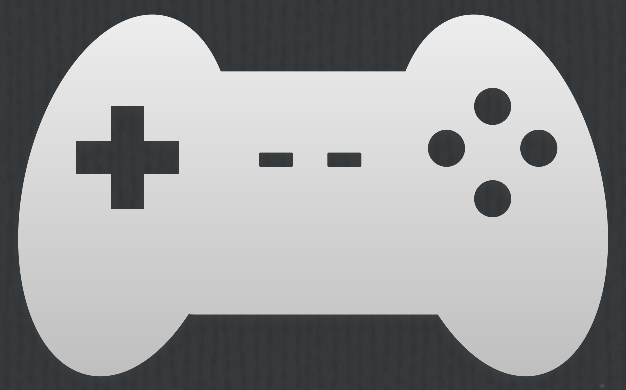
Deep Reinforcement Learning

Yuchu Luo
Computer Animation and Multimedia Analysis LAB
Monday, 28 August 2017


reward
Agent
action
Goal
state
Environment

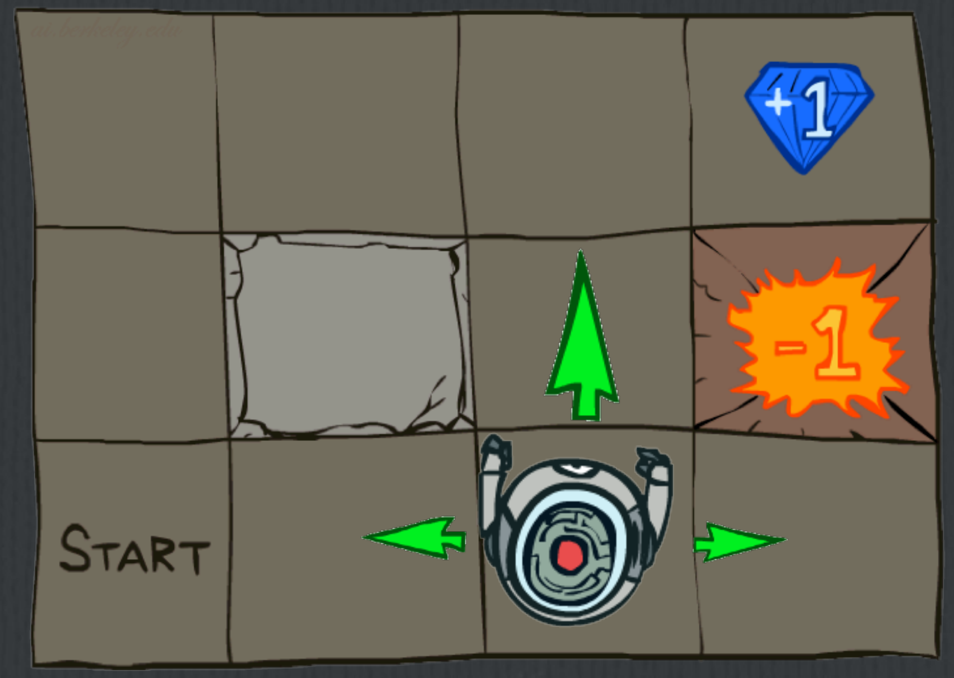


Play Ataria Game

- **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 Process

- A set of states $s \in S$
- A set of actions (per state) $a \in A$
- A model $T(s,a,s')$
- A reward function $R(s,a,s')$
- Looking for a policy $\pi^*(s)$ that maximizes cumulative discounted reward: $\sum \gamma^t r_t$



Policy Learning

Find optimal policy $\pi^*(s)$

$$a \sim \pi^*(s)$$

local information

Value Learning

Find optimal Q-Value Function $Q^*(s,a)$

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

global information

$$Q^*(s_t, a_t) = \max_{\pi} E\left[\sum_{i=t}^{T} \gamma^{i-t} r_i\right]$$

Maximum expected future rewards starting at **state s_i**, choosing **action a_i**, and then following an **optimal policy π^***

Bellman Equation

$$Q^*(s,a) = E_{s'\sim\varepsilon}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right]$$

Intuition:

从最佳选择的路径末端截取一小部分，余下的路径仍然是最佳路径

Solving Optimal Q-Value

Value Iteration

Convergent

$$\boxed{Q_{k+1}(s,a) = \mathrm{E}\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \mid s,a \right] = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]}$$

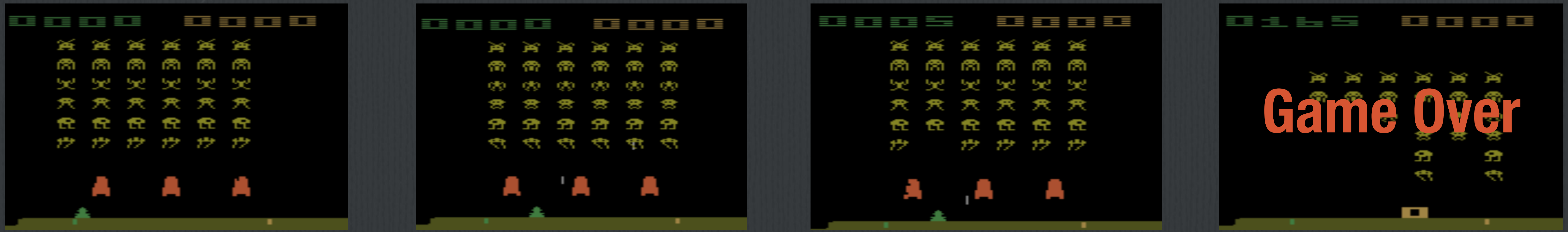
Reinforcement Learning

Life is always hard — We don't know P(s,a,s') and R(s,a,s')

Partially Observed MDP (POMDP)

Episode: sequence of states and actions

$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T$



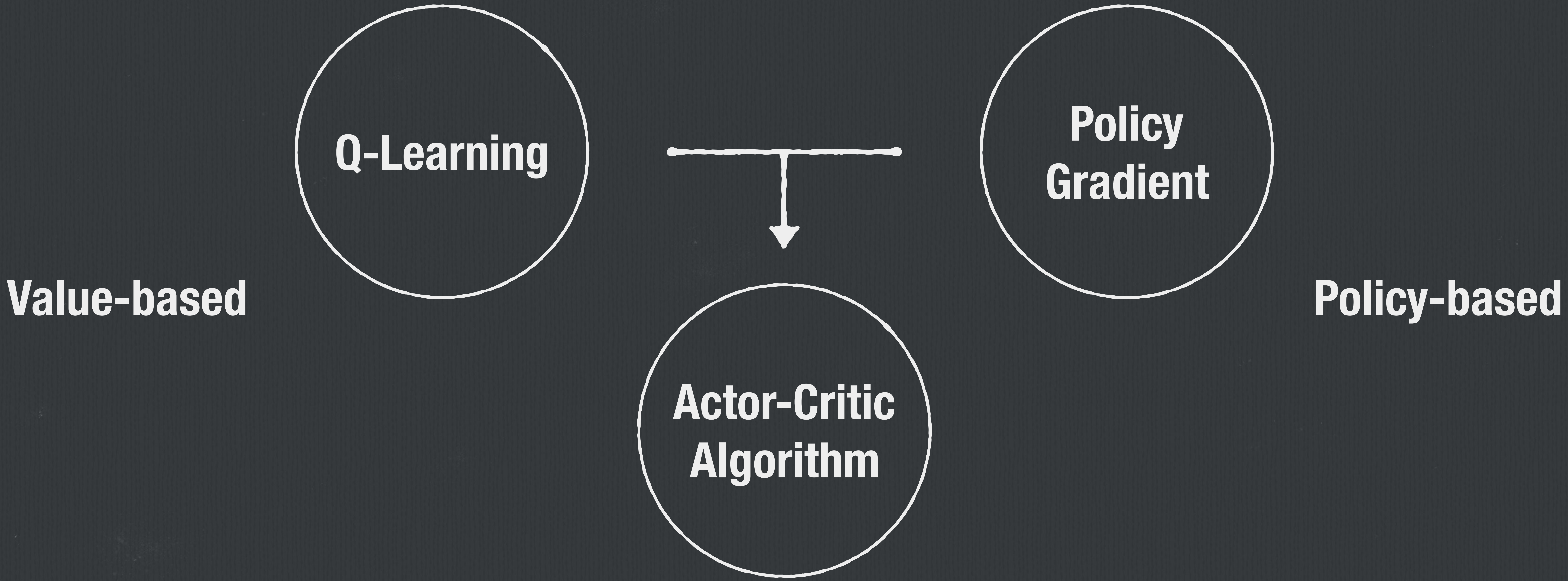
observations (states), actions

obtain reward R

Learn to maximize the expected cumulative reward per episode

Model-Based or Model-Free?

Model-Free


Q-Learning
Policy Gradient
Actor-Critic Algorithm
Value-based
Policy-based

Recap: Approximate Q-Learning

Linear Value Functions

$$Q(s,a) = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n(s,a)$$

Feature-Based Representations

- Distance to closest ghost
- Distance to closest dot
- Number of ghosts
- 1 / (dist to dot)2
- Is Pacman in a tunnel? (0/1)
- …

Update

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$$

Historical experience **Learned from new (s,a,r,s') pair**

$$difference = [r + \gamma \max_{a'} Q(s',a')] - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot difference$$

$$w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a)$$

Now, we have deep learning

Deep Q-Learning

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

Make the function approximate be a deep neural network

Loss function: $L_i(\theta_i) = \mathrm{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s,a;\theta_i))^2]$

Where $y_i = \mathrm{E}_{s'\sim\varepsilon}[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \mid s,a]$

Deep Q-Learning

Feedforward Pass

Cutest state s_t
(84x84x4) stack of last 4 frames

Convolutional Neural Network

Fully Connected Layer

Output (4 Q-Values)

Experience Replay

Learning from batches of consecutive samples is problematic:

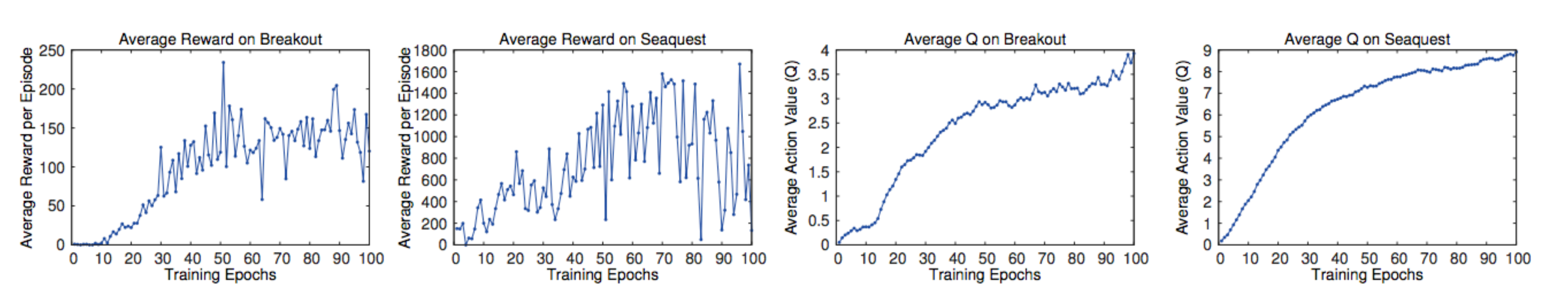
- Samples are correlated => inefficient learning
- current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size => can lead to bad feedback loops

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 mini batches of transitions from the replay memory, instead of consecutive samples
- Each transition can also contribute to multiple weight updates => greater data efficiency

From CS231n

Experiments



Policy Gradients

Instead of learning exact value of every (state, action) pair,
just riding the best policy from a collection of policies

Neural Network

left	0.6
right	0.1
fire	0.3

Probability

REINFORCE algorithm

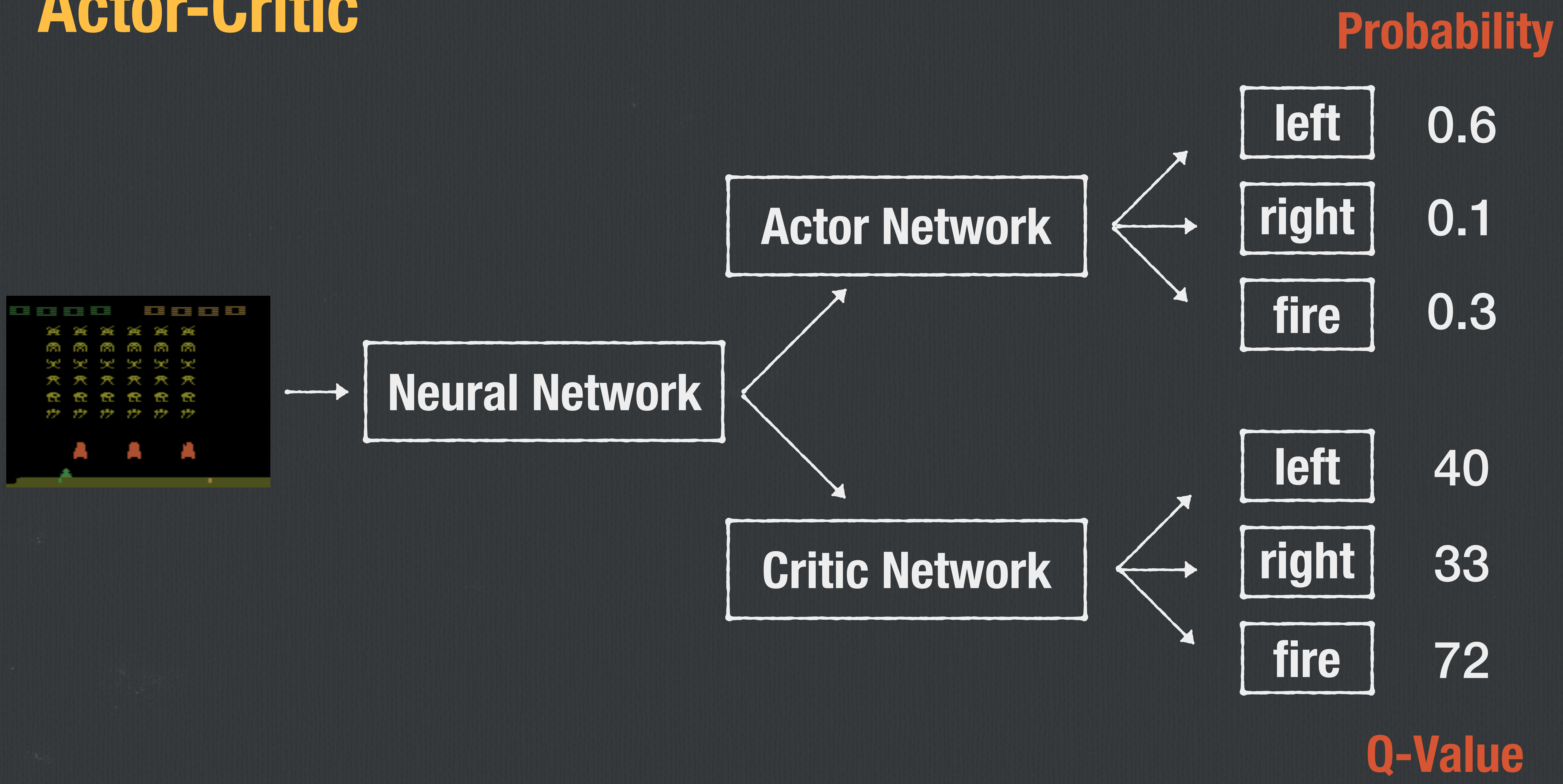
$$\boxed{\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)}$$

Intuition:

- **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**

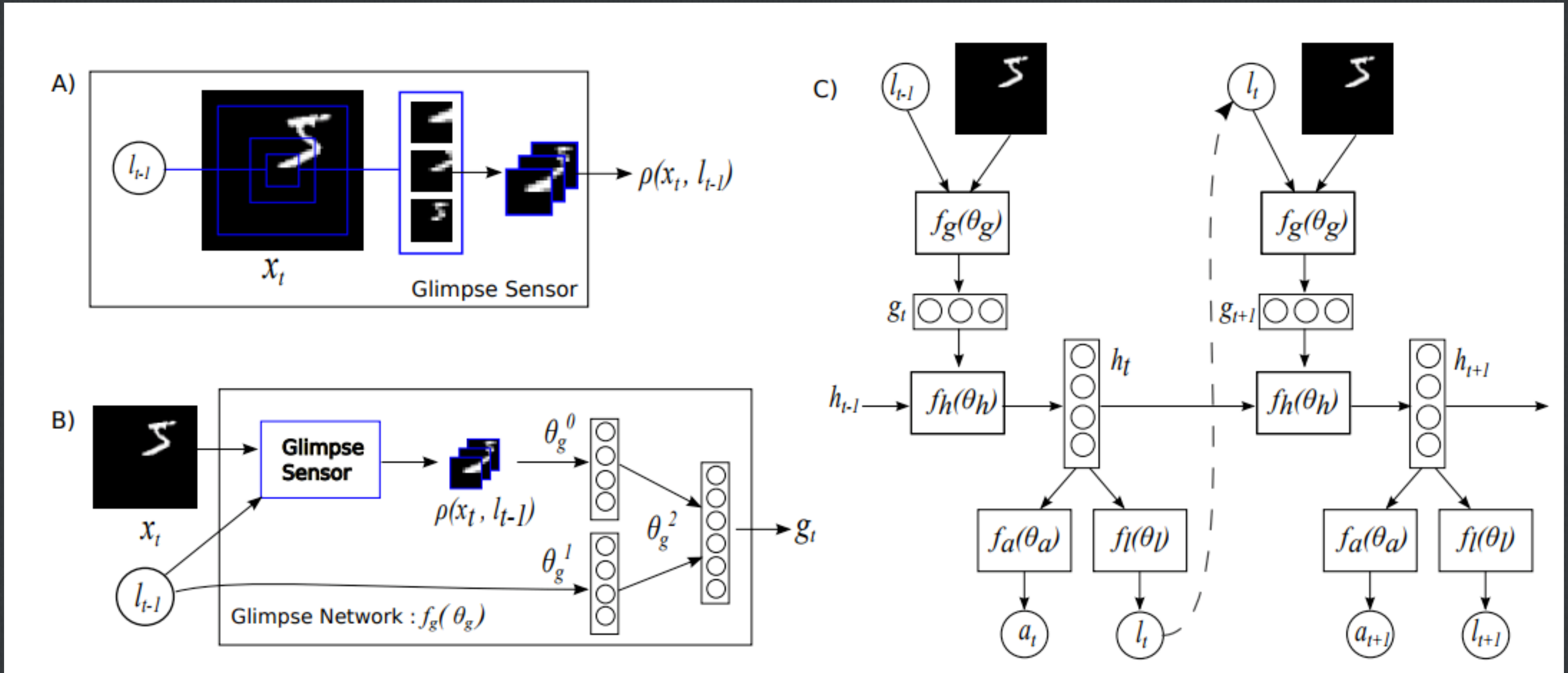
Learn more in supplied materials

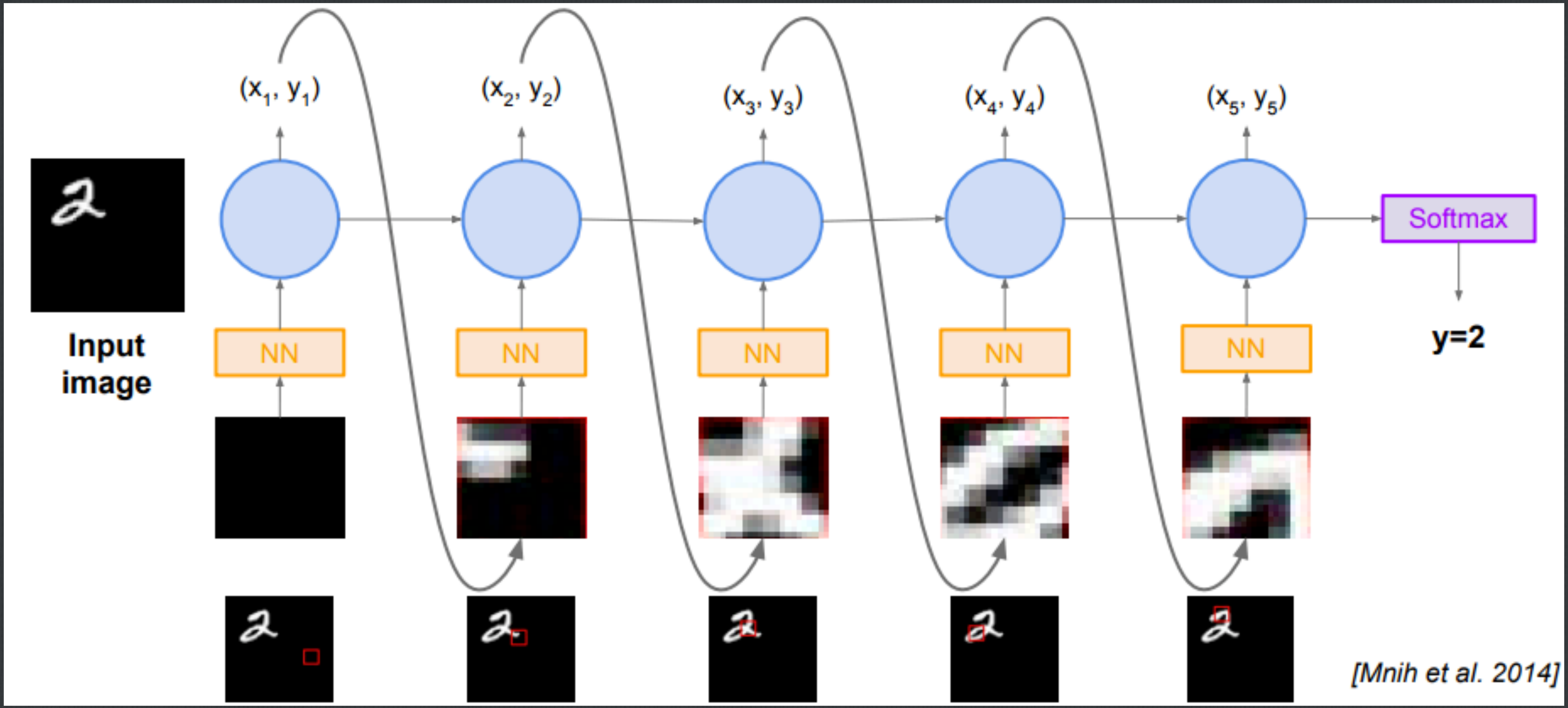
Actor-Critic



Example: Recurrent Attention Model (RAM)

Considered as a **control problem**




(x1, y1)
(x2, y2)
(x3, y3)
(x4, y4)
(x5, y5)
Input image
NN
NN
NN
NN
NN
Softmax
y=2
[Mnih et al. 2014]

Summary

- **Policy gradients**: 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(ʒ), often good enough!
  - **Q-learning**: Zero guarantees since you are approximating Bellman equation with a complicated function approximator

From CS231n