

Brief Introduction to Reinforcement Learning

Saurabh Patil, Dhwanit Balwani

BCS IITK

What Reinforcement Learning is, and how rewards are the central idea?

The idea behind Reinforcement Learning is that an agent will learn from the environment by interacting with it and receiving rewards for performing actions. think of this as how we learn not to touch hot objects. Say a baby(our agent) touches the fire, it cries(i.e it is negatively rewarded) whereas if he stays just enough close to the fireplace to get its warmth it is positively rewarded. Reinforcement Learning is just a computational approach of this kind of learning.



Courtesy - OpenAI

In recent years, we've seen a lot of improvements in this fascinating area of research. like beating the champion of the game of Go with AlphaGo in 2016, OpenAI and the PPO in 2017.

**I believe by now you
have a good idea of
what Machine Learning
is about, so i want you
to spot the difference
between RL and
Supervised learning**

Things to keep in mind in RL

1. Objective
2. State
3. Action
4. Reward

Reinforcement Learning

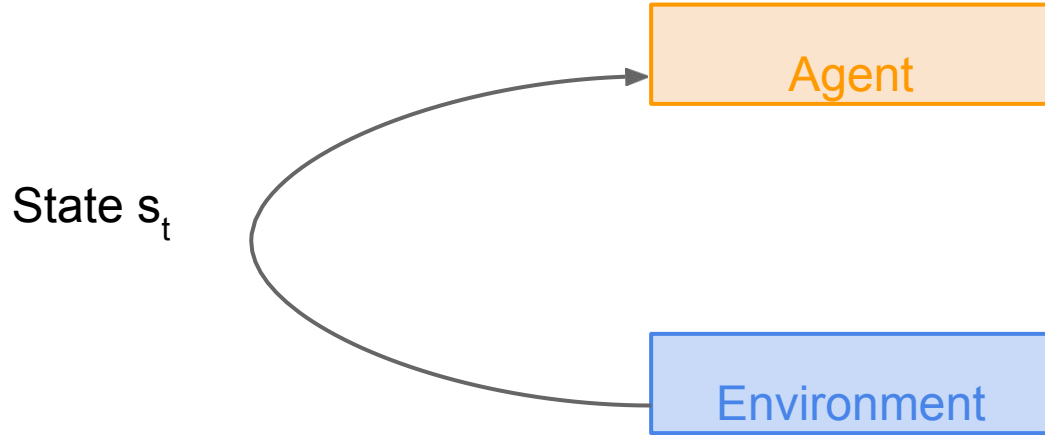
Agent



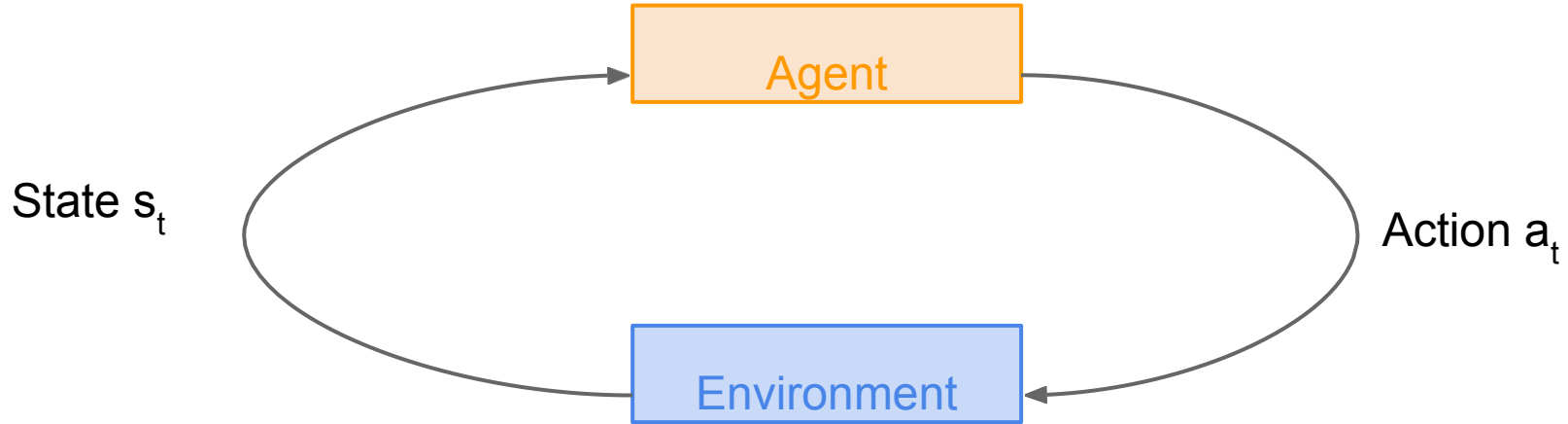
The diagram consists of two rectangular boxes. The top box is light orange with an orange border and contains the word 'Agent' in orange text. The bottom box is light blue with a blue border and contains the word 'Environment' in blue text. The boxes are vertically aligned and centered horizontally.

Environment

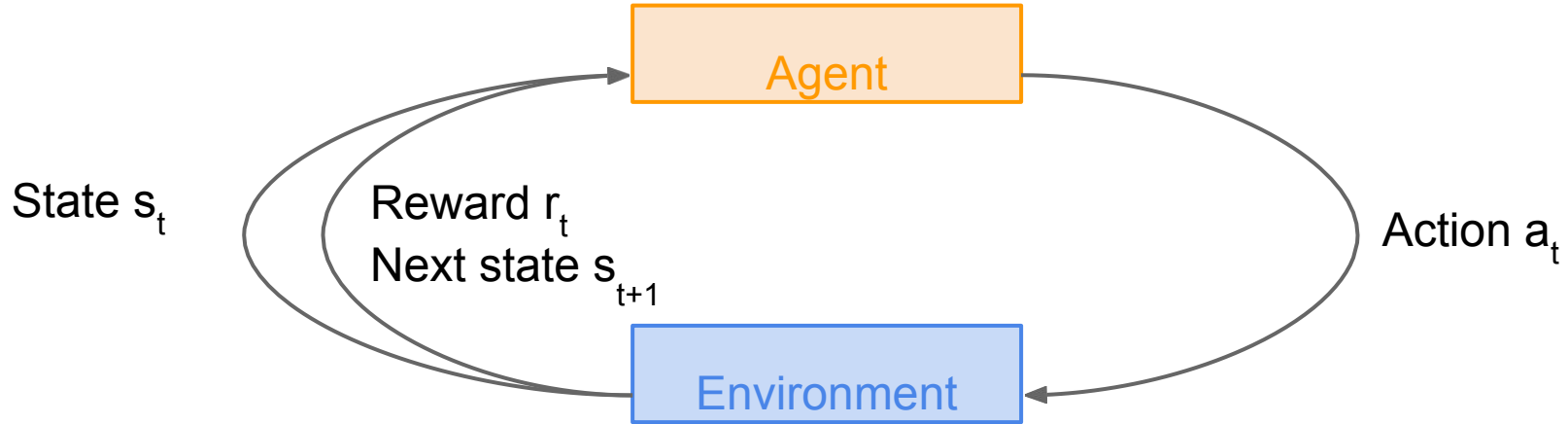
Reinforcement Learning



Reinforcement Learning

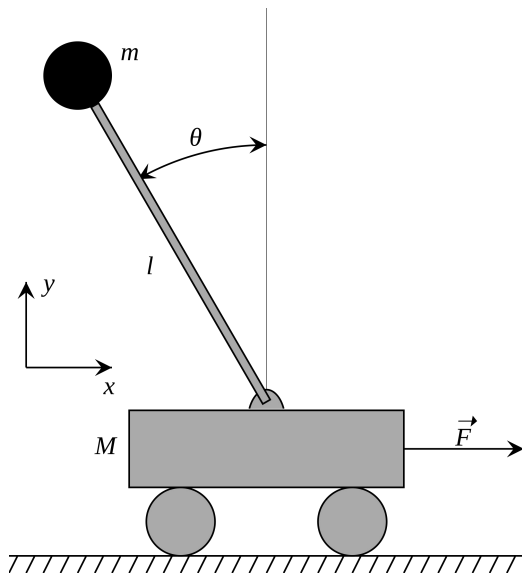


Reinforcement Learning



SOME EXAMPLES

Cart-Pole Problem



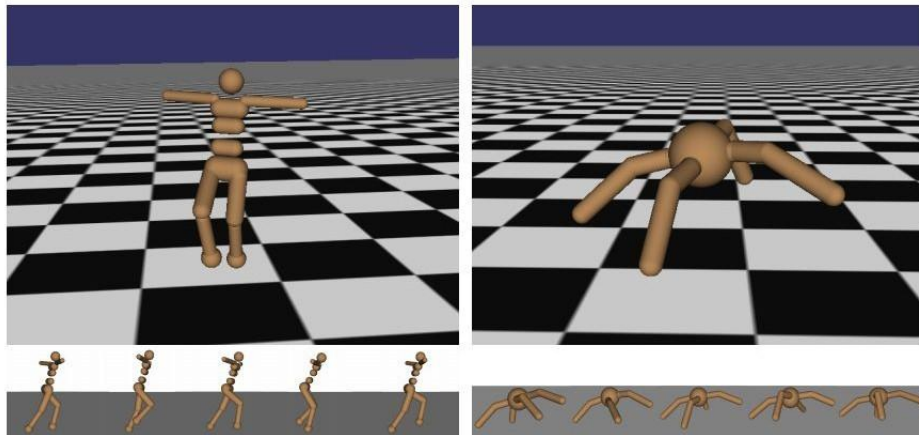
Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity

Action: horizontal force applied on the cart

Reward: 1 at each time step if the pole is upright

Robot Locomotion



Objective: Make the robot move forward

State: Angle and position of the joints

Action: Torques applied on joints

Reward: 1 at each time step upright + forward movement

Atari Games



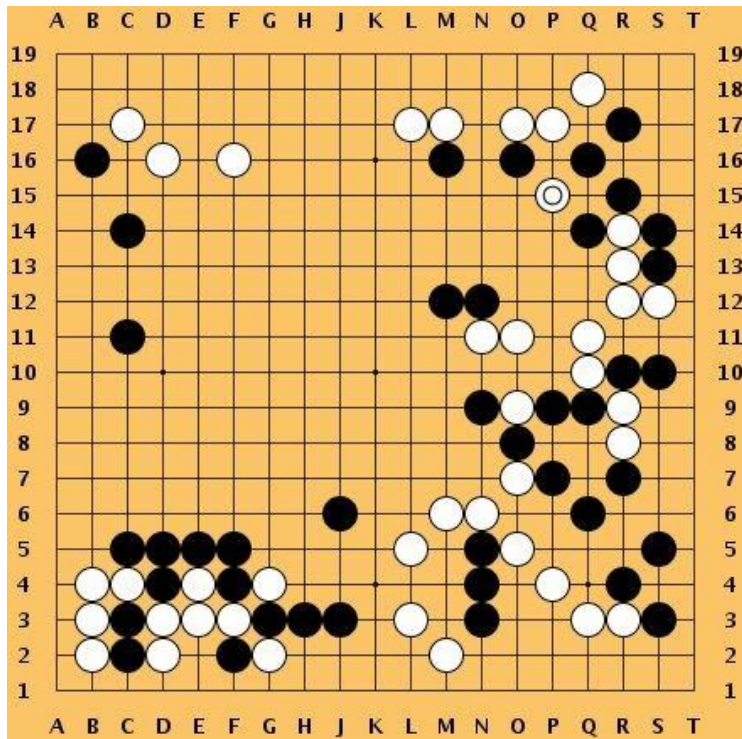
Objective: Complete the game with the highest score

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

Go



Objective: Win the game!

State: Position of all pieces

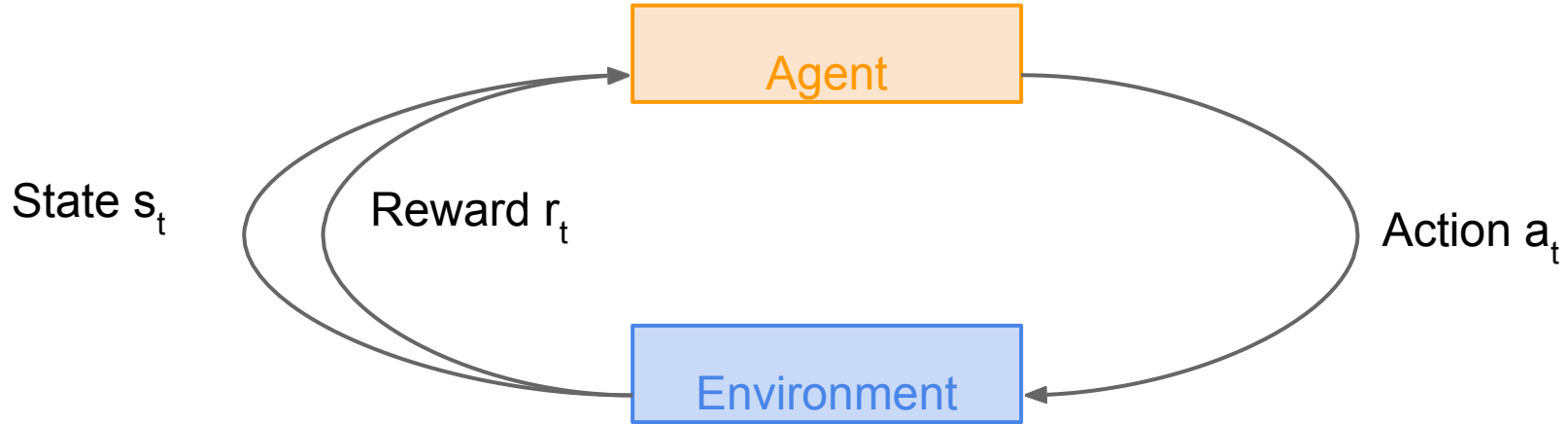
Action: Where to put the next piece down

Reward: 1 if win at the end of the game, 0 otherwise

Overview

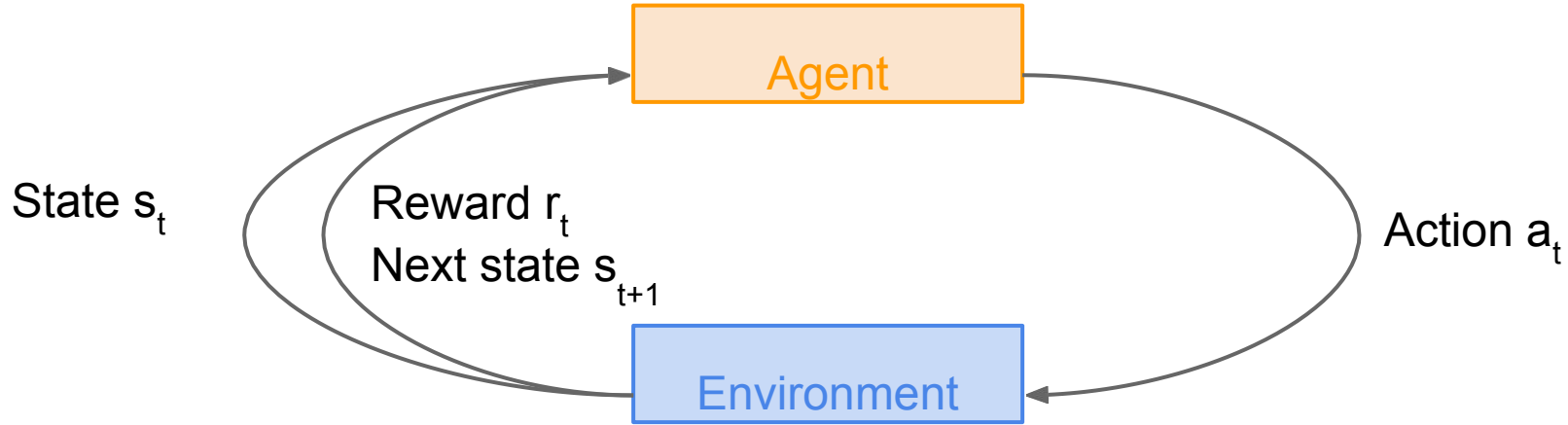
- Markov Decision Processes
- Q-Learning
- Policy Gradients

Reinforcement Learning



Now you got a proper feel for it

So how can we mathematically formalize the RL problem?



Markov Decision Process

- Mathematical formulation of the RL problem
- **Markov property**: Current state completely characterises the state of the world

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

\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 Process

- At time step $t=0$, environment samples initial state $s_0 \sim p(s_0)$
- Then, for $t=0$ until done:
 - Agent selects action a_t
 - Environment samples reward $r_t \sim R(\cdot | s_t, a_t)$
 - Environment samples next state $s_{t+1} \sim P(\cdot | s_t, a_t)$
 - Agent receives reward r_t and next state s_{t+1}
- A policy u is a function from S to A that specifies what action to take in each state
- **Objective:** find policy u^* that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$


A simple MDP: Grid World

actions = {

1. right 

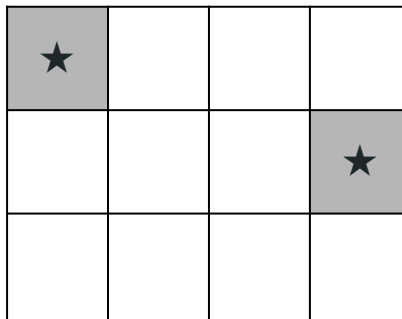
2. left 

3. up 

4. down 

}

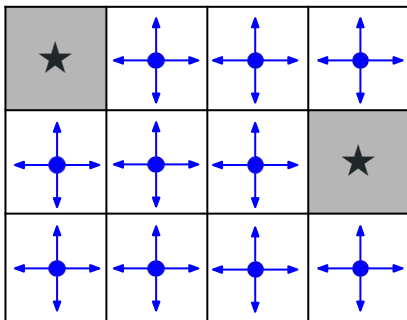
states



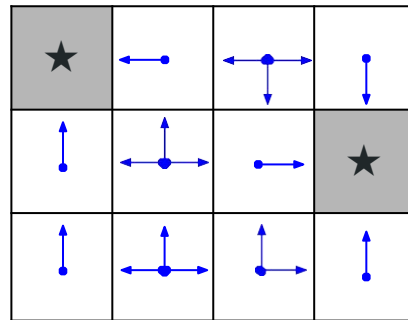
Set a negative “reward”
for each transition
(e.g. $r = -1$)

Objective: reach one of terminal states (greyed out) in
least number of actions

A simple MDP: Grid World



Random Policy



Optimal Policy

The optimal policy u^*

We want to find optimal policy u^* that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability...)? Maximize the **expected sum of rewards!**

Formally: $\pi^* = \arg \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid \pi \right]$ with $s_0 \sim p(s_0)$, $a_t \sim \pi(\cdot \mid s_t)$, $s_{t+1} \sim p(\cdot \mid s_t, a_t)$

Definitions: Value function and Q-value function

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

How good is a state?

The **value function** at state s , is the expected cumulative reward from following the policy from state s :

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

How good is a state-action pair?

The **Q-value function** at state s and action a , is the expected cumulative reward from taking action a in state s and then following the policy:

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

Bellman equation

The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given (state, action) pair:

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

Q^* satisfies the following **Bellman equation**:

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

Intuition: if the optimal state-action values for the next time-step $Q^*(s', a')$ are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s', a')$

The optimal policy u^* corresponds to taking the best action in any state as specified by Q^*

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 $i \rightarrow \infty$

What's the problem with this?

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

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

Solving for the optimal policy: Q-learning

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

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

Forward Pass

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

Loss function:

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$

i

Backward Pass

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

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

Takeaway :

From the working and the applications we can see that we can make a RL model do a variety of tasks that a normal person can do and that too better than a skilled professional.

What will you do
next?



Assignment

Notebook link :

https://colab.research.google.com/drive/16kTpN_vTz1P3oOyT4xDABF8EhHLBGog3?usp=sharing

Gym : <https://gym.openai.com/>