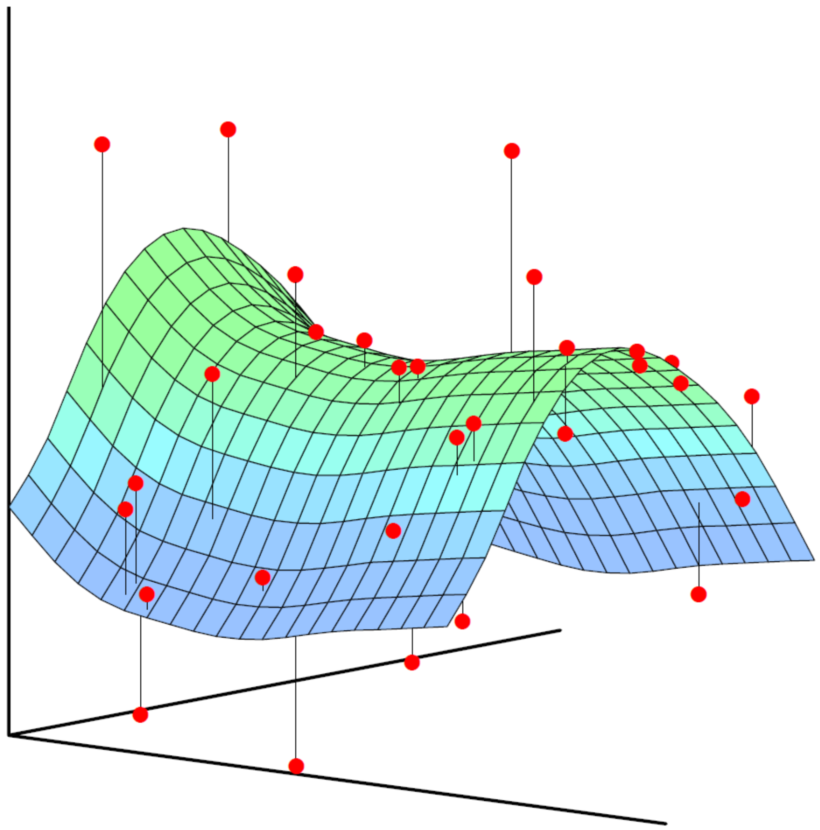
Machine Learning



第12讲 强化学习

Reinforcement Learning

刘 峤

电子科技大学计算机科学与工程学院

Roadmap

- Introduction
  ※ Terminologies.
  ※ Rewards, Returns, and Value functions
  ※ Play games using reinforcement learning
- Value-based learning
  ※ Deep Qnetwork (DQN) for approximating Q*(s,a).
  ※ Learn the network
  ※ parameters using temporal different (TD).
- Policy-based learning.
  ※ Policy network for approximating πt(a|s).
  ※ Learn the network parameters using policy gradient.
- Actor-critic method. (Policy network + valtue network.) 自学

12.1 Introduction

12.1.1 Terminologies

Random Variable

- Random variable:
  - ※ a variable whose values depend on outcomes of a random event.
  - ※ Uppercase letter ***X*** for random variable.
  - ※ Lowercase letter ***x*** for an observed value.
  - ※ For example, we flipped a coin 4 times and observed:
  - ※ $x_1 = 1, \quad x_2 = 1, \quad x_3 = 0, \quad x_4 = 1$
- Probability Density Function (PDF)
  - ※ PDF provides a relative likelihood that the value of the random variable would equal to that sample.

Expectation

- Random variable ***X*** is in the domain $\mathcal{X}$.
- For continuous distribution, the expectation of *f(X)* is:

$$\mathbb{E}[f(x)] = \int_{\chi} p(x) \cdot f(x) dx$$

- For discrete distribution, the expectation of *f(X)* is:

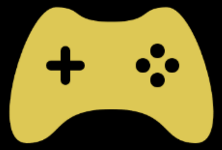
$$\mathbb{E}[f(x)] = \sum_{x \in \chi} p(x) \cdot f(x)$$

Random Sampling

- Sample red ball w.p. 0.2, green ball w.p.0.5, and blue ball w.p.0.3.
  - ※ Randomly sample a ball.
  - ※ What will be the outcome?

```
from numpy.random import choice

samples = choice( ['R', 'G', 'B' ],size=100,p=[0.2, 0.5, 0.3])
print (samples)
```

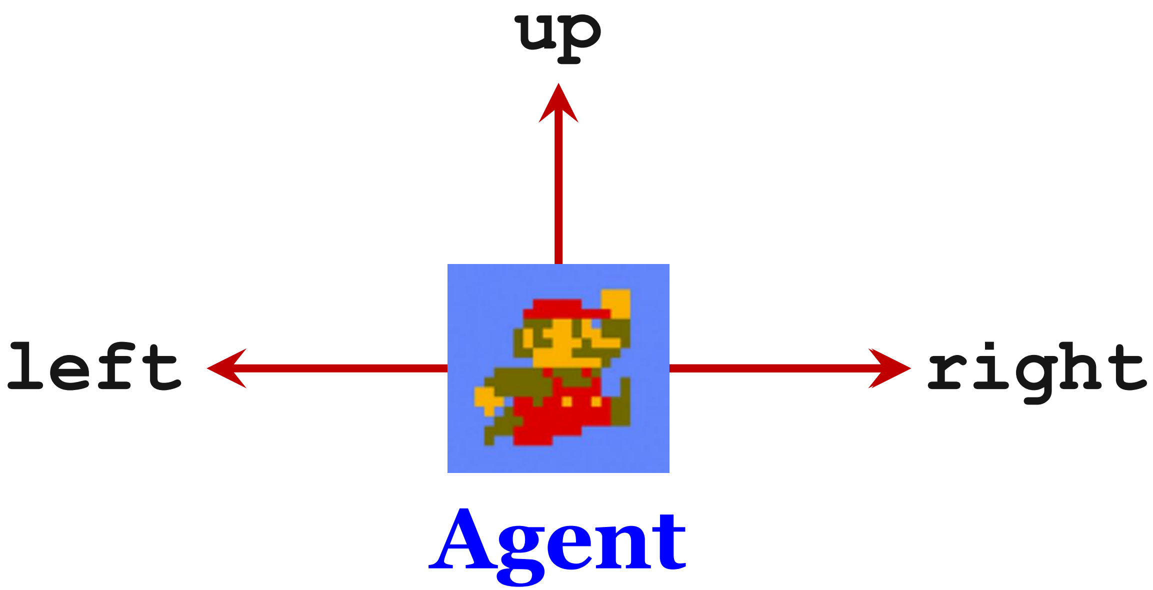
```
['G' 'G' 'G' 'R' 'R' 'G' 'B' 'G' 'G' 'B' 'B' 'G' 'G' 'R' 'R' 'G' 'R' 'B'
 'R' 'G' 'G' 'B' 'B' 'G' 'B' 'R' 'G' 'G' 'B' 'B' 'G' 'G' 'G' 'G' 'G' 'G'
 'B' 'G' 'B' 'B' 'G' 'R' 'G' 'G' 'G' 'B' 'B' 'R' 'G' 'R' 'G' 'B' 'G' 'B'
 'R' 'R' 'G' 'R' 'R' 'R' 'R' 'B' 'R' 'B' 'B' 'G' 'G' 'R' 'B' 'G' 'G' 'G'
 'G' 'G' 'R' 'B' 'B' 'G' 'G' 'G' 'B' 'B' 'B' 'B' 'B' 'B' 'R' 'B' 'B' 'G'
 'R' 'B' 'B' 'G' 'G' 'G' 'G' 'G' 'B' 'G']
```

			
Paradigm	Supervised Learning	Unsupervised Learning	Reinforcement Learning
Objective	$p_\theta(y\|x)$	$p_\theta(x)$	$\pi_\theta(a\|s)$
Applications	→ Classification → Regression	→ Inference → Generation	→ Prediction → Control

Terminology: state and action

State s (this frame)

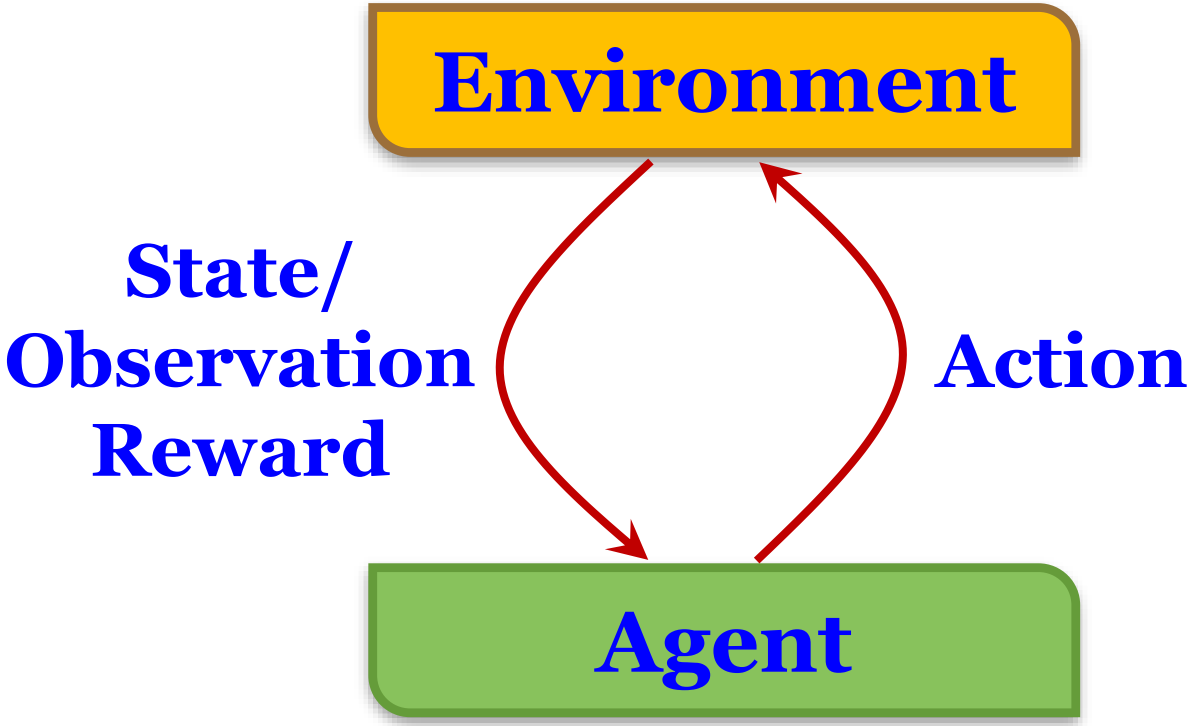
Action $a \in$ {left, right, up}



Terminology: policy

Setting

policy π



using policy $\pi_\theta(a|s)$

- Policy function $\pi : (s, a) \to [0, 1]$

$$\pi(a|s) = P(A = a|S = s)$$

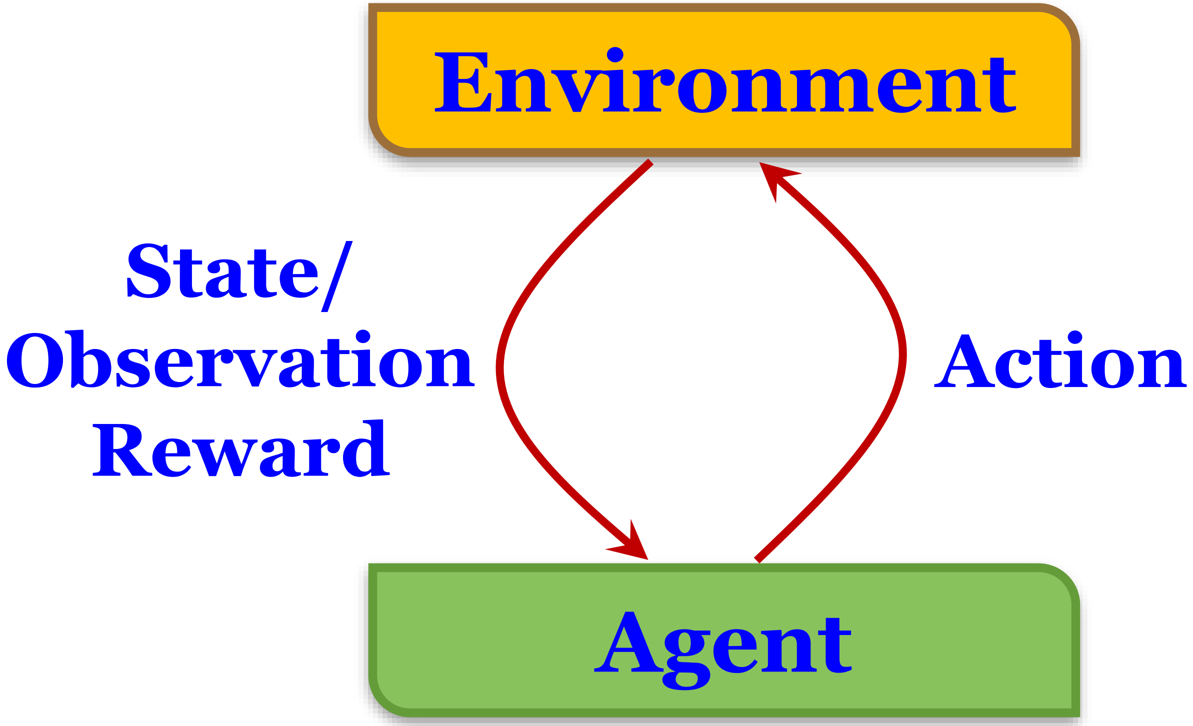
- It is the probability of taking action $A = a$ given state s , e.g.,

$$\pi(\text{left}|s) = 0.2$$
$$\pi(\text{right}|s) = 0.1$$
$$\pi(\text{up}|s) = 0.7$$

- Upon observing state $S = s$, the agent's action A can be random.

Terminology: reward

Setting



using policy $\pi_\theta(a|s)$

reward R

- Collect a coin: R= + 1
- Win the game: R=+10000
- Touch a Goomba: R =-10000

  ※ game over
- Nothing happens: R =0

Terminology: state transition

- state transition

$$\text{old state} \xrightarrow{\text{action}} \text{new state}$$

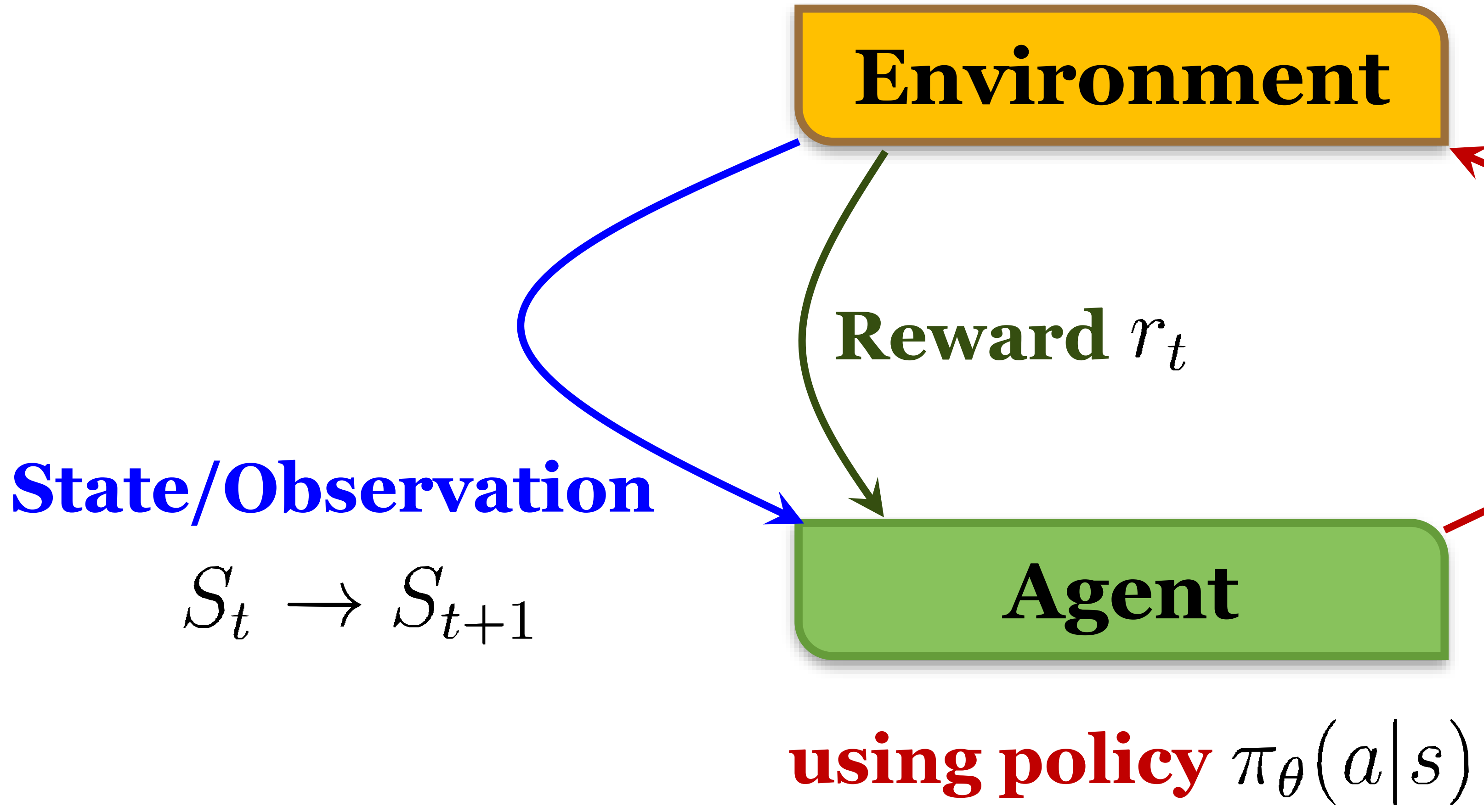
- E.g.,"up"action leads to a new state.
- State transition can be random.

※ Randomness is from the environment.

※ E.g., the Goombas’ next move is random

$$p(s'|s,a) = P(S' = s' | S = s, A = a)$$

Terminology: agent environment interaction



Randomness in Reinforcement Learning

- Actions have randomness. $A \sim \pi(\cdot|s)$

  ※ Given state s, the action can be random, e.g.,

$$\pi(\text{left}|s) = 0.2 \qquad \pi(\text{right}|s) = 0.1 \qquad \pi(\text{up}|s) = 0.7$$

- State transitions have randomness. $S' \sim (\cdot|s,a)$

  ※ Given state S = s and action A = a,

  ➢ the environment randomly generates a new state S'.

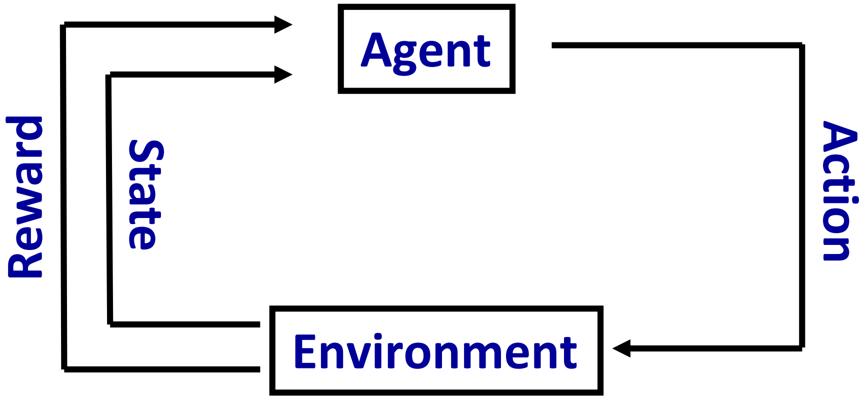
- Play the game using AI.

  ※ (state, action, reward) trajectory:

$$s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T, a_T, r_T.$$

Reinforcement Learning

- Learning to interact with an environment
  - ※ Robots, games, process control
  - ※ With limited human training
  - ※ Where the ‘right thing’ isn’t obvious



- Reinforcement Learning:
  - ※ Goal: Maximize $\sum_{i=1}^{\infty} \mathrm{Reward}(\mathrm{State}_i, \mathrm{Acton}_i)$
  - ※ Data: $\mathrm{Reward}_i,\ \mathrm{State}_{i+1} = \mathrm{Interact}(\mathrm{State}_i, \mathrm{Acton}_i)$

12.1.2 Rewards and Returns

Return

- Definition: Return (aka cumulative future reward).

  ※ $U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \ldots$ until the game is over.

- Question: Are R and R_{t+1} equally important?

  ※ Which of the followings do you prefer?

  - I give you $100 right now.
  - I will give you $100 one year later.
  - Or how about: I will give you $200 one year later.

- Future reward is less valuable than present reward.

  ※ R_{t+1} should be given less weight than R_t.

Return

- Definition: Return (aka cumulative future reward).

  ※ $U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \dots$ until the game is over.

- Definition: Discounted return (aka cumulative *discounted* future reward).

  ※ γ : *discount rate* -- tuning hyper-parameter.

  ※ $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \dots$

- At time step t, the return U_t is random.

  ※ Two sources of randomness:

  1. Action can be random: $P[A = a | S = s] = \pi(a|s)$
  2. New state can be random: $P[S' = s' | S = s, A = a] = p(s'|s, a)$

Randomness in Returns

- Definition: Discounted Return (at time step t).

  ※ $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \dots$

- At time step t, the return U_t is random.

  1. Action can be random: $P[A = a | S = s] = \pi(a|s)$
  2. New state can be random: $P[S' = s' | S = s, A = a] = p(s'|s, a)$

- For any $i \geq t$, the reward R_i depends on S_i and A_i.

  ※ Thus, given s_t, the return U_t depends on the random variables:

  - $A_t, A_{t+1}, A_{t+2}, \dots$ and $S_{t+1}, S_{t+2}, \dots$

12.1.3 Value functions

Action-Value Function Q(s,a)

- Definition: Discounted Return (cumulative discounted future reward).

※ $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \ldots$

- Definition: Action-value function for policy π

※ U_t depends on actions $A_t, A_{t+1}, A_{t+2}, \ldots$ and states $S_{t+1}, S_{t+2}, \ldots$

※ $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$

※ Actions are random: $P[A = a | S = s] = \pi(a|s)$ (Policy function)

※ States are random: $P[S' = s' | S = s, A = a] = p(s'|s, a)$ (State transition)

Action-Value Function Q(s,a)

- Definition: Discounted Return (cumulative discounted future reward).

※ $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \ldots$

- Definition: Action-value function for policy $\boldsymbol{\pi}$

※ $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$

- Definition: Optimal action-value function

※ $Q^*(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t)$

State-Value Function V(s)

- Definition: Discounted Return (cumulative discounted future reward).

※ $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \dots$

- Definition: Action-value function for policy π

※ $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$

- Definition: State-value function

※ $V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)]$ $\qquad A \sim \pi(\cdot|s)$

※ $V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$ (if Actions are discrete)

※ $V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)] = \int \pi(a|s_t) \cdot Q_\pi(s_t, a) da$ (Actions are continuous)

Understanding the Value Functions

- Action-value function: $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$

※ For policy π, $Q_\pi(s, a)$ evaluates how good it is for an agent to pick action a while being in state s.

- State-value function: $V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)]$

※ For fixed policy π, $V_\pi(s)$ evaluates how good the situation is in state s.

※ $E_s[V_\pi(s)]$ evaluates how good the policy π is.

12.1.4 Play games using reinforcement learning

How does AI control the agent?

- Suppose we have a good policy $\pi(a|s)$.
  - ※ Upon observe the state s_t,
  - ※ random sampling: $a_t \sim \pi(\cdot|s_t)$.

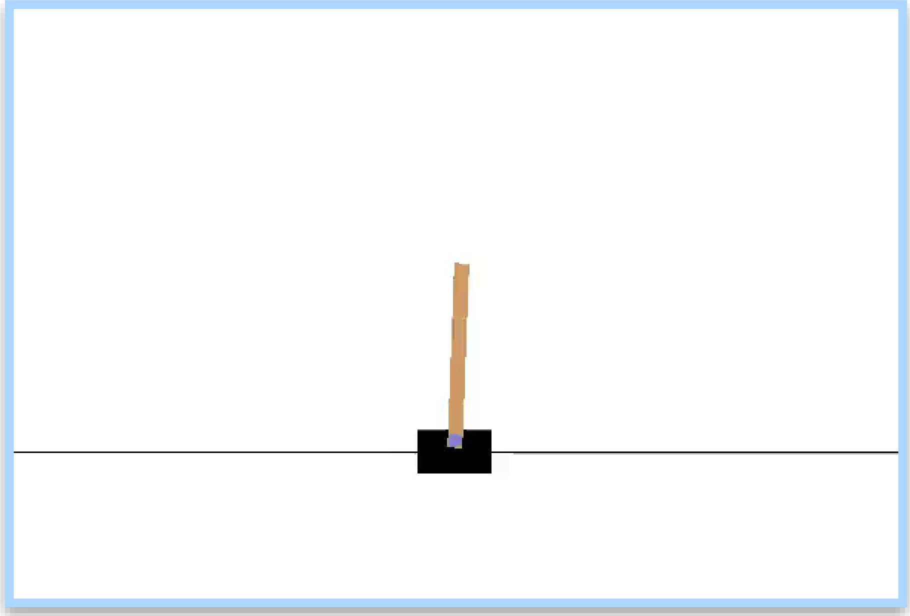
- Suppose we know the optimal action-value function $Q^*(s, a)$
  - ※ Upon observe the state s_t,
  - ※ choose the action that maximizes the value: $a_t = \text{argmax}_{\boldsymbol{a}} Q^*(s_t, a)$.

OpenAI Gym

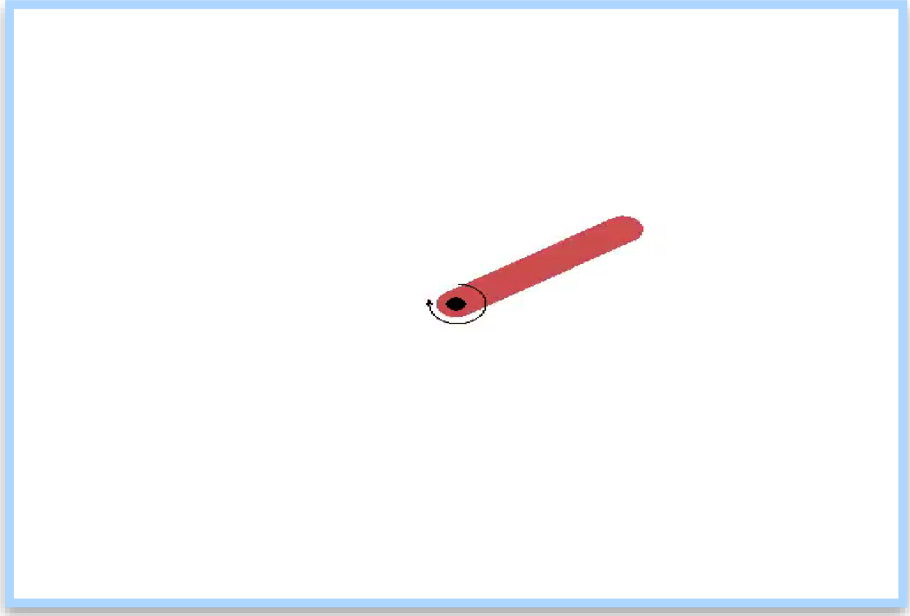
- Gym is a toolkit for developing reinforcement learning algorithms.

  ※ https://gym.openai.com/

- Problem setting 1： Classical control problems



Cart Pole

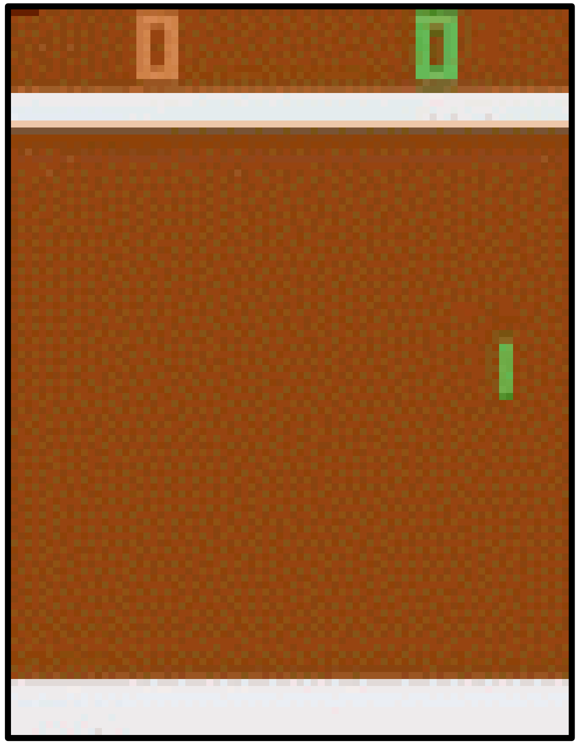


Pendulum

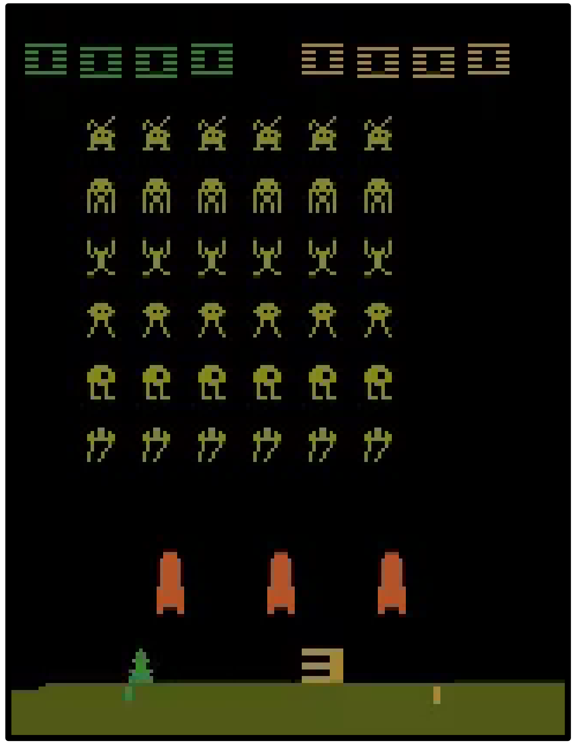
OpenAI Gym

- Gym is a toolkit for developing reinforcement learning algorithms.

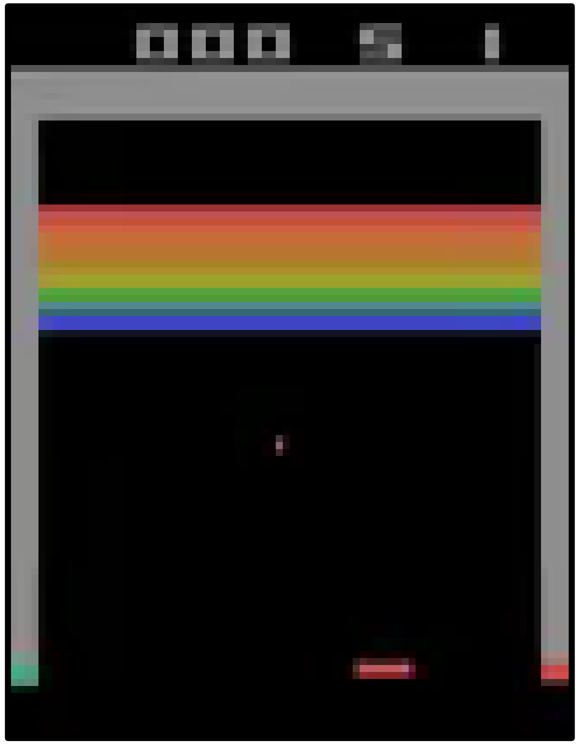
- Problem setting 2： Atari Games



Pong



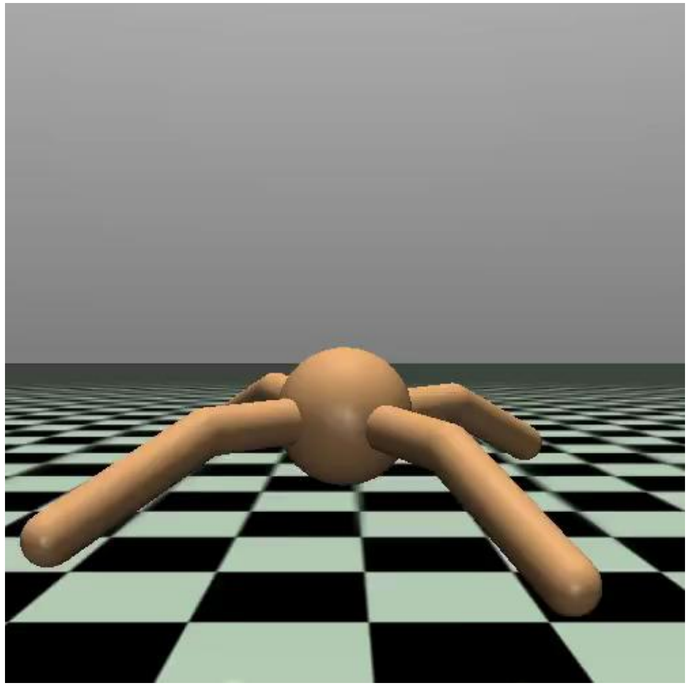
Space Invader



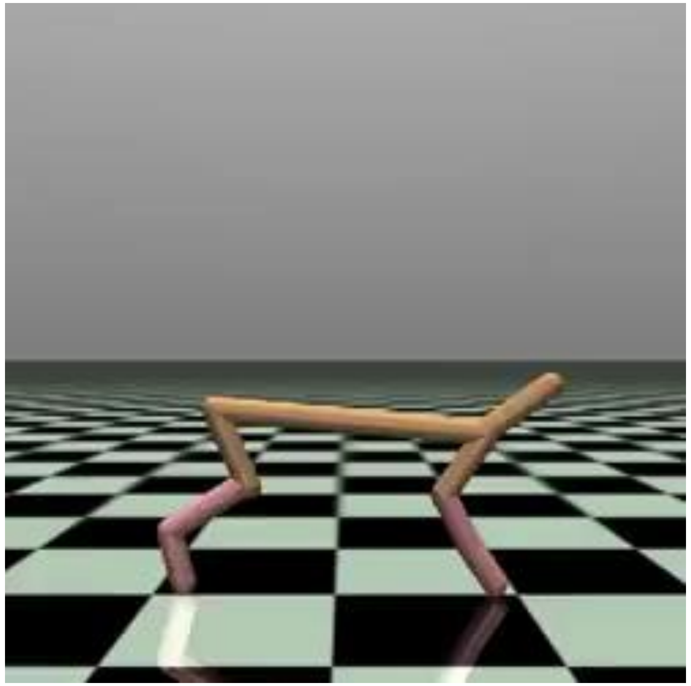
Breakout

OpenAI Gym

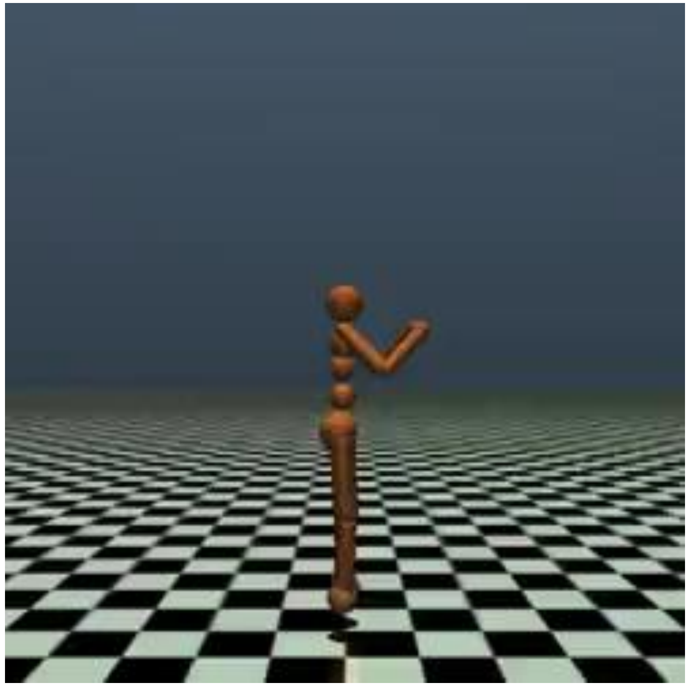
- Gym is a toolkit for developing reinforcement learning algorithms.
- Problem setting 3： MuJoCo (Advanced Physics Simulation)



Ant



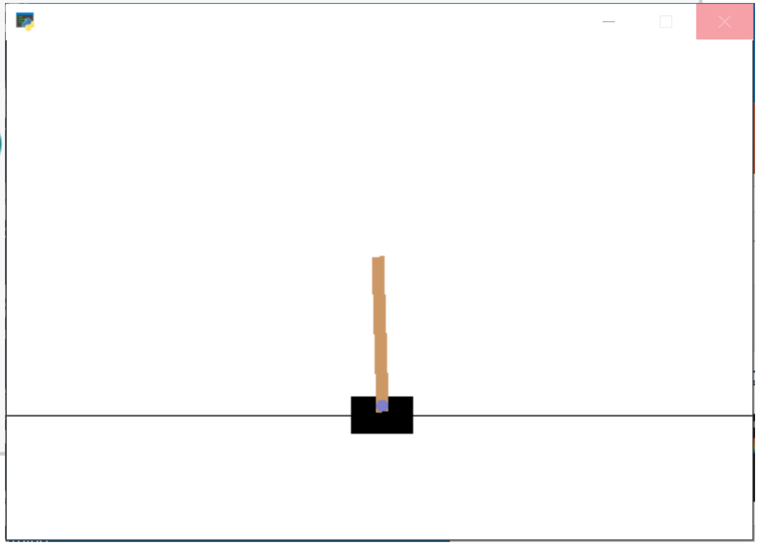
HalfCheetah



Humanoid

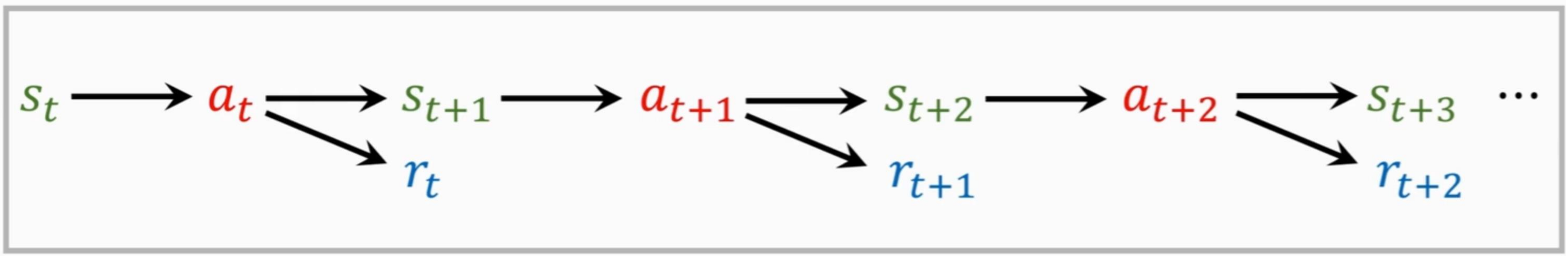
Play CartPole Game

```
import gym
env = gym.make('CartPole-v0') # 生成环境
state = env.reset()
for t in range(100):
    env.render()  # 弹出环境渲染窗口
    print(state)  # [0.01850658  0.01749877 -0.03132206  0.01806279]

    action = env.action_space.sample() # take a random action
    state,reward, done, info = env.step(action)

    if done: # done == 1 means finished (win or loose)
        print('Mission terminated.')
        break
env.close()
```



Play games using reinforcement learning

- Observe state s_t, make action a_t, environment gives s_{t+1} and reward r_t.



- The agent can be controlled by either $\pi(a|s)$ or $Q^*(s, a)$.

12.2 Value-based Reinforcement learning

Recall: Discounted Return & Action-Value Function

- Definition: Discounted Return (cumulative discounted future reward).

※ $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \dots$

- Definition: Action-value function for policy π

※ U_t depends on actions $A_t, A_{t+1}, A_{t+2}, \dots$ and states $S_{t+1}, S_{t+2}, \dots$

※ $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$ $\quad Q^*(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t)$

※ Actions are random: $P[A = a | S = s] = \pi(a|s)$ (Policy function)

※ States are random: $P[S' = s' | S = s, A = a] = p(s'|s, a)$ (State transition)

Recall: Action-Value Function Q(s,a)

- Definition: Discounted Return (cumulative discounted future reward).

※ $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \ldots$

- Definition: Action-value function for policy π

※ $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$

- Definition: Optimal action-value function

※ $Q^*(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t)$

※ Whatever policy function T is used, the result of taking at at state st cannot be better than Q* (st, at).

12.2.1 Deep Q Network (DQN)

Approximate the Q Function

- Goal: Win the game ($\approx$ maximize the total reward.)
- Question: If we know $Q^*(s, a)$, what is the best action?

  ※ Obviously, the best action is: $a^* = \underset{a}{\operatorname{argmax}} \; Q^*(s, a)$

- Q* is an indication for how good it is

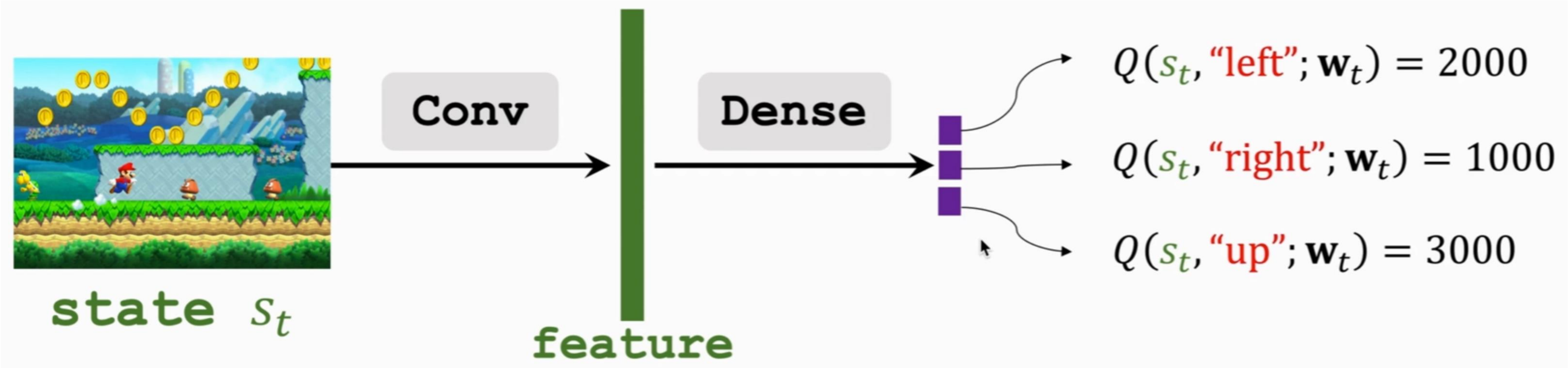
  ※ for an agent to pick action a while being in state s.

- Challenge: We do not know $Q^*(s, a)$.
- Solution: Deep Q Network (DQN)

  ※ Use neural network $Q(s, a; \mathbf{w})$ to approximate $Q^*(s, a)$.

Deep Q Network (DQN)

- Input shape: size of the screenshot.
- Output shape: dimension of action space.



- Question: Based on the predictions, what should be the action?

Apply DQN to Play Game

$$s_{t+2} \sim p(\cdot \,|\, s_{t+1}, a_{t+1})$$

$$s_{t+1} \sim p(\cdot \,|\, s_t, a_t) \qquad\qquad s_{t+3} \sim p(\cdot \,|\, s_{t+2}, a_{t+2})$$

$$\begin{tikzcd}
s_t \arrow[r] & a_t \arrow[r] \arrow[dr] & s_{t+1} \arrow[r] & a_{t+1} \arrow[r] \arrow[dr] & s_{t+2} \arrow[r] & a_{t+2} \arrow[r] \arrow[dr] & s_{t+3} \;\cdots\cdots \\
 & & r_t & & r_{t+1} & & r_{t+2}
\end{tikzcd}$$

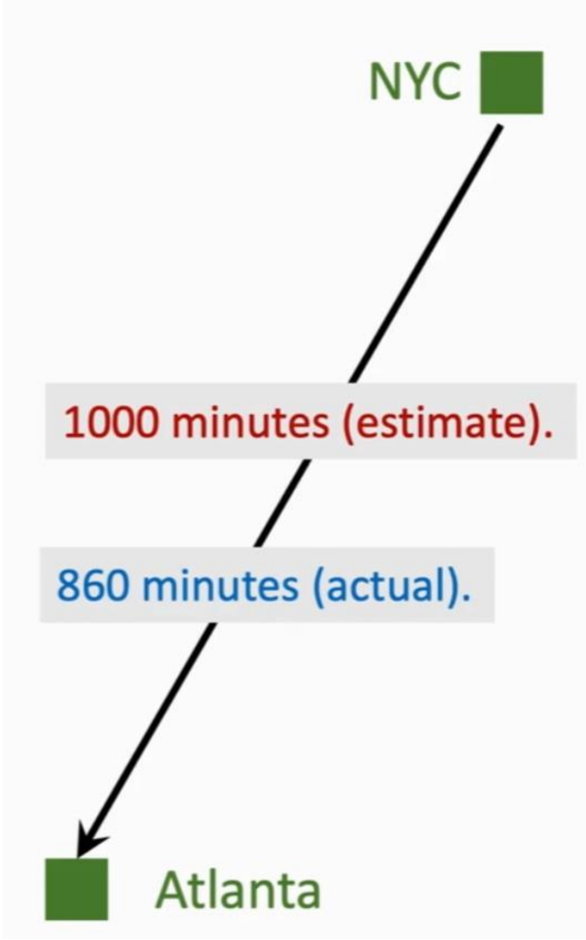
$$a_t = \underset{a}{argmax}\, Q(s_t, a; \mathbf{w}) \qquad\qquad a_{t+2} = \underset{a}{argmax}\, Q(s_{t+2}, a; \mathbf{w})$$

$$a_{t+1} = \underset{a}{argmax}\, Q(s_{t+1}, a; \mathbf{w})$$

12.2.2 Temporal Different (TD)

Example

- Alice want to drive from NYC to Atlanta.
  - ※ Model Q(w) estimates the time cost, e.g., 1000 minutes.
- Question: How do l update the model?
  - ※ Make a prediction: q = Q(w), e.g., q =1000.
  - ※ Finish the trip and get the target y, e.g., y = 860.
  - ※ Loss: $L = \frac{1}{2}(q - y)^2$
  - ※ Gradient: $\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial q}{\partial \mathbf{w}} \cdot \frac{\partial L}{\partial q} = (q - y) \cdot \frac{\partial Q(\mathbf{w})}{\partial \mathbf{w}}$
  - ※ Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L}{\partial \mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}_t}$



Example

- Alice want to drive from NYC to Atlanta.
  - ※ Model Q(w) estimates the time cost, e.g., 1000 minutes.
- Question: How do l update the model?
  - ※ Can we update the model before finishing the trip?
  - ※ Can we get a better **w** as soon as we arrived DC?

Temporal Different (TD) Learning

- Model's estimate:

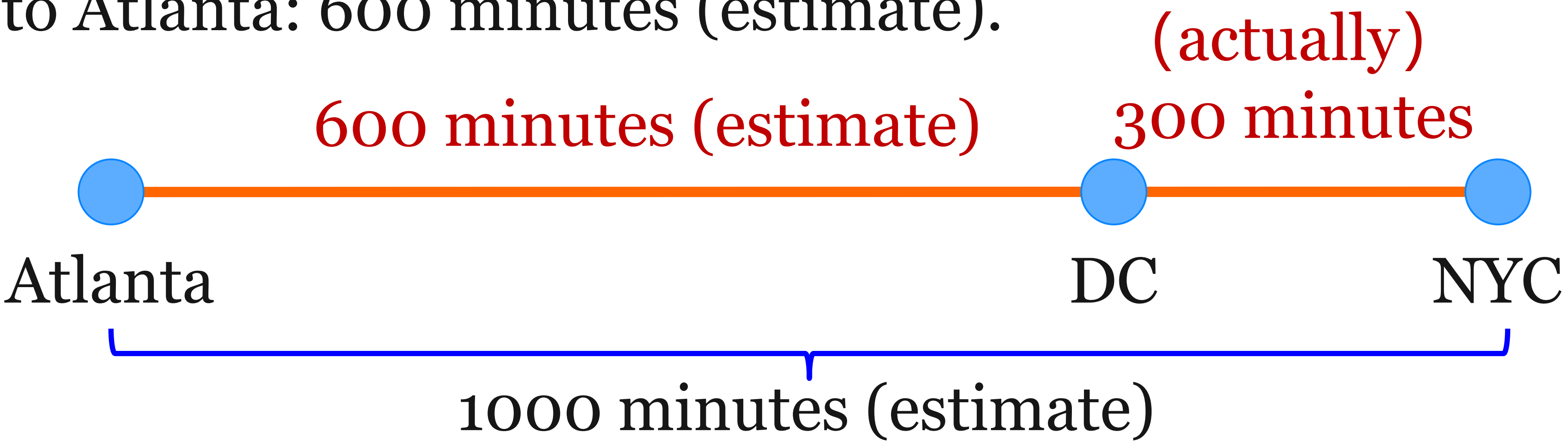
  ※ NYC to Atlanta: 1000 minutes (estimate).

- Alice arrived at DC; actual time cost:

  ※ NYC to DC: 300 minutes (actual).

- Model now updates its estimate:

  ※ DC to Atlanta: 600 minutes (estimate).



Temporal Different (TD) Learning

- Model's estimate: Q(w) = 1000 minutes..

900 minutes --> TD target

  ※ Updated estimate: 300+600 = 900 minutes.

- TD target y = 900 is a more reliable estimate than 1000.

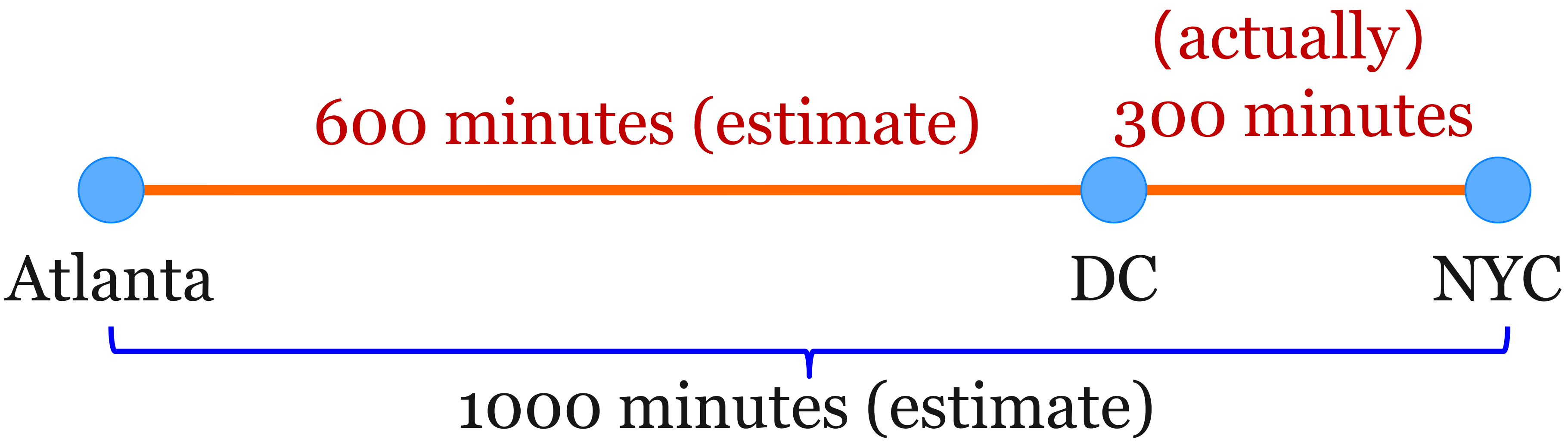
  ※ Loss: $L = \frac{1}{2}(Q(\mathbf{w}) - y)^2$

  ※ Gradient: $\frac{\partial L}{\partial \mathbf{w}} = (1000 - 900) \cdot \frac{\partial Q(\mathbf{w})}{\partial \mathbf{w}}$

  ※ Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L}{\partial \mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}_t}$

Why does TD Learning Work

- Model's estimates:
  - ※ NYC to Atlanta: 1000 minutes.
  - ※ DC to Atlanta: 600 minutes.
  - ※ → NYC to DC: 400 minutes.
- Ground truth:
  - ※ NYC to DC: 300 minutes.
  - ※ TD error: $\delta = 400 - 300 = 100$



How to apply TD learning to DQN?

- In the "driving time" example, we have the equation:

$$T_{\text{NYC}\to\text{ATL}} \approx T_{\text{NYC}\to\text{DC}} + T_{\text{DC}\to\text{ATL}}$$

Model's estimate = Actual time + Model's estimate

- In deep reinforcement learning:

$$Q(s_t, a_t; \mathbf{w}) \approx r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w})$$

How to apply TD learning to DQN?

- Recall: Definition: Discounted Return

※ $$U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \gamma^4 R_{t+4} + \ldots$$
$$= R_t + \gamma \cdot (R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots)$$
$$= R_t + \gamma \cdot U_{t+1}$$

- TD learning for DQN:

※ DQN's output, $Q(s_t, a_t; \mathbf{w})$, is estimate of $E[U_t]$.

※ DQN's output, $Q(s_{t+1}, a_{t+1}; \mathbf{w})$, is estimate of $E[U_{t+1}]$.

※ Thus, $\underbrace{Q(s_t, a_t; \mathbf{w})}_{\text{Prediction}} \approx \underbrace{r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w})}_{\text{TD Target}}$

Train DQN using TD learning

- Prediction: $Q(s_t, a_t; \mathbf{w}_t)$
- TD target:

$$y_t = r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w}_t)$$

$$= r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t)$$

- Loss: $L_t = \frac{1}{2}[Q(s_t, a_t; \mathbf{w}) - y_t]^2$
- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L_t}{\partial \mathbf{w}}\big|_{\mathbf{w}=\mathbf{w}_t}$

Summary: Value-Based Reinforcement Learning

- Definition: Optimal action-value function

$$Q^*(s_t, a_t) = \max_{\pi} \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$$

- DQN: Approximate Q*(s, a) using a neural network (DQN).

※ Q(s, a; **w**) is a neural network parameterized by **w**.

※ Input: observed state *s*.

※ Output: scores for every action $a \in \mathcal{A}$

Temporal Difference (TD) Learning

- Algorithm: One iteration of TD learning.

1. Observe state $S_t = s_t$ and action $A_t = a_t$.
2. Predict the value: $q_t = Q(s_t, a_t; \mathbf{w}_t)$
3. Differentiate the value network: $d_t = \frac{\partial Q(s_t, a_t; \mathbf{w})}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$
4. Environment provides new state s_{t+1} and reward r_t.
5. Compute TD target: $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t)$
6. Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot (q_t - y_t) \cdot d_t$

DeepMind's DQN playing Breakout

12.3 Policy-based Reinforcement learning

12.3.1 Policy network

Policy Function Approximation

Recall:Policy function $\pi(a|s)$

- Policy function $\pi(a|s)$ is a probability density function (PDF)
- It takes state *s* as input.
- It output the probabilities for all the actions, e.g.,

$$\pi(\text{left}|s) = 0.2$$

$$\pi(\text{right}|s) = 0.1$$

$$\pi(\text{up}|s) = 0.7$$

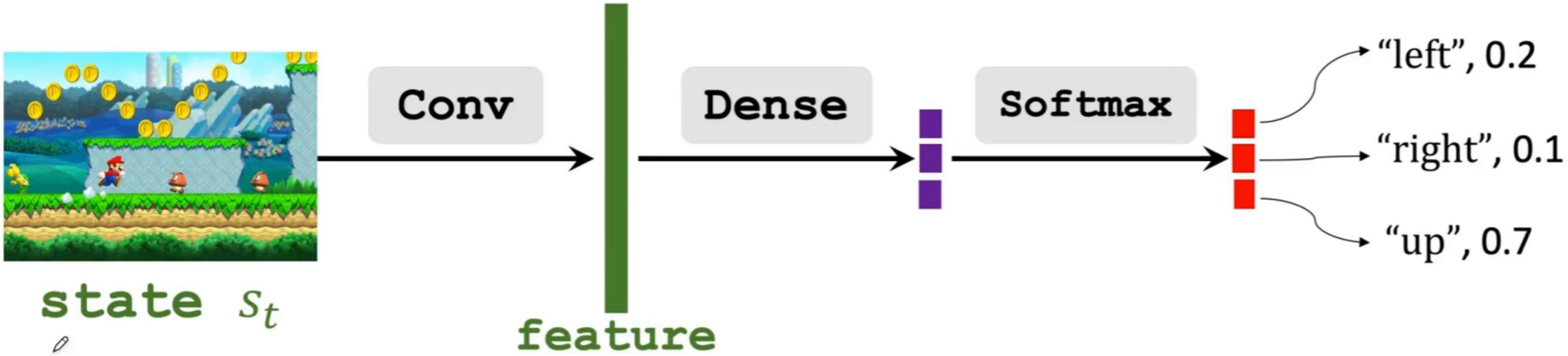
- The agent performs an action *a* random drawn from the distribution.

Policy Network $\pi(a|s,\theta)$

- Policy network: Use a neural net to approximate $\pi(a|s)$

※ Use policy network $\pi(a|s, \boldsymbol{\theta})$ to approximate $\pi(a|s)$.

※ $\boldsymbol{\theta}$: trainable parameters of the neural net.

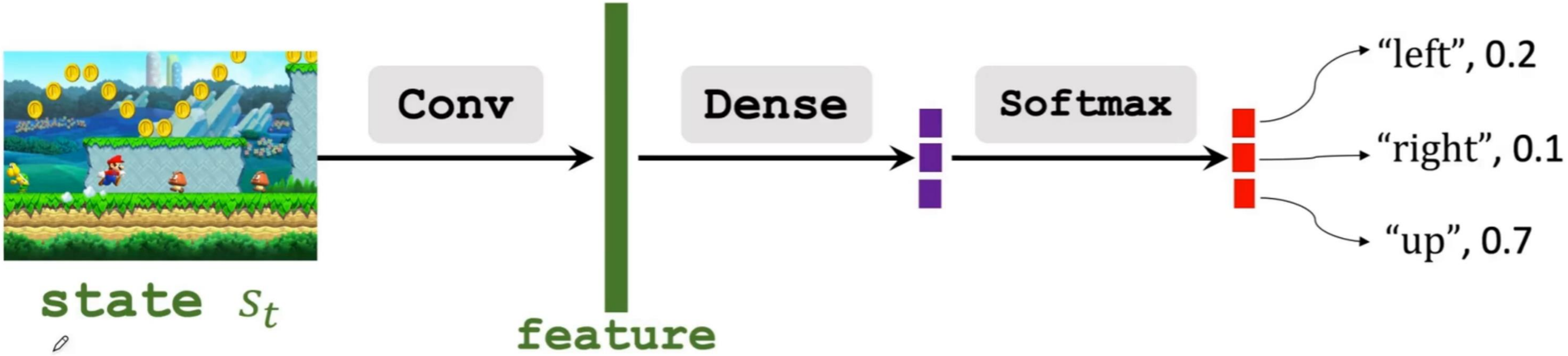


Policy Network $\pi(a|s, \theta)$

$$\sum_{a \in \mathcal{A}} \pi(a|s, \theta) = 1$$

- Where: $\mathcal{A} = \{\text{"left"}, \text{"right"}, \text{"up"}\}$ is the set of all actions.

※ That is why we use *softmax* activation



12.3.2 State-Value Function Approximation

Recall: State-Value Function V(s)

- Definition: Discounted Return (cumulative discounted future reward).

※ $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \ldots$

- Definition: Action-value function for policy π

※ $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$

- Definition: State-value function

※ $V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)]$ $\qquad A \sim \pi(\cdot|s)$

※ $V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$ (if Actions are discrete)

※ $V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)] = \int \pi(a|s_t) \cdot Q_\pi(s_t, a) da$ (Actions are continuous)

Policy-Based Reinforcement Learning

- Definition: State-value function.

  ※ $V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$

- Approximate state-value function.

  ※ Approximate policy function $\pi(a|s_t)$ by policy network $\pi(a|s_t, \boldsymbol{\theta})$

  ※ Approximate value function $V_\pi(s_t)$ by:

$$V(s_t; \boldsymbol{\theta}) = \sum_a \pi(a|s_t; \theta) \cdot Q_\pi(s_t, a)$$

Policy-Based Reinforcement Learning

- Definition: Approximate state-value function.

$$V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$$

- Policy-based learning: Learn $\boldsymbol{\theta}$ that maximizes $J(\boldsymbol{\theta}) = \mathbb{E}_S[V(S;\boldsymbol{\theta})]$

- How to improve $\boldsymbol{\theta}$? Policy gradient ascent!

※ observe state s.

※ Update policy by: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \cdot \frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$

12.3.3 Policy gradient

Policy Gradient

- Definition: Approximate state-value function.

$$V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$$

- Policy gradient: Derivative of $V(s;\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$

$$\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial \sum_a \pi(a|s;\boldsymbol{\theta}) \cdot Q_\pi(s,a)}{\partial \boldsymbol{\theta}}$$

$$= \sum_a \frac{\partial \pi(a|s;\boldsymbol{\theta}) \cdot Q_\pi(s,a)}{\partial \boldsymbol{\theta}}$$ Push derivative inside the summation

$$= \sum_a \frac{\partial \pi(a|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s,a)$$ Pretend Q_π is independent of θ.

(It may not be true.)

Policy Gradient

- Definition: Approximate state-value function.

$$V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$$

- Policy gradient: Derivative of $V(s;\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$

$$\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_a \frac{\partial \pi(a|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s,a)$$

$$= \sum_a \pi(a|s;\boldsymbol{\theta}) \cdot \frac{\partial \log \pi(a|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s,a)$$

Chain rule: $\dfrac{\partial \log[\pi(\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} = \dfrac{1}{\pi(\boldsymbol{\theta})} \cdot \dfrac{\partial \pi(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$

$$\pi(\boldsymbol{\theta}) \cdot \frac{\partial \log[\pi(\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} = \pi(\boldsymbol{\theta}) \cdot \frac{1}{\pi(\boldsymbol{\theta})} \cdot \frac{\partial \pi(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial \pi(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

Policy Gradient

- Definition: Approximate state-value function.

$$V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$$

- Policy gradient: Derivative of $V(s;\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$

$$\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_a \frac{\partial \pi(a|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s,a)$$

$$= \sum_a \pi(a|s;\boldsymbol{\theta}) \cdot \frac{\partial \log \pi(a|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s,a)$$

$$= \mathbb{E}_A\Big[\frac{\partial \log[\pi(A|s;\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s,A)\Big]$$

Note: This derivation is over-simplified and not rigorous.

Policy Gradient

- Two forms of policy gradient:

※ Form 1: $$\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_a \frac{\partial \pi(a|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s, a)$$

※ Form 2: $$\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbb{E}_{A \sim \pi(\cdot|s;\boldsymbol{\theta})} \left[\frac{\partial \log \pi(A|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s, A) \right]$$

Calculate Policy Gradient for Discrete Actions

- If the actions are discrete, e.g., action space $\mathcal{A} = \{\text{"left"}, \text{"right"}, \text{"up"}\}$

- Use Form 1: $\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_a \frac{\partial \pi(a|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s,a)$

  1. Calculate $f(a,\theta) = \frac{\partial \pi(a|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s,a)$, for every action $a \in \mathcal{A}$

  2. Policy gradient: $\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = f(\text{"left"}, \boldsymbol{\theta}) + f(\text{"right"}, \boldsymbol{\theta}) + f(\text{"up"}, \boldsymbol{\theta})$

- This approach does not work for continuous actions.

Calculate Policy Gradient for Continuous Actions

- If the actions are continuous, e.g., action space $\mathcal{A} = [0, 1], \ldots$

- Use Form 2: $\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbb{E}_{A\sim\pi(\cdot|s;\boldsymbol{\theta})}\left[\frac{\partial \log \pi(A|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s, A)\right]$

  1. Randomly sample an action a according to the PDF $\pi(\cdot|s;\boldsymbol{\theta})$

  2. Calculate $g(\hat{a}, \boldsymbol{\theta}) = \frac{\partial \log \pi(\hat{a}|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s, \hat{a})$

- Obviously, $\mathbb{E}_A[g(A, \boldsymbol{\theta})] = \frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$

- $g(\hat{a}, \boldsymbol{\theta})$ is an unbiased estimate of $\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$

Calculate Policy Gradient for Continuous Actions

- If the actions are continuous, e.g., action space $\mathcal{A} = [0, 1], \dots$

- Use Form 2: $\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbb{E}_{A \sim \pi(\cdot|s;\boldsymbol{\theta})} \left[\frac{\partial \log \pi(A|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s, A) \right]$

1. Randomly sample an action a according to the PDF $\pi(\cdot|s;\boldsymbol{\theta})$

2. Calculate $g(\hat{a}, \boldsymbol{\theta}) = \frac{\partial \log \pi(\hat{a}|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s, \hat{a})$

3. Use $g(\hat{a}, \boldsymbol{\theta})$ as an approximation to the policy gradient $\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$

- This approach work for discrete actions.

Update policy network using policy gradient

Algorithm

1. Observe the state s_t.

2. Randomly sample action a_t according to $\pi(\cdot|s_t;\boldsymbol{\theta}_t)$

3. Compute $q_t \approx Q_\pi(s_t, a_t)$ (some estimate).

4. Differentiate policy network: $d_{\boldsymbol{\theta},t} = \frac{\partial \log \pi(a_t|s_t;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$

5. (Approximate) policy gradient: $g(a_t, \boldsymbol{\theta}_t) = q_t \cdot d_{\boldsymbol{\theta},t}$

6. Update policy network: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \beta \cdot g(a_t, \boldsymbol{\theta}_t)$

Algorithm

3. Compute $q_t \approx Q_\pi(s_t, a_t)$ (some estimate). How?

- Option 1: REINFORCE.

※ Play the game to the end and generate the trajectory:

$$s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T, a_T, r_T$$

※ Compute the discounted return $u_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$, for all t.

※ Since $Q_\pi(s_t, a_t) = \mathbb{E}[U_t]$, we can use u_t to approximate $Q_\pi(s_t, a_t)$

※ Therefore: Use $q_t = u_t$

Algorithm

3. Compute $q_t \approx Q_\pi(s_t, a_t)$ (some estimate). How?

- Option 2: Approximate Q_π using a neural network.

※ This leads to the actor-critic method.

Policy-Based Method

- If a good policy function π is known, the agent can be controlled by the policy: randomly sample $a_t \sim \pi(\cdot \mid s_t)$.

- Approximate policy function $\pi(a|s)$ by policy network $\pi(a|s; \theta)$

- Learn the policy network by policy gradient.

- Policy gradient algorithm learn θ that maximizes $\mathbb{E}_S[V(S; \boldsymbol{\theta})]$

12.4 Actor-Critic Methods

12.4.1 Value Network and Policy Network

State-Value Function Approximation

- Definition: State-value function.

$$V_\pi(s) = \sum_a \pi(a|s) \cdot Q_\pi(s,a) \;\approx \sum_a \pi(a|s;\theta) \cdot q(s,a,\mathbf{w})$$

- Policy network (actor):

※ Use neural net $\pi(a|s;\theta)$ to approximate $\pi(a|s)$

※ θ: trainable parameters of the neural net.

- Value network (critic):

※ Use neural net $q(s,a;\mathbf{w})$ to approximate $Q_\pi(s,a)$

※ $\mathbf{w}$: trainable parameters of the neural net.

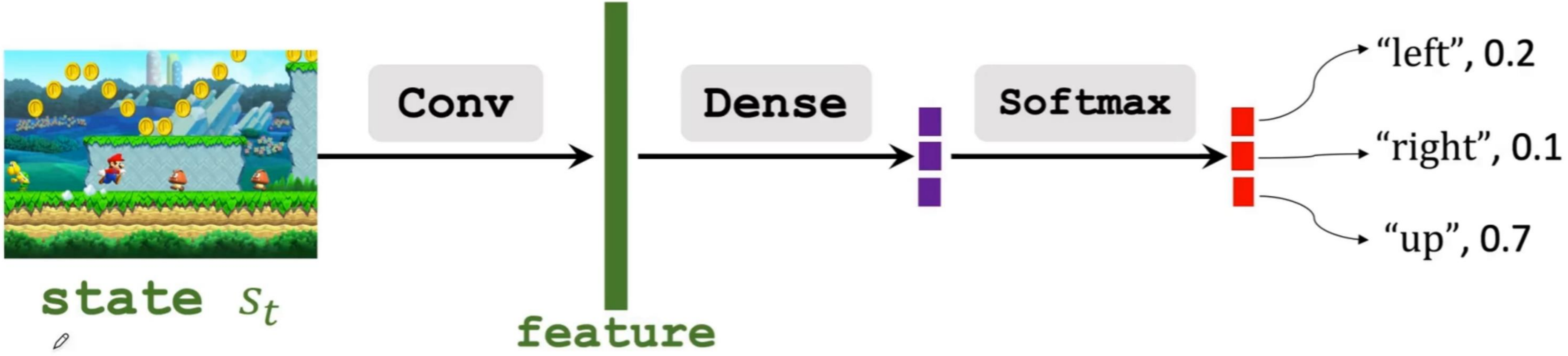
Actor: Policy Network $\pi(a|s, \theta)$

- Input: state s, e.g., a screenshot of Super Mario.
- Output: probability distribution over the actions.

- Let $\mathcal{A} = \{\text{"left"}, \text{"right"}, \text{"up"}\}$ be the set of all actions

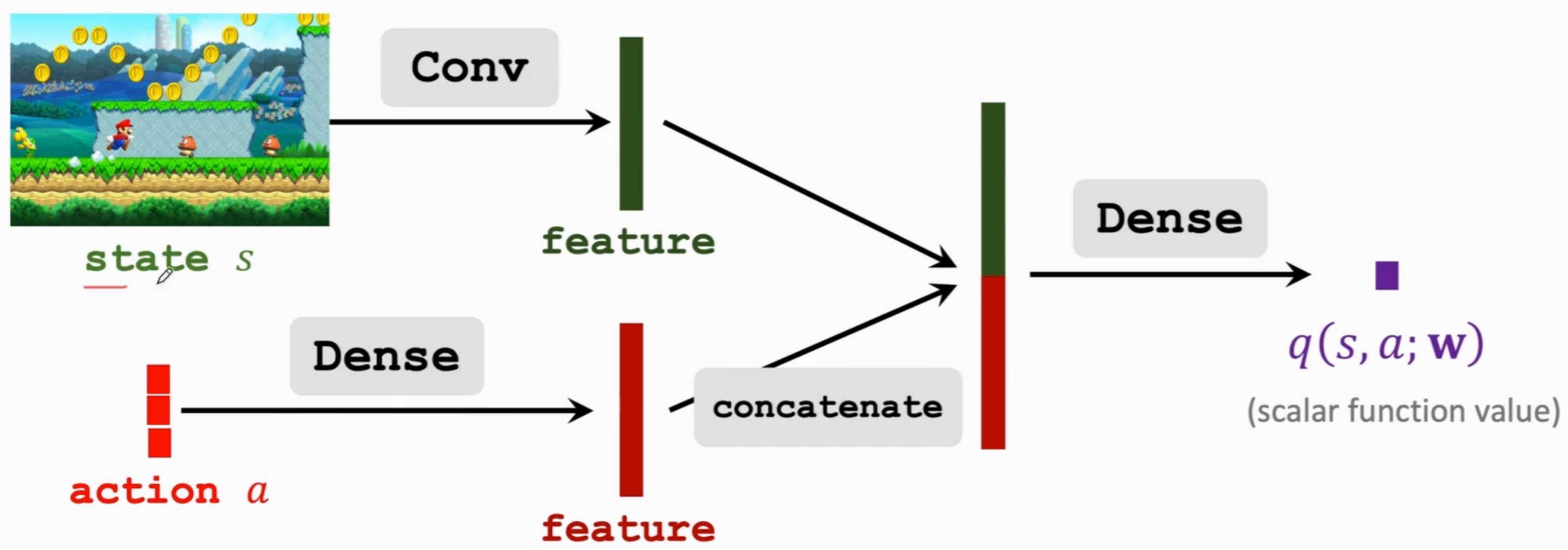
$$\sum_{a \in \mathcal{A}} \pi(a|s, \theta) = 1$$

That is why we use softmax activation



Critic: Value Network $q(s, a; \mathbf{w})$

- Inputs: state s and action a.
- Output: approximate action-value (scalar).



Train the networks

- Definition: State-value function approximated using neural networks.

$$V(s;\boldsymbol{\theta},\mathbf{w}) = \sum_a \pi(a|s;\boldsymbol{\theta}) \cdot q(s,a;\mathbf{w})$$

- Training: Update the parameters $\boldsymbol{\theta}$ and $\mathbf{w}$.

※ Update policy network $\pi(a|s;\boldsymbol{\theta})$ to increase the state-value $V(s;\boldsymbol{\theta},\mathbf{w})$.

➢ Actor gradually performs better.

➢ Supervision is purely from the value network (critic).

※ Update value network $q(s, a; w)$ to better estimate the return.

➢ Critic's judgement becomes more accurate.

➢ Supervision is purely from the rewards.

Train the networks

- Definition: State-value function approximated using neural networks.

$$V(s;\boldsymbol{\theta},\mathbf{w}) = \sum_a \pi(a|s;\boldsymbol{\theta}) \cdot q(s,a;\mathbf{w})$$

- Training: Update the parameters $\boldsymbol{\theta}$ and $\mathbf{w}$.

1. Observe the state s_t.
2. Randomly sample action a_t according to $\pi(\cdot \mid s_t; \boldsymbol{\theta}_t)$.
3. Perform a_t and observe new state s_{t+1} and reward r_t.
4. Update $\mathbf{w}$ (in value network) using temporal difference (TD).
5. Update $\boldsymbol{\theta}$ (in policy network) using policy gradient.

Update value network q using TD

- Compute $q(s_t, a_t; \mathbf{w}_t)$ and $q(s_{t+1}, a_{t+1}; \mathbf{w}_t)$.

- TD target: $y_t = r_t + \gamma \cdot q\big(s_{t+1}, a_{t+1}; \mathbf{w}_t\big)$

- Loss: $L(\mathbf{w}) = \frac{1}{2}\big[q\big(s_t, a_t; \mathbf{w}\big) - y_t\big]^2$

- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}\big|_{\mathbf{w}=\mathbf{w}_t}$

Update policy network π using policy gradient

- Definition: State-value function approximated using neural networks.

$$V(s;\boldsymbol{\theta},\mathbf{w}) = \sum_a \pi(a|s;\boldsymbol{\theta}) \cdot q(s,a;\mathbf{w})$$

- Policy gradient: Derivative of $V(s_t;\boldsymbol{\theta},\mathbf{w})$ w.r.t. $\boldsymbol{\theta}$.

※ Let $g(a,\boldsymbol{\theta}) = \frac{\partial \log \pi(a|s,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot q(s_t,a;\mathbf{w})$

※ $\frac{\partial V(s;\boldsymbol{\theta},\mathbf{w})}{\partial \boldsymbol{\theta}} = \mathbb{E}_A[g(A,\boldsymbol{\theta})]$

- Algorithm: Update policy network using stochastic policy gradient.

※ Random sampling: $a \sim \pi(\cdot|s_t;\boldsymbol{\theta}_t)$ (Thus $g(a,\boldsymbol{\theta})$ is unbiased.)

※ Stochastic gradient ascent: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \beta \cdot g(a,\boldsymbol{\theta}_t)$

Summary of Algorithm

1. Observe state s_t and randomly sample $a_t \sim \pi(\cdot \mid s_t; \theta_t)$.
2. Perform a_t; then environment gives new state s_{t+1} and reward r_t.
3. Randomly sample $\tilde{a}_{t+1} \sim \pi(\cdot | s_{t+1}; \boldsymbol{\theta}_t)$. (Do not perform a_{t+1} !)
4. Evaluate value network: $q_t = q(s_t, a_t; \mathbf{w}_t)$ and $q_{t+1} = q(s_{t+1}, a_{t+1}; \mathbf{w}_t)$
5. Compute TD error: $\delta_t = q_t - (r_t + \gamma \cdot q_{t+1})$
6. Differentiate value network: $d_{\mathbf{w},t} = \frac{\partial q(s_t, a_t; \mathbf{w})}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$
7. Update value network: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \delta_t \cdot d_{\mathbf{w},t}$
8. Differentiate policy network: $d_{\boldsymbol{\theta},t} = \frac{\partial \log \pi(a_t | s_t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$
9. Update policy network: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \beta \cdot q_t \cdot d_{\boldsymbol{\theta},t}$

Summary of Algorithm

……

5. Compute TD error: $\delta_t = q_t - (r_t + \gamma \cdot q_{t+1})$

……

9. Update policy network: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \beta \cdot q_t \cdot d_{\boldsymbol{\theta},t}$

9. Update policy network: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \beta \cdot \delta_t \cdot d_{\boldsymbol{\theta},t}$

Summary : Policy Network and Value Network

- Definition: State-value function.

$$V_\pi(s) = \sum_a \pi(a|s) \cdot Q_\pi(s, a)$$

- Definition: function approximation using neural networks.

※ Approximate policy function $\pi(a|s)$ by $\pi(a|s; \boldsymbol{\theta})$ (**actor**).

※ Approximate value function $Q_\pi(s, a)$ by $q(s, a; \mathbf{w})$ (**critic**).

Roles of Actor and Critic

- During training

  ※ Agent is controlled by policy network (actor): $a_t \sim \pi(\cdot \mid s_t; \boldsymbol{\theta})$.

  ※ Value network q (critic) provides the actor with supervision.

- After training

  ※ Agent is controlled by policy network (actor): $a_t \sim \pi(\cdot \mid s_t; \boldsymbol{\theta})$.

  ※ Value network q (critic) will not be used.

Training

- Learning: Update the policy network (actor) by policy gradient.
  - ※ Seek to increase state-value: $V(s; \boldsymbol{\theta}, \mathbf{w}) = \sum_a \pi(a|s; \boldsymbol{\theta}) \cdot q(s, a; \mathbf{w})$
  - ※ Compute policy gradient: $\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbb{E}\left[\frac{\partial \log \pi(A|s,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot q(s, A; \mathbf{w})\right]$
  - ※ Perform gradient ascent.
- Learning: Update the value network (critic) by TD learning.
  - ※ Predicted action-value: $q_t = q(s_t, a_t; \mathbf{w})$
  - ※ TD target: $y_t = r_t + \gamma \cdot q(s_{t+1}, a_{t+1}; \mathbf{w}_t)$
  - ※ Gradient: $\frac{\partial (q_t - y_t)^2/2}{\partial \mathbf{w}} = (q_t - y_t) \cdot \frac{\partial q(s_t, a_t; \mathbf{w})}{\partial \mathbf{w}}$
  - ※ Perform gradient descent.

References

- Reinforcement Learning: An Introduction
  - ※ By Richard S. Sutton and Andrew G. Barto （代码）
  - ※ http://incompleteideas.net/sutton/book/the-book-2nd.html
- Statistical Reinforcement Learning: Modern Machine Learning Approaches
  - ※ By Masashi Sugiyama （链接）
- Deep Reinforcement Learning (open course)
  - ※ By Shusen Wang
  - ※ https://www.youtube.com/watch?v=vmkRMvhCW5c

End of the course, wish you have a pleasant journey!