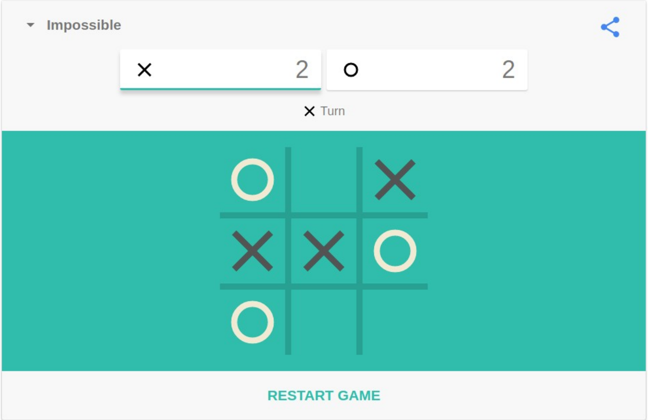
Reinforcement Learning

Sayantan Auddy

universität innsbruck
Department of Computer Science

Motivation

How should a computer learn to play tic-tac-toe?

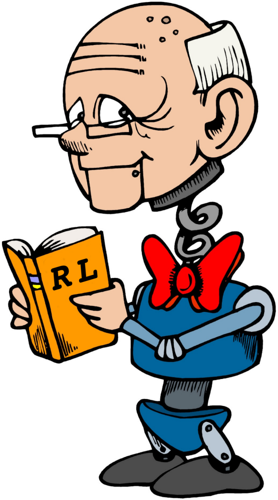


Content

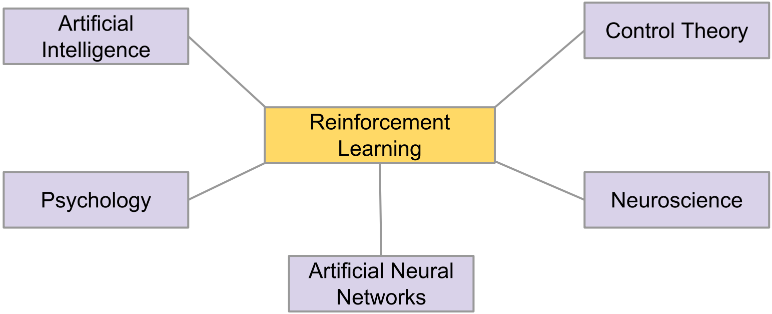
- Introduction
  - Simplified view of Reinforcement Learning
  - Different ML techniques
  - Applications of RL
  - RL Terminology
  - Categories or RL algorithms
- Markov Decision Process
  - Definition
  - Markov Property
  - Reward, Goal, Episodes, Returns, Policy, Value Functions
  - Bellman Equations
  - Optimal Policies and Values

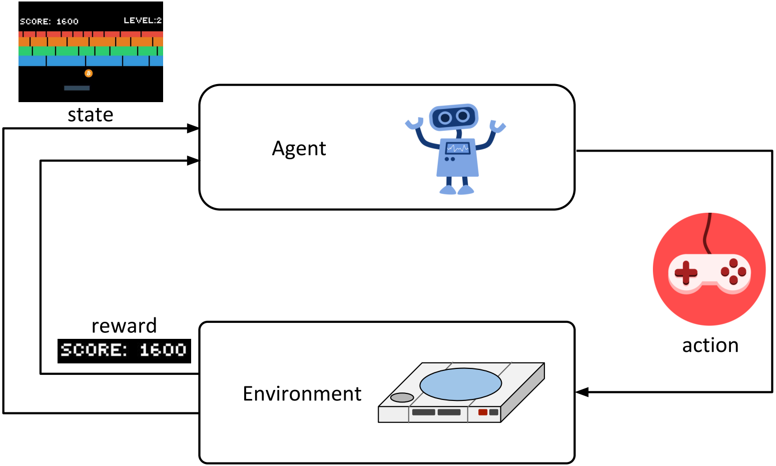
Content

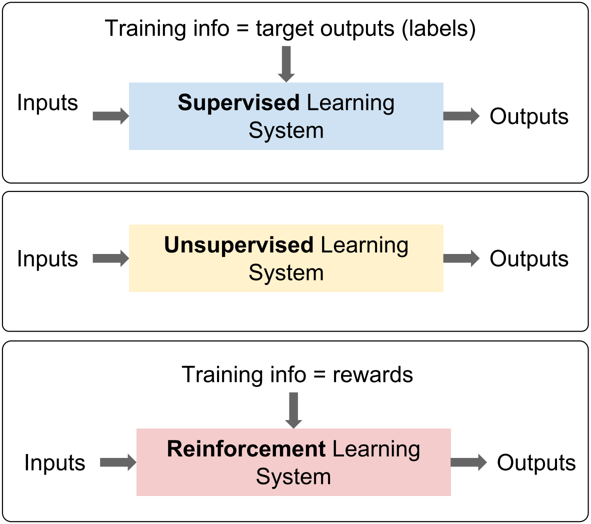
- Introduction
  - Simplified view of Reinforcement Learning
  - Different ML techniques
  - Applications of RL
  - RL Terminology
  - Categories or RL algorithms
- Markov Decision Process
  - Definition
  - Markov Property
  - Reward, Goal, Episodes, Returns, Policy, Value Functions
  - Bellman Equations
  - Optimal Policies and Values


RL

Origins of RL




SCORE: 1600
LEVEL:2
state
Agent
action
reward
SCORE: 1600
Environment


Training info = target outputs (labels)
Inputs
Supervised Learning System
Outputs
Inputs
Unsupervised Learning System
Outputs
Training info = rewards
Inputs
Reinforcement Learning System
Outputs

Key Features of RL

- No explicit teacher
- Learning by trial and error
- Learning through repeated agent-environment interactions
- Goal-oriented learning by maximizing cumulative reward
- Delayed rewards
- Need to balance exploitation vs exploration

- No explicit teacher
- Learning by trial and error
- Learning through repeated agent-environment interactions
- Goal-oriented learning by maximizing cumulative reward
- Delayed rewards
- Need to balance exploitation vs exploration

- No explicit teacher
- Learning by trial and error
- Learning through repeated agent-environment interactions
- Goal-oriented learning by maximizing cumulative reward
- Delayed rewards
- Need to balance exploitation vs exploration

Key Features of RL

- No explicit teacher
- Learning by trial and error
- Learning through repeated agent-environment interactions
- Goal-oriented learning by maximizing cumulative reward
- Delayed rewards
- Need to balance exploitation vs exploration

Key Features of RL

- No explicit teacher
- Learning by trial and error
- Learning through repeated agent-environment interactions
- Goal-oriented learning by maximizing cumulative reward
- Delayed rewards
- Need to balance exploitation vs exploration

Key Features of RL

- No explicit teacher
- Learning by trial and error
- Learning through repeated agent-environment interactions
- Goal-oriented learning by maximizing cumulative reward
- Delayed rewards
- Need to balance exploitation vs exploration

- AlphaGo [Silver et al., 2016]
  [https://deepmind.com/research/alphago/]
- Deep Q Network [Mnih et al., 2015]
  [https://youtu.be/W2CAghUiofY]
- Dexterous Object Manipulation [Andrychowicz et al., 2018]
  [https://youtu.be/jwSbzNHGflM?t=38s]
- Ball in a Cup [Peters et al., 2009]
  [https://youtu.be/Fhb26WdqVuE?t=47s]
- Autonomous flying [Abbeel et al., 2010]
  [https://www.youtu.be/VCdxqn0fcnE]

- Agent
- Environment
- State
- Action
- Reward
- Return
- Goal
- Policy
- Value Function
- Model

The artificial entity that is being trained to perform a task by learning from its own experience. A learning agent must be able to sense the state of its environment and take actions to affect the state.

- Agent
- Environment
- State
- Action
- Reward
- Return
- Goal
- Policy
- Value Function
- Model

Comprises of everything outside the purview of the agent. The environment has its own internal dynamics and rules which are usually not visible to the agent.

- Agent
- Environment
- State
- Action
- Reward
- Return
- Goal
- Policy
- Value Function
- Model

Current situation of the environment (as observed by the agent), which forms the basis for the decisions taken by the agent.

- Agent
- Environment
- State
- Action
- Reward
- Return
- Goal
- Policy
- Value Function
- Model

Choices made by the agent to change the state of the environment.

- Agent
- Environment
- State
- Action
- Reward
- Return
- Goal
- Policy
- Value Function
- Model

Scalar quantity that is emitted by the environment in response to the action taken by the agent.

- Agent
- Environment
- State
- Action
- Reward
- Return
- Goal
- Policy
- Value Function
- Model

The cumulative sum of rewards to be received by the agent.

- Agent
- Environment
- State
- Action
- Reward
- Return
- Goal
- Policy
- Value Function
- Model

The agent's goal is to maximize the the expected return it receives.

- Agent
- Environment
- State
- Action
- Reward
- Return
- Goal
- Policy
- Value Function
- Model

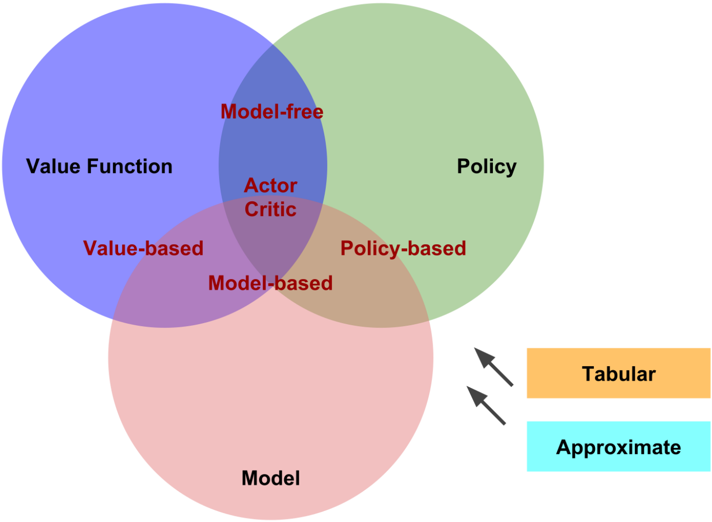
Defines the learning agent's behavior. The policy can be viewed as a function that provides a mapping from perceived states to actions (or probability of actions) to be taken in those states.

- Agent
- Environment
- State
- Action
- Reward
- Return
- Goal
- Policy
- Value Function
- Model

Specifies what is good in the long run. Value of a state is the total amount of reward that an agent can expect to accumulate starting from that state.

- Agent
- Environment
- State
- Action
- Reward
- Return
- Goal
- Policy
- Value Function
- Model

Something that mimics the behavior of the environment and allows inferences to be made about how the environment will behave.



Based on `http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching_files/intro_RL.pdf`

An agent in RL must be capable of:

- sensing the state of the environment
- taking actions for affecting the state
- realizing goals related to the state of the environment

What is a Markov Decision Process?

A Markov Decision Process (MDP) is a formal mathematical framework that is used to define the interation between the agent and its environment in terms of states, actions and rewards.

An agent in RL must be capable of:

- sensing the state of the environment
- taking actions for affecting the state
- realizing goals related to the state of the environment

What is a Markov Decision Process?

A Markov Decision Process (MDP) is a formal mathematical framework that is used to define the interation between the agent and its environment in terms of states, actions and rewards.

An agent in RL must be capable of:

- sensing the state of the environment
- taking actions for affecting the state
- realizing goals related to the state of the environment

What is a Markov Decision Process?

A Markov Decision Process (MDP) is a formal mathematical framework that is used to define the interation between the agent and its environment in terms of states, actions and rewards.

An agent in RL must be capable of:

- sensing the state of the environment
- taking actions for affecting the state
- realizing goals related to the state of the environment

What is a Markov Decision Process?

A Markov Decision Process (MDP) is a formal mathematical framework that is used to define the interation between the agent and its environment in terms of states, actions and rewards.

Markov Decision Process (MDP)

An MDP is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{P}$ is a state transition probability function, which defines the probability of transitioning to the next state S_{t+1} from the current state S_t on taking the action A_t.

$$\mathcal{P}(s'|s,a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$$

- r is a reward function, which defines the expected reward to be received on taking a particular action in a given state.

$$r(s,a,s') = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s']$$

- γ is a discount factor for assigning more importance to immediate rewards.

Markov Decision Process (MDP)

An MDP is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{P}$ is a state transition probability function, which defines the probability of transitioning to the next state S_{t+1} from the current state S_t on taking the action A_t.

$$\mathcal{P}(s'|s, a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$$

- r is a reward function, which defines the expected reward to be received on taking a particular action in a given state.

$$r(s, a, s') = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s']$$

- γ is a discount factor for assigning more importance to immediate rewards.

Markov Decision Process (MDP)

An MDP is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{P}$ is a state transition probability function, which defines the probability of transitioning to the next state S_{t+1} from the current state S_t on taking the action A_t.

$$\mathcal{P}(s'|s,a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

- r is a reward function, which defines the expected reward to be received on taking a particular action in a given state.

$$r(s,a,s') = \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s']$$

- γ is a discount factor for assigning more importance to immediate rewards.

Markov Decision Process (MDP)

An MDP is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{P}$ is a state transition probability function, which defines the probability of transitioning to the next state S_{t+1} from the current state S_t on taking the action A_t.

$$\mathcal{P}(s'|s, a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$$

- r is a reward function, which defines the expected reward to be received on taking a particular action in a given state.

$$r(s, a, s') = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s']$$

- γ is a discount factor for assigning more importance to immediate rewards.

Markov Decision Process (MDP)

An MDP is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{P}$ is a state transition probability function, which defines the probability of transitioning to the next state S_{t+1} from the current state S_t on taking the action A_t.

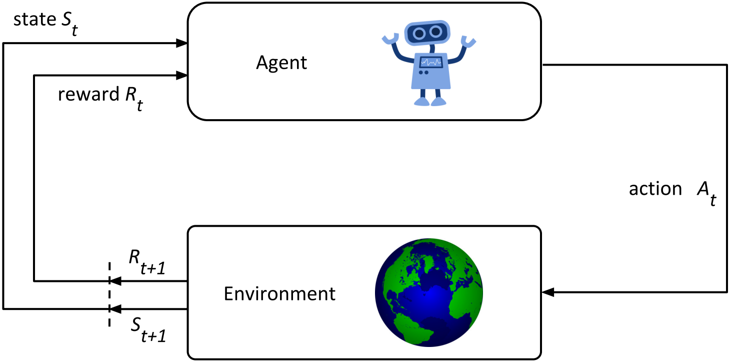
$$\mathcal{P}(s'|s, a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$$

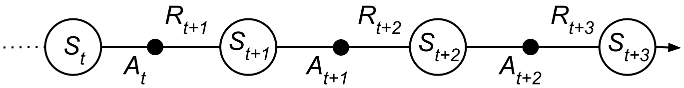
- r is a reward function, which defines the expected reward to be received on taking a particular action in a given state.

$$r(s, a, s') = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s']$$

- γ is a discount factor for assigning more importance to immediate rewards.

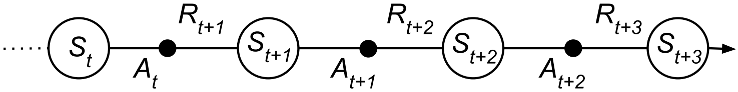
Agent interacts with its environment at $t = 0, 1, 2, 3, \dots$

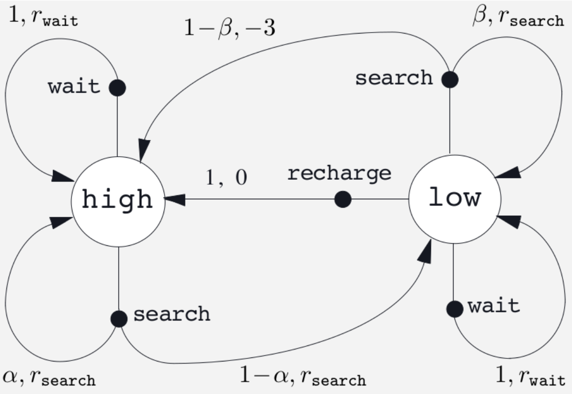




Markov Property

A state is said to possess the **Markov Property** when it includes information about all aspects of the past agent-environment interaction that make a difference for the future (future is independent of past states, actions and rewards).




1, $r_{\texttt{wait}}$
1−β, −3
β, $r_{\texttt{search}}$
wait
search
1, 0
recharge
high
low
search
wait
α, $r_{\texttt{search}}$
1−α, $r_{\texttt{search}}$
1, $r_{\texttt{wait}}$

Exercise: Markovian and Non-Markovian Env.

1. Devise an example task that fits into the MDP framework, identifying for each its states, actions, and rewards.
2. Can you think of an environment in which states do not have the Markov property?

Reward

The **reward** $R_t \in \mathbb{R}$ is a scalar quantity that forms the basis of evaluating the action taken by an agent.

- Reward is a measure of the immediate benefit of taking a particular action
- The agent must be able to measure how well it is performing frequently over its lifespan
- If rewards are **sparse** figuring out good actions can be difficult

Exercise: **Maze Runner**

What is a good choice of rewards for a maze solver?

Reward

The **reward** $R_t \in \mathbb{R}$ is a scalar quantity that forms the basis of evaluating the action taken by an agent.

- Reward is a measure of the immediate benefit of taking a particular action
- The agent must be able to measure how well it is performing frequently over its lifespan
- If rewards are **sparse** figuring out good actions can be difficult

Exercise: **Maze Runner**

What is a good choice of rewards for a maze solver?

Reward

The **reward** $R_t \in \mathbb{R}$ is a scalar quantity that forms the basis of evaluating the action taken by an agent.

- Reward is a measure of the immediate benefit of taking a particular action
- The agent must be able to measure how well it is performing frequently over its lifespan
- If rewards are **sparse** figuring out good actions can be difficult

Exercise: Maze Runner

What is a good choice of rewards for a maze solver?

Reward

The **reward** $R_t \in \mathbb{R}$ is a scalar quantity that forms the basis of evaluating the action taken by an agent.

- Reward is a measure of the immediate benefit of taking a particular action
- The agent must be able to measure how well it is performing frequently over its lifespan
- If rewards are **sparse** figuring out good actions can be difficult

Exercise: Maze Runner

What is a good choice of rewards for a maze solver?

Reward

The **reward** $R_t \in \mathbb{R}$ is a scalar quantity that forms the basis of evaluating the action taken by an agent.

- Reward is a measure of the immediate benefit of taking a particular action
- The agent must be able to measure how well it is performing frequently over its lifespan
- If rewards are **sparse** figuring out good actions can be difficult

Exercise: **Maze Runner**

What is a good choice of rewards for a maze solver?

Goal

The **goal** of an RL agent is to maximize the cumulative reward over the long run.

- Any goal can be thought of as the maximization of the expected value of the cumulative reward
- A goal should be outside the agent's direct control

Goal

The **goal** of an RL agent is to maximize the cumulative reward over the long run.

- Any goal can be thought of as the maximization of the expected value of the cumulative reward
- A goal should be outside the agent's direct control

Goal

The **goal** of an RL agent is to maximize the cumulative reward over the long run.

- Any goal can be thought of as the maximization of the expected value of the cumulative reward
- A goal should be outside the agent's direct control

Episodic Tasks

- Agent-environment interaction breaks down naturally into subsequences known as episodes
- Agent's state reset after terminal state S_T

Continuing Tasks

Interaction does not break down into sub-sequences (e.g. gas pipeline monitoring, heating system monitoring)

Episodic Tasks

- Agent-environment interaction breaks down naturally into subsequences known as episodes
- Agent's state reset after terminal state S_T

Continuing Tasks

Interaction does not break down into sub-sequences (e.g. gas pipeline monitoring, heating system monitoring)

Episodic Tasks

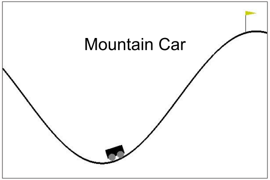
- Agent-environment interaction breaks down naturally into subsequences known as episodes
- Agent's state reset after terminal state S_T

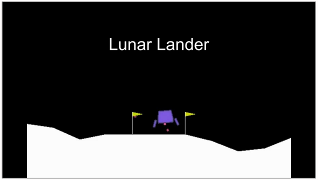
Continuing Tasks

Interaction does not break down into sub-sequences (e.g. gas pipeline monitoring, heating system monitoring)

Episodic Tasks

- Agent-environment interaction breaks down naturally into subsequences known as episodes
- Agent's state reset after terminal state S_T



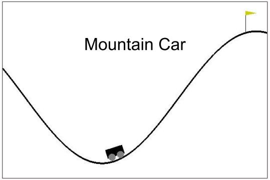


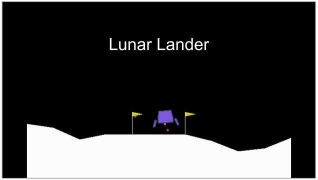
Continuing Tasks

Interaction does not break down into sub-sequences (e.g. gas pipeline monitoring, heating system monitoring)

Episodic Tasks

- Agent-environment interaction breaks down naturally into subsequences known as episodes
- Agent's state reset after terminal state S_T





Continuing Tasks

Interaction does not break down into sub-sequences (e.g. gas pipeline monitoring, heating system monitoring)

- For episodic tasks, if the agent expects to receive rewards $R_{t+1}, R_{t+2}, R_{t+3}, \ldots, R_T$ from time t till time T, the **return** G_t is defined as:

$$G_t = R_{t+1} + R_{t+2} + \ldots + R_T = \sum_{k=0}^{T} R_{t+k+1}$$

- For continuing tasks, $T = \infty$, so G_t can evaluate to ∞
- A discounting factor $\gamma \in [0, 1)$ used to limit the value of G_t to a finite quantity.

Return G_t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- For episodic tasks, if the agent expects to receive rewards $R_{t+1}, R_{t+2}, R_{t+3}, ..., R_T$ from time t till time T, the **return** G_t is defined as:

$$G_t = R_{t+1} + R_{t+2} + ... + R_T = \sum_{k=0}^{T} R_{t+k+1}$$

- For continuing tasks, $T = \infty$, so G_t can evaluate to ∞
- A discounting factor $\gamma \in [0, 1)$ used to limit the value of G_t to a finite quantity.

Return G_t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- For episodic tasks, if the agent expects to receive rewards $R_{t+1}, R_{t+2}, R_{t+3}, ..., R_T$ from time t till time T, the **return** G_t is defined as:

$$G_t = R_{t+1} + R_{t+2} + ... + R_T = \sum_{k=0}^{T} R_{t+k+1}$$

- For continuing tasks, $T = \infty$, so G_t can evaluate to ∞
- A discounting factor $\gamma \in [0, 1)$ used to limit the value of G_t to a finite quantity.

Return G_t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

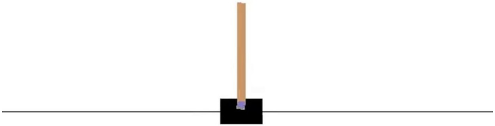
The relationship between returns at successive steps can be easily derived:

Recursive relationship between G_t and G_{t+1}

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \ldots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Exercise: **Calculate the Return**

Suppose $\gamma = 0.5$ and the following sequence of rewards is received $R_1 = 1, R_2 = 2, R_3 = 6, R_4 = 3$, and $R_5 = 2$, with $T = 5$. What are $G_0, G_1, \ldots, G_5$? Hint: Work backwards.

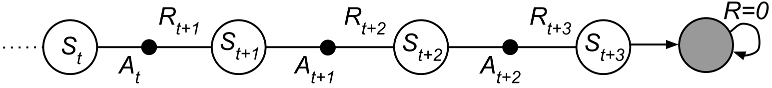


- **Episodic Task** (undiscounted)
  - reward = +1 for each step before failure
  - return at each step = # steps to failure
- **Continuing Task**
  - reward = -1 on failure, 0 otherwise
  - return at each step is related to $-\gamma^k$, for k steps before failure

In both cases return is maximized by avoiding failure as long as possible.

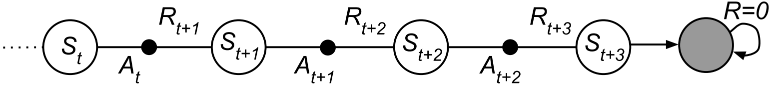
- Episodic tasks can be viewed as a special case of continuing tasks
- Terminal state S_T acts as an absorbing state for which the reward is always 0
- Both continuing and episodic tasks using $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

- Episodic tasks can be viewed as a special case of continuing tasks
- Terminal state S_T acts as an absorbing state for which the reward is always 0



- Both continuing and episodic tasks using $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

- Episodic tasks can be viewed as a special case of continuing tasks
- Terminal state S_T acts as an absorbing state for which the reward is always 0



- Both continuing and episodic tasks using $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

Policy

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

- Policy $\pi(a|s)$ is a mapping from states to probabilities of selecting an action.
- RL methods specify how the agent changes its policy based on its experience
- A good policy is one that results in a lot of rewards in the long run

Policy

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

- Policy $\pi(a|s)$ is a mapping from states to probabilities of selecting an action.
- RL methods specify how the agent changes its policy based on its experience
- A good policy is one that results in a lot of rewards in the long run

Policy

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

- Policy $\pi(a|s)$ is a mapping from states to probabilities of selecting an action.
- RL methods specify how the agent changes its policy based on its experience
- A good policy is one that results in a lot of rewards in the long run

Policy

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

- Policy $\pi(a|s)$ is a mapping from states to probabilities of selecting an action.
- RL methods specify how the agent changes its policy based on its experience
- A good policy is one that results in a lot of rewards in the long run

State-value Function

The state-value function of a state s under a policy π is defined as the expected return when starting in s and following π thereafter.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \; \forall s \in \mathcal{S}$$

Action-value Function

The action-value function of a state s and action a under a policy π is defined as the expected return when starting in s, taking the action a (which may not necessarily be predicted by π) and following π thereafter.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

$$\forall s \in \mathcal{S}, a \in \mathcal{A}$$

State-value Function

The state-value function of a state s under a policy π is defined as the expected return when starting in s and following π thereafter.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \ \forall s \in \mathcal{S}$$

Action-value Function

The action-value function of a state s and action a under a policy π is defined as the expected return when starting in s, taking the action a (which may not necessarily be predicted by π) and following π thereafter.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

$$\forall s \in \mathcal{S}, a \in \mathcal{A}$$

- **Markov Decision Process**
  - Recap
  - Bellman Equations
  - Optimal Policies and Values
- **Dynamic Programming**
  - Policy Evaluation
  - Policy Iteration
  - Value Iteration

- **Markov Decision Process**
  - Recap
  - Bellman Equations
  - Optimal Policies and Values
- **Dynamic Programming**
  - Policy Evaluation
  - Policy Iteration
  - Value Iteration

Term	Description	Expression
MDP	Framework defining agent-environment interaction	$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ where ▶ $\mathcal{S}$ is a finite set of states. ▶ $\mathcal{A}$ is a finite set of actions. ▶ $\mathcal{P}$ is a state transition prob. func. $\mathcal{P}(s'\|s,a) = \mathbb{P}[S_{t+1} = s' \| S_t = s, A_t = a]$ ▶ r is a reward function $r(s,a,s') = \mathbb{E}[R_{t+1} \| S_t = s, A_t = a, S_{t+1} = s']$ ▶ γ is a discount factor, $0 \leq \gamma \leq 1$
Markov Property	Current state includes all information about the past	-
Reward	Scalar quantity for evaluating the agent's action.	$R_t \in \mathbb{R}$
Return	Discounted sum of future rewards.	$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ $= R_{t+1} + \gamma G_{t+1}$
Goal	Maximize expected Return	$maximize(\mathbb{E}[G_t])$ at each t

Term	Description	Expression
MDP	Framework defining agent-environment interaction	$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ where ▶ $\mathcal{S}$ is a finite set of states. ▶ $\mathcal{A}$ is a finite set of actions. ▶ $\mathcal{P}$ is a state transition prob. func. $\mathcal{P}(s'\|s,a) = \mathbb{P}[S_{t+1} = s' \| S_t = s, A_t = a]$ ▶ r is a reward function $r(s,a,s') = \mathbb{E}[R_{t+1} \| S_t = s, A_t = a, S_{t+1} = s']$ ▶ γ is a discount factor, $0 \leq \gamma \leq 1$
Markov Property	Current state includes all information about the past	-
Reward	Scalar quantity for evaluating the agent's action.	$R_t \in \mathbb{R}$
Return	Discounted sum of future rewards.	$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$
Goal	Maximize expected Return	$maximize(\mathbb{E}[G_t])$ at each t

Term	Description	Expression
MDP	Framework defining agent-environment interaction	$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ where ▶ $\mathcal{S}$ is a finite set of states. ▶ $\mathcal{A}$ is a finite set of actions. ▶ $\mathcal{P}$ is a state transition prob. func. $\mathcal{P}(s'\|s,a) = \mathbb{P}[S_{t+1} = s' \| S_t = s, A_t = a]$ ▶ r is a reward function $r(s,a,s') = \mathbb{E}[R_{t+1} \| S_t = s, A_t = a, S_{t+1} = s']$ ▶ γ is a discount factor, $0 \leq \gamma \leq 1$
Markov Property	Current state includes all information about the past	-
Reward	Scalar quantity for evaluating the agent's action.	$R_t \in \mathbb{R}$
Return	Discounted sum of future rewards.	$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$
Goal	Maximize expected Return	$maximize(\mathbb{E}[G_t])$ at each t

Term	Description	Expression
MDP	Framework defining agent-environment interaction	$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ where ▶ $\mathcal{S}$ is a finite set of states. ▶ $\mathcal{A}$ is a finite set of actions. ▶ $\mathcal{P}$ is a state transition prob. func. $\mathcal{P}(s'\|s,a) = \mathbb{P}[S_{t+1} = s'\|S_t = s, A_t = a]$ ▶ r is a reward function $r(s,a,s') = \mathbb{E}[R_{t+1}\|S_t = s, A_t = a, S_{t+1} = s']$ ▶ γ is a discount factor, $0 \leq \gamma \leq 1$
Markov Property	Current state includes all information about the past	-
Reward	Scalar quantity for evaluating the agent's action.	$R_t \in \mathbb{R}$
Return	Discounted sum of future rewards.	$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ $= R_{t+1} + \gamma G_{t+1}$
Goal	Maximize expected Return	$maximize(\mathbb{E}[G_t])$ at each t

Term	Description	Expression
MDP	Framework defining agent-environment interaction	$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ where ▶ $\mathcal{S}$ is a finite set of states. ▶ $\mathcal{A}$ is a finite set of actions. ▶ $\mathcal{P}$ is a state transition prob. func. $\mathcal{P}(s'\|s,a) = \mathbb{P}[S_{t+1} = s'\|S_t = s, A_t = a]$ ▶ r is a reward function $r(s,a,s') = \mathbb{E}[R_{t+1}\|S_t = s, A_t = a, S_{t+1} = s']$ ▶ γ is a discount factor, $0 \leq \gamma \leq 1$
Markov Property	Current state includes all information about the past	-
Reward	Scalar quantity for evaluating the agent's action.	$R_t \in \mathbb{R}$
Return	Discounted sum of future rewards.	$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ $= R_{t+1} + \gamma G_{t+1}$
Goal	Maximize expected Return	$maximize(\mathbb{E}[G_t])$ at each t

Term	Description	Expression
Policy	Mapping from states to probabilities of actions.	$\pi(a\|s) = \mathbb{P}[A_t = a\|S_t = s]$
State-value Function	Expected **Return** when starting in s and following π thereafter.	$v_\pi(s) = \mathbb{E}_\pi[G_t\|S_t = s]$ $= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\|S_t = s] \ \forall s \in \mathcal{S}$
Action-value Function	Expected **Return** when starting in s, taking action a and following π thereafter.	$q_\pi(s, a) = \mathbb{E}_\pi[G_t\|S_t = s, A_t = a]$ $= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\|S_t = s, A_t = a]$ $\forall s \in \mathcal{S}, a \in \mathcal{A}$

Term	Description	Expression
Policy	Mapping from states to probabilities of actions.	$\pi(a\|s) = \mathbb{P}[A_t = a\|S_t = s]$
State-value Function	Expected **Return** when starting in s and following π thereafter.	$v_\pi(s) = \mathbb{E}_\pi[G_t\|S_t = s]$ $= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\|S_t = s] \; \forall s \in \mathcal{S}$
Action-value Function	Expected **Return** when starting in s, taking action a and following π thereafter.	$q_\pi(s,a) = \mathbb{E}_\pi[G_t\|S_t = s, A_t = a]$ $= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\|S_t = s, A_t = a]$ $\forall s \in \mathcal{S}, a \in \mathcal{A}$

Term	Description	Expression
Policy	Mapping from states to probabilities of actions.	$\pi(a\|s) = \mathbb{P}[A_t = a\|S_t = s]$
State-value Function	Expected **Return** when starting in s and following π thereafter.	$v_\pi(s) = \mathbb{E}_\pi[G_t\|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\|S_t = s] \; \forall s \in \mathcal{S}$
Action-value Function	Expected **Return** when starting in s, taking action a and following π thereafter.	$q_\pi(s, a) = \mathbb{E}_\pi[G_t\|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\|S_t = s, A_t = a] \; \forall s \in \mathcal{S}, a \in \mathcal{A}$

- A fundamental property of value functions is that they satisfy recursive relationships
- Bellman Expectation Equations formally express this relationship between values of states and their successors
- System of linear equations which has a unique solution

- A fundamental property of value functions is that they satisfy recursive relationships
- Bellman Expectation Equations formally express this relationship between values of states and their successors
- System of linear equations which has a unique solution

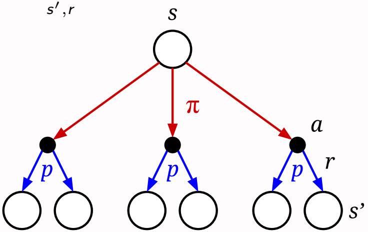
- A fundamental property of value functions is that they satisfy recursive relationships

- Bellman Expectation Equations formally express this relationship between values of states and their successors

- System of linear equations which has a unique solution

Bellman Expectation Equation for v_π

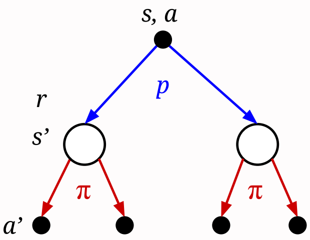
$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[G_t \,\middle|\, S_t = s\right] = \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right] \,\middle|\, S_t = s\right] \quad \text{(law of total } \mathbb{E}\text{)}^{1} \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(s') \,\middle|\, S_t = s\right] \\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right] \quad \forall s, s' \in \mathcal{S} \; a \in \mathcal{A}
\end{aligned}
$$



[1] Law of total expectation [https://en.wikipedia.org/wiki/Law_of_total_expectation]

Bellman Expectation Equation for q_π

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t \,\middle|\, S_t = s, A_t = a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \,\middle|\, S_t = s, A_t = a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s', A_{t+1} = a'\right] \,\middle|\, S_t = s, A_t = a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(s',a') \,\middle|\, S_t = s, A_t = a\right] \\
&= \sum_{s'} p(s' \mid s,a)\left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')\right] \quad \forall s,s' \in \mathcal{S}\ a,a' \in \mathcal{A}
\end{aligned}
$$



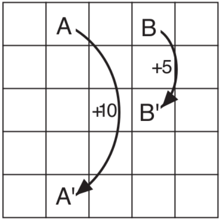
Exercise: **Random Walk**

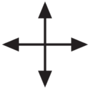
- **States**: A and B
- **Actions**: stay or switch (equal probabilities, deterministic effect)
- **Rewards**: +2 for A→B; -1 for B→A; 0 otherwise
- **Discount Factor**: $\gamma = 0.9$

Calculate the state-value function for the states A and B using the Bellman expectation equation for v_π:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]$$

$$\forall s, s' \in \mathcal{S} \; a \in \mathcal{A}$$





Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

- **Rewards**: -1 for going off the edge; 0 otherwise except for the special cases shown
- **Discount Factor**: $\gamma = 0.9$
- **Environment**: Deterministic as shown
- **Policy**: Uniform random
- Negative values near edge, A's return less than immediate reward and B's is greater than immediate reward
- Solved using Bellman expectation equations

Ordering of Policies

Value functions define a partial ordering over policies.
A policy π is considered to be better than a policy π':

$$\pi > \pi' \text{ if and only if } v_\pi(s) \geq v_{\pi'}(s) \ \forall s \in \mathcal{S}$$

Optimal Policy π_*

π_* is better than or equal to all other policies.

$$\pi_* = \arg\max_\pi v_*(s) \ \forall s \in \mathcal{S}$$

v_* is the maximum state-value function among all policies.

$$v_*(s) = \max_\pi v_\pi(s) \ \forall s \in \mathcal{S}$$

Optimal πs share the same optimal action-value func. q_*

$$\begin{aligned} q_*(s,a) &= \max_\pi q_\pi(s,a) \ \forall s \in \mathcal{S} \ a \in \mathcal{A} \\ &= \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right] \end{aligned}$$

Ordering of Policies

Value functions define a partial ordering over policies.
A policy π is considered to be better than a policy π':

$$\pi > \pi' \text{ if and only if } v_\pi(s) \geq v_{\pi'}(s) \ \forall s \in \mathcal{S}$$

Optimal Policy π_*

π_* is better than or equal to all other policies.

$$\pi_* = \arg\max_\pi v_*(s) \ \forall s \in \mathcal{S}$$

v_* is the maximum state-value function among all policies.

$$v_*(s) = \max_\pi v_\pi(s) \ \forall s \in \mathcal{S}$$

Optimal πs share the same optimal action-value func. q_*

$$\begin{aligned} q_*(s,a) &= \max_\pi q_\pi(s,a) \ \forall s \in \mathcal{S} \ a \in \mathcal{A} \\ &= \mathbb{E}\big[R_{t+1} + \gamma v_*(S_{t+1}) \,\big|\, S_t = s, A_t = a\big] \end{aligned}$$

Bellman optimality equations:

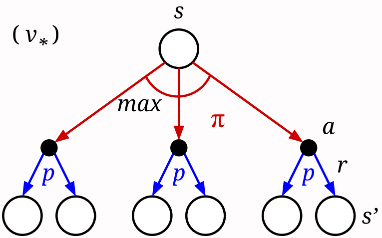
- Enables calculation of best value functions $\Rightarrow$ optimal policies
- Value of a state under π_* must equal the expected return for the best action from that state

Bellman Optimality Equation for v_*

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\ &= \max_a \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma v_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right] \\ &= \max_a \sum_{s', r} p\big(s', r \mid s, a\big)\big[r + \gamma v_*(s')\big] \end{aligned}$$

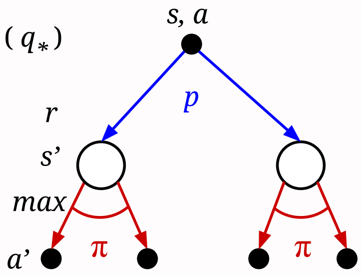
Bellman Optimality Equation for v_*

$$
\begin{aligned}
v_*(s) &= \max_a \ \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma v_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right]
\end{aligned}
$$



Bellman Optimality Equation for q_*

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a') \;\big|\; S_t = s, A_t = a\big] \\
&= \sum_{s',r} p\big(s',r \;\big|\; s,a\big)\big[r + \gamma \max_{a'} q_*(s',a')\big]
\end{aligned}
$$



- For finite MDPs, Bellman optimality equation for v_* has a unique solution
- Bellman optimality equation $\Rightarrow$ System of n non-linear equations for n unknowns (if there are n states)
- If the environment dynamics p is known, exact solution can be obtained.

Determining π_* from v_* or q_*

- v_* is known: Take an action that is greedy w.r.t. v_* (need to search through actions)
- q_* is known: In state s take the action a that has the maximum value of $q_*(s, a)$ (no search is necessary)

- For finite MDPs, Bellman optimality equation for v_* has a unique solution
- Bellman optimality equation $\Rightarrow$ System of n non-linear equations for n unknowns (if there are n states)
- If the environment dynamics p is known, exact solution can be obtained.

Determining π_* from v_* or q_*

- v_* is known: Take an action that is greedy w.r.t. v_* (need to search through actions)
- q_* is known: In state s take the action a that has the maximum value of $q_*(s, a)$ (no search is necessary)

- For finite MDPs, Bellman optimality equation for v_* has a unique solution
- Bellman optimality equation $\Rightarrow$ System of n non-linear equations for n unknowns (if there are n states)
- If the environment dynamics p is known, exact solution can be obtained.

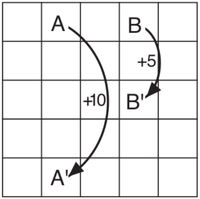
Determining π_* from v_* or q_*

- v_* is known: Take an action that is greedy w.r.t. v_* (need to search through actions)
- q_* is known: In state s take the action a that has the maximum value of $q_*(s, a)$ (no search is necessary)

- For finite MDPs, Bellman optimality equation for v_* has a unique solution
- Bellman optimality equation $\Rightarrow$ System of n non-linear equations for n unknowns (if there are n states)
- If the environment dynamics p is known, exact solution can be obtained.

Determining π_* from v_* or q_*

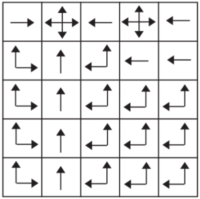
- v_* is known: Take an action that is greedy w.r.t. v_* (need to search through actions)
- q_* is known: In state s take the action a that has the maximum value of $q_*(s, a)$ (no search is necessary)



Gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*



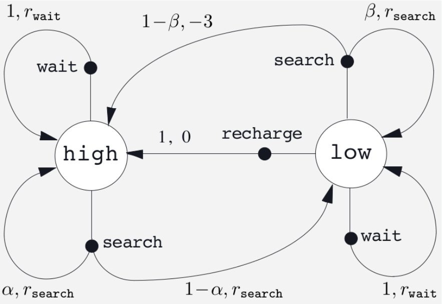
π_*

- **Rewards**: -1 for going off the edge; 0 otherwise except for the special cases shown
- **Discount Factor**: $\gamma = 0.9$
- **Environment**: Deterministic as shown
- **Policy**: greedy
- Solved using Bellman optimality equations

Exercise: v_* for Recycling Robot

Write down the Bellman optimality equation(s) for v_* for the recycling robot.

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right]$$



- Dynamic Programming (DP) is a technique for solving problems by:
  - Breaking the problem into sub-problems
  - Solving the sub-problems and combining their solutions
- Full knowledge of the MDP (environment dynamics) is assumed $\rightarrow$ Limited utility in practice
- Theoretically important $\rightarrow$ basis for other techniques which approximate the effects of DP with less computation and without MDP.

- Dynamic Programming (DP) is a technique for solving problems by:
  - Breaking the problem into sub-problems
  - Solving the sub-problems and combining their solutions
- Full knowledge of the MDP (environment dynamics) is assumed $\rightarrow$ Limited utility in practice
- Theoretically important $\rightarrow$ basis for other techniques which approximate the effects of DP with less computation and without MDP.

- Dynamic Programming (DP) is a technique for solving problems by:
  - Breaking the problem into sub-problems
  - Solving the sub-problems and combining their solutions
- Full knowledge of the MDP (environment dynamics) is assumed $\rightarrow$ Limited utility in practice
- Theoretically important $\rightarrow$ basis for other techniques which approximate the effects of DP with less computation and without MDP.

Prediction: How good is a policy?

- **Problem** Calculate the value function for a policy
- **Input** MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ and the policy π
- **Output** Value function v_π

Control: What is the best policy?

- **Problem** Calculate the optimal policy
- **Input** MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$
- **Output** Optimal value function v_* and the optimal policy π_*

Prediction: How good is a policy?

- **Problem** Calculate the value function for a policy
- **Input** MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ and the policy π
- **Output** Value function v_π

Control: What is the best policy?

- **Problem** Calculate the optimal policy
- **Input** MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$
- **Output** Optimal value function v_* and the optimal policy π_*

Prediction: How good is a policy?

- **Problem** Calculate the value function for a policy
- **Input** MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ and the policy π
- **Output** Value function v_π

Control: What is the best policy?

- **Problem** Calculate the optimal policy
- **Input** MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$
- **Output** Optimal value function v_* and the optimal policy π_*

- **Problem** Compute v_π for an arbitrary π.
- **Solution** Iterative application of the Bellman Expectation backup (synchronously).
- $\forall$ states $s \in \mathcal{S}$, at each iteration $k+1$, update $v_{k+1}(s)$ from $v_k(s)$:

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_k(S_{t+1}) \;\middle|\; S_t = s \right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s') \right] \end{aligned}$$

- From initial value v_0, repeated updates result in a sequence of value functions which converge to v_π as $k \approx \infty$ (in practice sooner):

$$v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$$

- Convergence: When the change in the updated value is negligible.

- **Problem** Compute v_π for an arbitrary π.
- **Solution** Iterative application of the Bellman Expectation backup (synchronously).
- $\forall$ states $s \in \mathcal{S}$, at each iteration $k+1$, update $v_{k+1}(s)$ from $v_k(s)$:

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_k(s')\right] \end{aligned}$$

- From initial value v_0, repeated updates result in a sequence of value functions which converge to v_π as $k \approx \infty$ (in practice sooner):

$$v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$$

- Convergence: When the change in the updated value is negligible.

- **Problem** Compute v_π for an arbitrary π.
- **Solution** Iterative application of the Bellman Expectation backup (synchronously).
- $\forall$ states $s \in \mathcal{S}$, at each iteration $k+1$, update $v_{k+1}(s)$ from $v_k(s)$:

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1}) \,\middle|\, S_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_k(s')\right] \end{aligned}$$

- From initial value v_0, repeated updates result in a sequence of value functions which converge to v_π as $k \approx \infty$ (in practice sooner):

$$v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$$

- Convergence: When the change in the updated value is negligible.

- **Problem** Compute v_π for an arbitrary π.
- **Solution** Iterative application of the Bellman Expectation backup (synchronously).
- $\forall$ states $s \in \mathcal{S}$, at each iteration $k+1$, update $v_{k+1}(s)$ from $v_k(s)$:

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1}) \,\middle|\, S_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_k(s')\right] \end{aligned}$$

- From initial value v_0, repeated updates result in a sequence of value functions which converge to v_π as $k \approx \infty$ (in practice sooner):

$$v_0 \to v_1 \to v_2 \to \dots \to v_\pi$$

- Convergence: When the change in the updated value is negligible.

- **Problem** Compute v_π for an arbitrary π.
- **Solution** Iterative application of the Bellman Expectation backup (synchronously).
- $\forall$ states $s \in \mathcal{S}$, at each iteration $k+1$, update $v_{k+1}(s)$ from $v_k(s)$:

$$
\begin{aligned}
v_{k+1}(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1}) \,\middle|\, S_t = s\right] \\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_k(s')\right]
\end{aligned}
$$

- From initial value v_0, repeated updates result in a sequence of value functions which converge to v_π as $k \approx \infty$ (in practice sooner):

$$
v_0 \to v_1 \to v_2 \to \ldots \to v_\pi
$$

- Convergence: When the change in the updated value is negligible.

Input: π, the policy to be evaluated
Parameter: $\theta > 0$, threshold for the accuracy of estimation
Initialize: $V(s)\ \forall s \in \mathcal{S}^+$, a table used for approximating v_π. Initialization can be arbitrary (or set everything to 0), except for $V(\mathit{terminal}) = 0$

repeat

    $\Delta \leftarrow 0$

    for each $s \in \mathcal{S}$ **do**

        $v \leftarrow V(s)$

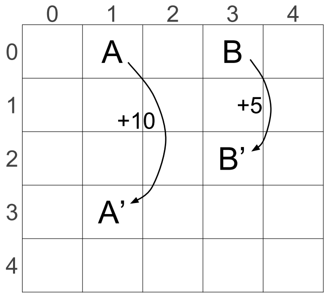
        $V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

        $\Delta \leftarrow \max(\Delta, \left| v - V(s) \right|)$

    end for

until $\Delta < \theta$

- **Rewards**: -1 for going off the edge; 0 otherwise, except for the special cases shown
- **Discount Factor**: $\gamma = 0.9$
- **Environment**: Deterministic as shown
- **Policy**: Uniform random (L,R,U,D)



https://iis.uibk.ac.at/uibk/auddy/rl/code/dynamic-programming/pol_eval.html

Exercise: **Policy Evaluation Algorithm**

The final value-function for the uniform random policy is shown below. Verify that the value of the state V(row:2,col:2)=0.71 does not change using the update step in the Policy Evaluation algorithm:

$$V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s')\right]$$

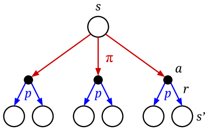
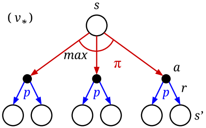
3.36	8.83	4.47	5.36	1.53
1.57	3.04	2.29	1.95	0.59
0.10	0.78	0.71	0.40	-0.37
-0.93	-0.39	-0.32	-0.55	-1.15
-1.81	-1.30	-1.19	-1.39	-1.94

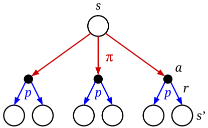
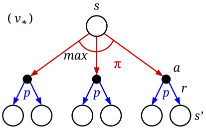
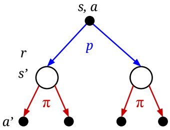
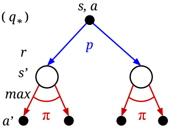
- Recap
- **Dynamic Programming**
  - Policy Evaluation
  - Policy Iteration
  - Value Iteration
- **Model-free Prediction**
  - Monte Carlo learning
  - Temporal Difference Learning
- **Model-free Control**
  - SARSA
  - Q-learning

- Recap
- **Dynamic Programming**
  - Policy Evaluation
  - Policy Iteration
  - Value Iteration
- **Model-free Prediction**
  - Monte Carlo learning
  - Temporal Difference Learning
- **Model-free Control**
  - SARSA
  - Q-learning

- Recap
- **Dynamic Programming**
  - Policy Evaluation
  - Policy Iteration
  - Value Iteration
- **Model-free Prediction**
  - Monte Carlo learning
  - Temporal Difference Learning
- **Model-free Control**
  - SARSA
  - Q-learning

- Recap
- **Dynamic Programming**
  - Policy Evaluation
  - Policy Iteration
  - Value Iteration
- **Model-free Prediction**
  - Monte Carlo learning
  - Temporal Difference Learning
- **Model-free Control**
  - SARSA
  - Q-learning

Name	Backup Diagram	Equation
Bellman Expect. Eq. for v_π		$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(s') \,\middle|\, S_t = s\right]$ $= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]$
Bellman Optimality Eq. for v_*		$v_*(s) = \max_a \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma v_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right]$ $= \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right]$
Bellman Expect. Eq. for q_π		$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(s', a') \,\middle|\, S_t = s, A_t = a\right]$ $= \sum_{s'} p(s' \mid s, a)\left[r + \gamma \sum_{a'} \pi(a' \vert s') q_\pi(s', a')\right]$
Bellman Optimality Eq. for q_*		$q_*(s,a) = \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\middle|\, S_t = s, A_t = a\right]$ $= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right]$

Name	**Backup Diagram**	**Equation**
Bellman Expect. Eq. for v_π		$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(s') \,\middle\vert\, S_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right] \end{aligned}$$
Bellman Optimality Eq. for v_*		$$\begin{aligned} v_*(s) &= \max_a \ \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right] \end{aligned}$$
Bellman Expect. Eq. for q_π		$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(s', a') \,\middle\vert\, S_t = s, A_t = a\right] \\ &= \sum_{s'} p(s' \mid s, a)\left[r + \gamma \sum_{a'} \pi(a' \vert s') q_\pi(s', a')\right] \end{aligned}$$
Bellman Optimality Eq. for q_*		$$\begin{aligned} q_*(s,a) &= \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right] \end{aligned}$$

- $\forall\ s \in \mathcal{S}$, at each iteration $k+1$, update $v_{k+1}(s)$ from $v_k(s)$:

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_k(s')\right] \end{aligned}$$

- From initial value v_0, repeated updates result in a sequence of value functions which converge to v_π as $k \approx \infty$:

$$v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$$

0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00

→

-0.50	10.00	2.00	5.00	0.63
-0.36	2.17	0.94	1.34	0.19
-0.33	0.41	0.30	0.37	-0.12
-0.32	0.02	0.07	0.10	-0.26
-0.57	-0.37	-0.32	-0.30	-0.62

→

1.44	9.66	3.71	5.33	1.02
0.41	2.57	1.78	1.73	0.38
-0.21	0.60	0.64	0.53	-0.13
-0.50	-0.04	0.08	0.01	-0.47
-0.95	-0.63	-0.51	-0.57	-1.02

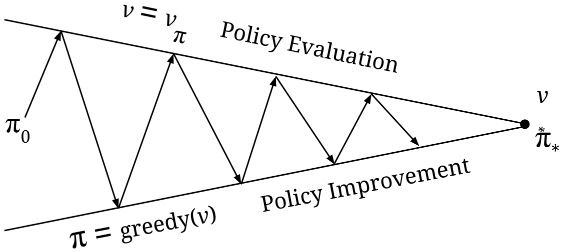
→ ⋯ →

3.36	8.83	4.47	5.36	1.53
1.57	3.04	2.29	1.95	0.59
0.10	0.78	0.71	0.40	-0.37
-0.93	-0.39	-0.32	-0.55	-1.15
-1.81	-1.30	-1.19	-1.39	-1.94

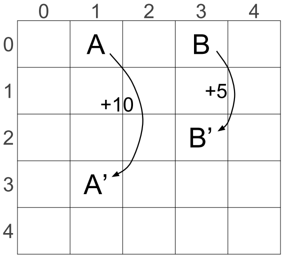
https://iis.uibk.ac.at/uibk/auddy/rl/code/dynamic-programming/pol_eval.html

- Alternatively uses policy evaluation and policy improvement
- Ultimately converges to π_* and v_*

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$



- **Rewards**: -1 for going off the edge; 0 otherwise, except for the special cases shown
- **Discount Factor**: $\gamma = 0.9$
- **Environment**: Deterministic as shown
- **Initial Policy**: Uniform random (L,R,U,D)
- What is the best policy?



https://iis.uibk.ac.at/uibk/auddy/rl/code/dynamic-programming/pol_iter.html

- For some state s, is it better to follow π or to choose a different action $a \neq \pi(s)$?
- If we take a and then follow π, then

$$q_\pi(s,a) = \mathbb{E}\big[R_{t+1} + \gamma v_\pi(S_{t+1}) \,\big|\, S_t = s, A_t = a\big]$$

- If $q_\pi(s,a) \geq v_\pi(s)$, then it is better to follow the modified policy π' (take action a whenever state s is encountered)[2]
- Act greedily to choose the policy that gives the best value

$$\pi'(s) = \arg\max_a q_\pi(s,a) = \arg\max_a \mathbb{E}\big[R_{t+1} + \gamma v_\pi(S_{t+1}) \,\big|\, S_t = s, A_t = a\big]$$

$$v_{\pi'}(s) = \max_a \mathbb{E}\big[R_{t+1} + \gamma v_\pi(S_{t+1}) \,\big|\, S_t = s, A_t = a\big]$$

- This is the Bellman optimality equation $\Rightarrow \pi' = \pi_*$ and $v_{\pi'} = v_*$

[2] $q_\pi(s,a) \geq v_\pi(s) \Rightarrow v_{\pi'}(s) \geq v_\pi(s)$ [Sutton and Barto, 2018](p. 78)

- For some state s, is it better to follow π or to choose a different action $a \neq \pi(s)$?
- If we take a and then follow π, then

$$q_\pi(s,a) = \mathbb{E}\big[R_{t+1} + \gamma v_\pi\big(S_{t+1}\big) \,\big|\, S_t = s, A_t = a\big]$$

- If $q_\pi(s,a) \geq v_\pi(s)$, then it is better to follow the modified policy π' (take action a whenever state s is encountered)[2]
- Act greedily to choose the policy that gives the best value

$$\pi'(s) = \arg\max_a q_\pi(s,a) = \arg\max_a \mathbb{E}\big[R_{t+1} + \gamma v_\pi(S_{t+1}) \,\big|\, S_t = s, A_t = a\big]$$

$$v_{\pi'}(s) = \max_a \mathbb{E}\big[R_{t+1} + \gamma v_\pi(S_{t+1}) \,\big|\, S_t = s, A_t = a\big]$$

- This is the Bellman optimality equation $\Rightarrow \pi' = \pi_*$ and $v_{\pi'} = v_*$

[2] $q_\pi(s,a) \geq v_\pi(s) \Rightarrow v_{\pi'}(s) \geq v_\pi(s)$ [Sutton and Barto, 2018](p. 78)

- For some state s, is it better to follow π or to choose a different action $a \neq \pi(s)$?
- If we take a and then follow π, then

$$q_\pi(s,a) = \mathbb{E}\big[R_{t+1} + \gamma v_\pi\big(S_{t+1}\big) \,\big|\, S_t = s, A_t = a\big]$$

- If $q_\pi(s,a) \geq v_\pi(s)$, then it is better to follow the modified policy π' (take action a whenever state s is encountered)[2]
- Act greedily to choose the policy that gives the best value

$$\pi'(s) = \arg\max_a q_\pi(s,a) = \arg\max_a \mathbb{E}\big[R_{t+1} + \gamma v_\pi(S_{t+1}) \,\big|\, S_t = s, A_t = a\big]$$

$$v_{\pi'}(s) = \max_a \mathbb{E}\big[R_{t+1} + \gamma v_\pi(S_{t+1}) \,\big|\, S_t = s, A_t = a\big]$$

- This is the Bellman optimality equation $\Rightarrow \pi' = \pi_*$ and $v_{\pi'} = v_*$

[2] $q_\pi(s,a) \geq v_\pi(s) \Rightarrow v_{\pi'}(s) \geq v_\pi(s)$ [Sutton and Barto, 2018](p. 78)

- For some state s, is it better to follow π or to choose a different action $a \neq \pi(s)$?
- If we take a and then follow π, then

$$q_\pi(s, a) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) \,\middle|\, S_t = s, A_t = a\right]$$

- If $q_\pi(s, a) \geq v_\pi(s)$, then it is better to follow the modified policy π' (take action a whenever state s is encountered)[2]
- Act greedily to choose the policy that gives the best value

$$\pi'(s) = \arg\max_a q_\pi(s, a) = \arg\max_a \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right]$$

$$v_{\pi'}(s) = \max_a \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right]$$

- This is the Bellman optimality equation $\Rightarrow \pi' = \pi_*$ and $v_{\pi'} = v_*$

[2] $q_\pi(s, a) \geq v_\pi(s) \Rightarrow v_{\pi'}(s) \geq v_\pi(s)$ [Sutton and Barto, 2018](p. 78)

- For some state s, is it better to follow π or to choose a different action $a \neq \pi(s)$?
- If we take a and then follow π, then

$$q_\pi(s,a) = \mathbb{E}\big[R_{t+1} + \gamma v_\pi\big(S_{t+1}\big) \bigm| S_t = s, A_t = a\big]$$

- If $q_\pi(s,a) \geq v_\pi(s)$, then it is better to follow the modified policy π' (take action a whenever state s is encountered)[2]
- Act greedily to choose the policy that gives the best value

$$\pi'(s) = \arg\max_a q_\pi(s,a) = \arg\max_a \mathbb{E}\big[R_{t+1} + \gamma v_\pi(S_{t+1}) \bigm| S_t = s, A_t = a\big]$$

$$v_{\pi'}(s) = \max_a \mathbb{E}\big[R_{t+1} + \gamma v_\pi(S_{t+1}) \bigm| S_t = s, A_t = a\big]$$

- This is the Bellman optimality equation $\Rightarrow \pi' = \pi_*$ and $v_{\pi'} = v_*$

[2] $q_\pi(s,a) \geq v_\pi(s) \Rightarrow v_{\pi'}(s) \geq v_\pi(s)$ [Sutton and Barto, 2018](p. 78)

1. Initialization

   $V(s) \in \mathbb{R}$ and $\pi(s) = \mathcal{A}(s)\ \forall s \in \mathcal{S}$

2. Policy Evaluation (algorithm from earlier, input: π, output: V)

3. Policy Improvement

policy-stable $\leftarrow$ *true*
for each $s \in \mathcal{S}$ **do**
  old-action $\leftarrow \pi(s)$
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
  If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
end for
If *policy-stable*, then stop and return $V \approx v*$ and $\pi \approx \pi_*$; else go to 2.

- Drawback of Policy Iteration: Involves Policy Evaluation
- Value Iteration turns the Bellman optimality equation into an update rule

$$
\begin{aligned}
v_{k+1}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_k(s')\right] \ \forall s \in \mathcal{S}
\end{aligned}
$$

- For any arbitrary initialization v_0, this converges to v_*
- The optimal policy π_* is obtained by acting greedily with respect to v_*

- Drawback of Policy Iteration: Involves Policy Evaluation
- Value Iteration turns the Bellman optimality equation into an update rule

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}\big[R_{t+1} + \gamma v_k(S_{t+1}) \;\big|\; S_t = s, A_t = a\big] \\ &= \max_a \sum_{s',r} p\big(s', r \mid s, a\big)\big[r + \gamma v_k(s')\big] \;\forall s \in \mathcal{S} \end{aligned}$$

- For any arbitrary initialization v_0, this converges to v_*
- The optimal policy π_* is obtained by acting greedily with respect to v_*

- Drawback of Policy Iteration: Involves Policy Evaluation
- Value Iteration turns the Bellman optimality equation into an update rule

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p\left(s', r \,\middle|\, s, a\right)\left[r + \gamma v_k(s')\right] \ \forall s \in \mathcal{S} \end{aligned}$$

- For any arbitrary initialization v_0, this converges to v_*
- The optimal policy π_* is obtained by acting greedily with respect to v_*

- Drawback of Policy Iteration: Involves Policy Evaluation
- Value Iteration turns the Bellman optimality equation into an update rule

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_k(s')\right] \; \forall s \in \mathcal{S} \end{aligned}$$

- For any arbitrary initialization v_0, this converges to v_*
- The optimal policy π_* is obtained by acting greedily with respect to v_*

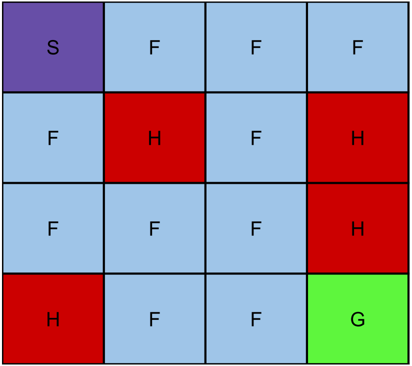
Algorithm parameter $\theta > 0$
Initialize $V(s),\ \forall s \in \mathcal{S}^+$ arbitrarily, except $V(\mathit{terminal}) = 0$
repeat
  $\Delta \leftarrow 0$
  for each $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$
    $\Delta \leftarrow \max \left(\Delta, |v - V(s)| \right)$
  end for
until $\Delta < \theta$
Output a deterministic policy $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evalutaion
Control	Bellman Expectation Equation + greedy policy improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- DP relies on knowledge of MDP
- Model-free prediction utilizes the agent's experience
- Two groups of methods
    - Monte Carlo Learning
    - Temporal Difference Learning
- Closer to real-world scenario

Monte Carlo Policy Evaluation

- Value function directly learned from episodes of experience.
- Learns from complete episodes (no bootstrapping) and is only applicable for episodic tasks.
- Uses the empirical mean return after an episode (instead of the expected return) as the value.
- As more and more returns are observed, the average should converge to the expected value.



Frozen Lake

Code: `https://iis.uibk.ac.at/uibk/auddy/rl/code/model-free/model-free.tar.gz`

Incremental Mean

The means $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally as:

$$
\begin{aligned}
\mu_k &= \frac{1}{k} \sum_{j=1}^{k} x_j \\
&= \frac{1}{k} \Big(x_k + \sum_{j=1}^{k-1} x_j \Big) \\
&= \frac{1}{k} \Big(x_k + (k-1)\mu_{k-1} \Big) \\
&= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})
\end{aligned}
$$

Input:
  Policy π to be evaluated
Initialize
  $V(s) \in \mathbb{R},\ \forall s \in \mathcal{S}$ arbitrarily
  $N(s) \in \mathbb{Z}$ an integer counter $\forall s \in \mathcal{S}$
repeat
  Generate an episode by choosing actions according to policy π:
  $S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for each step of episode, $t = T-1, T-2, ..., 0$ **do**
    $G \leftarrow R_{t+1} + \gamma G$
    if S_t does not appear in $S_0, S_1, .., S_{t-1}$ **then**
      $N(S_t) \leftarrow N(S_t) + 1$
      $V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}\Big(G - V(S_t)\Big)$
    end if
  end for
until forever

MC Policy Evaluation Update for learning V and Q

$$V(S_t) \leftarrow V(S_t) + \alpha\big(G_t - V(S_t)\big)$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big(G_t - Q(S_t, A_t)\big)$$

MC target: The actual return, and we update the value towards this target.

- DP $\rightarrow$ Bootstraps, but needs MDP
  (sampling: ✗, bootstrapping ✓)
- MC $\rightarrow$ Does not need MDP, but needs complete episodes
  (sampling: ✓, bootstrapping ✗)
- **TD** $\rightarrow$ Combines the best of DP and MC
  (sampling: ✓, bootstrapping ✓)
- **TD** is applicable to non-episodic tasks

TD Policy Evaluation Update for learning V and Q

$$V(S) \leftarrow V(S) + \alpha\big(R + \gamma V(S') - V(S)\big)$$
$$Q(S,A) \leftarrow Q(S,A) + \alpha\big(R + \gamma Q(S',A') - Q(S,A)\big)$$

TD target: An estimate of the expected return, and we update the value towards this target.

- DP $\rightarrow$ Bootstraps, but needs MDP
  (sampling: ✗, bootstrapping ✓)
- MC $\rightarrow$ Does not need MDP, but needs complete episodes
  (sampling: ✓, bootstrapping ✗)
- **TD** $\rightarrow$ Combines the best of DP and MC
  (sampling: ✓, bootstrapping ✓)
- **TD** is applicable to non-episodic tasks

TD Policy Evaluation Update for learning V and Q

$$V(S) \leftarrow V(S) + \alpha\big(R + \gamma V(S') - V(S)\big)$$
$$Q(S,A) \leftarrow Q(S,A) + \alpha\big(R + \gamma Q(S',A') - Q(S,A)\big)$$

TD target: An estimate of the expected return, and we update the value towards this target.

- DP $\rightarrow$ Bootstraps, but needs MDP
  (sampling: ✗, bootstrapping ✓)
- MC $\rightarrow$ Does not need MDP, but needs complete episodes
  (sampling: ✓, bootstrapping ✗)
- **TD** $\rightarrow$ Combines the best of DP and MC
  (sampling: ✓, bootstrapping ✓)
- **TD** is applicable to non-episodic tasks

TD Policy Evaluation Update for learning V and Q

$$V(S) \leftarrow V(S) + \alpha\big(R + \gamma V(S') - V(S)\big)$$
$$Q(S,A) \leftarrow Q(S,A) + \alpha\big(R + \gamma Q(S',A') - Q(S,A)\big)$$

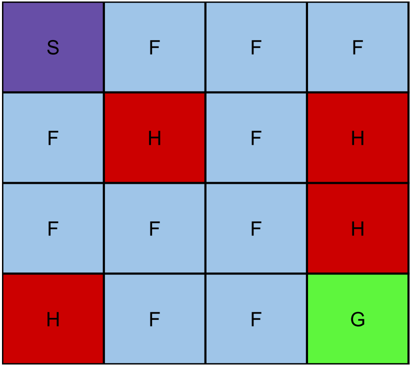
TD target: An estimate of the expected return, and we update the value towards this target.

- DP $\rightarrow$ Bootstraps, but needs MDP
(sampling: ✗, bootstrapping ✓)
- MC $\rightarrow$ Does not need MDP, but needs complete episodes
(sampling: ✓, bootstrapping ✗)
- **TD** $\rightarrow$ Combines the best of DP and MC
(sampling: ✓, bootstrapping ✓)
- **TD** is applicable to non-episodic tasks

TD Policy Evaluation Update for learning V and Q

$$V(S) \leftarrow V(S) + \alpha\big(R + \gamma V(S') - V(S)\big)$$
$$Q(S, A) \leftarrow Q(S, A) + \alpha\big(R + \gamma Q(S', A') - Q(S, A)\big)$$

TD target: An estimate of the expected return, and we update the value towards this target.

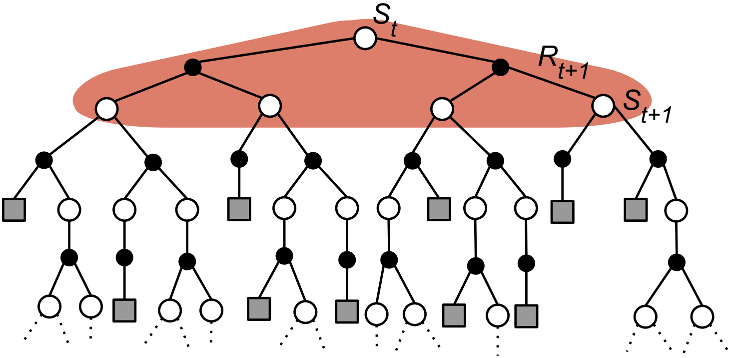


Frozen Lake

Code: `https://iis.uibk.ac.at/uibk/auddy/rl/code/model-free/model-free.tar.gz`

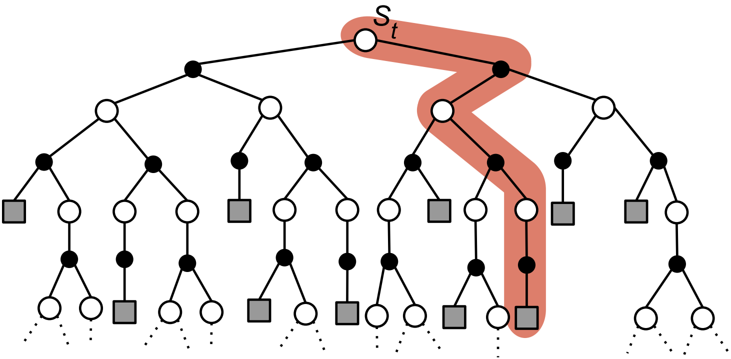
Tabular TD(0) Algorithm

Input: Policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize: $V(s) \in \mathbb{R}$, $\forall s \in \mathcal{S}^+$ arbitrarily, except $V(terminal) = 0$
for each episode **do**
  Initialize S
  for each step of episode until S is terminal **do**
    $A \leftarrow$ action taken by π for S
    Take action A, observe reward R and next state S'
    $V(S) \leftarrow V(S) + \alpha\big(R + \gamma V(S') - V(S)\big)$
    $S \leftarrow S'$
  end for
end for



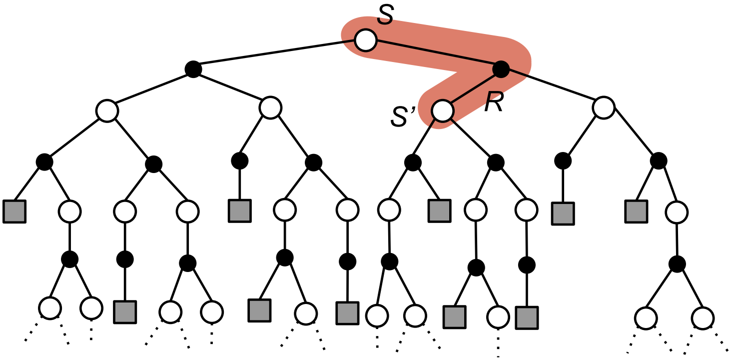
DP: Updates use $\mathbb{E}$ over all states/actions.

$$V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$$



MC: Updates use true return at the end of the episode.

$$V(S_t) \leftarrow V(S_t) + \alpha\big(G_t - V(S_t)\big)$$



TD(0): Updates are done after a single step.

$$V(S) \leftarrow V(S) + \alpha\big(R + \gamma V(S') - V(S)\big)$$

- **On-policy Learning**: The *behavior policy* μ (policy used for generating samples) and the *target policy* π (policy being learned) are the same.
- **Off-policy Learning**: The *target policy* π is learned from experience sampled from a different *behavior policy* μ.

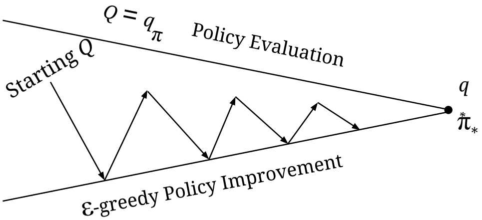
- If action-value function Q for π is known, we can improve the policy using $\pi'(s) = \arg\max_a Q(s,a)$
- TD updates are used for updating Q
  $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
- ϵ-greedy action selection is used to improve π

- SARSA is an on-policy TD control algorithm

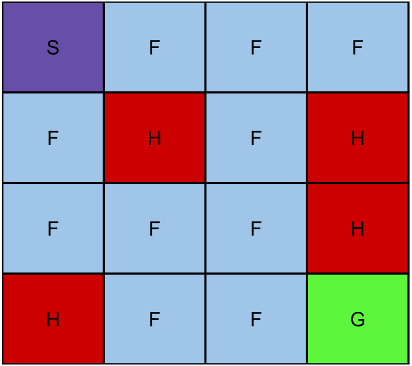
- If action-value function Q for π is known, we can improve the policy using $\pi'(s) = \arg\max_a Q(s, a)$
- TD updates are used for updating Q
  $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
- ϵ-greedy action selection is used to improve π
- SARSA is an on-policy TD control algorithm

- If action-value function Q for π is known, we can improve the policy using $\pi'(s) = \arg\max_a Q(s, a)$
- TD updates are used for updating Q
  $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
- ϵ-greedy action selection is used to improve π
- SARSA is an on-policy TD control algorithm

- If action-value function Q for π is known, we can improve the policy using $\pi'(s) = \arg\max_a Q(s, a)$
- TD updates are used for updating Q
  $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
- ϵ-greedy action selection is used to improve π



- SARSA is an on-policy TD control algorithm



Frozen Lake

Code: `https://iis.uibk.ac.at/uibk/auddy/rl/code/model-free/model-free.tar.gz`

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize: $Q(s, a)$, $\forall s \in \mathcal{S}^+, a \in \mathcal{A}$ arbitrarily, except $Q(terminal, \cdot) = 0$
for each episode **do**
    Initialize S
    Choose A from S using policy derived from Q (eg. ϵ-greedy)
    for each step of episode until S is terminal **do**
        Take action A, observe reward R and next state S'
        Choose A' from S' using policy derived from Q (eg. ϵ-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'; A \leftarrow A'$
    end for
end for

Exercise: SARSA

- Why is SARSA considered an on-policy algorithm?
- What is the backup diagram for SARSA?

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize: $Q(s, a)$, $\forall s \in \mathcal{S}^+, a \in \mathcal{A}$ arbitrarily, except $Q(\textit{terminal}, \cdot) = 0$
for each episode **do**
  Initialize S
  Choose A from S using policy derived from Q (eg. ϵ-greedy)
  for each step of episode until S is terminal **do**
    Take action A, observe reward R and next state S'
    Choose A' from S' using policy derived from Q (eg. ϵ-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'; A \leftarrow A'$
  end for
end for

Exercise: SARSA

- Why is SARSA considered an on-policy algorithm?
- What is the backup diagram for SARSA?

- Q-Learning uses TD updates to learn the optimal action-value function $Q \approx q_*$ independent of the policy followed

$$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$$

- Converges as long as all state-action pairs are visited and updated
- Q-Learning is an off-policy TD control algorithm

- Q-Learning uses TD updates to learn the optimal action-value function $Q \approx q_*$ independent of the policy followed

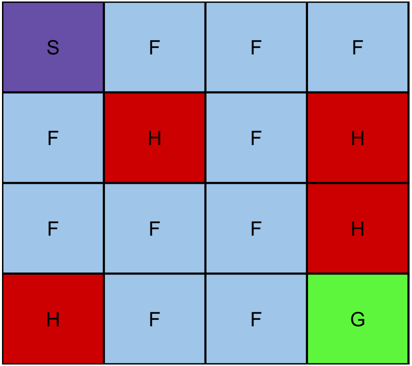
$$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$$

- Converges as long as all state-action pairs are visited and updated
- Q-Learning is an off-policy TD control algorithm

- Q-Learning uses TD updates to learn the optimal action-value function $Q \approx q_*$ independent of the policy followed

$$Q(S, A) \leftarrow Q(S, A) + \alpha \big[R + \gamma \max_a Q(S', a) - Q(S, A) \big]$$

- Converges as long as all state-action pairs are visited and updated
- Q-Learning is an off-policy TD control algorithm



Frozen Lake

Code: `https://iis.uibk.ac.at/uibk/auddy/rl/code/model-free/model-free.tar.gz`

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize: $Q(s, a)$, $\forall s \in \mathcal{S}^+, a \in \mathcal{A}$ arbitrarily, except $Q(terminal, \cdot) = 0$
for each episode **do**
  Initialize S
  for each step of episode until S is terminal **do**
    Choose A from S using policy derived from Q (eg. ϵ-greedy)
    Take action A, observe reward R and next state S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  end for
end for

Exercise: Q-Learning

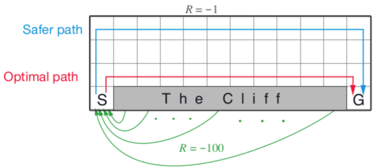
- Why is Q-Learning considered an off-policy algorithm?
- What is the backup diagram for Q-Learning?

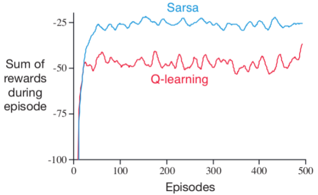
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize: $Q(s, a)$, $\forall s \in \mathcal{S}^+, a \in \mathcal{A}$ arbitrarily, except $Q(terminal, \cdot) = 0$
for each episode **do**
  Initialize S
  for each step of episode until S is terminal **do**
    Choose A from S using policy derived from Q (eg. ϵ-greedy)
    Take action A, observe reward R and next state S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  end for
end for

Exercise: Q-Learning

- Why is Q-Learning considered an off-policy algorithm?
- What is the backup diagram for Q-Learning?

Exercise: **Cliff Walking** (example 6.6 from [Sutton and Barto, 2018])





Undiscounted episodic task with start and goal states. Reward = -100 for falling off cliff, -1 otherwise. Actions have deterministic effect (ϵ-greedy). Both SARSA and Q-learning find solutions.

- Why are the solutions (paths) different?
- Why is the performance of Q-learning worse than SARSA?
- How can we get the same solution from both algorithms?
- If action selection is always performed greedily, is Q-learning exactly the same algorithm as SARSA?

- Recap
- **Value Function Approximation**
  - Learning Value Functions with SGD
  - Linear Approximation
  - Algorithms for Prediction and Control
- **Policy Gradients**
  - Policy Optimization
  - Finite Differences
  - Score Function
  - MC Policy Gradient Control

- Recap
- **Value Function Approximation**
  - Learning Value Functions with SGD
  - Linear Approximation
  - Algorithms for Prediction and Control
- **Policy Gradients**
  - Policy Optimization
  - Finite Differences
  - Score Function
  - MC Policy Gradient Control

- Recap
- **Value Function Approximation**
  - Learning Value Functions with SGD
  - Linear Approximation
  - Algorithms for Prediction and Control
- **Policy Gradients**
  - Policy Optimization
  - Finite Differences
  - Score Function
  - MC Policy Gradient Control

Recap

- Model-free Prediction
  - Monte Carlo
  - TD(0)
- Model-free Control
  - SARSA
  - Q-Learning

- Previously represented V and Q as tables (table entries updated by RL algorithms)
- For large or infinite state spaces this is not possible
- For such problems value functions are represented with parameterized functions

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

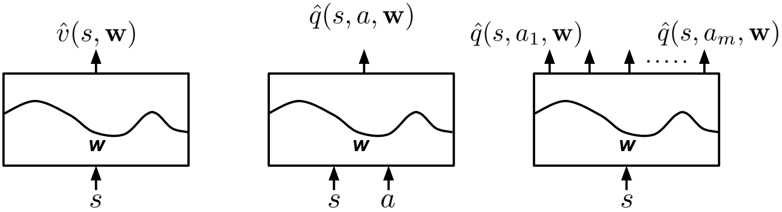
- Generalization is possible
- RL algorithms update the weight vector $\mathbf{w}$
- Use differentiable function approximators
- $\mathbf{w}$ updated using **gradient descent**

Tabular value functions

s	*V(s)*

s	*a*	*Q(s,a)*

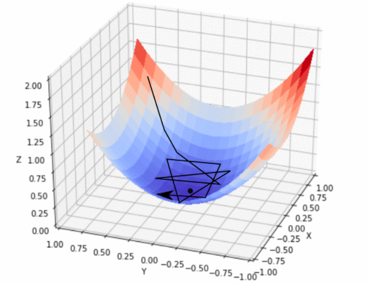
Parameterized value functions



- How to update function parameters $\mathbf{w}$ $\Rightarrow$ Gradient Descent
- If $J(\mathbf{w})$ is a differentiable function of the parameter vector $\mathbf{w}$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \left[\frac{\partial J(\mathbf{w})}{\partial w_1}, ..., \frac{\partial J(\mathbf{w})}{\partial w_n}\right]^T$$

- To find local minimum of $J(\mathbf{w})$, adjust $\mathbf{w}$ in the direction of the negative gradient
- Change in $\mathbf{w}$: $\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_{\mathbf{w}} J(\mathbf{w})$



- For parameterized state-value function $\hat{v}(s, \mathbf{w})$, the objective function is $J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)^2\right]$
- $\nabla_{\mathbf{w}} J(\mathbf{w}) = -2\mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_{\mathbf{w}}\hat{v}(s, \mathbf{w})\right]$
- $\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_{\mathbf{w}} J(\mathbf{w}) = \alpha\mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_w\hat{v}(s, \mathbf{w})\right]$
- SGD samples from this expectation
  $\Delta\mathbf{w} = \alpha\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_w\hat{v}(s, \mathbf{w})$
- Similarly for the action-value function $\hat{q}(s, a, \mathbf{w})$:
  $\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_{\mathbf{w}} J(\mathbf{w}) = \alpha\left(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\right)\nabla_{\mathbf{w}}\hat{q}(s, a, \mathbf{w})$

- For parameterized state-value function $\hat{v}(s, \mathbf{w})$, the objective function is $J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)^2\right]$
- $\nabla_\mathbf{w} J(\mathbf{w}) = -2\mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_\mathbf{w}\hat{v}(s, \mathbf{w})\right]$
- $\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_\mathbf{w} J(\mathbf{w}) = \alpha\mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_w\hat{v}(s, \mathbf{w})\right]$
- SGD samples from this expectation
  $\Delta\mathbf{w} = \alpha\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_w\hat{v}(s, \mathbf{w})$
- Similarly for the action-value function $\hat{q}(s, a, \mathbf{w})$:
  $\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_\mathbf{w} J(\mathbf{w}) = \alpha\left(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\right)\nabla_\mathbf{w}\hat{q}(s, a, \mathbf{w})$

- For parameterized state-value function $\hat{v}(s, \mathbf{w})$, the objective function is $J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)^2\right]$
- $\nabla_\mathbf{w} J(\mathbf{w}) = -2\mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_\mathbf{w}\hat{v}(s, \mathbf{w})\right]$
- $\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_\mathbf{w} J(\mathbf{w}) = \alpha\mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_w\hat{v}(s, \mathbf{w})\right]$
- SGD samples from this expectation
  $\Delta\mathbf{w} = \alpha\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_w\hat{v}(s, \mathbf{w})$
- Similarly for the action-value function $\hat{q}(s, a, \mathbf{w})$:
  $\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_\mathbf{w} J(\mathbf{w}) = \alpha\left(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\right)\nabla_\mathbf{w}\hat{q}(s, a, \mathbf{w})$

- For parameterized state-value function $\hat{v}(s, \mathbf{w})$, the objective function is $J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)^2\right]$
- $\nabla_\mathbf{w} J(\mathbf{w}) = -2\mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_\mathbf{w}\hat{v}(s, \mathbf{w})\right]$
- $\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_\mathbf{w} J(\mathbf{w}) = \alpha\mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_w\hat{v}(s, \mathbf{w})\right]$
- SGD samples from this expectation
  $\Delta\mathbf{w} = \alpha\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_w\hat{v}(s, \mathbf{w})$
- Similarly for the action-value function $\hat{q}(s, a, \mathbf{w})$:
  $\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_\mathbf{w} J(\mathbf{w}) = \alpha\left(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\right)\nabla_\mathbf{w}\hat{q}(s, a, \mathbf{w})$

- For parameterized state-value function $\hat{v}(s, \mathbf{w})$, the objective function is $J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)^2\right]$
- $\nabla_{\mathbf{w}} J(\mathbf{w}) = -2\mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_{\mathbf{w}}\hat{v}(s, \mathbf{w})\right]$
- $\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_{\mathbf{w}} J(\mathbf{w}) = \alpha\mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_w\hat{v}(s, \mathbf{w})\right]$
- SGD samples from this expectation
  $\Delta\mathbf{w} = \alpha\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_w\hat{v}(s, \mathbf{w})$
- Similarly for the action-value function $\hat{q}(s, a, \mathbf{w})$:
  $\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_{\mathbf{w}} J(\mathbf{w}) = \alpha\left(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\right)\nabla_{\mathbf{w}}\hat{q}(s, a, \mathbf{w})$

- Simplest form of function approximation
- state s is represented using a feature vector Φ such as $\Phi(s) = \left[\phi_1(s), \phi_2(s), ..., \phi_n(s)\right]^T$
- $\hat{v}(s, \mathbf{w})$ can be written as a linear combination of $\mathbf{w}$ and Φ $\hat{v}(s, \mathbf{w}) = \Phi(s)^T \mathbf{w} = \sum_{j=1}^{n} \phi_j(s) w_j$
- Loss function: $J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)^2\right] = \mathbb{E}_\pi\left[\left(v_\pi(s) - \Phi(s)^T \mathbf{w}\right)^2\right]$
- Since $\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) = \nabla_{\mathbf{w}} \Phi(s)^T \mathbf{w} = \Phi(s)$ $\Delta \mathbf{w} = \alpha\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right) \nabla_w \hat{v}(s, \mathbf{w}) = \alpha\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\Phi(s)$
- Similarly for the action-value function $\Delta \mathbf{w} = \alpha\left(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\right) \nabla_w \hat{q}(s, a, \mathbf{w}) = \alpha\left(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\right)\Phi(s, a)$

- Simplest form of function approximation
- state s is represented using a feature vector Φ such as $\mathbf{\Phi}(s) = \Big[\phi_1(s), \phi_2(s), ..., \phi_n(s)\Big]^T$
- $\hat{v}(s, \mathbf{w})$ can be written as a linear combination of $\mathbf{w}$ and $\mathbf{\Phi}$ $\hat{v}(s, \mathbf{w}) = \mathbf{\Phi}(s)^T \mathbf{w} = \sum_{j=1}^{n} \phi_j(s) w_j$
- Loss function: $J(\mathbf{w}) = \mathbb{E}_\pi \Big[\big(v_\pi(s) - \hat{v}(s, \mathbf{w})\big)^2 \Big] = \mathbb{E}_\pi \Big[\big(v_\pi(s) - \mathbf{\Phi}(s)^T \mathbf{w}\big)^2 \Big]$
- Since $\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) = \nabla_{\mathbf{w}} \mathbf{\Phi}(s)^T \mathbf{w} = \mathbf{\Phi}(s)$ $\Delta \mathbf{w} = \alpha\big(v_\pi(s) - \hat{v}(s, \mathbf{w})\big) \nabla_w \hat{v}(s, \mathbf{w}) = \alpha\Big(v_\pi(s) - \hat{v}(s, \mathbf{w})\Big)\mathbf{\Phi}(s)$
- Similarly for the action-value function $\Delta \mathbf{w} = \alpha\big(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\big) \nabla_w \hat{q}(s, a, \mathbf{w}) = \alpha\Big(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\Big)\mathbf{\Phi}(s, a)$

- Simplest form of function approximation
- state s is represented using a feature vector Φ such as
  $\mathbf{\Phi}(s) = \left[\phi_1(s), \phi_2(s), ..., \phi_n(s)\right]^T$
- $\hat{v}(s, \mathbf{w})$ can be written as a linear combination of $\mathbf{w}$ and $\mathbf{\Phi}$
  $\hat{v}(s, \mathbf{w}) = \mathbf{\Phi}(s)^T \mathbf{w} = \sum_{j=1}^{n} \phi_j(s) w_j$
- Loss function: $J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)^2\right] = \mathbb{E}_\pi\left[\left(v_\pi(s) - \mathbf{\Phi}(s)^T \mathbf{w}\right)^2\right]$
- Since $\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) = \nabla_{\mathbf{w}} \mathbf{\Phi}(s)^T \mathbf{w} = \mathbf{\Phi}(s)$
  $\Delta \mathbf{w} = \alpha\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right) \nabla_w \hat{v}(s, \mathbf{w}) = \alpha\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right) \mathbf{\Phi}(s)$
- Similarly for the action-value function
  $\Delta \mathbf{w} = \alpha\left(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\right) \nabla_w \hat{q}(s, a, \mathbf{w}) = \alpha\left(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\right) \mathbf{\Phi}(s, a)$

- Simplest form of function approximation
- state s is represented using a feature vector Φ such as
  $\mathbf{\Phi}(s) = \Big[\phi_1(s), \phi_2(s), ..., \phi_n(s)\Big]^T$
- $\hat{v}(s, \mathbf{w})$ can be written as a linear combination of $\mathbf{w}$ and $\mathbf{\Phi}$
  $\hat{v}(s, \mathbf{w}) = \mathbf{\Phi}(s)^T \mathbf{w} = \sum_{j=1}^{n} \phi_j(s) w_j$
- Loss function: $J(\mathbf{w}) = \mathbb{E}_\pi\Big[\big(v_\pi(s) - \hat{v}(s, \mathbf{w})\big)^2\Big] = \mathbb{E}_\pi\Big[\big(v_\pi(s) - \mathbf{\Phi}(s)^T \mathbf{w}\big)^2\Big]$
- Since $\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) = \nabla_{\mathbf{w}} \Phi(s)^T \mathbf{w} = \Phi(s)$
  $\Delta \mathbf{w} = \alpha\big(v_\pi(s) - \hat{v}(s, \mathbf{w})\big) \nabla_w \hat{v}(s, \mathbf{w}) = \alpha\Big(v_\pi(s) - \hat{v}(s, \mathbf{w})\Big)\Phi(s)$
- Similarly for the action-value function
  $\Delta \mathbf{w} = \alpha\big(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\big) \nabla_w \hat{q}(s, a, \mathbf{w}) = \alpha\Big(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\Big)\Phi(s, a)$

- Simplest form of function approximation
- state s is represented using a feature vector Φ such as
  $\mathbf{\Phi}(s) = \Big[\phi_1(s), \phi_2(s), ..., \phi_n(s)\Big]^T$
- $\hat{v}(s, \mathbf{w})$ can be written as a linear combination of $\mathbf{w}$ and $\mathbf{\Phi}$
  $\hat{v}(s, \mathbf{w}) = \mathbf{\Phi}(s)^T \mathbf{w} = \sum_{j=1}^{n} \phi_j(s) w_j$
- Loss function: $J(\mathbf{w}) = \mathbb{E}_\pi\Big[\big(v_\pi(s) - \hat{v}(s, \mathbf{w})\big)^2\Big] = \mathbb{E}_\pi\Big[\big(v_\pi(s) - \mathbf{\Phi}(s)^T \mathbf{w}\big)^2\Big]$
- Since $\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) = \nabla_{\mathbf{w}} \mathbf{\Phi}(s)^T \mathbf{w} = \mathbf{\Phi}(s)$
  $\Delta \mathbf{w} = \alpha\big(v_\pi(s) - \hat{v}(s, \mathbf{w})\big) \nabla_w \hat{v}(s, \mathbf{w}) = \alpha\Big(v_\pi(s) - \hat{v}(s, \mathbf{w})\Big) \mathbf{\Phi}(s)$
- Similarly for the action-value function
  $\Delta \mathbf{w} = \alpha\big(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\big) \nabla_w \hat{q}(s, a, \mathbf{w}) = \alpha\Big(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\Big) \mathbf{\Phi}(s, a)$

- Simplest form of function approximation
- state s is represented using a feature vector Φ such as $\boldsymbol{\Phi}(s) = \Big[\phi_1(s), \phi_2(s), ..., \phi_n(s)\Big]^T$
- $\hat{v}(s, \mathbf{w})$ can be written as a linear combination of $\mathbf{w}$ and $\boldsymbol{\Phi}$ $\hat{v}(s, \mathbf{w}) = \boldsymbol{\Phi}(s)^T \mathbf{w} = \sum_{j=1}^{n} \phi_j(s) w_j$
- Loss function: $J(\mathbf{w}) = \mathbb{E}_\pi\Big[\big(v_\pi(s) - \hat{v}(s, \mathbf{w})\big)^2\Big] = \mathbb{E}_\pi\Big[\big(v_\pi(s) - \boldsymbol{\Phi}(s)^T \mathbf{w}\big)^2\Big]$
- Since $\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) = \nabla_{\mathbf{w}} \boldsymbol{\Phi}(s)^T \mathbf{w} = \boldsymbol{\Phi}(s)$ $\Delta \mathbf{w} = \alpha\big(v_\pi(s) - \hat{v}(s, \mathbf{w})\big) \nabla_w \hat{v}(s, \mathbf{w}) = \alpha\Big(v_\pi(s) - \hat{v}(s, \mathbf{w})\Big) \boldsymbol{\Phi}(s)$
- Similarly for the action-value function $\Delta \mathbf{w} = \alpha\big(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\big) \nabla_w \hat{q}(s, a, \mathbf{w}) = \alpha\Big(q_\pi(s, a) - \hat{q}(s, a, \mathbf{w})\Big) \boldsymbol{\Phi}(s, a)$

- Polynomials: $\phi_i(\mathbf{s}) = \prod_{j=1}^{k} s_j^{c_{i,j}}$ where $\mathbf{s} = [s_1, s_2, ..., s_k]^T$, $c_{i,j} \in \{0, 1, ..., n\}$ for $n \geq 0$. For k dimensions, $\mathbf{\Phi}$ has $(n+1)^k$ different features.

> Exercise: **Polynomial Features**
>
> Let $\mathbf{s} = [s_1, s_2]^T$, thus $k = 2$ and let $n = 2$. Calculate $\mathbf{\Phi}$?

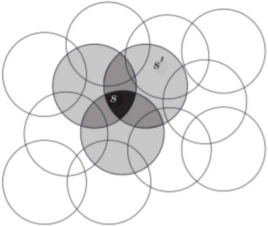
- Coarse Coding and Tile Coding
- Radial Basis Functions $\phi_i(s) = exp\left(-\frac{||s-c_i||^2}{2\sigma_i^2}\right)$

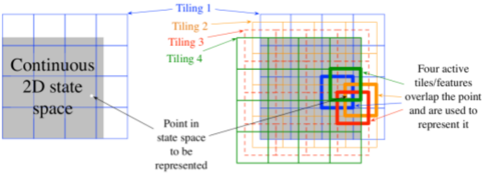
- Polynomials: $\phi_i(\mathbf{s}) = \prod_{j=1}^{k} s_j^{c_{i,j}}$ where $\mathbf{s} = [s_1, s_2, ..., s_k]^T$, $c_{i,j} \in \{0, 1, ..., n\}$ for $n \geq 0$. For k dimensions, $\mathbf{\Phi}$ has $(n+1)^k$ different features.

Exercise: Polynomial Features

Let $\mathbf{s} = [s_1, s_2]^T$, thus $k = 2$ and let $n = 2$. Calculate $\mathbf{\Phi}$?

- Coarse Coding and Tile Coding





- Polynomials: $\phi_i(\mathbf{s}) = \prod_{j=1}^{k} s_j^{c_{i,j}}$ where $\mathbf{s} = [s_1, s_2, ..., s_k]^T$, $c_{i,j} \in \{0, 1, ..., n\}$ for $n \geq 0$. For k dimensions, $\mathbf{\Phi}$ has $(n+1)^k$ different features.

> Exercise: **Polynomial Features**
>
> Let $\mathbf{s} = [s_1, s_2]^T$, thus $k = 2$ and let $n = 2$. Calculate $\mathbf{\Phi}$?

- Coarse Coding and Tile Coding
- Radial Basis Functions $\phi_i(s) = exp\Big(-\frac{||s-c_i||^2}{2\sigma_i^2}\Big)$

Weight Update for MC prediction

$$\Delta \mathbf{w} = \alpha\Big(G_t - \hat{v}(S_t, \mathbf{w})\Big)\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

Input:
    Policy π to be evaluated
    Algorithm parameter: step size $\alpha > 0$
    A differentiable value function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Initialize
    Value-function weight vector $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g. $\mathbf{w} = 0$)
repeat
    Generate episode using π: $S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T$
    for each step of episode, $t = 0, 1, ..., T-1$ **do**
        $\mathbf{w} = \mathbf{w} + \alpha\Big(G_t - \hat{v}(S_t, \mathbf{w})\Big)\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$
    end for
until forever

Weight Update for TD prediction

$$\Delta \mathbf{w} = \alpha\Big(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})\Big)\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

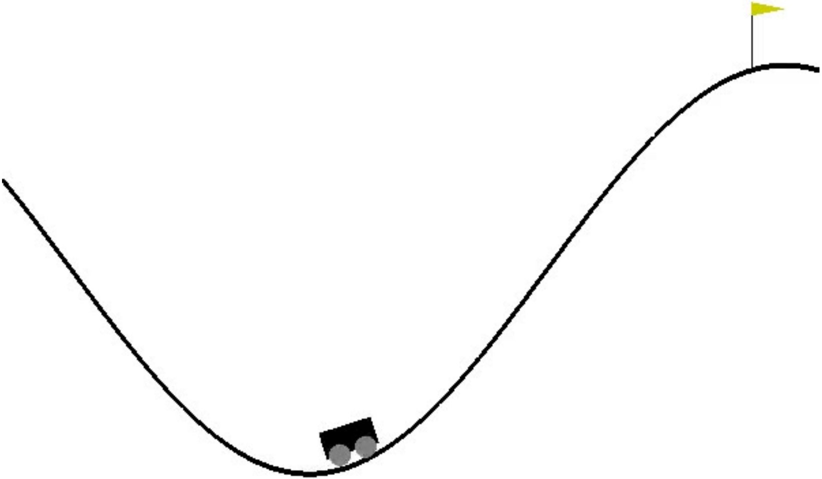
Input:
  Policy π to be evaluated
  Algorithm parameter: step size $\alpha > 0$
  A differentiable value function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\mathit{terminal}) = 0$
Initialize
  Value-function weight vector $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g. $\mathbf{w} = [0, ..., 0]^T$)
repeat
  Initialize S
  for each step of episode until S is terminal **do**
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{w} = \mathbf{w} + \alpha\Big(R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})\Big)\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$
    $S \leftarrow S'$
  end for
until forever

Weight Update for TD(0) control

$$\Delta \mathbf{w} = \alpha\Big(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})\Big)\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

Input: A differentiable action-value function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize value function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g. $\mathbf{w} = [0, ..., 0]^T$)
for each episode **do**
    Initialize S
    Choose A from S using policy derived from $\hat{q}$ (eg. ϵ-greedy)
    for each step of episode **do**
        Take action A, observe reward R and next state S'
        if S' is terminal **then**
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(R - \hat{q}(S, A, \mathbf{w})\big)\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$
            Go to next episode
        end if
        Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (eg. ϵ-greedy)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(R + \gamma \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big)\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$
        $S \leftarrow S'$; $A \leftarrow A'$
    end for
end for

Semi-gradient SARSA - Demo



Code:
https:
//iis.uibk.ac.at/uibk/auddy/rl/code/value-func-approximation/value-func-approximation.tar.gz

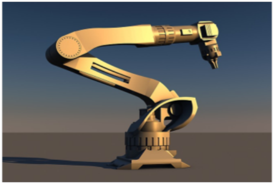
Deadly Triad:

- **Function Approximation** - Using a parametric function to significantly generalize from a large number of examples.
- **Bootstrapping** - Learning value estimates from other value estimates.
- **Off-policy Learning** - Learning about a policy from data not due to that policy.

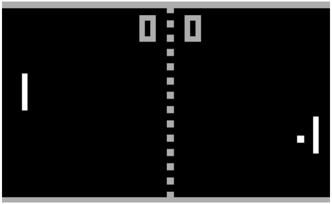
Algorithm	**Tabular**	**Linear FA**	**Non-linear FA**
SARSA	*Yes*	*Chatters around optimal*	*No*
Q-Learning	*Yes*	*No*	*No*

- Earlier: $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$ and $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$
- Policy Gradient: Directly parameterize the policy using $\theta \in \mathbb{R}^d$
- Change θ so that we can compute the best possible policy
- Parameterized policy $\pi_\theta(s, a) = \mathbb{P}\left[a | s, \theta\right]$
- Optimize θ directly without going through value-functions.

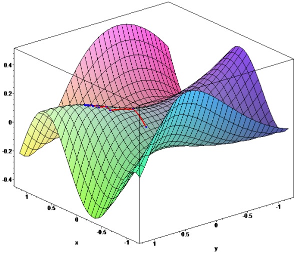
- Better convergence
- Continuous actions



- Simpler to learn policy directly



- Goal: Find the best possible θ
- Need to determine the quality of the policy π_θ
- For episodic tasks: $J(\theta) = v_{\pi_\theta}(s_0)$
- Maximize $J(\theta) \Rightarrow$ *Gradient Ascent*



- $\Delta\theta = \alpha \nabla_\theta J(\theta)$ where $\nabla_\theta J(\theta) = [\frac{\partial J(\theta)}{\partial \theta_1}, ..., \frac{\partial J(\theta)}{\partial \theta_n}]^T$

Input: Policy parameterization instance θ_h (n-dimensional vector)
for $i = 1$ to n **do**
  Generate policy variation $\Delta\theta_i$ by perturbing θ_h in dimension i
  Estimate $\hat{J}_i \approx J(\theta_h + \Delta\theta_i) = \sum_{k=0}^{H} \gamma^k R_k$ from roll-out
  Estimate $\hat{J}_{ref} = J(\theta_h)$ from roll-out
  Compute $\Delta\hat{J}_i \approx J(\theta_h + \Delta\theta_i) - \hat{J}_{ref}$
end for
Return the policy gradient estimate
$\mathbf{g}_{FD} \approx \nabla_\theta J|_{\theta=\theta_h} = \left(\Delta\theta^T \Delta\theta\right)^{-1} \Delta\theta^T \Delta\hat{\mathbf{J}}$
where $\Delta\theta = [\Delta\theta_1, ..., \Delta\theta_n]^T$ and $\Delta\hat{\mathbf{J}} = [\Delta\hat{J}_1, ..., \Delta\hat{J}_n]^T$
(performing regression over the data points $[\Delta\theta_1, \Delta\hat{J}_1], ..., [\Delta\theta_n, \Delta\hat{J}_n]$)

Exercise: PG with Finite Differences

Let the objective function $J(\boldsymbol{\theta}) = \theta_1^2 + \theta_1\theta_2 + \theta_2^2$ where $\boldsymbol{\theta} = [\theta_1, \theta_2]^T$. Calculate the policy gradient using Finite Differences and verify analytically that it approximates the true gradient at $\boldsymbol{\theta}_h = [\theta_1 = 1.0, \theta_2 = 1.0]^T$.

- Assume π_θ is differentiable when it is $\neq 0$ and $\nabla_\theta \pi_\theta(s,a)$ is known
- Liklihood Ratio:
  $\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a)\dfrac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} = \pi_\theta(s,a)\ \nabla_\theta \ln \pi_\theta(s,a)$
  (since $\nabla_\theta \ln(z) = \frac{1}{z}\nabla_\theta z$)
- Score function tells us how to adjust our policy so that the liklihood of **good actions** is increased.

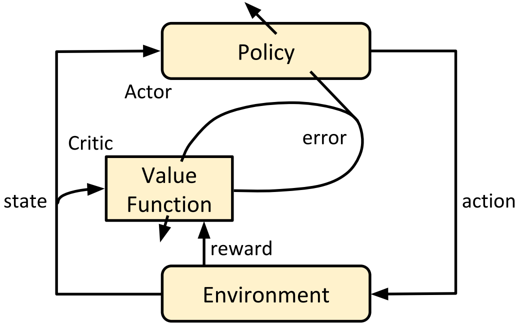
Action-space	$\boldsymbol{\pi_\theta(s,a)}$	$\boldsymbol{\nabla_\theta \ln \pi_\theta(s,a)}$
Discrete $\pi_\theta(s,a) \propto e^{(\Phi(s,a)^T\theta)}$	$\dfrac{exp\left(\Phi(s,a)^T\theta\right)}{\sum_b exp\left(\Phi(s,b)^T\theta\right)}$	$\Phi(s,a) - \mathbb{E}_{\pi_\theta}\left[\Phi(s,\cdot)\right]$
Continuous $\mu(s,\theta) = \Phi(s)^T\theta$ $a \sim \mathcal{N}(\mu(s,\theta),\sigma^2)$	$\dfrac{1}{\sigma\sqrt{2\pi}}exp\left(-\dfrac{\left(a-\mu(s,\theta)\right)^2}{2\sigma^2}\right)$	$\dfrac{\left(a-\Phi(s)^T\theta\right)\Phi(s)}{\sigma^2}$

Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ln \pi_\theta(s,a) \cdot q_{\pi_\theta}(s,a)\right]$$

Input: A differentiable policy parameterization $\pi(a|s,\theta)$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\theta \in \mathbb{R}^d$ (e.g. to 0)
for each episode **do**
  Generate episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$ by following $\pi(\cdot|\cdot,\theta)$
  for each step of episode $t = 0, 1, ..., T-1$ **do**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\theta \leftarrow \theta + \alpha \nabla_\theta \ln \pi(A_t|S_t,\theta) \cdot \gamma^t G$
  end for
end for

- MC Policy Gradient using only $\pi_\theta(s, a)$ has high variance
- Actor-Critic methods use $\hat{q}(s, a, \mathbf{w})$ as a **critic**
- **Critic** $\rightarrow$ Updates $\mathbf{w}$ using TD(0) update
  $r + \gamma\hat{q}(s', a', \mathbf{w}) - \hat{q}(s, a, \mathbf{w})$
- **Actor** $\rightarrow$ Updates θ in direction suggested by critic
  $\Delta\theta = \alpha\nabla_\theta \ln \pi_\theta(s, a)\hat{q}(s, a, \mathbf{w})$



Wrap Up

References I

[Abbeel et al., 2010] Abbeel, P., Coates, A., and Ng, A. Y. (2010).
Autonomous helicopter aerobatics through apprenticeship learning.
The International Journal of Robotics Research, 29(13):1608–1639.

[Andrychowicz et al., 2018] Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. (2018).
Learning dexterous in-hand manipulation.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015).
Human-level control through deep reinforcement learning.
Nature, 518(7540):529.

[Peters et al., 2009] Peters, J., Kober, J., Muelling, K., Nguyen-Tuong, D., and Kroemer, O. (2009).
Towards motor skill learning for robotics.
In *Proceedings of the International Symposium on Robotics Research (ISRR), Invited Paper*.

References II

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016).
Mastering the game of go with deep neural networks and tree search.
nature, 529(7587):484.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018).
Reinforcement Learning.
The MIT Press.