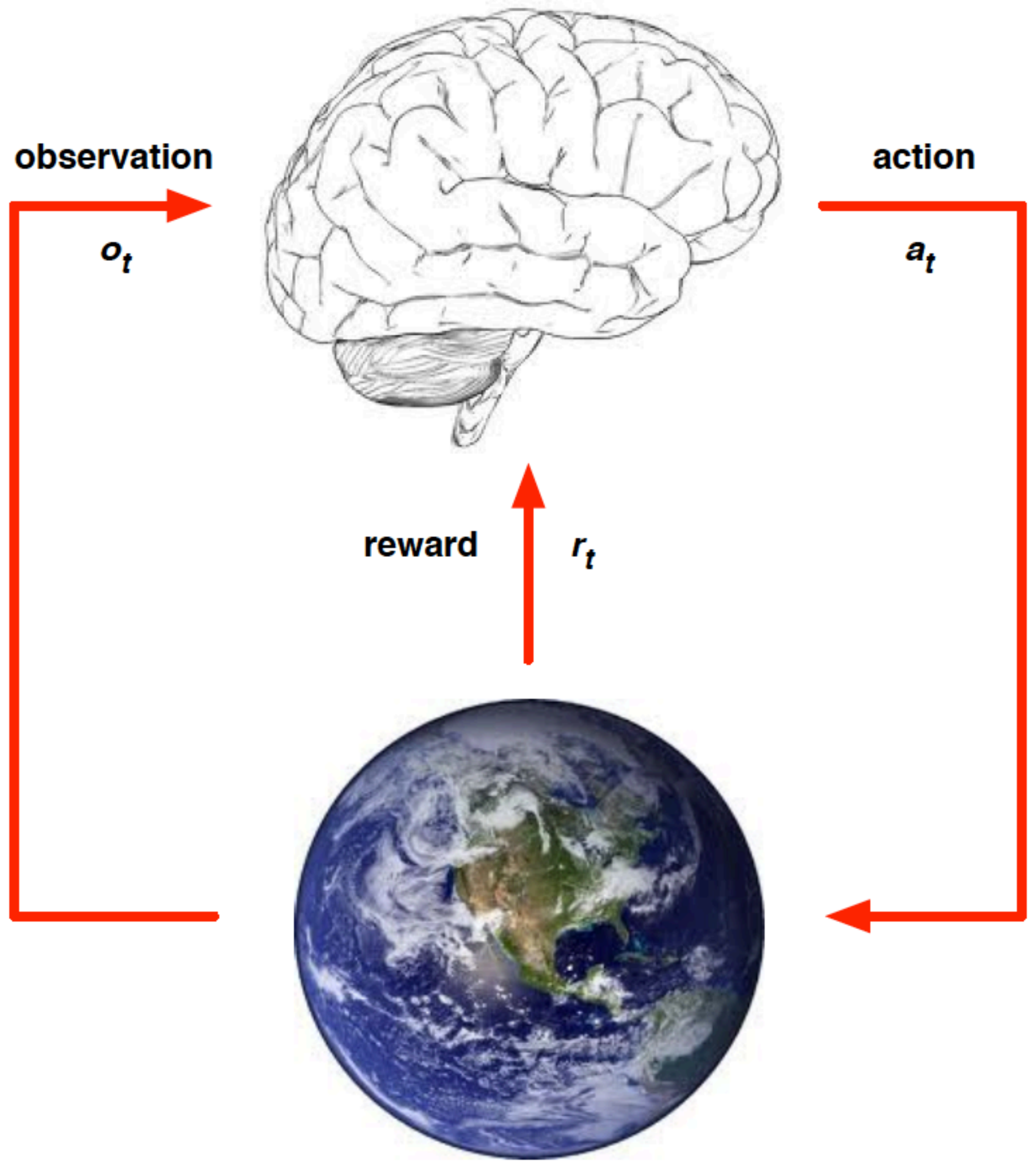
THE UNIVERSITY OF BRITISH COLUMBIA

Topics in AI (CPSC 532S):
Multimodal Learning with Vision, Language and Sound

Lecture 20: Deep Reinforcement Learning (cont)

How does **RL** work?



- At each step t the agent:
  - Executes action a_t
  - Receives observation o_t
  - Receives scalar reward r_t
- The environment:
  - Receives action a_t
  - Emits observation o_{t+1}
  - Emits scalar reward r_{t+1}

* slide from David Silver

Atari Games

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

Markov Decision Processes

— Mathematical **formulation** of the RL problem

Defined by:

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

Markov Decision Processes

At times step t=0, environment samples initial state

For time t=0 until done:

- Agent selects action
- Environment samples the reward
- Environment samples the next state
- Agent receives reward and next state

Markov Decision Processes

- Mathematical **formulation** of the RL problem

Defined by:

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

- Life is **trajectory**: $\ldots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, \ldots$

Markov Decision Processes

— Mathematical **formulation** of the RL problem

Defined by:

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

— Life is **trajectory**: $\ldots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, \ldots$

— **Markov property**: Current state completely characterizes the state of the world

$$p(r, s'|s, a) = Prob\left[R_{t+1} = r, S_{t+1} = s' \,\middle|\, S_t = s, A_t = a\right]$$

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Components of the RL Agent

Policy

— How does the agent behave?

Value Function

— How good is each state and/or action pair?

Model

— Agent's representation of the environment

* slide from Dhruv Batra

Policy

— The policy is how the agent acts

— Formally, map from states to actions:

Deterministic policy: $a = \pi(s)$

Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

* slide from Dhruv Batra

Policy

- The policy is how the agent acts
- Formally, map from states to actions:

  Deterministic policy: $a = \pi(s)$

  Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

e.g.

State	Action
A $\longrightarrow$	2
B $\longrightarrow$	1

* slide from Dhruv Batra

The **Optimal** Policy

What is a good policy?

* slide from Dhruv Batra

The **Optimal** Policy

What is a good policy?

Maximizes current reward? Sum of all future rewards?

* slide from Dhruv Batra

The **Optimal** Policy

What is a good policy?

Maximizes current reward? Sum of all future rewards?

Discounted future rewards!

The **Optimal** Policy

What is a good policy?

Maximizes current reward? Sum of all future rewards?

Discounted future rewards!

Formally: $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi\right]$

with $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$

* slide from Dhruv Batra

Components of the RL Agent

✓ **Policy**

— How does the agent behave?

Value Function

— How good is each state and/or action pair?

Model

— Agent's representation of the environment

* slide from Dhruv Batra

Value Function

A value function is a prediction of future reward

“State Value Function" or simply “**Value Function**”

- How good is a state?
- Am I screwed? Am I winning this game?

“Action Value Function” or **Q-function**

- How good is a state action-pair?
- Should I do this now?

* slide from Dhruv Batra

Value Function and **Q-value** Function

Following a policy produces sample trajectories (or paths) s_0, a_0, r_0, s_1, a_1, r_1, …

— The **value function** (how good is the state) at state s, is the expected cumulative reward from state s (and following the policy thereafter):

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

Value Function and **Q-value** Function

Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

— The **value function** (how good is the state) at state s, is the expected cumulative reward from state s (and following the policy thereafter):

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

— The **Q-value function** (how good is a state-action pair) at state s and action a, is the expected cumulative reward from taking action a in state s (and following the policy thereafter):

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Components of the RL Agent

✓ **Policy**

— How does the agent behave?

✓ **Value Function**

— How good is each state and/or action pair?

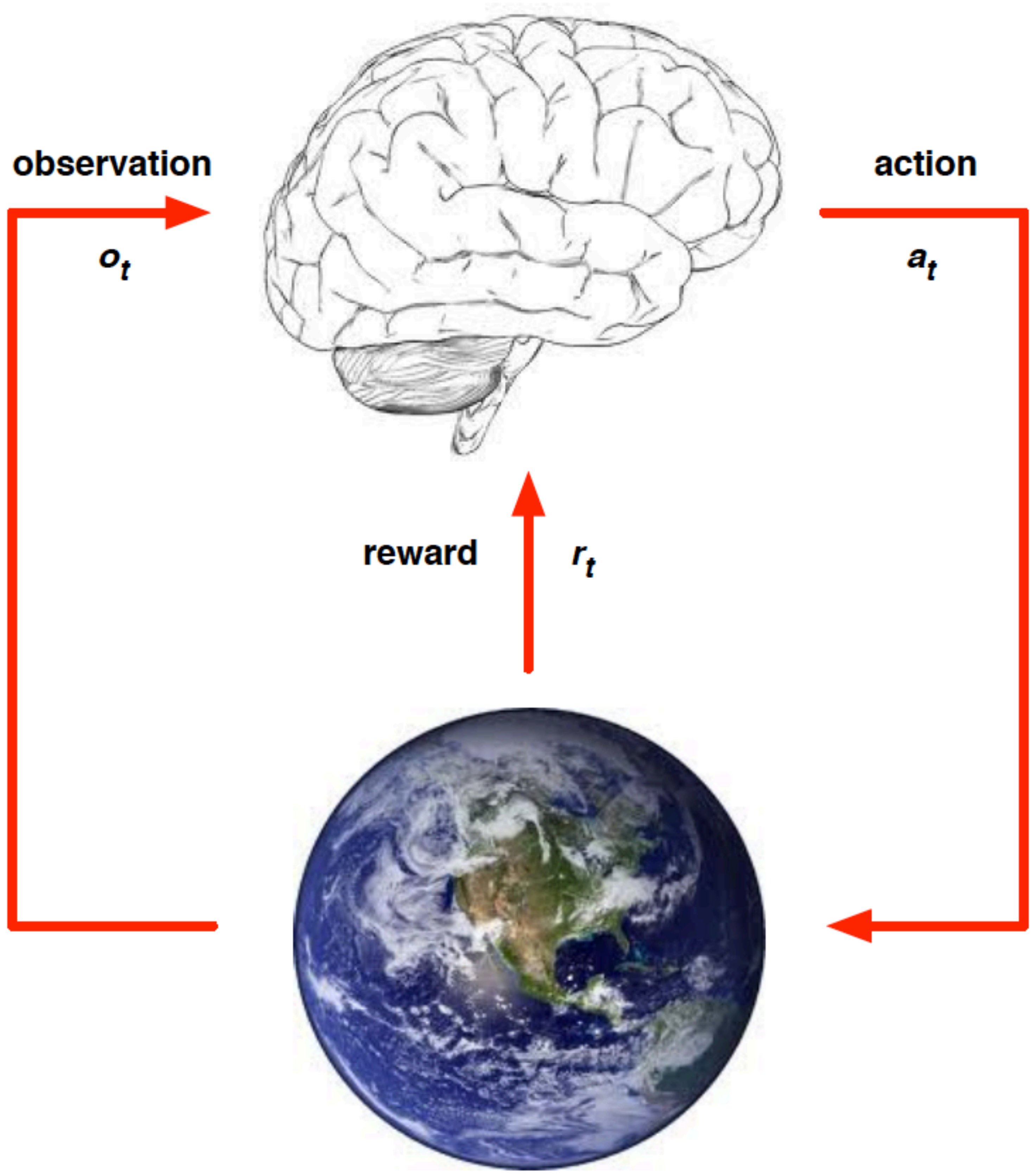
Model

— Agent’s representation of the environment

* slide from Dhruv Batra

Model

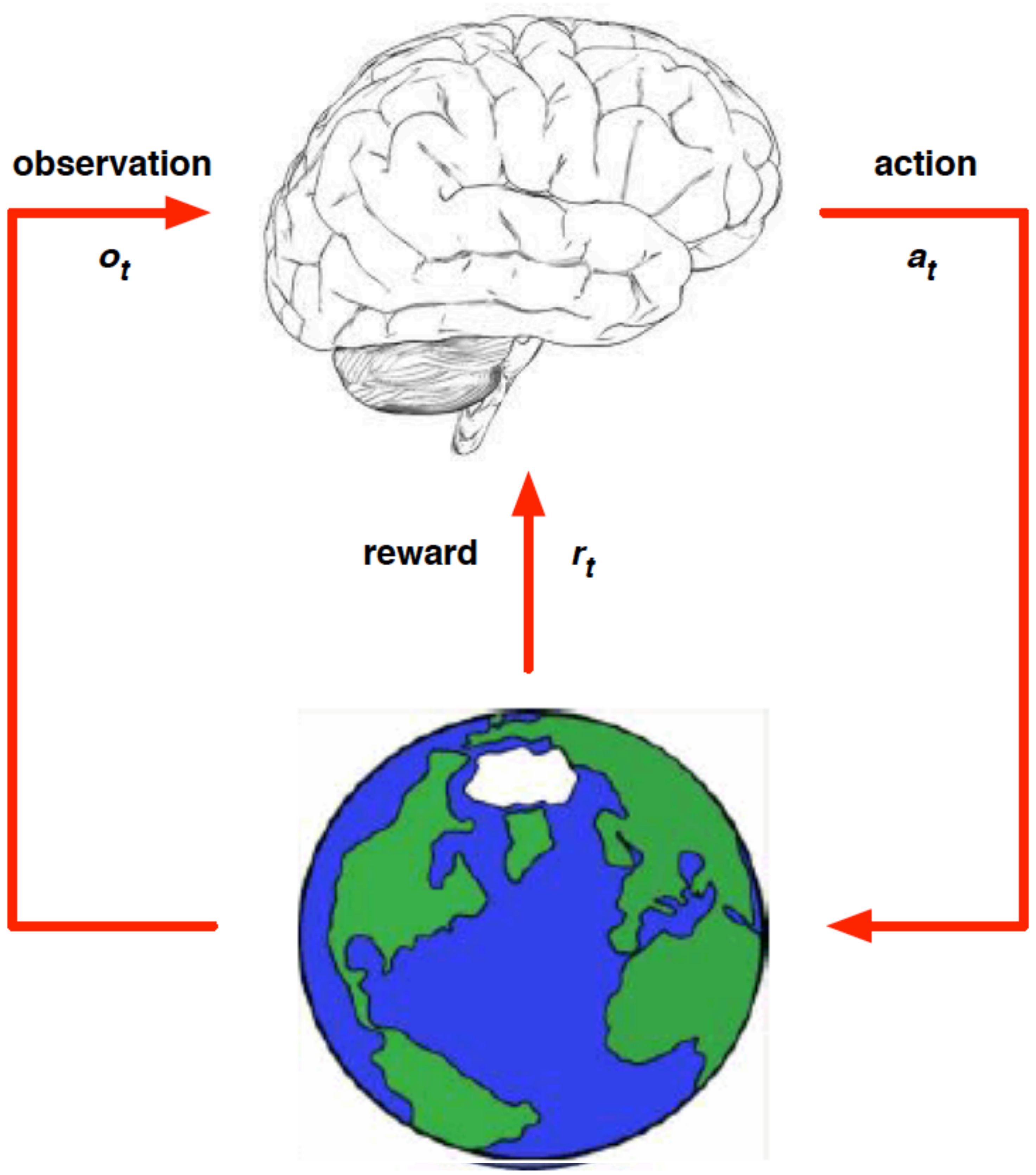
Model predicts what the world will do next



* slide from David Silver

Model

Model predicts what the world will do next



* slide from David Silver

Components of the RL Agent

✓ **Policy**

— How does the agent behave?

✓ **Value Function**

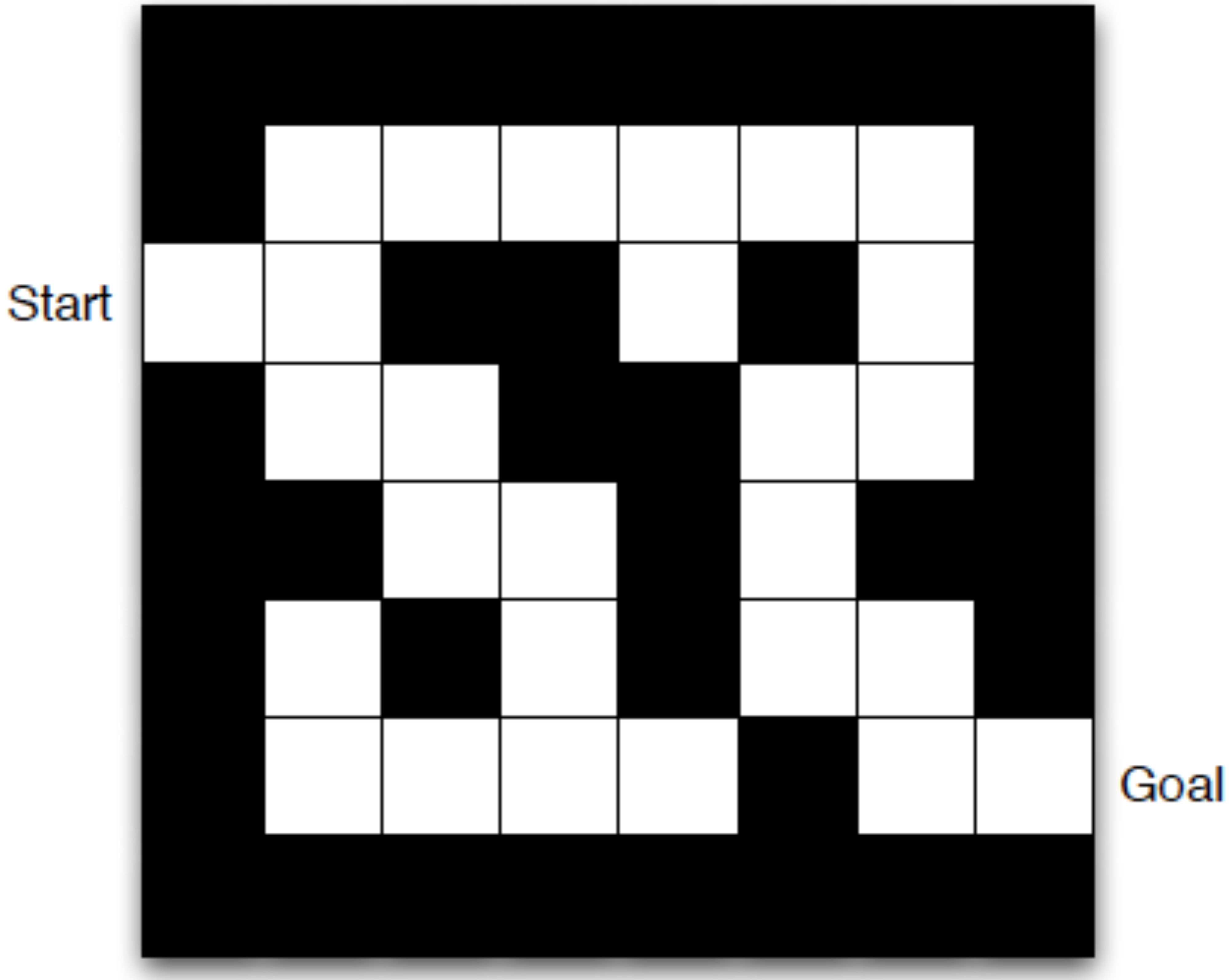
— How good is each state and/or action pair?

✓ **Model**

— Agent’s representation of the environment

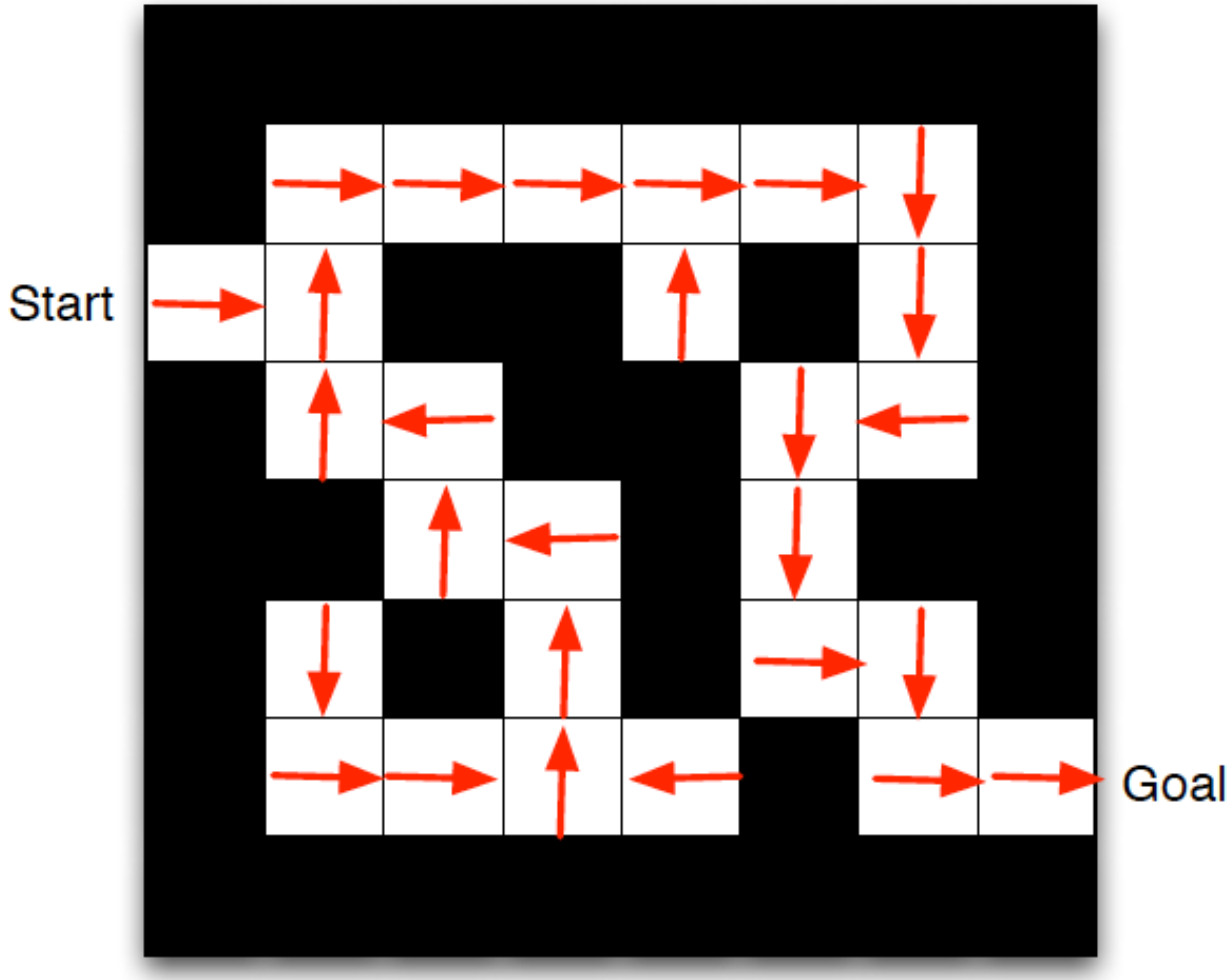
* slide from Dhruv Batra

Maze Example



Reward: -1 per time-step
Actions: N, E, S, W
States: Agent's location

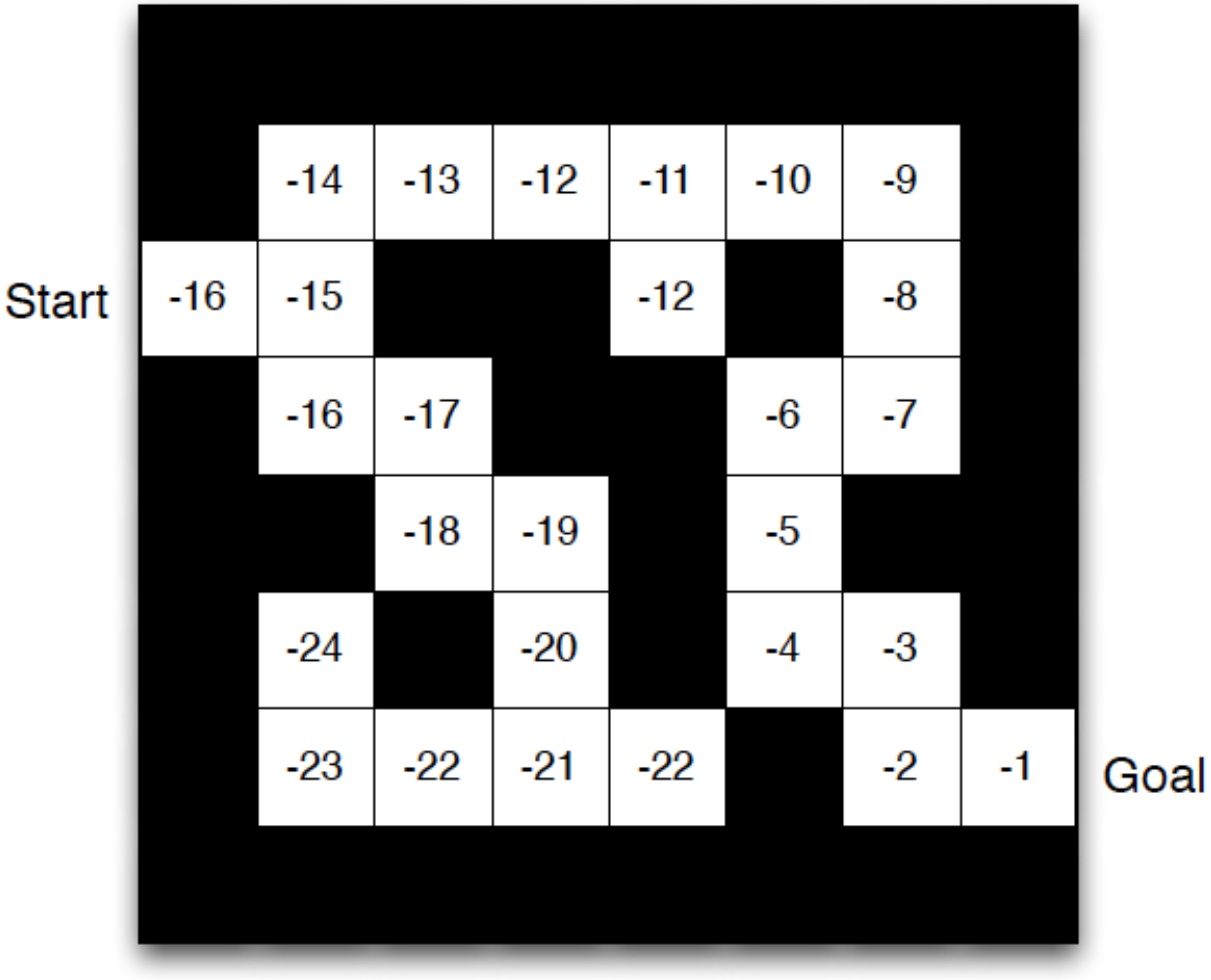
Maze Example: Policy



Arrows represent a policy $\pi(s)$ for each state s

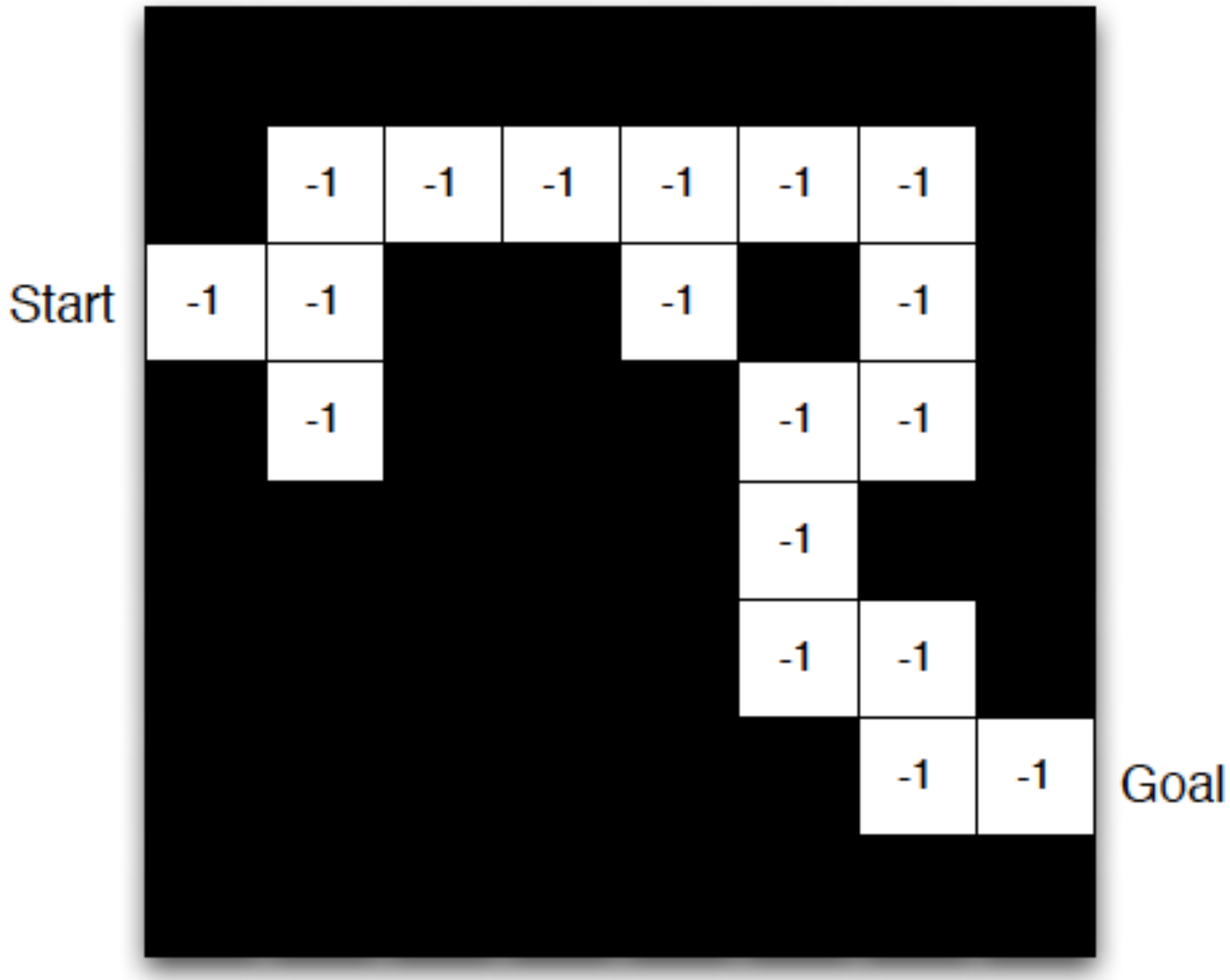
* slide from David Silver

Maze Example: Value



Numbers represent value $v_\pi(s)$ of each state s

Maze Example: Model



Grid layout represents transition model

Numbers represent the immediate reward for each state (same for all states)

* slide from David Silver

Components of the RL Agent

Policy

— How does the agent behave?

Value Function

— How good is each state and/or action pair?

Model

— Agent's representation of the environment

* slide from Dhruv Batra

Approaches to RL: Taxonomy

Model-free RL

Value-based RL

- — Estimate the optimal action-value function $Q^*(s,a)$
- — No policy (implicit)

Policy-based RL

- — Search directly for the optima policy π^*
- — No value function

Model-based RL

— Build a model of the world

— Plan (e.g., by look-ahead) using model

* slide from Dhruv Batra

Approaches to RL: Taxonomy

Model-free RL

Value-based RL

- — Estimate the optimal action-value function $Q^*(s, a)$
- — No policy (implicit)

Policy-based RL

- — Search directly for the optima policy π^*
- — No value function

Actor-critic RL

- — Value function
- — Policy function

Model-based RL

— Build a model of the world

— Plan (e.g., by look-ahead) using model

* slide from Dhruv Batra

Deep RL

Value-based RL

— Use neural nets to represent Q function $Q(s, a; \theta)$

$$Q(s, a; \theta^*) \approx Q^*(s, a)$$

Policy-based RL

— Use neural nets to represent the policy π_θ

$$\pi_{\theta^*} \approx \pi^*$$

Model-based RL

— Use neural nets to represent and learn the model

* slide from Dhruv Batra

Approaches to RL

Value-based RL

- Estimate the optimal action-value function $Q^*(s, a)$
- No policy (implicit)

Optimal Value Function

Optimal Q-function is the maximum achievable value

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

* slide from David Silver

Optimal Value Function

Optimal Q-function is the maximum achievable value

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

Once we have it, we can act optimally

$$\pi^*(s) = \operatorname*{argmax}_{a} Q^*(s,a)$$

Optimal Value Function

Optimal Q-function is the maximum achievable value

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) = Q^{\pi^*}(s,a)$$

Once we have it, we can act optimally

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

Optimal value maximizes over all future decisions

$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

* slide from David Silver

Optimal Value Function

Optimal Q-function is the maximum achievable value

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) = Q^{\pi^*}(s,a)$$

Once we have it, we can act optimally

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q^*(s,a)$$

Optimal value maximizes over all future decisions

$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

Formally, Q* satisfied Bellman Equations

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]$$

* slide from David Silver

Solving for the Optimal Policy

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a')|s,a\right]$$

Q_i will converge to Q^* as i -> infinity

Solving for the Optimal Policy

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a')|s,a\right]$$

Q_i will converge to Q^* as i -> infinity

What's the problem with this?

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Solving for the Optimal Policy

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a')|s,a\right]$$

Q_i will converge to Q^* as i -> infinity

What's the problem with this?

Not scalable. Must compute Q(s,a) for every state-action pair. If state is e.g. game pixels, computationally infeasible to compute for entire state space!

Solving for the Optimal Policy

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a')|s,a\right]$$

Q_i will converge to Q^* as i -> infinity

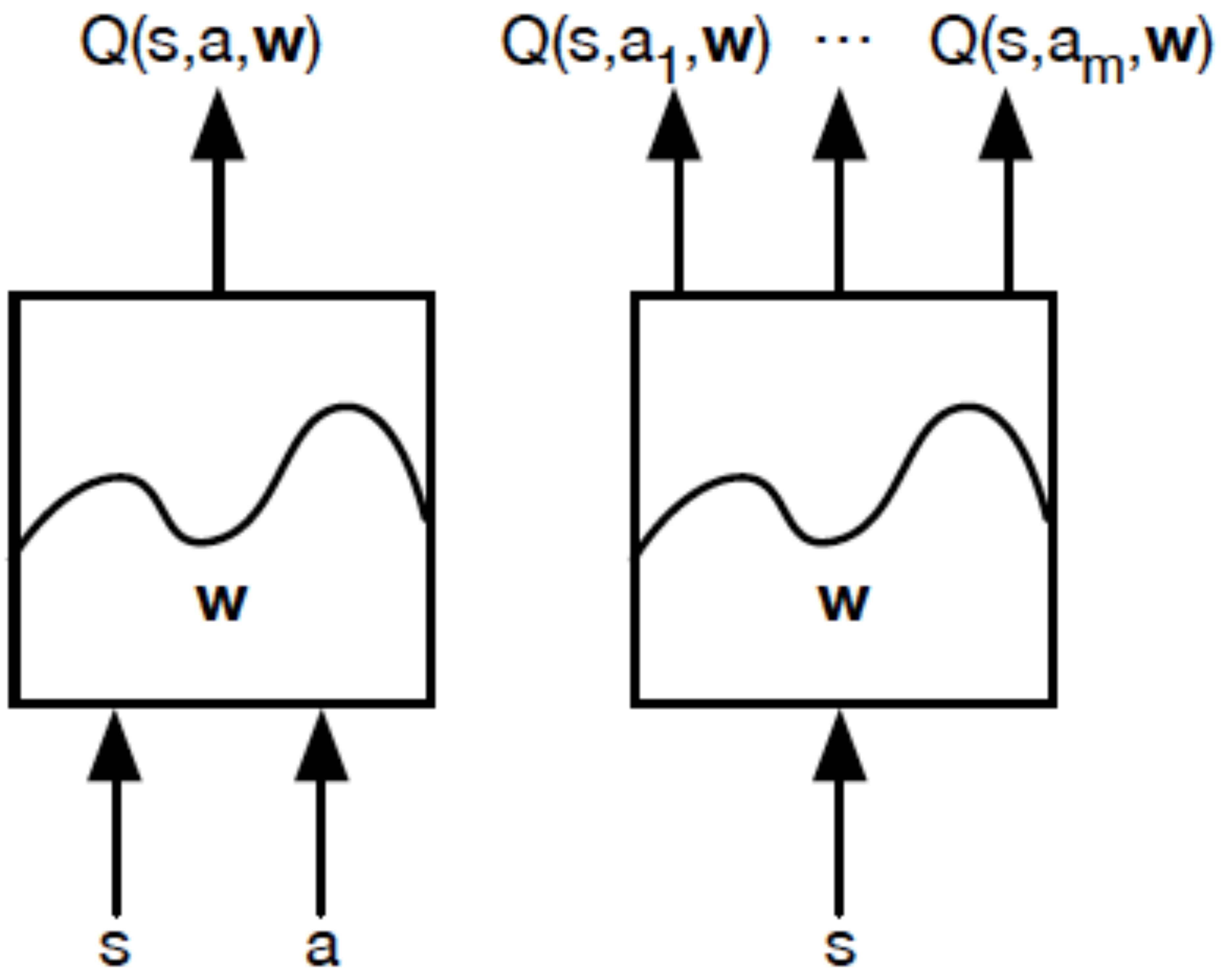
What's the problem with this?

Not scalable. Must compute Q(s,a) for every state-action pair. If state is e.g. game pixels, computationally infeasible to compute for entire state space!

Solution: use a function approximator to estimate Q(s,a). E.g. a neural network!

Q-Networks

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$



* slide from David Silver

Case Study: Playing **Atari** Games

[Mnih *et al.*, 2013; Nature 2015]

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

Q-Network Architecture

[Mnih *et al.*, 2013; Nature 2015]

$Q(s, a; \theta)$: neural network with weights θ

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

Current state s_t: 84x84x4 stack of last 4 frames (after RGB->grayscale conversion, downsampling, and cropping)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Q-Network Architecture

[Mnih *et al.*, 2013; Nature 2015]

$Q(s, a; \theta)$: neural network with weights θ

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

← **Input**: state s_t

Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Q-Network Architecture

[Mnih *et al.*, 2013; Nature 2015]

$Q(s, a; \theta)$: neural network with weights θ

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

← familiar conv and fc layers

Current state s_t: 84x84x4 stack of last 4 frames (after RGB->grayscale conversion, downsampling, and cropping)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Q-Network Architecture

[Mnih *et al.*, 2013; Nature 2015]

$Q(s, a; \theta)$: neural network with weights θ

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$

Current state s_t: 84x84x4 stack of last 4 frames (after RGB->grayscale conversion, downsampling, and cropping)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Q-Network Architecture

[Mnih *et al.*, 2013; Nature 2015]

$Q(s, a; \theta)$: neural network with weights θ

FC-4 (Q-values) ⟵ Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

Number of actions between 4-18 depending on Atari game

Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

Q-Network Architecture

[Mnih *et al.*, 2013; Nature 2015]

$Q(s, a; \theta)$: neural network with weights θ

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$

A single feedforward pass to compute Q-values for all actions from the current state => efficient!

Number of actions between 4-18 depending on Atari game

Current state s_t: 84x84x4 stack of last 4 frames (after RGB->grayscale conversion, downsampling, and cropping)

Q-Network Learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a]$$

Q-Network Learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a]$$

Forward Pass:

Loss function: $L_i(\theta_i) = \mathbb{E}\left[(y_i - Q(s,a;\theta_i)^2\right]$

where $y_i = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a]$

Q-Network Learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a]$$

Forward Pass:

Loss function: $L_i(\theta_i) = \mathbb{E}\left[(y_i - Q(s,a;\theta_i))^2\right]$

where $y_i = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a]$

Backward Pass:

Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

Q-Network Learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a]$$

Forward Pass:

Loss function: $$L_i(\theta_i) = \mathbb{E}\left[(y_i - Q(s,a;\theta_i)^2\right]$$

where $$y_i = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a]$$

Iteratively try to make the Q-value close to the target value (y_i) it should have, if Q-function corresponds to optimal Q* (and optimal policy **π***)

Backward Pass:

Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

Training the Q-Network: **Experience Replay**

Learning from **batches of consecutive samples is problematic**:

— Samples are correlated => inefficient learning

— Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

Address these problems using experience replay

— Continually update a replay memory table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played

— Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

Experience Replay

Experience Replay

To remove correlations, build data-set from agent's own experience

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\dots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$s_2, a_2, r_3, s_3 \rightarrow s, a, r, s'$

$s_t, a_t, r_{t+1}, s_{t+1} \rightarrow s_t, a_t, r_{t+1}, s_{t+1}$

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Putting it together: Deep Q-learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Putting it together: Deep Q-learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

Initialize replay memory, Q-network

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Putting it together: Deep Q-learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

Play M episodes (full games)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Putting it together: Deep Q-learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Initialize state (start geme screen pixes) at beggining of each episode

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Putting it together: Deep Q-learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

For each timestep T of the game
(T is max steps but can return early)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Putting it together: Deep Q-learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

With small probability take random action (explore)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Putting it together: Deep Q-learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Otherwise select greedy action from current policy (implicit in Q function)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Putting it together: Deep Q-learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Take action and observe the reward and next state

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Putting it together: Deep Q-learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Store transition replay in memory

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Putting it together: Deep Q-learning with Experience Replay

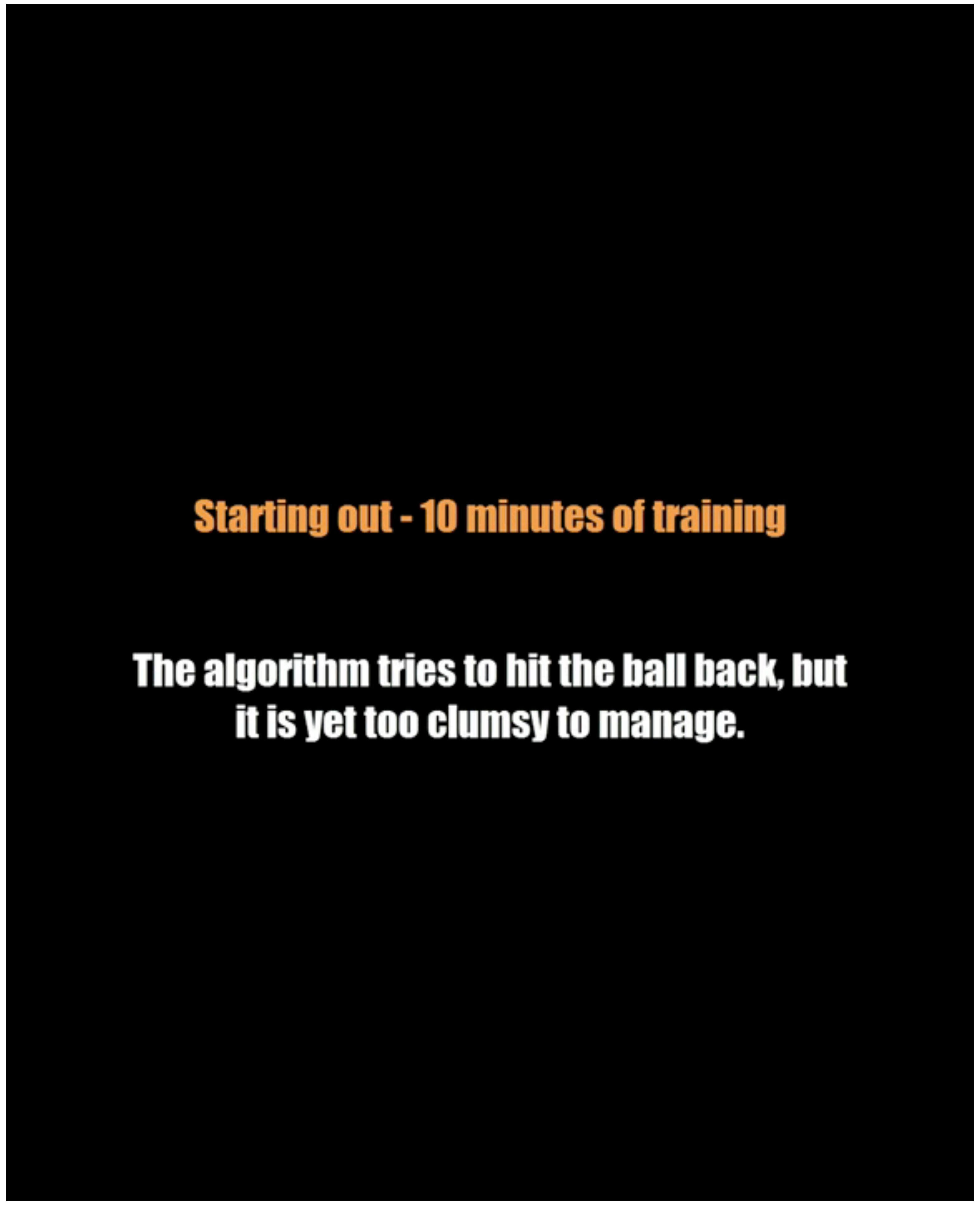
Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

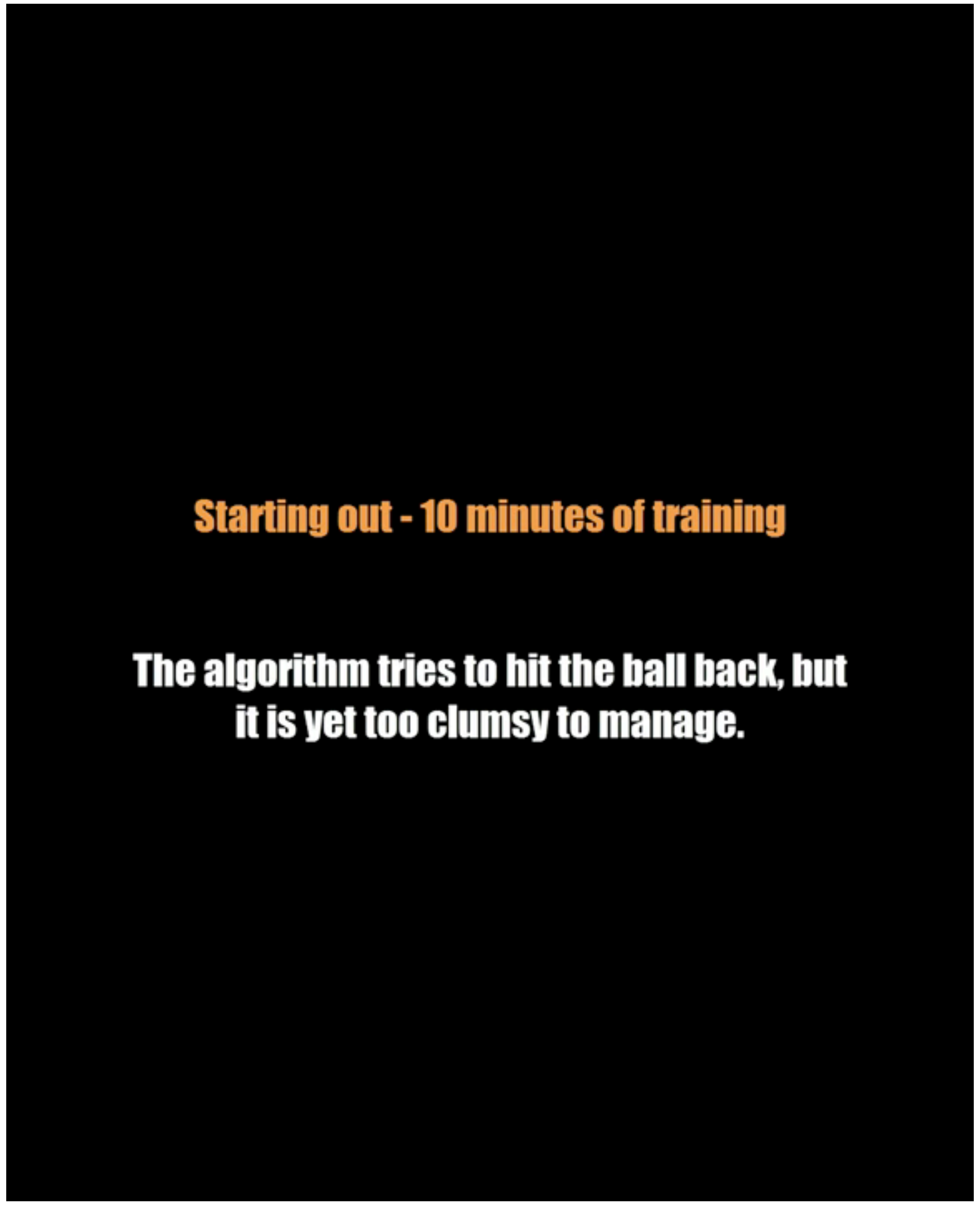
Sample a random mini-batch from replay memory and perform a gradient descent step

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Example: Atari Playing



Example: Atari Playing



Deep RL

Value-based RL

— Use neural nets to represent Q function $Q(s, a; \theta)$

$$Q(s, a; \theta^*) \approx Q^*(s, a)$$

Policy-based RL

— Use neural nets to represent the policy π_θ

$$\pi_{\theta^*} \approx \pi^*$$

Model-based RL

— Use neural nets to represent and learn the model

* slide from Dhruv Batra

Deep RL

Value-based RL

— Use neural nets to represent Q function $Q(s, a; \theta)$

$$Q(s, a; \theta^*) \approx Q^*(s, a)$$

Policy-based RL

— Use neural nets to represent the policy π_θ

$$\pi_{\theta^*} \approx \pi^*$$

Model-based RL

— Use neural nets to represent and learn the model

* slide from Dhruv Batra

Policy Gradients

Formally, let's define a class of parameterized policies:

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

Policy Gradients

Formally, let's define a class of parameterized policies:

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

Policy Gradients

Formally, let's define a class of parameterized policies:

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

How can we do this?

Policy Gradients

Formally, let's define a class of parameterized policies:

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

How can we do this?

Gradient ascent on policy parameters!

REINFORCE algorithm

Expected reward:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} [r(\tau)]$$
$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Where r(τ) is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

REINFORCE algorithm

Expected reward:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)]$$
$$= \int_\tau r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Where r(τ) is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$

Now let's differentiate this: $\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$

Intractable! Expectation of gradient is problematic when p depends on θ

REINFORCE algorithm

Expected reward:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)]$$
$$= \int_\tau r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Where r(τ) is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$

Now let's differentiate this: $\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$

However, we can use a nice trick: $\nabla_\theta p(\tau;\theta) = p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)$

If we inject this back:

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau) \nabla_\theta \log p(\tau;\theta) \right) p(\tau;\theta) \mathrm{d}\tau$$
$$= \mathbb{E}_{\tau \sim p(\tau;\theta)} \left[r(\tau) \nabla_\theta \log p(\tau;\theta) \right]$$

Can estimate with Monte Carlo sampling

Intuition

Gradient estimator:

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Intuition

Gradient estimator:

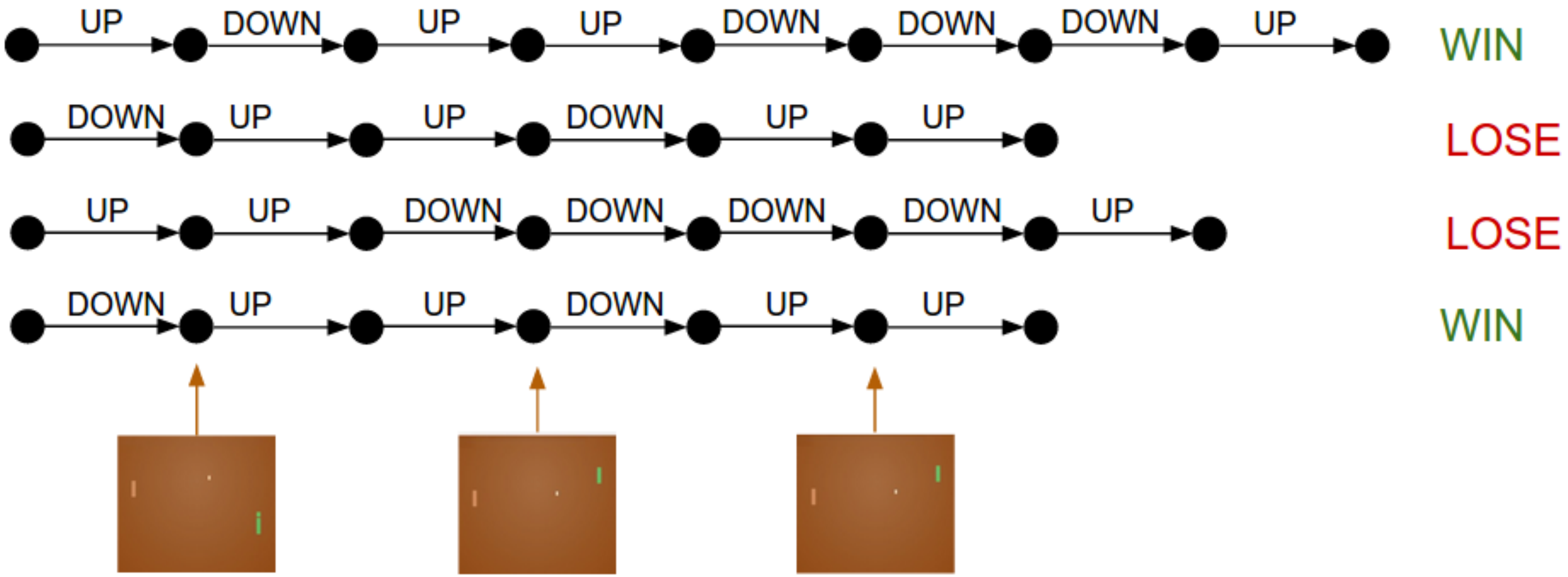
$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. **But in expectation, it averages out!**

Intuition



* slide from Dhruv Batra

Intuition

Gradient estimator:

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. **But in expectation, it averages out!**

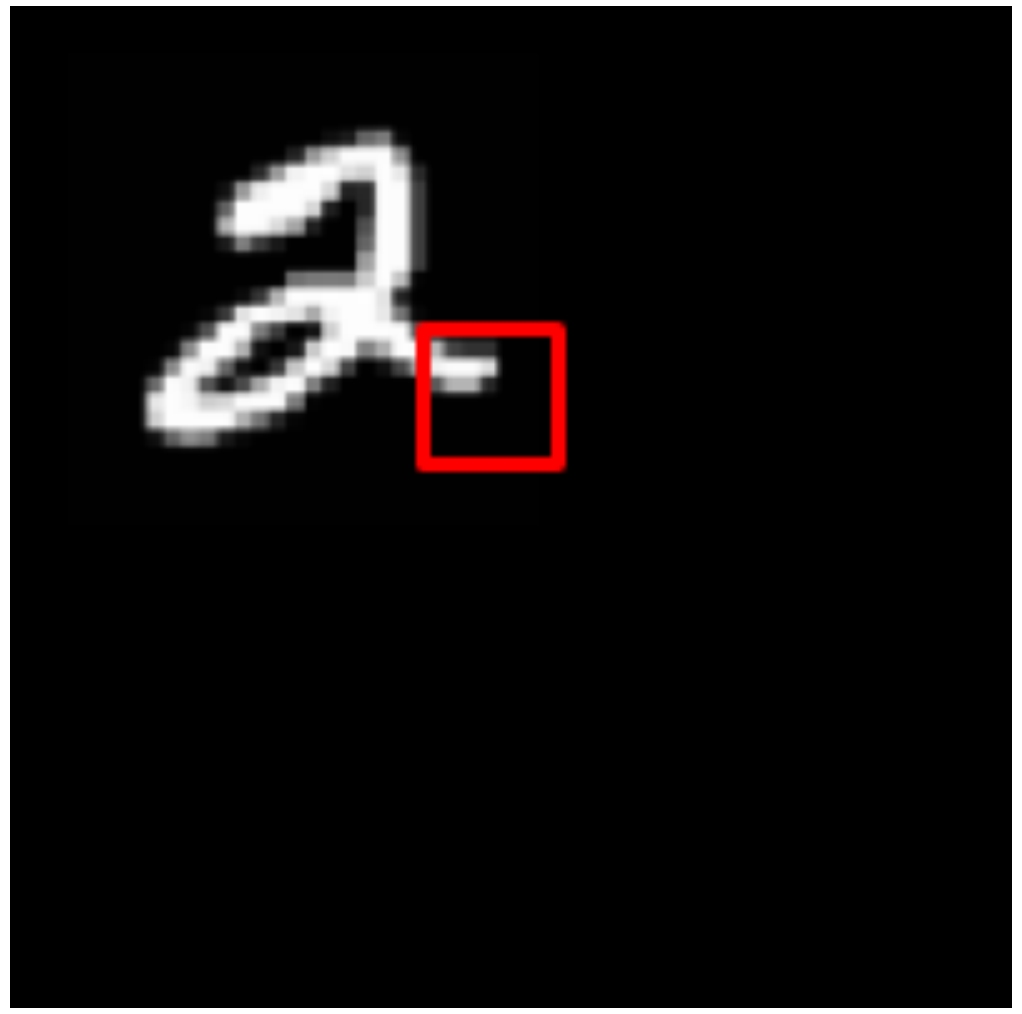
However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?

REINFORCE in Action: **Recurrent Attention Model** (REM)

Objective: Image Classification

Take a sequence of "glimpses" selectively focusing on regions of the image, to predict class

- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image



glimpse

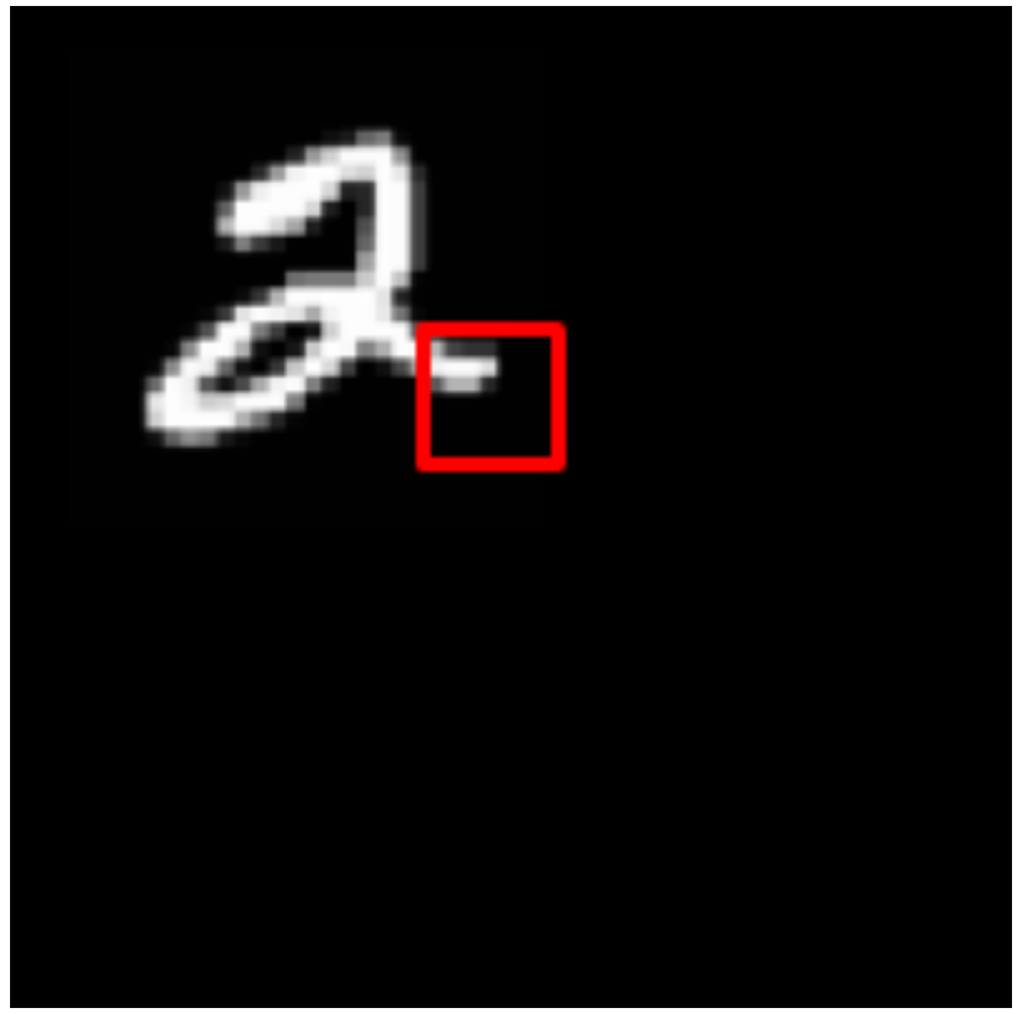
[Mnih *et al.*, 2014]

REINFORCE in Action: **Recurrent Attention Model** (REM)

Objective: Image Classification

Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class

- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image



glimpse

State: Glimpses seen so far
Action: (x,y) coordinates (center of glimpse) of where to look next in image
Reward: 1 at the final timestep if image correctly classified, 0 otherwise

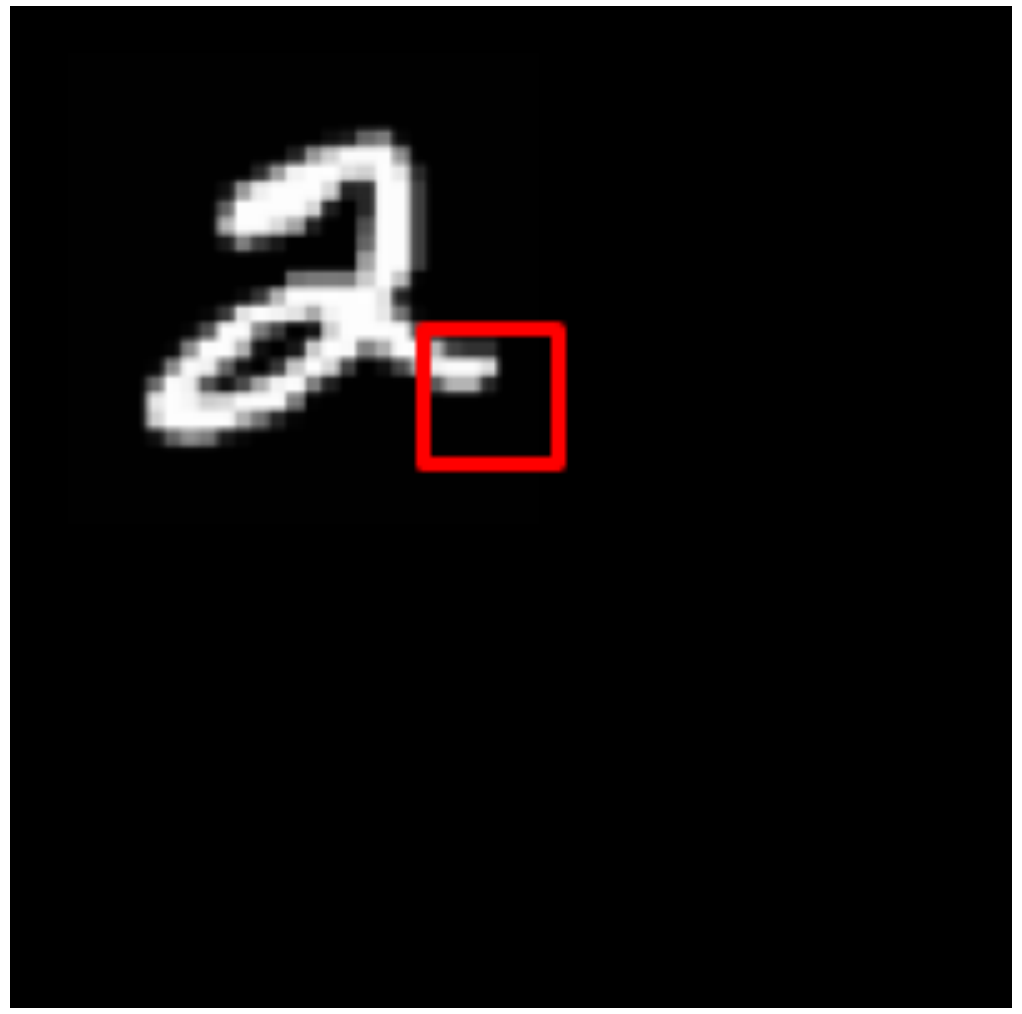
[Mnih *et al.*, 2014]

REINFORCE in Action: **Recurrent Attention Model** (REM)

Objective: Image Classification

Take a sequence of "glimpses" selectively focusing on regions of the image, to predict class

- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image



glimpse

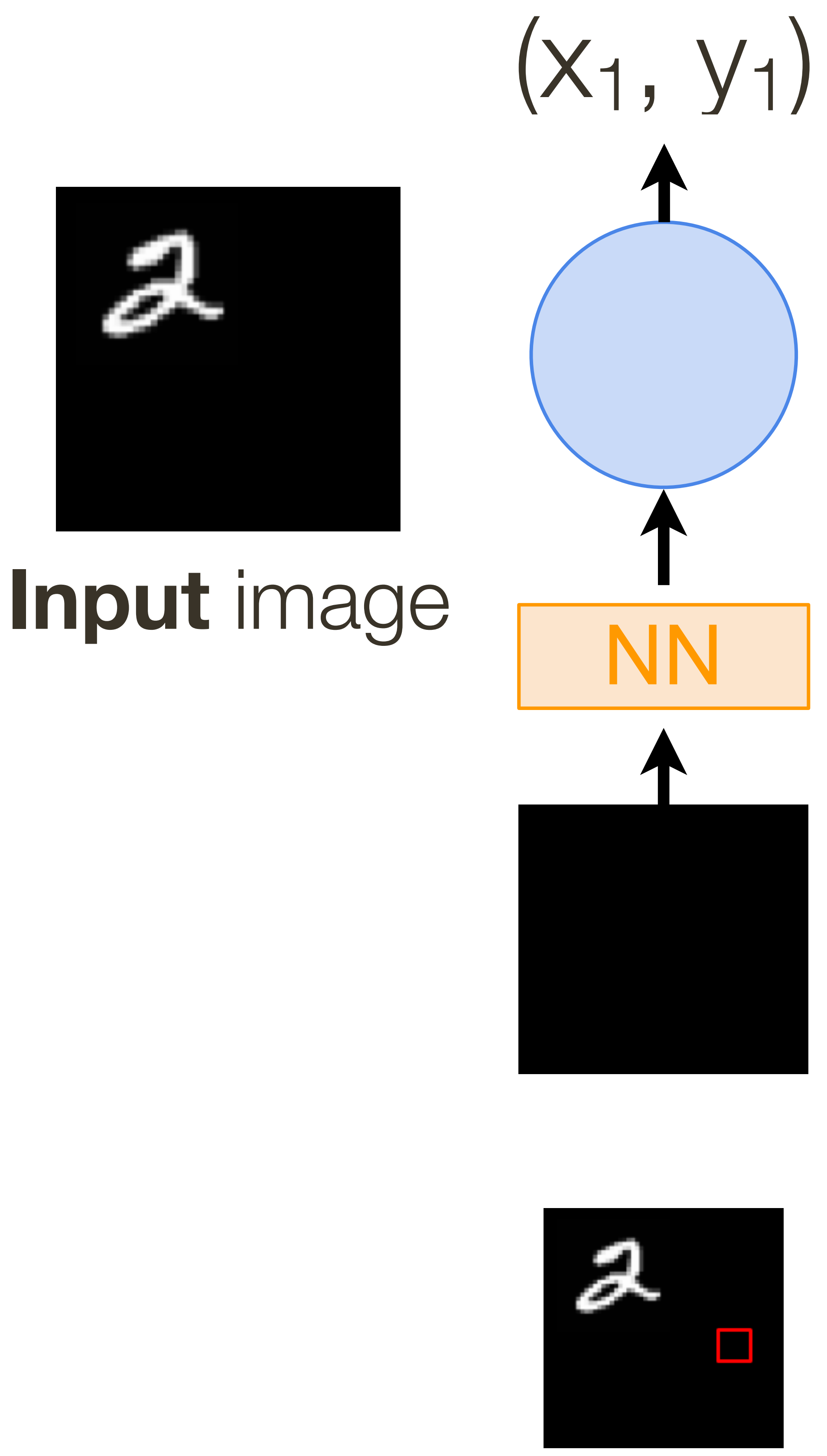
State: Glimpses seen so far
Action: (x,y) coordinates (center of glimpse) of where to look next in image
Reward: 1 at the final timestep if image correctly classified, 0 otherwise

Glimpsing is a **non-differentiable operation** => learn policy for how to take glimpse actions using REINFORCE
Given state of glimpses seen so far, use RNN to model the state and output next action

[Mnih *et al.*, 2014]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

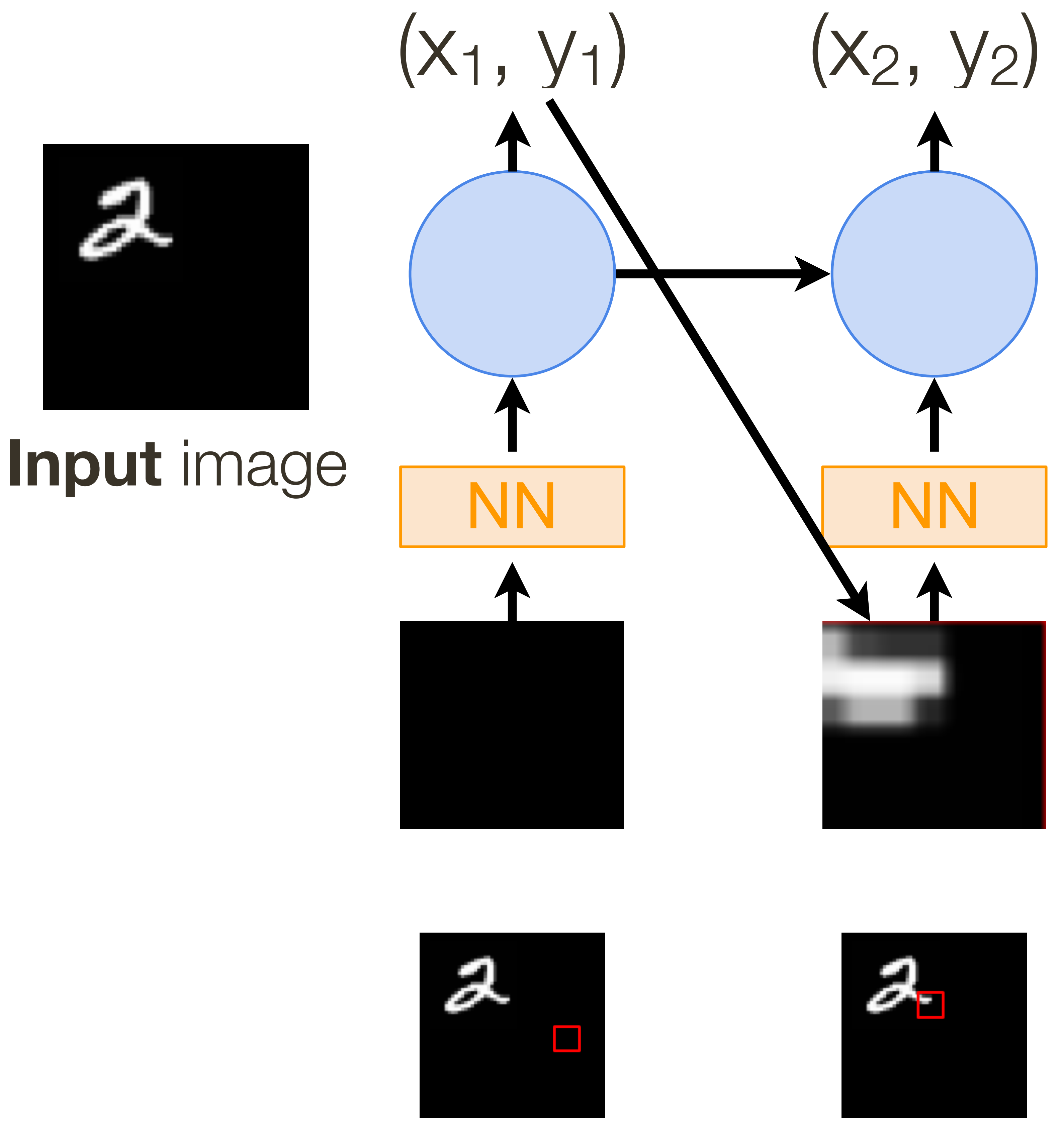
REINFORCE in Action: **Recurrent Attention Model** (REM)



[Mnih *et al.*, 2014]

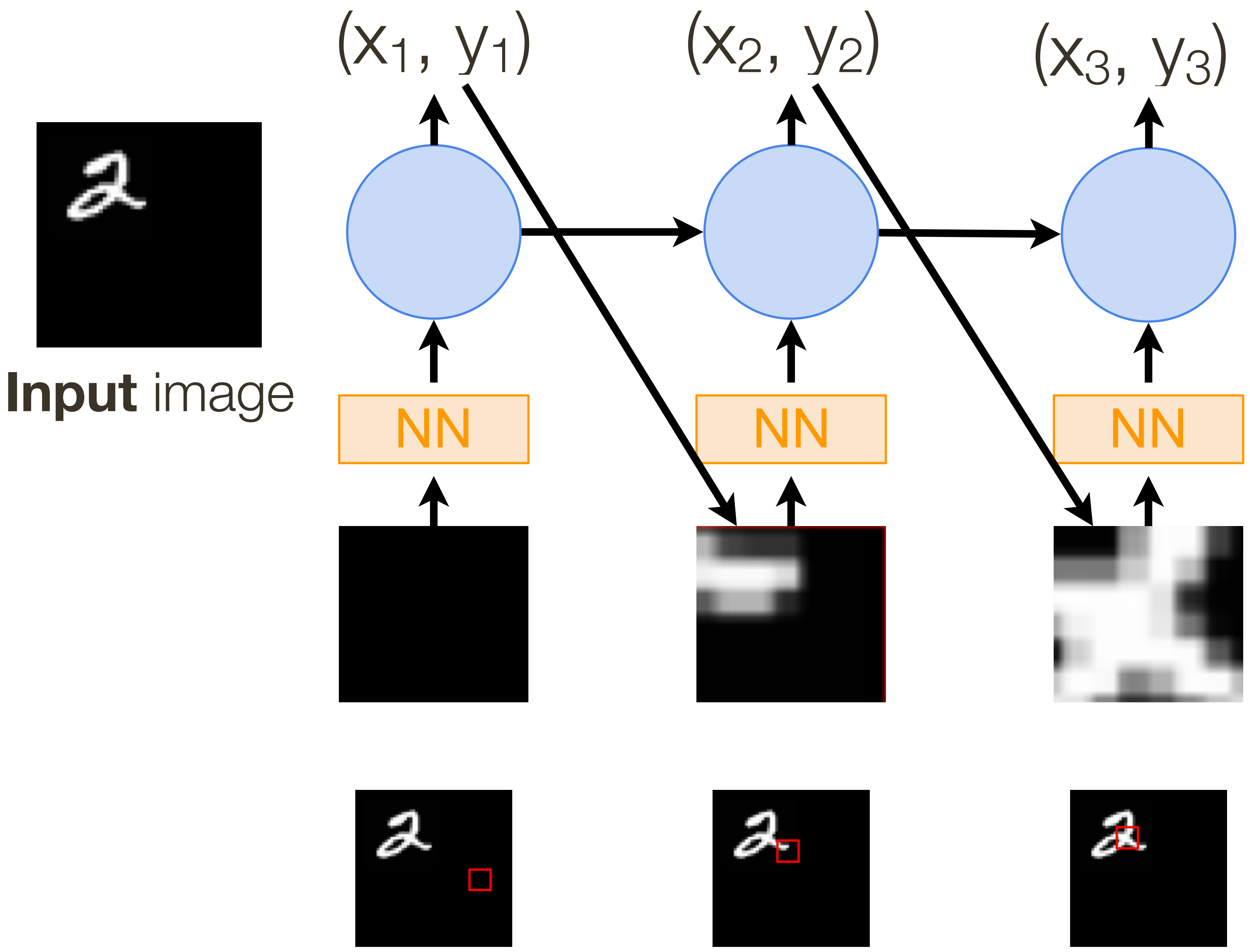
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

REINFORCE in Action: **Recurrent Attention Model** (REM)



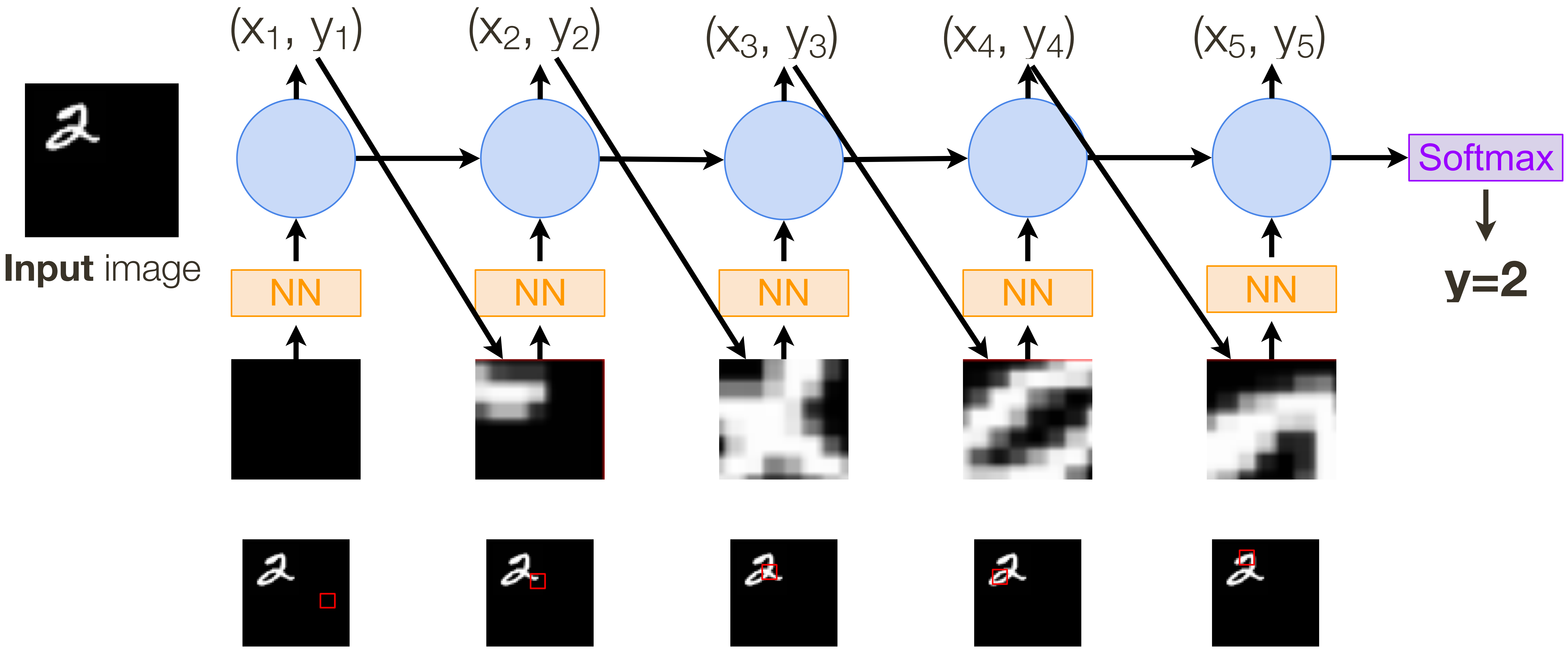
[Mnih *et al.*, 2014]

REINFORCE in Action: **Recurrent Attention Model** (REM)



[Mnih *et al.*, 2014]

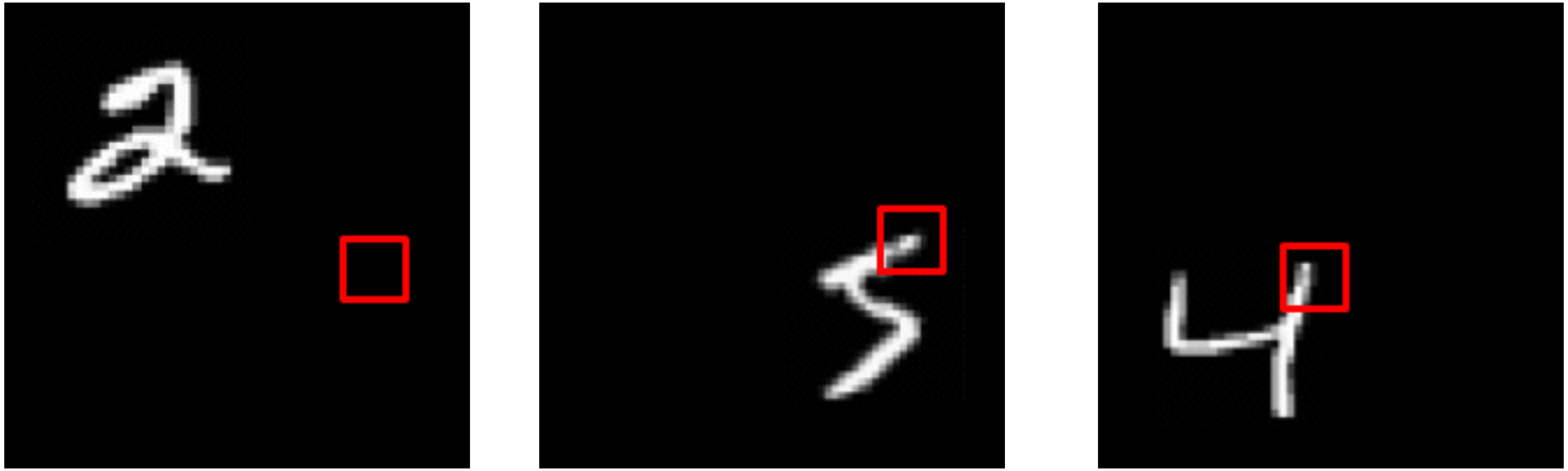
REINFORCE in Action: **Recurrent Attention Model** (REM)



[Mnih *et al.*, 2014]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

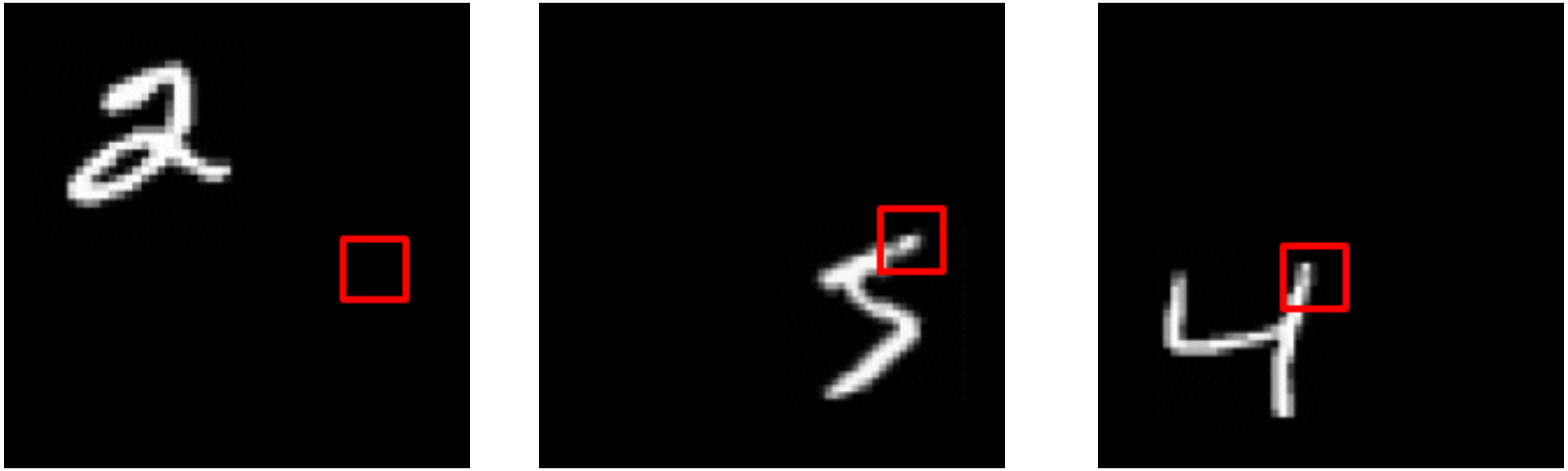
REINFORCE in Action: **Recurrent Attention Model** (REM)



Has also been used in many other tasks including fine-grained image recognition, image captioning, and visual question-answering!

[Mnih *et al.*, 2014]

REINFORCE in Action: **Recurrent Attention Model** (REM)

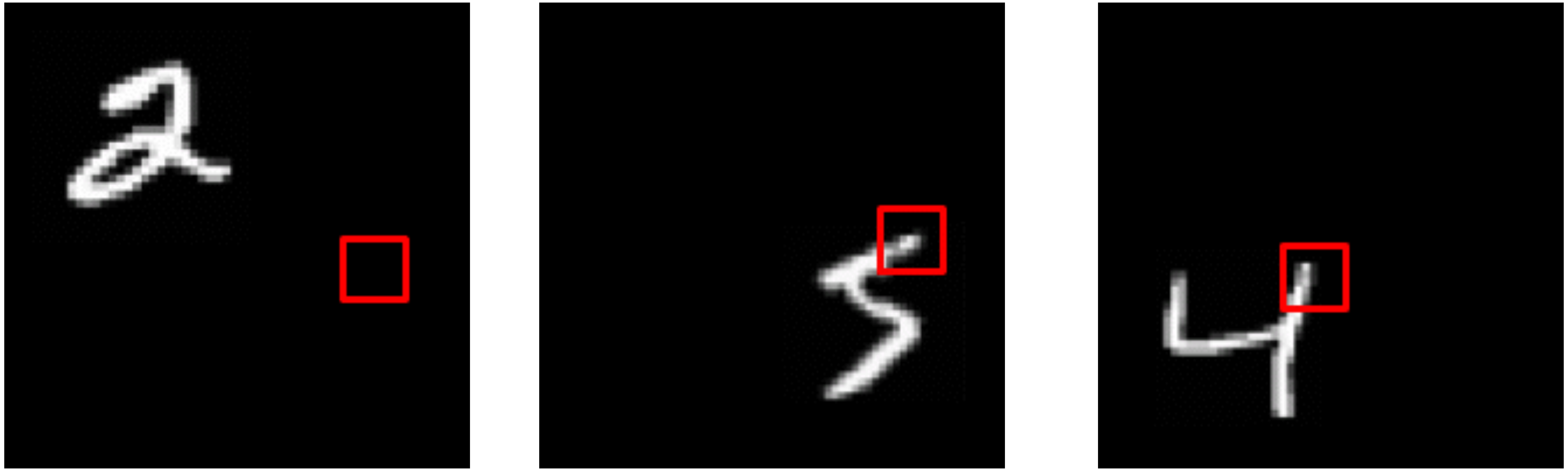


Has also been used in many other tasks including fine-grained image recognition, image captioning, and visual question-answering!

[Mnih *et al.*, 2014]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

REINFORCE in Action: **Recurrent Attention Model** (REM)

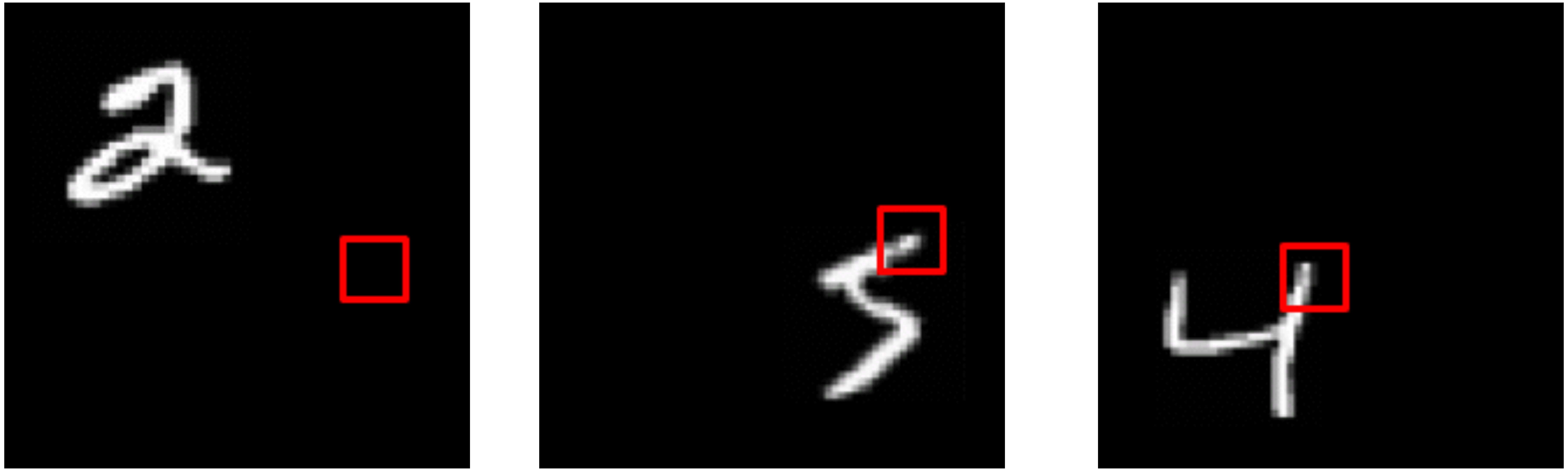


Has also been used in many other tasks including fine-grained image recognition, image captioning, and visual question-answering!

[Mnih *et al.*, 2014]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

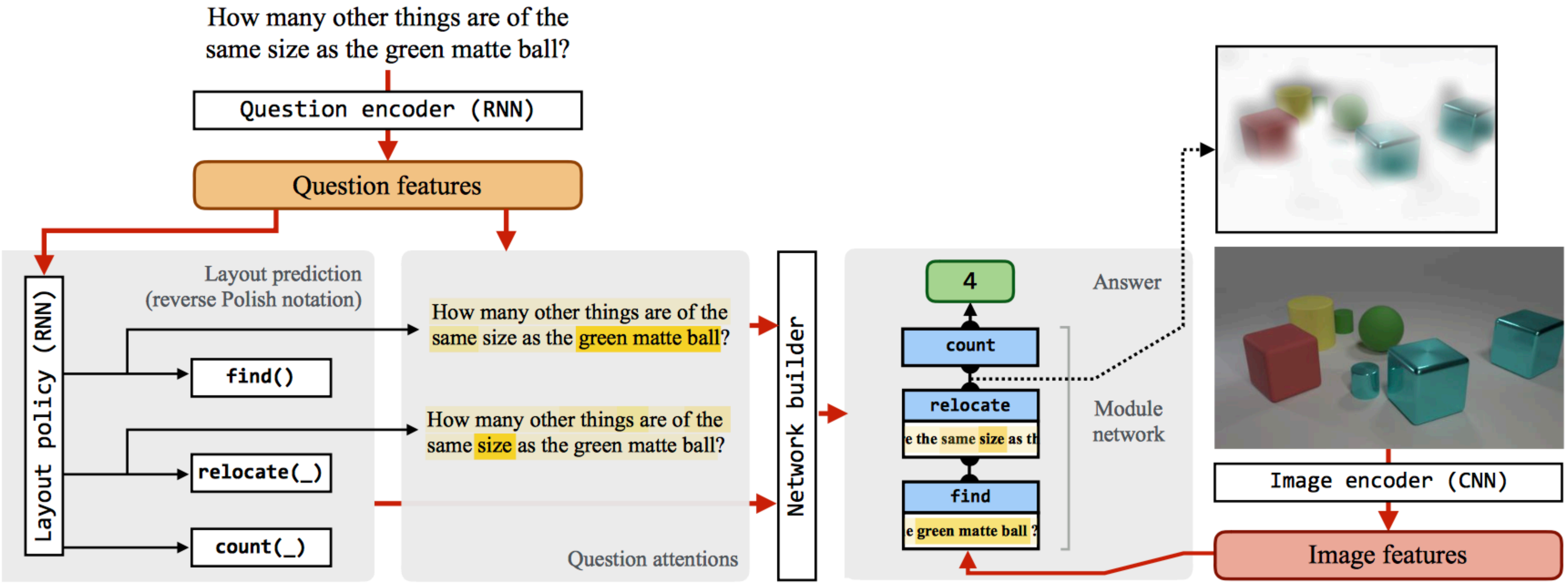
REINFORCE in Action: **Recurrent Attention Model** (REM)



Has also been used in many other tasks including fine-grained image recognition, image captioning, and visual question-answering!

[Mnih *et al.*, 2014]

Learning to **Reason**



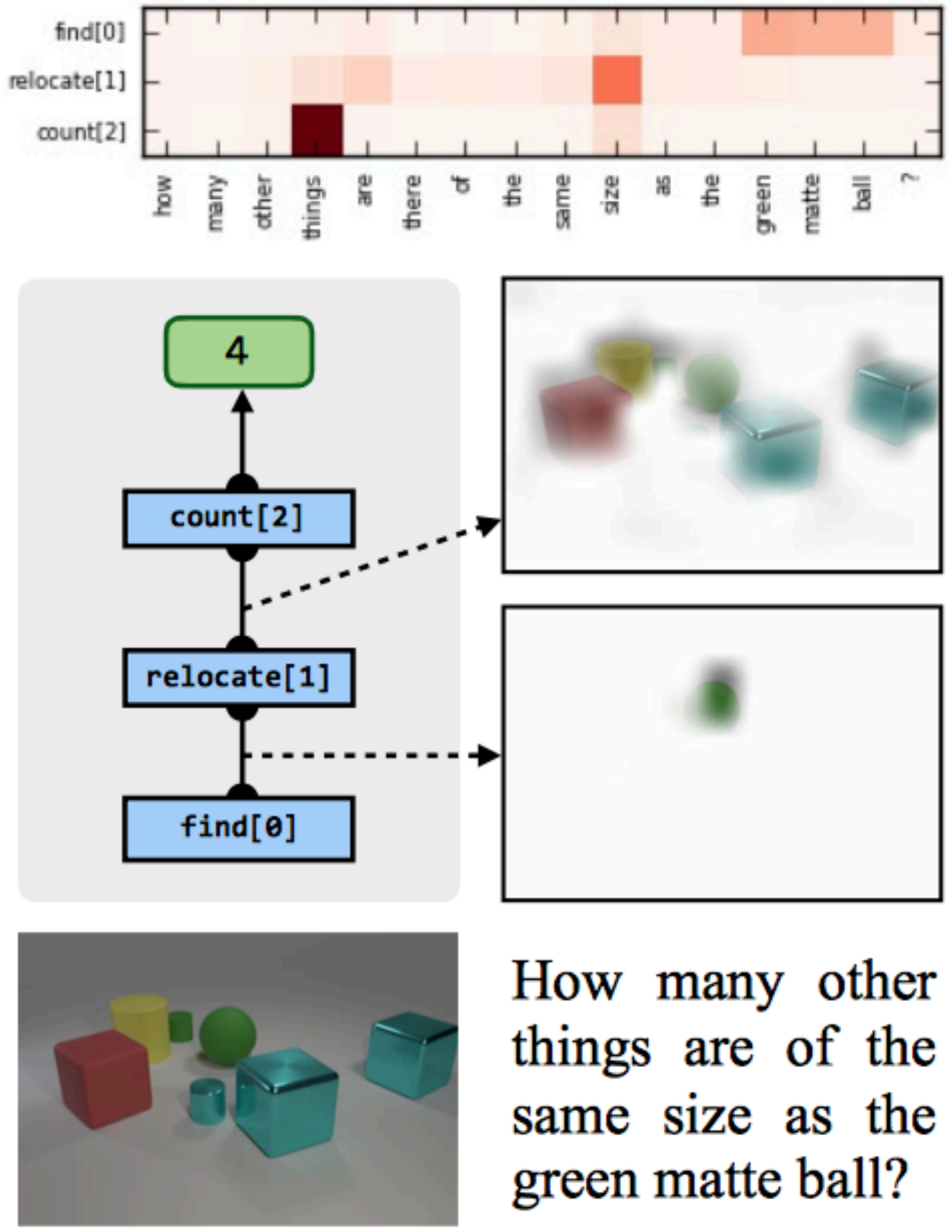
[Hu *et al.*, 2017]

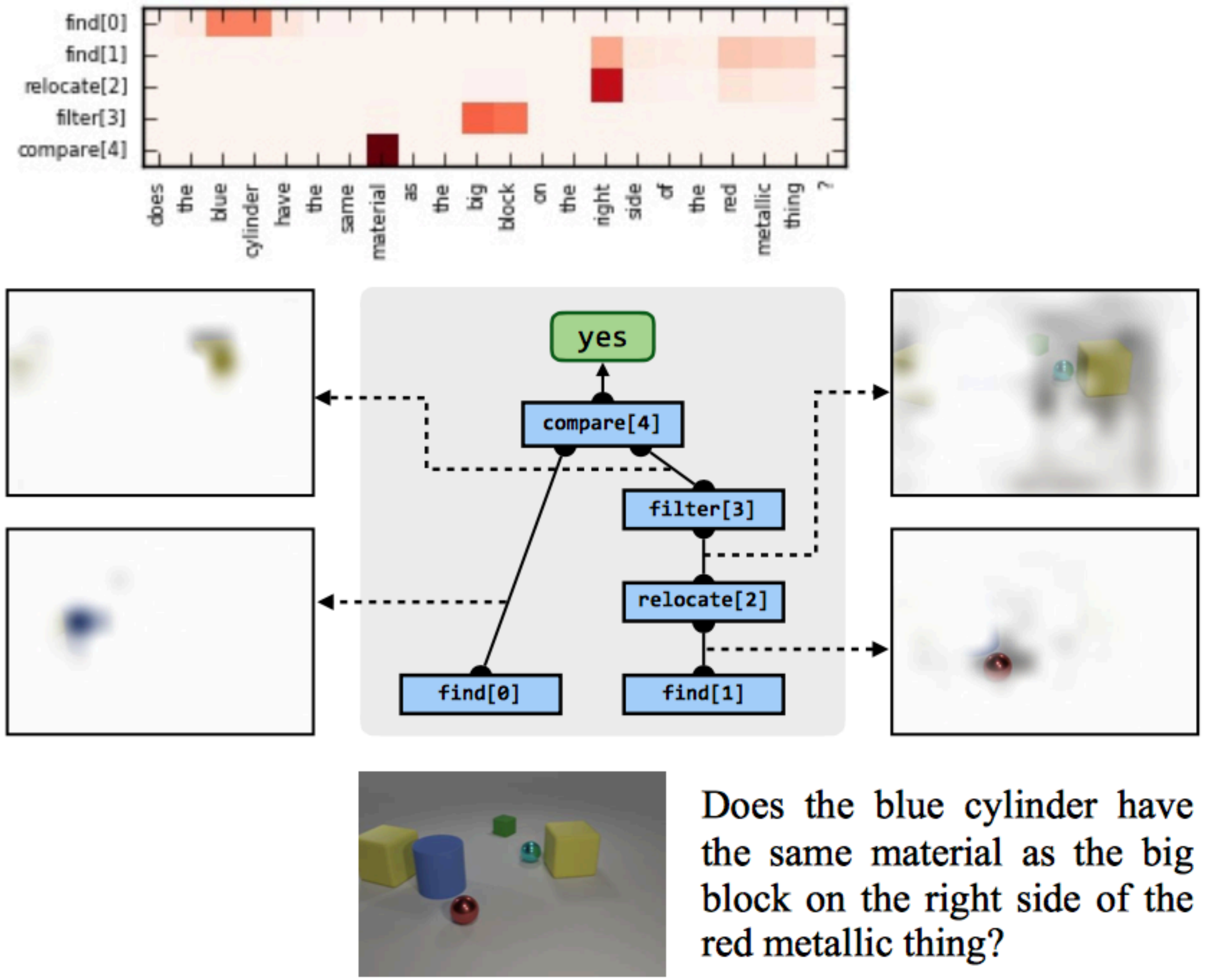
Learning to **Reason**

Module name	Att-inputs	Features	Output	Implementation details
`find`	(none)	x_{vis}, x_{txt}	att	$a_{out} = \text{conv}_2\left(\text{conv}_1(x_{vis}) \odot W x_{txt}\right)$
`relocate`	a	x_{vis}, x_{txt}	att	$a_{out} = \text{conv}_2\left(\text{conv}_1(x_{vis}) \odot W_1 \text{sum}(a \odot x_{vis}) \odot W_2 x_{txt}\right)$
`and`	a_1, a_2	(none)	att	$a_{out} = \text{minimum}(a_1, a_2)$
`or`	a_1, a_2	(none)	att	$a_{out} = \text{maximum}(a_1, a_2)$
`filter`	a	x_{vis}, x_{txt}	att	$a_{out} = \text{and}(a, \texttt{find}[x_{vis}, x_{txt}]())$, *i.e.* reusing `find` and `and`
`[exist, count]`	a	(none)	ans	$y = W^T \text{vec}(a)$
`describe`	a	x_{vis}, x_{txt}	ans	$y = W_1^T \left(W_2 \text{sum}(a \odot x_{vis}) \odot W_3 x_{txt}\right)$
`[eq_count, more, less]`	a_1, a_2	(none)	ans	$y = W_1^T \text{vec}(a_1) + W_2^T \text{vec}(a_2)$
`compare`	a_1, a_2	x_{vis}, x_{txt}	ans	$y = W_1^T \left(W_2 \text{sum}(a_1 \odot x_{vis}) \odot W_3 \text{sum}(a_2 \odot x_{vis}) \odot W_4 x_{txt}\right)$

[Hu *et al.*, 2017]

Learning to **Reason**





[Hu *et al.*, 2017]

Summary

Policy gradients: very general but suffer from high variance so requires a lot of samples. ***Challenge***: sample-efficiency

Q-learning: does not always work but when it works, usually more sample-efficient. ***Challenge***: exploration

Guarantees:
— Policy Gradients: Converges to a local minima of $J(\theta)$, often good enough!
— Q-learning: Zero guarantees since you are approximating Bellman equation with a complicated function approximator

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**