Reinforcement Learning

AICTE-MARGDARSHAN sponsored workshop on Computational   Intelligence for Multimedia

T. T. Mirnalinee
T. T. Mirnalinee
Prof/CSE
SSN College of Engineering
mirnalineett@ssn.edu.in

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

$\longrightarrow$ Cat

Classification

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

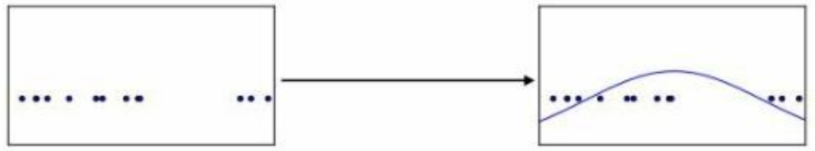
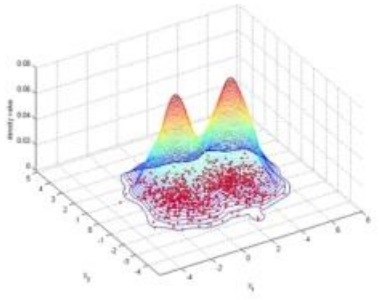
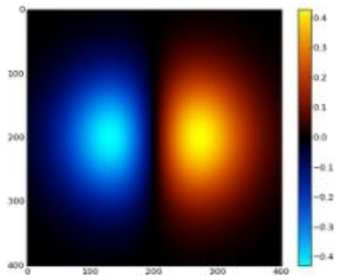


Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

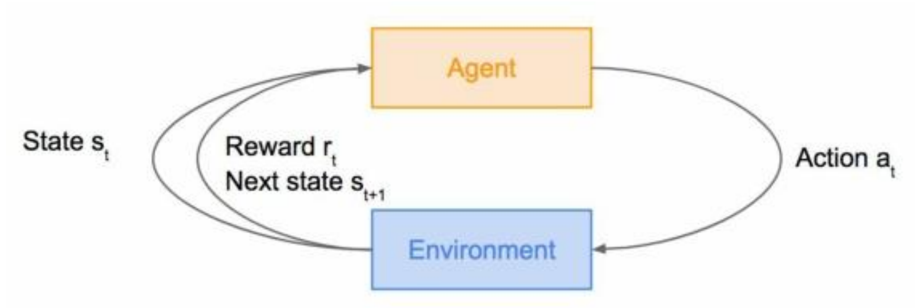




Reinforcement Learning

Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals

Goal: Learn how to take actions in order to maximize reward



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}

- Learning from interaction with an environment to achieve some long-term goal that is related to the state of the environment
- The goal is defined by reward signal, which must be maximised
- Agent must be able to partially/fully sense the environment state and take actions to influence the environment state
- The state is typically described with a feature-vector

Terminologies

- A reward R_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximise cumulative reward
- Policy: agent's behaviour function
- Value function: how good is each state and/or action
- Model: agent's representation of the environment

Policy

- A policy is the agent's behaviour
- It is a map from state to action
  - Deterministic policy
  - Stochastic policy

Value function

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions

Model

- A model predicts what the environment will do next
- P - predicts the next state
- R- predicts the next (immediate) reward

Agent

- Value based
- Policy based
- Exploration finds more information about the environment
- Exploitation exploits known information to maximise reward
- It is usually important to explore as well as exploit

Agents algorithm

```
Repeat:
    ◆ s ← sensed state
    ◆ If s is terminal then exit
    ◆ a ← Π(s)
    ◆ Perform a
```

Types of Reinforcement learning

- Search-based: evolution directly on a policy
  - E.g. optimization algorithm –GA, PSO

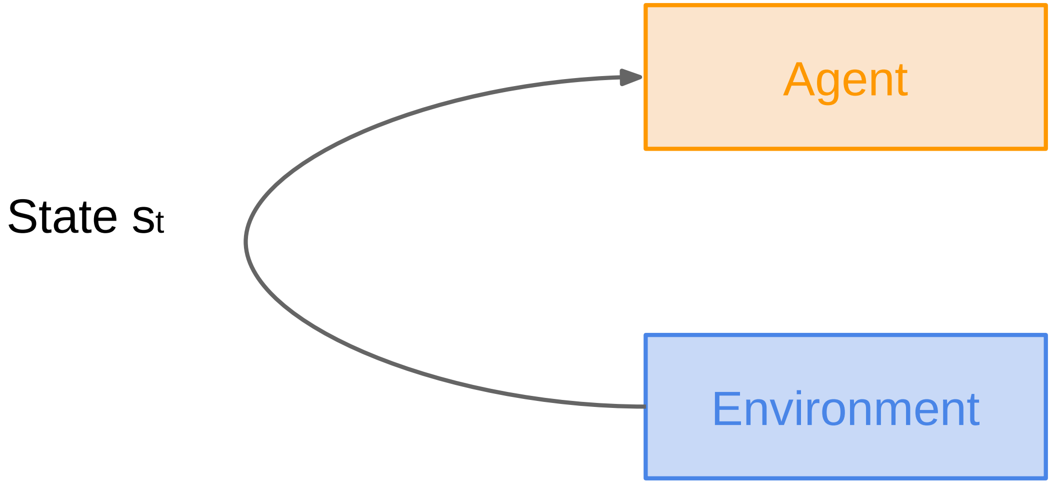
- Model-based: build a model of the environment
  - Then you can use dynamic programming
  - Memory-intensive learning method

- Model-free: learn a policy without any model
  - Temporal difference methods (TD)

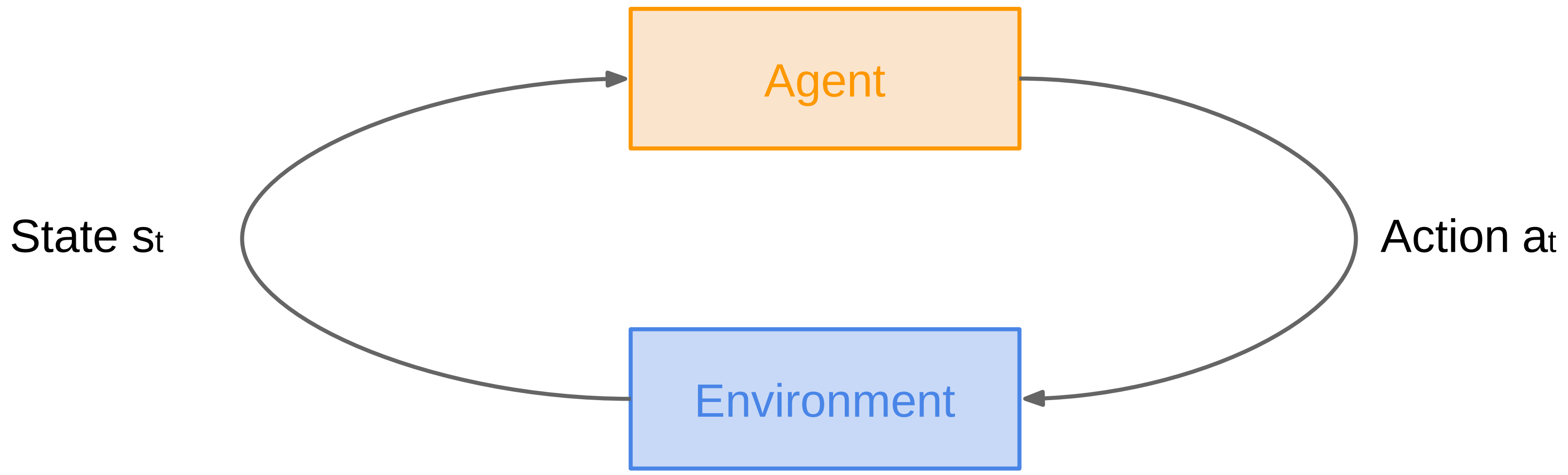
Reinforcement Learning

Agent
Environment

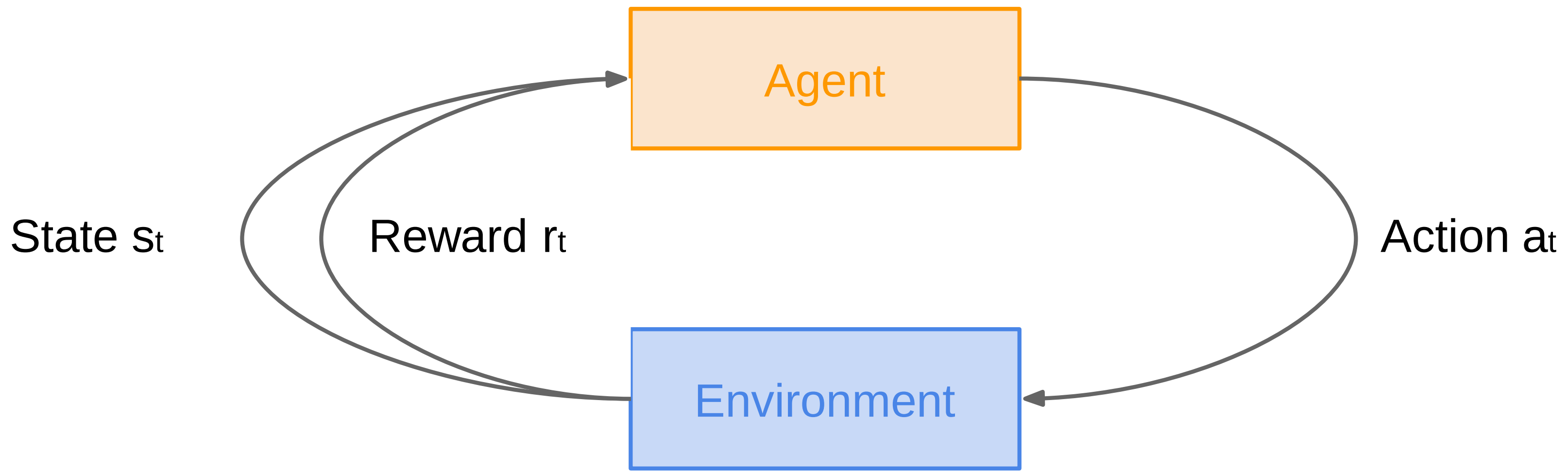
Reinforcement Learning



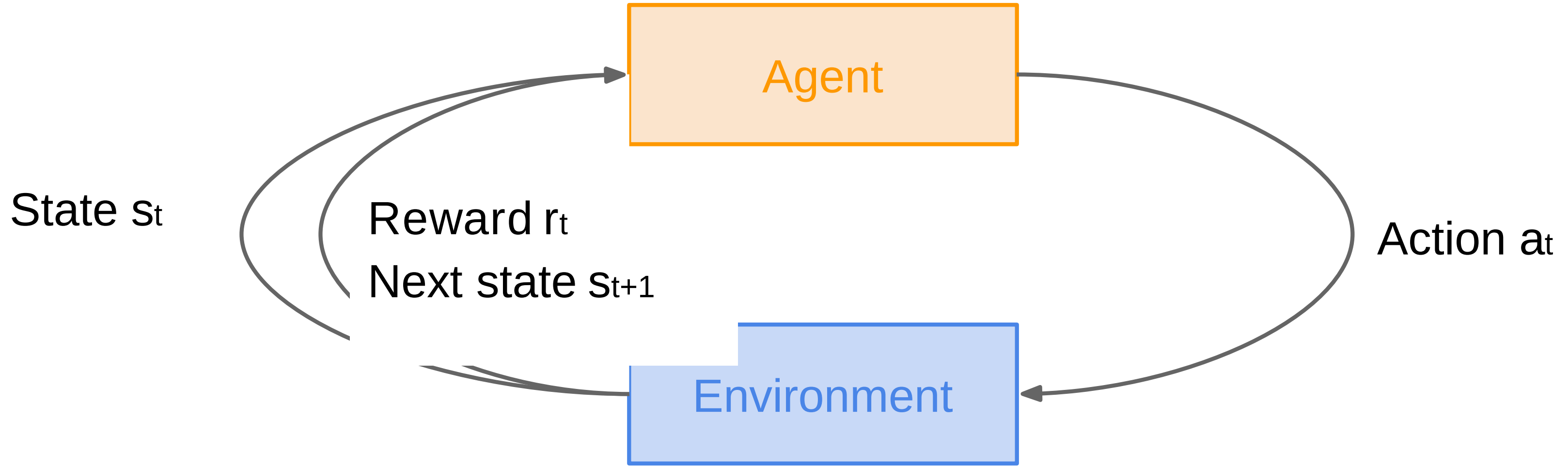
Reinforcement Learning



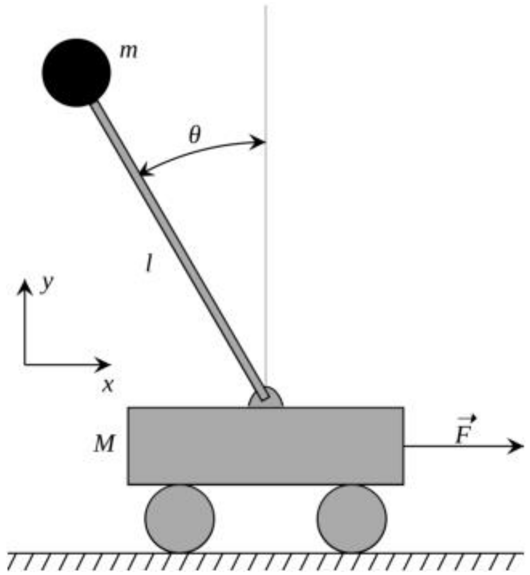
Reinforcement Learning



Reinforcement Learning



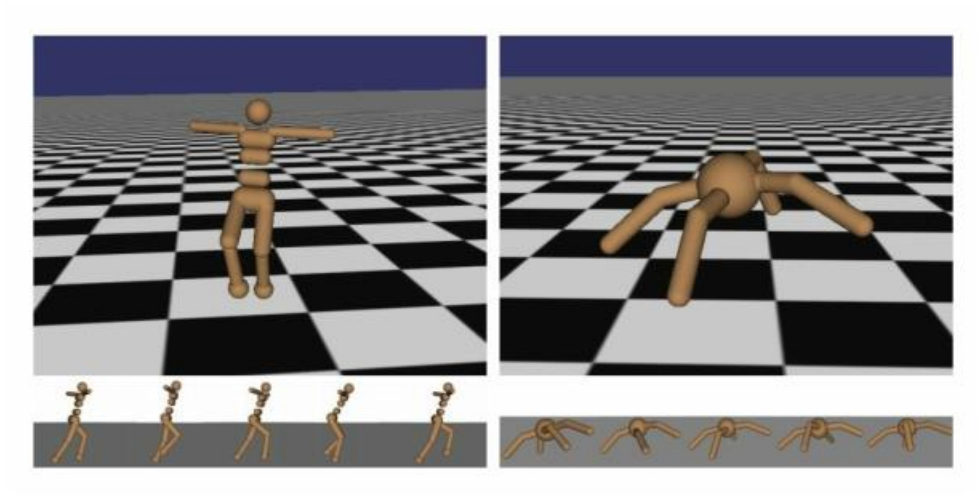
Cart-Pole Problem



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

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 $r_t \sim R(. \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. \mid s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

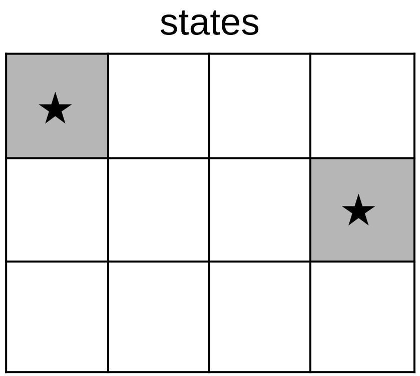
- A policy is a function from S to A that specifies what action to take in each state
- **Objective**: find policy * that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

A simple MDP: Grid World

actions = {

1. right →
2. left ←
3. up ↑
4. down ↓

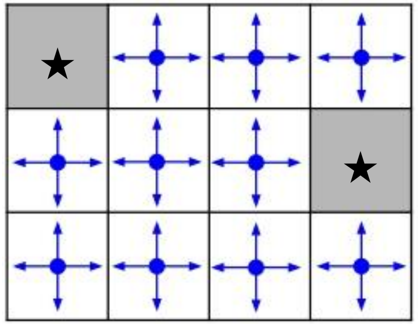
}

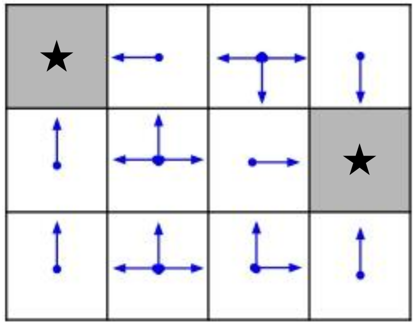


Set a negative "reward" for each transition (e.g. $r = -1$)

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 *

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

How do we handle the randomness (initial state, transition probability…)?

The optimal policy *

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

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

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

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, …

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, …

How good is a state?

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

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

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, …

How good is a state?

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

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

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:

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

Bellman equation

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

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

Bellman equation

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

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

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

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

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

$$r + \gamma Q^*(s',a')$$

Bellman equation

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

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

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

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

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 * 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 -> 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?

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. current game state 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. 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

Q-learning: Use a function approximator to estimate the action-value function

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

Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

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

If the function approximator is a deep neural network => **deep q-learning**!

Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

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

function parameters (weights)

If the function approximator is a deep neural network => **deep q-learning**!

Solving for the optimal policy: Q-learning

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

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

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

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

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

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

Backward Pass

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

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

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

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

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   *)

Backward Pass

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

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

- Current state: ***s***
- Current action: ***a***

- Transition function: ***δ(s, a) = s'***

- Reward function: ***r(s, a) ϵ R***

- Policy ***π(s) = a***

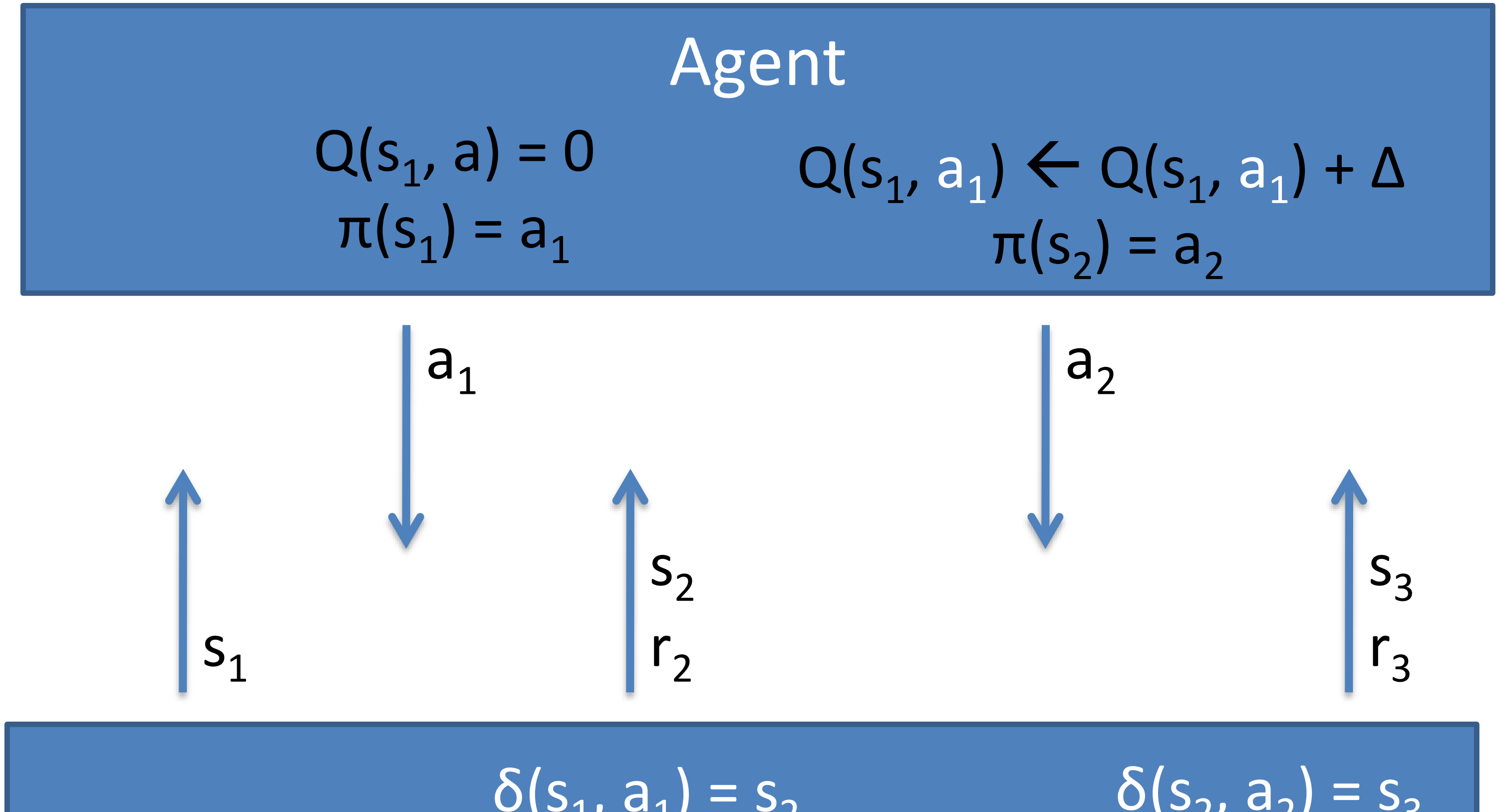
The Q-function

- ***Q(s, a)*** estimates the *discounted cumulative reward*
  - Starting in state ***s***
  - Taking action ***a***
  - Following the current policy thereafter

- Suppose we have the optimal Q-function
  - What's the optimal policy in state ***s***?
  - The action $\boldsymbol{argmax_b\ Q(s, b)}$

- But we don't have the optimal Q-function at first
  - Let's act as if we do

Q-Learning: The Procedure



Q-Learning: Updates

- The basic update equation

$$Q(s,a) \longleftarrow r(s,a) + \max_b Q(s',b)$$

- With a discount factor to give later rewards less impact

$$Q(s,a) \longleftarrow r(s,a) + \gamma \max_b Q(s',b)$$

- With a learning rate for non-deterministic worlds

$$Q(s,a) \longleftarrow [1-\alpha]Q(s,a) + \alpha[r(s,a) + \gamma \max_b Q(s',b)]$$

Q-Learning

-
  ```
  foreach state s
    foreach action a
      Q(s,a)=0
  s=currentstate
  do forever
    a = select an action
    do action a
    r = reward from doing a
    t = resulting state from doing a
    $Q(s,a) = (1 - \alpha)\, Q(s,a) + \alpha\, (r + \gamma\, Q(t))$
    s = t
  ```
- The *learning coefficient*, α, determines how quickly our estimates are updated
- Normally, α is set to a small positive constant less than 1

What about very large state-spaces?

- Value-Based: Learning a model and utility function
  - Can be difficult to learn good models for large complex environments (e.g. learning a DBN representation)
  - But if we can learn a model then learning utility function is simpler than learning Q(s,a)
  - Also can reuse the model for "related problems"
- Q-learning: Learning Q-function
  - Simpler to implement since we don't need to worry about representing and learning a model
  - But Q-functions can be substantially more complex than utility functions (must somehow make up for not having the model)

Exploration versus Exploitation

- We want a reinforcement learning agent to earn lots of reward
- The agent must prefer past actions that have been found to be effective at producing reward
- The agent must exploit what it already knows to obtain reward
- The agent must select untested actions to discover reward-producing actions
- The agent must explore actions to make better action selections in the future

- Exploitation: Maximize its reward
- Exploration: Maximize long-term

well being.

Summary

- There is no supervisor, only a reward signal
- Feedback is delayed, not instantaneous
- Time really matters
- Agent's actions act the subsequent data it receives

- Goal is to learn utility values of states and
- an optimal mapping from states to actions.
- Direct Utility Estimation ignores
- dependencies among states → we must
- follow Bellman Equations.
- Temporal difference updates values to
- match those of successor states.
- Active reinforcement learning learns the
- optimal mapping from states to actions