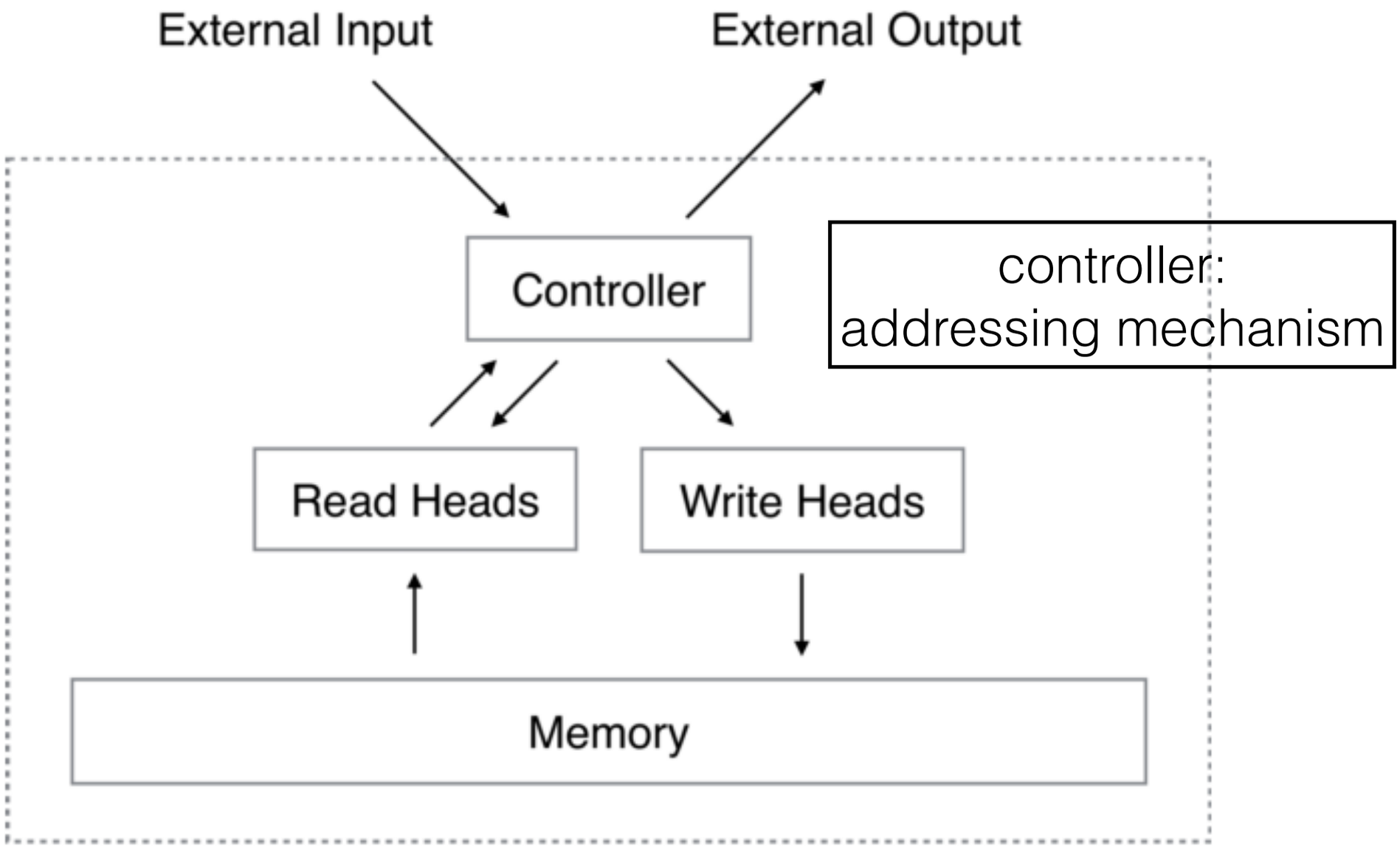
Mnih, Volodymyr, Nicolas Heess, and Alex Graves. "Recurrent models of visual attention."

results:



Another memory machine: neural turing machine

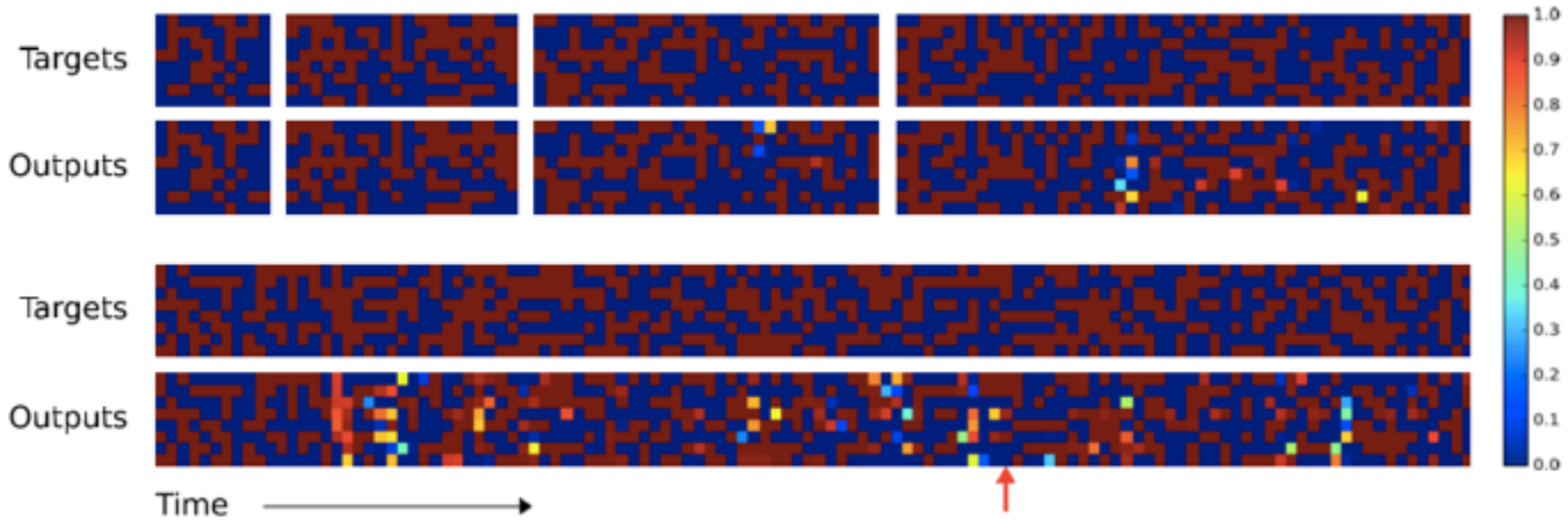
Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines." arXiv preprint arXiv:1410.5401 (2014).



Another memory machine: neural turing machine

Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines." arXiv preprint arXiv:1410.5401 (2014).

copy operation:



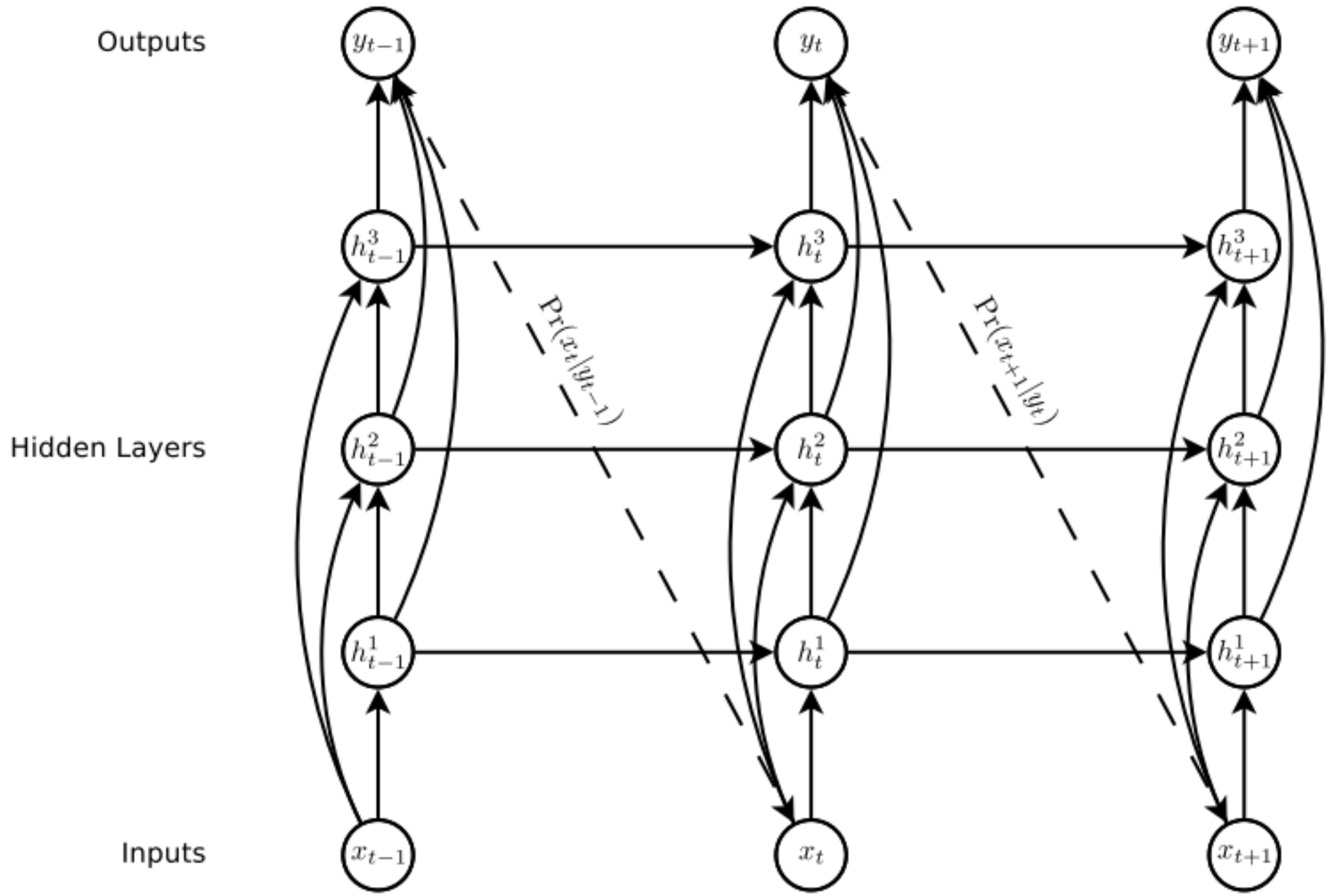
by analysing the interaction of the controller and the memory, the machine is able to learn “a copy algorithm”:

initialise: move head to start location
while input delimiter not seen **do**
  receive input vector
  write input to head location
  increment head location by 1
end while
return head to start location
while true **do**
  read output vector from head location
  emit output
  increment head location by 1
end while

Deep RNN

Graves, Alex. "Generating sequences with recurrent neural networks." arXiv preprint arXiv:1308.0850 (2013).

Instead of one single memory cell, we can actually stack several of them like feedforward networks.



Overview

1. What is Recurrent Neural Network
2. Learning algorithms of RNN
3. Vanishing Gradients and LSTM
4. Applications
5. Deep RNN

Code & Demo

http://www.cs.toronto.edu/~graves/

http://www.cs.toronto.edu/~graves/handwriting.html