Introduction to Artificial Intelligence

Reinforcement Learning

Andres Mendez-Vazquez

April 26, 2019

Outline

Outline

Reinforcement Learning is learning what to do

What does Reinforcement Learning want?

- Maximize a numerical reward signal

The learner is not told which actions to take

- A discovery must be performed to obtain the actions that yield the most rewards

Reinforcement Learning is learning what to do

What does Reinforcement Learning want?

- Maximize a numerical reward signal

The learner is not told which actions to take

- A discovery must be performed to obtain the actions that yield the most rewards

These ideas come from Dynamical Systems Theory

Specifically

- As the optimal control of incompletely-known **Markov Decision Processes** (MDP).

Thus, the device must do the following

- Interact with the environment to learn by using punishments and rewards.

These ideas come from Dynamical Systems Theory

Specifically

- As the optimal control of incompletely-known **Markov Decision Processes** (MDP).

Thus, the device must do the following

- Interact with the environment to learn by using punishments and rewards.

Outline

A mobile robot

decides whether it should enter a new room

- in search of more trash to collect
- or going back to charge its battery

It needs to take a decision based on the

- current charge level of its battery
- how quickly it can arrive to its base

A mobile robot

decides whether it should enter a new room

- in search of more trash to collect
- or going back to charge its battery

It needs to take a decision based on the

- current charge level of its battery
- how quickly it can arrive to its base

Outline

A K-Armed Bandit Problem

Definition

- You are faced repeatedly with a choice among k different options, or actions.

After each choice you receive a numerical reward chosen

- From an stationary probability distribution depending on the actions.

Definition Stationary Probability

- A stationary distribution of a Markov chain is a probability distribution that remains unchanged in the Markov chain as time progresses.

A K-Armed Bandit Problem

Definition

- You are faced repeatedly with a choice among k different options, or actions.

After each choice you receive a numerical reward chosen

- From an stationary probability distribution depending on the actions.

Definition Stationary Probability

- A stationary distribution of a Markov chain is a probability distribution that remains unchanged in the Markov chain as time progresses.

A K-Armed Bandit Problem

Definition

- You are faced repeatedly with a choice among k different options, or actions.

After each choice you receive a numerical reward chosen

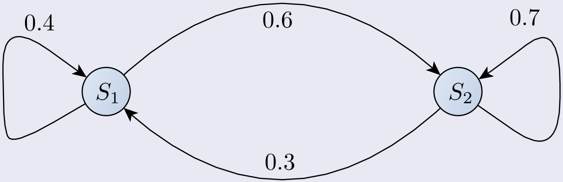
- From an stationary probability distribution depending on the actions.

Definition Stationary Probability

- A stationary distribution of a Markov chain is a probability distribution that remains unchanged in the Markov chain as time progresses.

Example

A Two State System



Therefore

Each of the k actions has an expected value

$$q_*\left(a\right) = E\left[R_t | A_t = a\right]$$

Something Notable

- If you knew the value of each action,
  - It would be trivial to solve the k-armed bandit problem.

Therefore

Each of the k actions has an expected value

$$q_*(a) = E[R_t | A_t = a]$$

Something Notable

- If you knew the value of each action,
  - It would be trivial to solve the k-armed bandit problem.

What do we want?

To find the estimated value of action a at time t, $Q_t(a)$

$$|Q_t(a) - q_*(a)| < \epsilon \text{ with } \epsilon \text{ as small as possible}$$

Outline

Greedy Actions

Intuitions

- To choose the greatest $E\left[R_t|A_t = a\right]$

This is know as Exploitation

- You are exploiting your current knowledge of the values of the actions.

Contrasting to Exploration

- When you select the non-gready actions to obtain a better knowledge of the non-gready actions.

Greedy Actions

Intuitions

- To choose the greatest $E\left[R_t \middle| A_t = a\right]$

This is know as **Exploitation**

- You are exploiting your current knowledge of the values of the actions.

Contrasting to **Exploration**

- When you select the non-gready actions to obtain a better knowledge of the non-gready actions.

Greedy Actions

Intuitions

- To choose the greatest $E\left[R_t|A_t = a\right]$

This is know as **Exploitation**

- You are exploiting your current knowledge of the values of the actions.

Contrasting to **Exploration**

- When you select the non-gready actions to obtain a better knowledge of the non-gready actions.

An Interesting Conundrum

It has been observed that

- Exploitation maximizes the expected reward on the one step,
- Exploration may produce the greater total reward in the long run.

Thus, the idea

- The value of a state s is the total amount of reward an agent can expect to accumulate over the future, starting from s.

Thus, Values

- They indicate the long-term profit of states after taking into account
  - The states that follow and their rewards.

An Interesting Conundrum

It has been observed that

- Exploitation maximizes the expected reward on the one step,
- Exploration may produce the greater total reward in the long run.

Thus, the idea

- The value of a state s is the total amount of reward an agent can expect to accumulate over the future, starting from s.

Thus, Values

- They indicate the long-term profit of states after taking into account
  - The states that follow and their rewards.

An Interesting Conundrum

It has been observed that

- Exploitation maximizes the expected reward on the one step,
- Exploration may produce the greater total reward in the long run.

Thus, the idea

- The value of a state s is the total amount of reward an agent can expect to accumulate over the future, starting from s.

Thus, Values

- They indicate the long-term profit of states after taking into account
  - The states that follow and their rewards.

Outline

Therefore, the parts of a Reinforcement Learning

A Policy

- It defines the learning agent's way of behaving at a given time.
  - The policy is the core of a reinforcement learning system
  - In general, policies may be stochastic

  - The system's sole objective is to maximize the total reward over the long run
  - This encourage Exploitation
  - It is immediate ... you might want something with more long term

  - This encourage Exploration

Therefore, the parts of a Reinforcement Learning

A Policy

- It defines the learning agent's way of behaving at a given time.
  - The policy is the core of a reinforcement learning system
  - In general, policies may be stochastic

A Reward Signal

- It defines the goal of a reinforcement learning problem.
  - The system's sole objective is to maximize the total reward over the long run
  - This encourage Exploitation
  - It is immediate... you might want something with more long term.

- This encourage Exploration

Therefore, the parts of a Reinforcement Learning

A Policy

- It defines the learning agent's way of behaving at a given time.
  - The policy is the core of a reinforcement learning system
  - In general, policies may be stochastic

A Reward Signal

- It defines the goal of a reinforcement learning problem.
  - The system's sole objective is to maximize the total reward over the long run
  - This encourage Exploitation
  - It is immediate... you might want something with more long term.

A Value Function

- It specifies what is good in the long run.
  - This encourage Exploration

Therefore, the parts of a Reinforcement Learning

A Policy

- It defines the learning agent's way of behaving at a given time.
  - The policy is the core of a reinforcement learning system
  - In general, policies may be stochastic

A Reward Signal

- It defines the goal of a reinforcement learning problem.
  - The system's sole objective is to maximize the total reward over the long run
  - This encourage Exploitation
  - It is immediate... you might want something with more long term.

A Value Function

- It specifies what is good in the long run.
  - This encourage Exploration

Therefore, the parts of a Reinforcement Learning

A Policy

- It defines the learning agent's way of behaving at a given time.
  - The policy is the core of a reinforcement learning system
  - In general, policies may be stochastic

A Reward Signal

- It defines the goal of a reinforcement learning problem.
  - The system's sole objective is to maximize the total reward over the long run
  - This encourage Exploitation
  - It is immediate... you might want something with more long term.

A Value Function

- It specifies what is good in the long run.
  - This encourage Exploration

Therefore, the parts of a Reinforcement Learning

A Policy

- It defines the learning agent's way of behaving at a given time.
  - The policy is the core of a reinforcement learning system
  - In general, policies may be stochastic

A Reward Signal

- It defines the goal of a reinforcement learning problem.
  - The system's sole objective is to maximize the total reward over the long run
  - This encourage Exploitation
  - It is immediate... you might want something with more long term.

A Value Function

- It specifies what is good in the long run.
  - This encourage Exploration

Therefore, the parts of a Reinforcement Learning

A Policy

- It defines the learning agent's way of behaving at a given time.
  - The policy is the core of a reinforcement learning system
  - In general, policies may be stochastic

A Reward Signal

- It defines the goal of a reinforcement learning problem.
  - The system's sole objective is to maximize the total reward over the long run
  - This encourage Exploitation
  - It is immediate... you might want something with more long term.

A Value Function

- It specifies what is good in the long run.
  - This encourage Exploration

Therefore, the parts of a Reinforcement Learning

A Policy

- It defines the learning agent's way of behaving at a given time.
  - The policy is the core of a reinforcement learning system
  - In general, policies may be stochastic

A Reward Signal

- It defines the goal of a reinforcement learning problem.
  - The system's sole objective is to maximize the total reward over the long run
  - This encourage Exploitation
  - It is immediate... you might want something with more long term.

A Value Function

- It specifies what is good in the long run.
  - This encourage Exploration

Therefore, the parts of a Reinforcement Learning

A Policy

- It defines the learning agent's way of behaving at a given time.
    - The policy is the core of a reinforcement learning system
    - In general, policies may be stochastic

A Reward Signal

- It defines the goal of a reinforcement learning problem.
    - The system's sole objective is to maximize the total reward over the long run
    - This encourage Exploitation
    - It is immediate... you might want something with more long term.

A Value Function

- It specifies what is good in the long run.
    - This encourage Exploration

Finally

A Model of the Environment

- This simulates the behavior of the environment
  - Allowing to make inferences on how the environment can behave

Something Notable

- Models are used for planning
  - Meaning we made decisions before they are executed
  - Thus, Reinforcement Learning Methods using models are called model-based

Finally

A Model of the Environment

- This simulates the behavior of the environment
  - Allowing to make inferences on how the environment can behave

Something Notable

- Models are used for planning
  - Meaning we made decisions before they are executed
  - Thus, Reinforcement Learning Methods using models are called model-based

Outline

Limitations

Given that

- Reinforcement learning relies heavily on the concept of state

Therefore, it has the same problem as in Markov Decision Processes

- Defining the state to obtain a better representation of the search space.

This is where the creative part of the problems come to be

- As we have seen, solutions in AI really a lot in the creativity of the solution...

Limitations

Given that

- Reinforcement learning relies heavily on the concept of state

Therefore, it has the same problem as in Markov Decision Processes

- Defining the state to obtain a better representation of the search space.

This is where the creative part of the problems come to be

- As we have seen, solutions in AI really a lot in the creativity of the solution...

Limitations

Given that

- Reinforcement learning relies heavily on the concept of state

Therefore, it has the same problem as in Markov Decision Processes

- Defining the state to obtain a better representation of the search space.

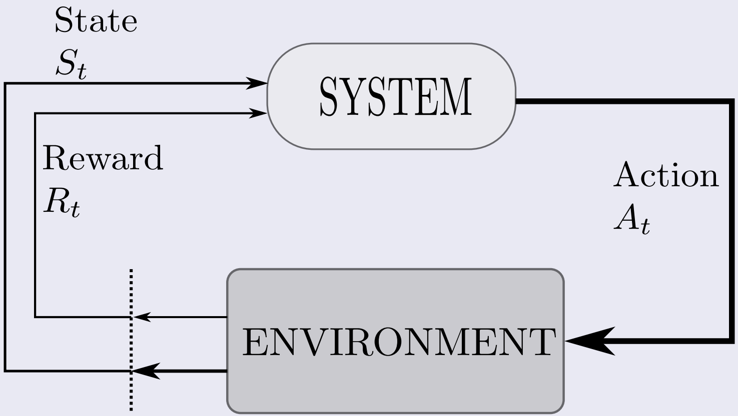
This is where the creative part of the problems come to be

- As we have seen, solutions in AI really a lot in the creativity of the solution...

Outline

The System–Environment Interaction

In a Markov Decision Process



Outline

Markov Process and Markov Decision Process

Definition of Markov Process

- A sequence of states is Markov if and only if the probability of moving to the next state s_{t+1} depends only on the present state s_t

$$P\left[s_{t+1}|s_t\right] = P\left[s_{t+1}|s_1, ..., s_t\right]$$

Something Notable

- We always talk about time-homogeneous Markov chain in Reinforcement Learning:
  - the probability of the transition is independent of t

$$P\left[s_{t+1} = s'|s_t = s\right] = P\left[s_t = s'|s_{t-1} = s\right]$$

Markov Process and Markov Decision Process

Definition of Markov Process

- A sequence of states is Markov if and only if the probability of moving to the next state s_{t+1} depends only on the present state s_t

$$P\left[s_{t+1}|s_t\right] = P\left[s_{t+1}|s_1, ..., s_t\right]$$

Something Notable

- We always talk about time-homogeneous Markov chain in Reinforcement Learning:
  - the probability of the transition is independent of t

$$P\left[s_{t+1} = s'|s_t = s\right] = P\left[s_t = s'|s_{t-1} = s\right]$$

Formally

Definition (Markov Process)

- a Markov Process (or Markov Chain) is a tuple $(\mathcal{S}, \mathcal{P})$, where
  1. $\mathcal{S}$ is a finite set of states
  2. $\mathcal{P}$ is a state transition probability matrix. $\mathcal{P}_{ss'} = P\left[s_{t+1} = s' | s_t = s\right]$

Thus, the dynamic of the Markov process is as follow

$$s_0 \xrightarrow{\mathcal{P}_{s_0 s_1}} s_1 \xrightarrow{\mathcal{P}_{s_1 s_2}} s_2 \xrightarrow{\mathcal{P}_{s_2 s_3}} s_3 \longrightarrow \cdots$$

Formally

Definition (Markov Process)

- a Markov Process (or Markov Chain) is a tuple $(\mathcal{S}, \mathcal{P})$, where
  1. $\mathcal{S}$ is a finite set of states
  2. $\mathcal{P}$ is a state transition probability matrix. $\mathcal{P}_{ss'} = P\left[s_{t+1} = s' | s_t = s\right]$

Thus, the dynamic of the Markov process is as follow

$$s_0 \xrightarrow{\mathcal{P}_{s_0 s_1}} s_1 \xrightarrow{\mathcal{P}_{s_1 s_2}} s_2 \xrightarrow{\mathcal{P}_{s_2 s_3}} s_3 \longrightarrow \cdots$$

Then, we get the following familiar definition

Definition

- MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \alpha \rangle$

1. $\mathcal{S}$ is a finite set of states
2. $\mathcal{A}$ is a finite set of actions
3. $\mathcal{P}$ is a state transition probability matrix.

$$\mathcal{P}_{ss'} = P\left[s_{t+1} = s' | s_t = s, A_t = a\right]$$

4. $\alpha \in [0, 1]$ is called the discount factor
5. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function

Then, we get the following familiar definition

Definition

- MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \alpha \rangle$

1. $\mathcal{S}$ is a finite set of states
2. $\mathcal{A}$ is a finite set of actions
3. $\mathcal{P}$ is a state transition probability matrix.

$$\mathcal{P}_{ss'} = P\left[s_{t+1} = s' | s_t = s, A_t = a\right]$$

4. $\alpha \in [0, 1]$ is called the discount factor
5. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function

Thus, the dynamic of the Markov Decision Process is as follow using a probabilty to pick an action

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \longrightarrow \cdots$$

Then, we get the following familiar definition

Definition

- MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \alpha \rangle$

1. $\mathcal{S}$ is a finite set of states
2. $\mathcal{A}$ is a finite set of actions
3. $\mathcal{P}$ is a state transition probability matrix,

$$\mathcal{P}_{ss'} = P\left[s_{t+1} = s' \middle| s_t = s, A_t = a\right]$$

4. $\alpha \in [0,1]$ is called the discount factor
5. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function

Thus, the dynamic of the Markov Decision Process is as follow using a probabilty to pick an action

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{} \cdots$$

Then, we get the following familiar definition

Definition

- MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \alpha \rangle$

1. $\mathcal{S}$ is a finite set of states
2. $\mathcal{A}$ is a finite set of actions
3. $\mathcal{P}$ is a state transition probability matrix.

$$\mathcal{P}_{ss'} = P\left[s_{t+1} = s' | s_t = s, A_t = a\right]$$

4. $\alpha \in [0, 1]$ is called the discount factor
5. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function

Thus, the dynamic of the Markov Decision Process is as follow using a probabilty to pick an action

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \longrightarrow \cdots$$

Then, we get the following familiar definition

Definition

- MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \alpha \rangle$

1. $\mathcal{S}$ is a finite set of states
2. $\mathcal{A}$ is a finite set of actions
3. $\mathcal{P}$ is a state transition probability matrix.

$$\mathcal{P}_{ss'} = P\left[s_{t+1} = s' | s_t = s, A_t = a\right]$$

4. $\alpha \in [0, 1]$ is called the discount factor
5. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function

Thus, the dynamic of the Markov Decision Process is as follow using a probabilty to pick an action

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \longrightarrow \cdots$$

Then, we get the following familiar definition

Definition

- MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \alpha \rangle$

1. $\mathcal{S}$ is a finite set of states
2. $\mathcal{A}$ is a finite set of actions
3. $\mathcal{P}$ is a state transition probability matrix.

$$\mathcal{P}_{ss'} = P\left[s_{t+1} = s' | s_t = s, A_t = a\right]$$

4. $\alpha \in [0, 1]$ is called the discount factor
5. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function

Thus, the dynamic of the Markov Decision Process is as follow using a probabilty to pick an action

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \longrightarrow \cdots$$

Then, we get the following familiar definition

Definition

- MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \alpha \rangle$

1. $\mathcal{S}$ is a finite set of states
2. $\mathcal{A}$ is a finite set of actions
3. $\mathcal{P}$ is a state transition probability matrix.

$$\mathcal{P}_{ss'} = P\left[s_{t+1} = s' | s_t = s, A_t = a\right]$$

4. $\alpha \in [0, 1]$ is called the discount factor
5. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function

Thus, the dynamic of the Markov Decision Process is as follow using a probabilty to pick an action

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \longrightarrow \cdots$$

Outline

The Goal of Reinforcement Learning

We want to maximize the expected value of the return

- i.e. To obtain the optimal policy!!!

Thus, as in our previous study of MDP's

$$G_t = R_{t+1} + \alpha R_{t+2} + \cdots = \sum_{k=0}^{\infty} \alpha^k R_{t+k+1}$$

The Goal of Reinforcement Learning

We want to maximize the expected value of the return

- i.e. To obtain the optimal policy!!!

Thus, as in our previous study of MDP's

$$G_t = R_{t+1} + \alpha R_{t+2} + \cdots = \sum_{k=0}^{\infty} \alpha^k R_{t+k+1}$$

Remarks

Interpretation

- The discounted future rewards can be interpreted as the **current value of future rewards**.

The reward α

- α close to 0 leads to "short-sighted" evaluation... you are only looking for immediate rewards!!!
- α close to 1 leads to "long-sighted" evaluation... you are willing to wait for a better reward!!!

Remarks

Interpretation

- The discounted future rewards can be interpreted as the **current value of future rewards**.

The reward α

- α close to 0 leads to "short-sighted" evaluation... you are only looking for immediate rewards!!!
- α close to 1 leads to "long-sighted" evaluation... you are willing to wait for a better reward!!!

Why to use this discount factor?

Basically

1. Discount factor avoids infinite returns in cyclic Markov process given the property of the geometry series.
2. There is certain uncertainty about the future reward.
3. If the reward is financial, immediate rewards may earn more interest than delayed rewards.
4. Animal/human behavior shows preference for immediate reward.

Why to use this discount factor?

Basically

1. Discount factor avoids infinite returns in cyclic Markov process given the property of the geometry series.
2. There is certain uncertainty about the future reward.
3. If the reward is financial, immediate rewards may earn more interest than delayed rewards.
4. Animal/human behavior shows preference for immediate reward.

Why to use this discount factor?

Basically

1. Discount factor avoids infinite returns in cyclic Markov process given the property of the geometry series.
2. There is certain uncertainty about the future reward.
3. If the reward is financial, immediate rewards may earn more interest than delayed rewards.
4. Animal/human behavior shows preference for immediate reward.

Why to use this discount factor?

Basically

1. Discount factor avoids infinite returns in cyclic Markov process given the property of the geometry series.
2. There is certain uncertainty about the future reward.
3. If the reward is financial, immediate rewards may earn more interest than delayed rewards.
4. Animal/human behavior shows preference for immediate reward.

A Policy

Using our MDP ideas...

- A policy π is a distribution over actions given states

$$\pi\left(a|s\right) = P\left[A_t = a|S_t = s\right]$$

  - A policy guides the choice of action at a given state.
  - And it is time independent

Then, we introduce two value functions

- State-value function.
- Action-value function.

A Policy

Using our MDP ideas...

- A policy π is a distribution over actions given states

$$\pi\left(a|s\right)=P\left[A_t=a|S_t=s\right]$$

  - A policy guides the choice of action at a given state.
  - And it is time independent

Then, we introduce two value functions

- State-value function.
- Action-value function.

The state-value function v_π of an MDP

It is defined as

$$v_\pi(s) = E_\pi\left[G_t | S_t = s\right]$$

The state-value function v_π of an MDP

It is defined as

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$

This gives the long-term value of the state s

$$\begin{aligned}
v_\pi(s) &= E_\pi[G_t | S_t = s] \\
&= E_\pi\left[\sum_{k=0}^{\infty} \alpha^k R_{t+k+1} | S_t = s\right] \\
&= E_\pi[R_{t+1} + \alpha G_{t+1} | S_t = s] \\
&= \underbrace{E_\pi[R_{t+1} | S_t = s]}_{\text{Immediate Reward}} + \underbrace{E_\pi[\alpha v_\pi(S_{t+1}) | S_t = s]}_{\text{Discount value of successor state}}
\end{aligned}$$

The state-value function v_π of an MDP

It is defined as

$$v_\pi(s) = E_\pi\left[G_t|S_t = s\right]$$

This gives the long-term value of the state s

$$\begin{aligned}
v_\pi(s) &= E_\pi\left[G_t|S_t = s\right] \\
&= E_\pi\left[\sum_{k=0}^{\infty} \alpha^k R_{t+k+1}|S_t = s\right] \\
&= E_\pi\left[R_{t+1} + \alpha G_{t+1}|S_t = s\right] \\
&= \underbrace{E_\pi\left[R_{t+1}|S_t = s\right]}_{\text{Immediate Reward}} + \underbrace{E_\pi\left[\alpha v_\pi(S_{t+1})|S_t = s\right]}_{\text{Discount value of successor state}}
\end{aligned}$$

The state-value function v_π of an MDP

It is defined as

$$v_\pi(s) = E_\pi\left[G_t | S_t = s\right]$$

This gives the long-term value of the state s

$$\begin{aligned} v_\pi(s) &= E_\pi\left[G_t | S_t = s\right] \\ &= E_\pi\left[\sum_{k=0}^{\infty} \alpha^k R_{t+k+1} | S_t = s\right] \\ &= E_\pi\left[R_{t+1} + \alpha G_{t+1} | S_t = s\right] \\ &= \underbrace{E_\pi\left[R_{t+1} | S_t = s\right]}_{\text{Immediate Reward}} + \underbrace{E_\pi\left[\alpha v_\pi(S_{t+1}) | S_t = s\right]}_{\text{Discount value of successor state}} \end{aligned}$$

The Action-Value Function $q_{\pi}(s,a)$

It is the expected return starting from state s

- Starting from state s, taking action a, and then following policy π

$$q_{\pi}(s,a) = E_{\pi}\left[G_t | S_t = a, A_t = a\right]$$

We can also obtain the following decomposition

$$q_{\pi}(s,a) = E_{\pi}\left[R_{t+1} + \alpha q_{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\right]$$

The Action-Value Function $q_\pi(s,a)$

It is the expected return starting from state s

- Starting from state s, taking action a, and then following policy π

$$q_\pi(s,a) = E_\pi[G_t | S_t = a, A_t = a]$$

We can also obtain the following decomposition

$$q_\pi(s,a) = E_\pi[R_{t+1} + \alpha q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

Now...

To simplify notations, we define

$$\mathcal{R}_s^a = E\left[R_{t+1} | S_t = s, A_t = a\right]$$

Then, we have

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

$$q_\pi(s,a) = \mathcal{R}_s^a + \alpha \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Now...

To simplify notations, we define

$$\mathcal{R}_s^a = E\left[R_{t+1} | S_t = s, A_t = a\right]$$

Then, we have

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

$$q_\pi(s,a) = \mathcal{R}_s^a + \alpha \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Then

Expressing $q_\pi(s,a)$ in terms of $v_\pi(s)$ in the expression of $v_\pi(s)$ - The Bellman Equation

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left[\mathcal{R}_s^a + \alpha \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')\right]$$

The Bellman equation relates the state-value function

- of one state with that of other states.

Then

Expressing $q_\pi(s,a)$ in terms of $v_\pi(s)$ in the expression of $v_\pi(s)$ - The Bellman Equation

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[\mathcal{R}_s^a + \alpha \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right]$$

The Bellman equation relates the state-value function

- of one state with that of other states.

Also

Bellman equation for $q_\pi(s,a)$

$$q_\pi(s,a) = \mathcal{R}_s^a + \alpha \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a \in \mathcal{A}} \pi(a|s)\, q_\pi(s,a)$$

How do we solve this problem?

- We have the following solutions

Also

Bellman equation for $q_\pi(s,a)$

$$q_\pi(s,a) = \mathcal{R}_s^a + \alpha \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a \in \mathcal{A}} \pi(a|s)\, q_\pi(s,a)$$

How do we solve this problem?

- We have the following solutions

Outline

Action-Value Methods

One natural way to estimate this is by averaging the rewards actually received

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i I_{A_i=a}}{\sum_{i=1}^{t-1} I_{A_i=a}}$$

Nevertheless

- If the denominator is zero, we define $Q_t(a)$ at some default value

Now, as the denominator goes to infinity, by the law of large numbers,

$$Q_t(a) \longrightarrow q^*(a)$$

Action-Value Methods

One natural way to estimate this is by averaging the rewards actually received

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i I_{A_i=a}}{\sum_{i=1}^{t-1} I_{A_i=a}}$$

Nevertheless

- If the denominator is zero, we define $Q_t(a)$ at some default value

Now, as the denominator goes to infinity, by the law of large numbers,

$$Q_t(a) \longrightarrow q^*(a)$$

Action-Value Methods

One natural way to estimate this is by averaging the rewards actually received

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i I_{A_i=a}}{\sum_{i=1}^{t-1} I_{A_i=a}}$$

Nevertheless

- If the denominator is zero, we define $Q_t(a)$ at some default value

Now, as the denominator goes to infinity, by the law of large numbers,

$$Q_t(a) \longrightarrow q^*(a)$$

Furthermore

We call this the sample-average method

- Not the best, but allows to introduce a way to select actions...

- It spends no time in inferior actions to see if they might really be better

Furthermore

We call this the sample-average method

- Not the best, but allows to introduce a way to select actions...

A classic one us the greedy estimate

$$A_t = \arg\max_a Q_t(a)$$

- It spends no time in inferior actions to see if they might really be better

Furthermore

We call this the sample-average method

- Not the best, but allows to introduce a way to select actions...

A classic one us the greedy estimate

$$A_t = \arg\max_a Q_t(a)$$

Therefore

- Greedy action selection always exploits current knowledge to maximize immediate reward.
- it spends no time in inferior actions to see if they might really be better.

Furthermore

We call this the sample-average method

- Not the best, but allows to introduce a way to select actions...

A classic one us the greedy estimate

$$A_t = \arg\max_a Q_t(a)$$

Therefore

- Greedy action selection always exploits current knowledge to maximize immediate reward.
- it spends no time in inferior actions to see if they might really be better.

And Here a Heuristic

A Simple Alternative is to act greedily most of the time

- every once in a while, say with small probability ϵ, select randomly from among all the actions with equal probability.

This is called ϵ-greedy method

- In the limit to infinity, this method ensures:

$$Q_t(a) \longrightarrow q^*(a)$$

And Here a Heuristic

A Simple Alternative is to act greedily most of the time

- every once in a while, say with small probability ϵ, select randomly from among all the actions with equal probability.

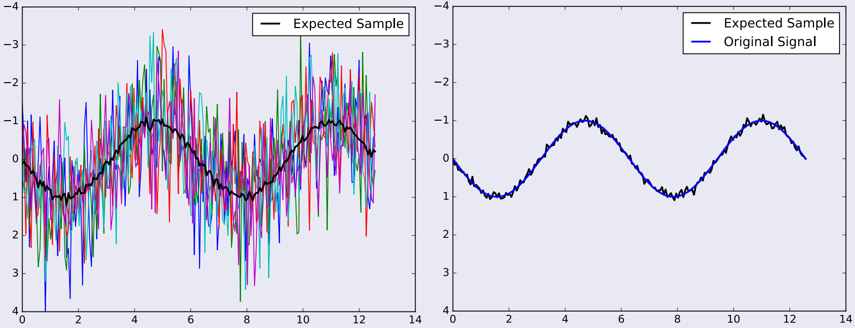
This is called ϵ-greedy method

- In the limit to infinity, this method ensures:

$$Q_t(a) \longrightarrow q^*(a)$$

It is like the Mean Filter

We have a nice result



Example

In the following example

- The $q^*(a)$ were selected according to a Gaussian Distribution with mean 0 and variance 1.
- Then, when an action is selected, the reward R_t was selected from a normal distribution with mean $q_*(A_t)$ and variance 1.

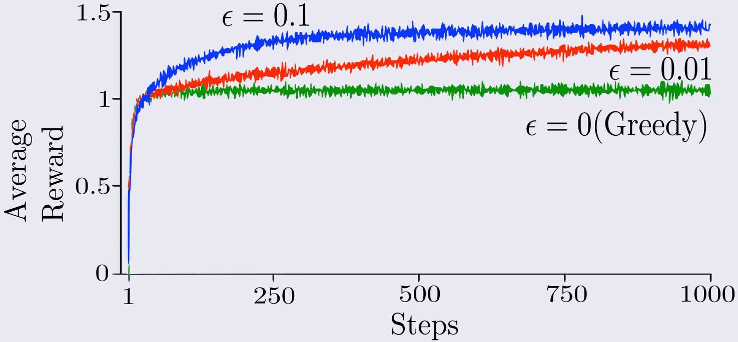
We have then

Example

In the following example

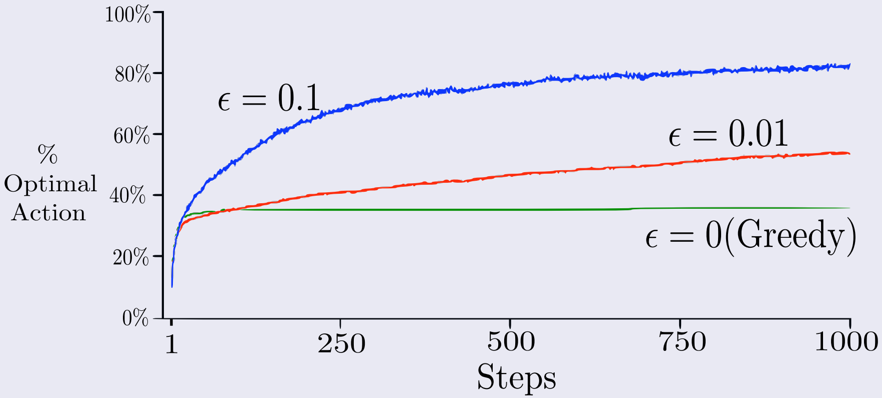
- The $q^*(a)$ were selected according to a Gaussian Distribution with mean 0 and variance 1.
- Then, when an action is selected, the reward R_t was selected from a normal distribution with mean $q_*(A_t)$ and variance 1.

We have then



Now, as the sampling goes up

We get



Outline

Imagine the following

Let R_i denotes the reward after the i^{th} of this action

$$Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

The problem of this technique

- The Amount of Memory and Computational Power to keep updates the estimates as more reward happens

Imagine the following

Let R_i denotes the reward after the i^{th} of this action

$$Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

The problem of this technique

- The Amount of Memory and Computational Power to keep updates the estimates as more reward happens

Therefore

We need something more efficient

- We can do something like that by realizing the following:

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_n \\
&= \frac{1}{n}\left[R_n + \sum_{i=1}^{n} E_i\right] \\
&= \frac{1}{n}\left[R_n + (n-1)\,Q_n\right] \\
&= Q_n + \frac{1}{n}\left[R_n - Q_n\right]
\end{aligned}
$$

Therefore

We need something more efficient

- We can do something like that by realizing the following:

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_n \\
&= \frac{1}{n}\left[R_n + \sum_{i=1}^{n} E_i\right] \\
&= \frac{1}{n}\left[R_n + (n-1)\,Q_n\right] \\
&= Q_n + \frac{1}{n}\left[R_n - Q_n\right]
\end{aligned}
$$

Therefore

We need something more efficient

- We can do something like that by realizing the following:

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_n \\
&= \frac{1}{n}\left[R_n + \sum_{i=1}^{n} E_i\right] \\
&= \frac{1}{n}\left[R_n + (n-1)\, Q_n\right] \\
&= Q_n + \frac{1}{n}\left[R_n - Q_n\right]
\end{aligned}
$$

Therefore

We need something more efficient

- We can do something like that by realizing the following:

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_n \\
&= \frac{1}{n}\left[R_n + \sum_{i=1}^{n} E_i\right] \\
&= \frac{1}{n}\left[R_n + (n-1)\,Q_n\right] \\
&= Q_n + \frac{1}{n}\left[R_n - Q_n\right]
\end{aligned}
$$

Outline

General Formula in Reinforcement Learning

The Pattern Appears almost all the time in RL

$$\text{New Estimate} = \text{Old Estimate} + \text{Step Size}\left[\text{Target} - \text{Old Estimate}\right]$$

Looks a lot as a Gradient Descent if we decide Assume a Taylor Series

$$f(x) = f(a) + f'(a)(x-a) \Longrightarrow f'(a) = \frac{f(x) - f(a)}{x-a}$$

Then

$$x_{n+1} = x_n + \gamma f'(a) = x_n + \gamma\left[\frac{f(x) - f(a)}{x-a}\right] = x_n + \gamma'\left[f(x) - f(a)\right]$$

General Formula in Reinforcement Learning

The Pattern Appears almost all the time in RL

$$\text{New Estimate} = \text{Old Estimate} + \text{Step Size}\left[\text{Target} - \text{Old Estimate}\right]$$

Looks a lot as a Gradient Descent if we decide Assume a Taylor Series

$$f(x) = f(a) + f'(a)(x-a) \Longrightarrow f'(a) = \frac{f(x) - f(a)}{x-a}$$

Then

$$x_{n+1} = x_n + \gamma f'(a) = x_n + \gamma\left[\frac{f(x) - f(a)}{x-a}\right] = x_n + \gamma'\left[f(x) - f(a)\right]$$

General Formula in Reinforcement Learning

The Pattern Appears almost all the time in RL

$$\text{New Estimate} = \text{Old Estimate} + \text{Step Size}\left[\text{Target} - \text{Old Estimate}\right]$$

Looks a lot as a Gradient Descent if we decide Assume a Taylor Series

$$f(x) = f(a) + f'(a)(x-a) \Longrightarrow f'(a) = \frac{f(x) - f(a)}{x - a}$$

Then

$$x_{n+1} = x_n + \gamma f'(a) = x_n + \gamma\left[\frac{f(x) - f(a)}{x - a}\right] = x_n + \gamma'\left[f(x) - f(a)\right]$$

Outline

A Simple Bandit Algorithm

Initialize for $a = 1$ to k

1. $Q(a) \leftarrow 0$
2. $N(a) \leftarrow 0$

Loop until certain threshold γ

1. $A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\epsilon \\ \text{A random action} & \text{with probability } \epsilon \end{cases}$
2. $R \leftarrow bandit(A)$
3. $N(A) \leftarrow N(A) + 1$
4. $Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$

A Simple Bandit Algorithm

Initialize for $a = 1$ to k

1. $Q(a) \leftarrow 0$
2. $N(a) \leftarrow 0$

Loop until certain threshold γ

1. $A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\epsilon \\ \text{A random action} & \text{with probability } \epsilon \end{cases}$
2. $R \leftarrow bandit(A)$
3. $N(A) \leftarrow N(A) + 1$
4. $Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Remarks

Remember

- This is works well for a stationary problem

For more in non-stationary problems

- "Reinforcement Learning: An Introduction" by Sutton et al. at chapter 2.

Remarks

Remember

- This is works well for a stationary problem

For more in non-stationary problems

- "Reinforcement Learning: An Introduction" by Sutton et al. at chapter 2.

Outline

Dynamic Programming

As in many things in Computer Science

- We love the Divide Et Impera...

Then

- In Reinforcement Learning, Bellman equation gives recursive decompositions, and value function stores and reuses solutions.

Dynamic Programming

As in many things in Computer Science

- We love the Divide Et Impera...

Then

- In Reinforcement Learning, Bellman equation gives recursive decompositions, and value function stores and reuses solutions.

Outline

Policy Evaluation

For this, we assume as always

- The problem can be solved with the Bellman Equation

We start from an initial guess v_1

- Then, we apply Bellman Iteratively

$$v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_\pi$$

For this, we use synchronous updates

- We update the value functions of the present iteration at the same time based on the that of the previous iteration.

Policy Evaluation

For this, we assume as always

- The problem can be solved with the Bellman Equation

We start from an initial guess v_1

- Then, we apply Bellman Iteratively

$$v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_\pi$$

For this, we use synchronous updates

- We update the value functions of the present iteration at the same time based on the that of the previous iteration.

Policy Evaluation

For this, we assume as always

- The problem can be solved with the Bellman Equation

We start from an initial guess v_1

- Then, we apply Bellman Iteratively

$$v_1 \to v_2 \to \cdots \to v_\pi$$

For this, we use synchronous updates

- We update the value functions of the present iteration at the same time based on the that of the previous iteration.

Then

At each iteration $k+1$, for all states $s \in \mathcal{S}$

- We update v_{k+1} from $v_k(s')$ according to Bellman Equations, where s' is a successor state of s:

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[\mathcal{R}_s^a + \alpha \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right]$$

Something Notable

- The iterative process stops when the maximum difference between value function at the current step and that at the previous step is smaller than some small positive constant.

Then

At each iteration $k+1$, for all states $s \in \mathcal{S}$

- We update v_{k+1} from $v_k(s')$ according to Bellman Equations, where s' is a successor state of s:

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[\mathcal{R}_s^a + \alpha \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right]$$

Something Notable

- The iterative process stops when the maximum difference between value function at the current step and that at the previous step is smaller than some small positive constant.

Outline

However

Our ultimate goal is to find the optimal policy

- For this, we can use policy iteration process

Algorithm

1. Initialize π randomly
2. Repeat until the previous policy and the current policy are the same.
   1. Evaluate v_π by policy evaluation.
   2. Using synchronous updates, for each state s, let

$$\pi(s) = \arg\max_{a \in \mathcal{A}} q(s, a) = \arg\max_{a \in \mathcal{A}} \left[\mathcal{R}_s^a + \alpha \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right]$$

However

Our ultimate goal is to find the optimal policy

- For this, we can use policy iteration process

Algorithm

1. Initialize π randomly
2. Repeat until the previous policy and the current policy are the same.
    1. Evaluate v_π by policy evaluation.
    2. Using synchronous updates, for each state s, let

$$\pi(s) = \arg\max_{a \in \mathcal{A}} q(s, a) = \arg\max_{a \in \mathcal{A}} \left[\mathcal{R}_s^a + \alpha \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right]$$

Now

Something Notable

- This policy iteration algorithm will improve the policy for each step.

Assume that

- We have a deterministic policy π and after one step we get π'.

We know that the value improves in one step because

$$q_{\pi}\left(s, \pi'\left(s\right)\right) = \max_{a \in \mathcal{A}} q_{\pi}\left(s, a\right) \geq q_{\pi}\left(s, \pi\left(s\right)\right) = v_{\pi}\left(s\right)$$

Now

Something Notable

- This policy iteration algorithm will improve the policy for each step.

Assume that

- We have a deterministic policy π and after one step we get π'.

We know that the value improves in one step because

$$q_{\pi}\left(s,\pi'\left(s\right)\right)=\max_{a\in\mathcal{A}} q_{\pi}\left(s,a\right)\geq q_{\pi}\left(s,\pi\left(s\right)\right)=v_{\pi}\left(s\right)$$

Now

Something Notable

- This policy iteration algorithm will improve the policy for each step.

Assume that

- We have a deterministic policy π and after one step we get π'.

We know that the value improves in one step because

$$q_{\pi}\left(s, \pi'\left(s\right)\right) = \max_{a \in \mathcal{A}} q_{\pi}\left(s, a\right) \geq q_{\pi}\left(s, \pi\left(s\right)\right) = v_{\pi}\left(s\right)$$

Then

We can see that

$$
\begin{aligned}
v_\pi(s) \leq q_\pi(s, \pi'(s)) &= E_{\pi'}\left[R_{t+1} + \alpha v_\pi(S_{t+1}) | S_t = s\right] \\
&\leq E_{\pi'}\left[R_{t+1} + \alpha q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s\right] \\
&\leq E_{\pi'}\left[R_{t+1} + \alpha R_{t+2} + \alpha^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) | S_t = s\right] \\
&\leq E_{\pi'}\left[R_{t+1} + \alpha R_{t+2} + \cdots | S_t = s\right] = v_{\pi'}(s)
\end{aligned}
$$

Then

We can see that

$$
\begin{aligned}
v_{\pi}(s) &\leq q_{\pi}(s, \pi'(s)) = E_{\pi'}\left[R_{t+1} + \alpha v_{\pi}(S_{t+1}) \mid S_t = s\right] \\
&\leq E_{\pi'}\left[R_{t+1} + \alpha q_{\pi}(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\
&\leq E_{\pi'}\left[R_{t+1} + \alpha R_{t+2} + \alpha^2 q_{\pi}(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right] \\
&\leq E_{\pi'}\left[R_{t+1} + \alpha R_{t+2} + \cdots \mid S_t = s\right] = v_{\pi'}(s)
\end{aligned}
$$

Then

We can see that

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi\left(s, \pi'(s)\right) = E_{\pi'}\left[R_{t+1} + \alpha v_\pi\left(S_{t+1}\right) | S_t = s\right] \\
&\leq E_{\pi'}\left[R_{t+1} + \alpha q_\pi\left(S_{t+1}, \pi'\left(S_{t+1}\right)\right) | S_t = s\right] \\
&\leq E_{\pi'}\left[R_{t+1} + \alpha R_{t+2} + \alpha^2 q_\pi\left(S_{t+2}, \pi'\left(S_{t+2}\right)\right) | S_t = s\right] \\
&\leq E_{\pi'}\left[R_{t+1} + \alpha R_{t+2} + \cdots | S_t = s\right] = v_{\pi'}(s)
\end{aligned}
$$

Then

We can see that

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi\left(s, \pi'(s)\right) = E_{\pi'}\left[R_{t+1} + \alpha v_\pi\left(S_{t+1}\right) | S_t = s\right] \\
&\leq E_{\pi'}\left[R_{t+1} + \alpha q_\pi\left(S_{t+1}, \pi'\left(S_{t+1}\right)\right) | S_t = s\right] \\
&\leq E_{\pi'}\left[R_{t+1} + \alpha R_{t+2} + \alpha^2 q_\pi\left(S_{t+2}, \pi'\left(S_{t+2}\right)\right) | S_t = s\right] \\
&\leq E_{\pi'}\left[R_{t+1} + \alpha R_{t+2} + \cdots | S_t = s\right] = v_{\pi'}(s)
\end{aligned}
$$

Then

If improvements stop

$$q_\pi\left(s, \pi'\left(s\right)\right) = \max_{a \in \mathcal{A}} q_\pi\left(s, a\right) = q_\pi\left(s, \pi\left(s\right)\right) = v_\pi\left(s\right)$$

This means that the Bellman equation has been satisfied

$$v_\pi\left(s\right) = \max_{a \in \mathcal{A}} q_\pi\left(s, a\right)$$

- Therefore, $v_\pi\left(s\right) = v_*\left(s\right)$ for all $s \in \mathcal{S}$ and π is an optimal policy.

Then

If improvements stop

$$q_\pi\left(s, \pi'\left(s\right)\right) = \max_{a \in \mathcal{A}} q_\pi\left(s, a\right) = q_\pi\left(s, \pi\left(s\right)\right) = v_\pi\left(s\right)$$

This means that the Bellman equation has been satisfied

$$v_\pi\left(s\right) = \max_{a \in \mathcal{A}} q_\pi\left(s, a\right)$$

- Therefore, $v_\pi\left(s\right) = v_*\left(s\right)$ for all $s \in \mathcal{S}$ and π is an optimal policy.

Outline

Once the policy has been improved

We can iteratively a sequence of monotonically improving policies and values

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

- with $E = evaluation$ and $I = improvement$.

Policy Iteration (using iterative policy evaluation)

Step 1. Initialization

- $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

Step 2. Policy Evaluation

1. Loop:
2. $\quad \Delta \leftarrow 0$
3. $\quad$ Loop for each $s \in \mathcal{S}$
4. $\quad\quad v \leftarrow V(s)$
5. $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s)) [r + \alpha V(s')]$
6. $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
7. until $\Delta < \theta$ (A small threshold for accuracy)

Policy Iteration (using iterative policy evaluation)

Step 1. Initialization

- $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

Step 2. Policy Evaluation

1. Loop:
2. $\quad \Delta \leftarrow 0$
3. $\quad$ Loop for each $s \in \mathcal{S}$
4. $\quad\quad v \leftarrow V(s)$
5. $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s)) [r + \alpha V(s')]$
6. $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
7. until $\Delta < \theta$ (A small threshold for accuracy)

Further

Step 3. Policy Improvement

1. Policy-stable $\leftarrow$ True
2. For each $s \in S$:
3. old_action $\leftarrow \pi(s)$
4. $\pi(s) \leftarrow \sum_{s',r} p(s'.r|s,a)\left[r + \alpha V(s')\right]$
5. if old_action$\neq \pi(s)$ then policy-stable $\leftarrow$ False
6. If policy-stable then stop and return $V \approx v_*$ and $\pi \approx \pi_*$ else go to 2.

Outline

Jack's Car Rental

Setup

- We have a Car's Rental with two places (s_1, s_2).
- And each time Jack is available he rents it out and is credited \$10 by the company.
- If Jack's is out of cars at that location, then the business is lost.
- The cars are available for renting the next day

Jack's Car Rental

Setup

- We have a Car's Rental with two places (s_1, s_2).
- And each time Jack is available he rents it out and is credited \$10 by the company.
- If Jack's is out of cars at that location, then the business is lost.
- The cars are available for renting the next day

Jack's has options

- It can move them between the two locations overnight, at a cost of \$2 dollars and only 5 of them can be moved.

Jack's Car Rental

Setup

- We have a Car's Rental with two places (s_1, s_2).
- And each time Jack is available he rents it out and is credited \$10 by the company.
- If Jack's is out of cars at that location, then the business is lost.
- The cars are available for renting the next day

Jack's has options

- It can move them between the two locations overnight, at a cost of \$2 dollars and only 5 of them can be moved.

Jack's Car Rental

Setup

- We have a Car's Rental with two places (s_1, s_2).
- And each time Jack is available he rents it out and is credited \$10 by the company.
- If Jack's is out of cars at that location, then the business is lost.
- The cars are available for renting the next day

Jack's has options

- It can move them between the two locations overnight, at a cost of \$2 dollars and only 5 of them can be moved.

Jack's Car Rental

Setup

- We have a Car's Rental with two places (s_1, s_2).
- And each time Jack is available he rents it out and is credited \$10 by the company.
- If Jack's is out of cars at that location, then the business is lost.
- The cars are available for renting the next day

Jack's has options

- It can move them between the two locations overnight, at a cost of \$2 dollars and only 5 of them can be moved.

We choose a probability for modeling requests and returns

Additionally, renting at each location has a Poisson distribution

$$P\left(n|\lambda\right)=\frac{\lambda^{n}}{n!}e^{-\lambda}$$

We simplify the problem by having an upper-bound $n = 20$

- Basically, any other car generated by the return simply disappear!!!

We choose a probability for modeling requests and returns

Additionally, renting at each location has a Poisson distribution

$$P\left(n|\lambda\right) = \frac{\lambda^n}{n!}e^{-\lambda}$$

We simplify the problem by having an upper-bound $n = 20$

- Basically, any other car generated by the return simply disappear!!!

Finally

We have the following $\lambda' s$ for request and returns

1. Request s_1 we have $\lambda_{r_1} = 3$
2. Request s_2 we have $\lambda_{r_2} = 4$
3. Return s_1 we have $\lambda_{r_1} = 3$
4. Return s_2 we have $\lambda_{r_1} = 2$

Finally

We have the following $\lambda's$ for request and returns

1. Request s_1 we have $\lambda_{r_1} = 3$
2. Request s_2 we have $\lambda_{r_2} = 4$
3. Return s_1 we have $\lambda_{r_1} = 3$
4. Return s_2 we have $\lambda_{r_1} = 2$

Actually, it defines how many cars are requested or returned in a time interval (One day)

Finally

We have the following $\lambda's$ for request and returns

1. Request s_1 we have $\lambda_{r_1} = 3$
2. Request s_2 we have $\lambda_{r_2} = 4$
3. Return s_1 we have $\lambda_{r_1} = 3$
4. Return s_2 we have $\lambda_{r_1} = 2$

Actually, it defines how many cars are requested or returned in a time interval (One day)

Finally

We have the following $\lambda's$ for request and returns

1. Request s_1 we have $\lambda_{r_1} = 3$
2. Request s_2 we have $\lambda_{r_2} = 4$
3. Return s_1 we have $\lambda_{r_1} = 3$
4. Return s_2 we have $\lambda_{r_1} = 2$

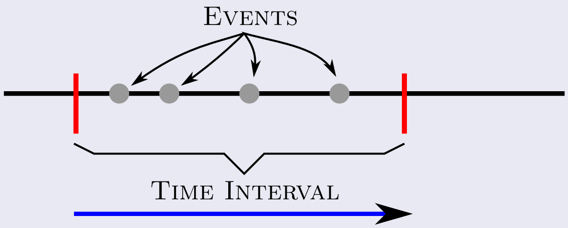
Actually, it defines how many cars are requested or returned in a time interval (One day)

Finally

We have the following $\lambda's$ for request and returns

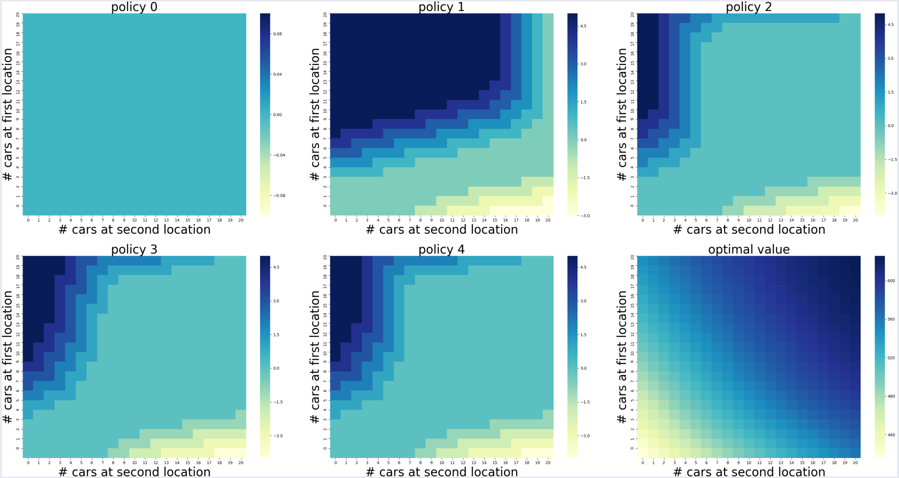
1. Request s_1 we have $\lambda_{r_1} = 3$
2. Request s_2 we have $\lambda_{r_2} = 4$
3. Return s_1 we have $\lambda_{r_1} = 3$
4. Return s_2 we have $\lambda_{r_1} = 2$

Actually, it defines how many cars are requested or returned in a time interval (One day)



Thus, we have the following running

With a $\alpha = 0.9$



Outline

History

There is an original earlier work

- By Arthur Samuel and it was used by Sutton to invent Temporal Difference-Lambda

It was famously applied by Gerald Tesauro

- To build TD-Gammon
  - A program that learned to play backgammon at the level of expert human players

History

There is an original earlier work

- By Arthur Samuel and it was used by Sutton to invent Temporal Difference-Lambda

It was famously applied by Gerald Tesauro

- To build TD-Gammon
  - A program that learned to play backgammon at the level of expert human players

Temporal-Difference (TD) Learning

Something Notable

- Possibly the central and novel idea of reinforcement learning,

It has similarity with Monte Carlo Methods

- They use experience to solve the prediction problem (For More look at Sutton's Book)

Monte Carlo methods wait until G_t is known

$$V(S_t) \leftarrow V(S_t) + \gamma \left[G_t - V(S_t) \right] \text{ and } G_t = \sum_{k=0}^{\infty} \alpha^k R_{t+k+1}$$

- Let us call this method constant-α Monte Carlo.

Temporal-Difference (TD) Learning

Something Notable

- Possibly the central and novel idea of reinforcement learning,

It has similarity with Monte Carlo Methods

- They use experience to solve the prediction problem (For More look at Sutton's Book)

Monte Carlo methods wait until G_t is known

$$V(S_t) \leftarrow V(S_t) + \gamma \left[G_t - V(S_t)\right] \text{ and } G_t = \sum_{k=0}^{\infty} \alpha^k R_{t+k+1}$$

- Let us call this method constant-α Monte Carlo.

Temporal-Difference (TD) Learning

Something Notable

- Possibly the central and novel idea of reinforcement learning,

It has similarity with Monte Carlo Methods

- They use experience to solve the prediction problem (For More look at Sutton's Book)

Monte Carlo methods wait until G_t is known

$$V(S_t) \leftarrow V(S_t) + \gamma \left[G_t - V(S_t) \right] \text{ and } G_t = \sum_{k=0}^{\infty} \alpha^k R_{t+k+1}$$

- Let us call this method constant-α Monte Carlo.

Outline

Fundamental Theorem of Simulation

Cumulative Distribution Function

- With every random variable, we associate a function called Cumulative Distribution Function (CDF) which is defined as follows:

$$F_X(x) = P(f(X) \leq x)$$

- With properties:
  - $F_X(x) \geq 0$
  - $F_X(x)$ in a non-decreasing function of X.

Fundamental Theorem of Simulation

Cumulative Distribution Function

- With every random variable, we associate a function called Cumulative Distribution Function (CDF) which is defined as follows:

$$F_X(x) = P(f(X) \leq x)$$

- With properties:
  - $F_X(x) \geq 0$
  - $F_X(x)$ in a non-decreasing function of X.

Fundamental Theorem of Simulation

Cumulative Distribution Function

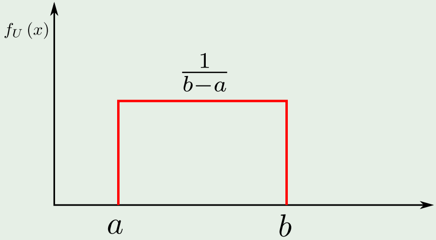
- With every random variable, we associate a function called Cumulative Distribution Function (CDF) which is defined as follows:

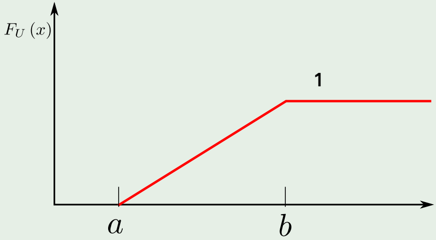
$$F_X(x) = P(f(X) \leq x)$$

- With properties:
  - $F_X(x) \geq 0$
  - $F_X(x)$ in a non-decreasing function of X.

Graphically

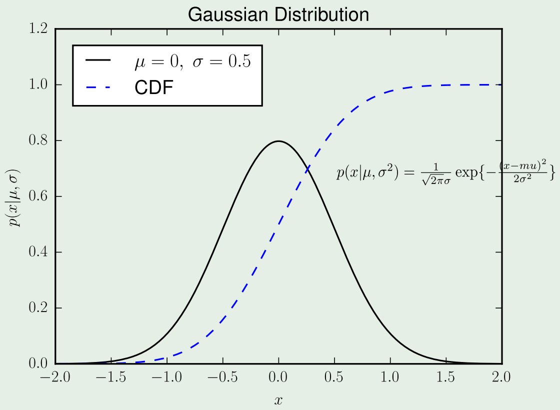
Example Uniform Distribution





Graphically

Example Gaussian Distribution



Outline

Fundamental Theorem of Simulation

Theorem

Simulating

$$X \sim f(x)$$

It is equivalent to simulating

$$(X, U) \sim U\left((x, u) \,|\, 0 < u < f(x)\right)$$

Fundamental Theorem of Simulation

Theorem

Simulating

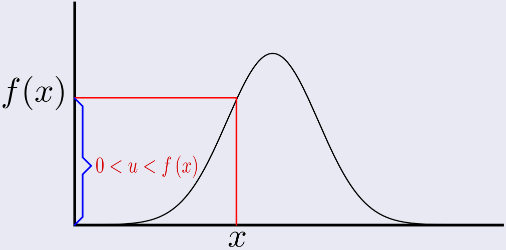
$$X \sim f(x)$$

It is equivalent to simulating

$$(X, U) \sim U\left((x, u) \,|\, 0 < u < f(x)\right)$$

Geometrically

We have

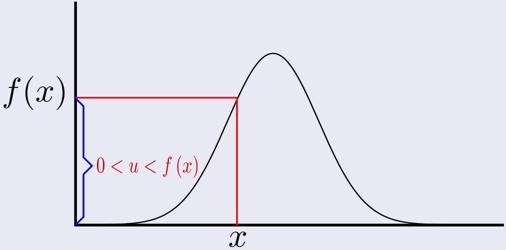


Proof

We have that $f(x) = \int_0^{f(x)} du$ the marginal pdf of (X, U).

Geometrically

We have



Proof

We have that $f(x) = \int_0^{f(x)} du$ the marginal pdf of (X, U).

Therefore

One thing that is made clear by this theorem is that we can generate X in three ways

1. First generate $X \sim f$, and then $U|X = x \sim U\left\{u|u \leq f(x)\right\}$, but this is useless because we already have X and do not need X.
2. First generate U, and then $X|U = u \sim U\left\{x|u \leq f(x)\right\}$.
3. Generate (X, U) jointly, a smart idea.
   - it allows us to generate data on a larger set were simulation is easier.
   - Not as simple as it looks.

Therefore

One thing that is made clear by this theorem is that we can generate X in three ways

1. First generate $X \sim f$, and then $U|X = x \sim U\left\{u|u \leq f(x)\right\}$, but this is useless because we already have X and do not need X.
2. First generate U, and then $X|U = u \sim U\left\{x|u \leq f(x)\right\}$.
3. Generate (X, U) jointly, a smart idea.
   - it allows us to generate data on a larger set were simulation is easier.
   - Not as simple as it looks.

Therefore

One thing that is made clear by this theorem is that we can generate X in three ways

1. First generate $X \sim f$, and then $U|X = x \sim U\left\{u|u \leq f(x)\right\}$, but this is useless because we already have X and do not need X.
2. First generate U, and then $X|U = u \sim U\left\{x|u \leq f(x)\right\}$.
3. Generate (X, U) jointly, a smart idea.
    - it allows us to generate data on a larger set were simulation is easier.
    - Not as simple as it looks.

Therefore

One thing that is made clear by this theorem is that we can generate X in three ways

1. First generate $X \sim f$, and then $U|X = x \sim U\left\{u|u \leq f(x)\right\}$, but this is useless because we already have X and do not need X.
2. First generate U, and then $X|U = u \sim U\left\{x|u \leq f(x)\right\}$.
3. Generate (X, U) jointly, a smart idea.
   - it allows us to generate data on a larger set were simulation is easier.
   - Not as simple as it looks.

Markov Chains

The random process $X_t \in S$ for $t = 1, 2, ..., T$ has a Markov property iif

$$p\left(X_T | X_{T-1}, X_{T-2}, ..., X_1\right) = p\left(X_T | X_{T-1}\right)$$

Finite-state Discrete Time Markov Chains $|S| < \infty$

- It can be completly specified by the transition matrix.
  - $P = [p_{ij}]$ with $p_{ij} = P[X_t = j | X_{t-1} = i]$
- For Irreducible chains, the stationary distribution π is long-term proportion of time that the chain spends in each state.
  - Computed by $\pi = \pi P$

Markov Chains

The random process $X_t \in S$ for $t = 1, 2, ..., T$ has a Markov property iif

$$p\left(X_T | X_{T-1}, X_{T-2}, ..., X_1\right) = p\left(X_T | X_{T-1}\right)$$

Finite-state Discrete Time Markov Chains $|S| < \infty$

- It can be completly specified by the transition matrix.
  - $P = [p_{ij}]$ with $p_{ij} = \mathbb{P}\left[X_t = j | X_{t-1} = i\right]$
- For Irreducible chains, the stationary distribution π is long-term proportion of time that the chain spends in each state.
  - Computed by $\pi = \pi P$.

Markov Chains

The random process $X_t \in S$ for $t = 1, 2, ..., T$ has a Markov property iif

$$p\left(X_T | X_{T-1}, X_{T-2}, ..., X_1\right) = p\left(X_T | X_{T-1}\right)$$

Finite-state Discrete Time Markov Chains $|S| < \infty$

- It can be completly specified by the transition matrix.
  - $P = [p_{ij}]$ with $p_{ij} = \mathbb{P}\left[X_t = j | X_{t-1} = i\right]$
- For Irreducible chains, the stationary distribution π is long-term proportion of time that the chain spends in each state.
  - Computed by $\pi = \pi P$.

Markov Chains

The random process $X_t \in S$ for $t = 1, 2, ..., T$ has a Markov property iif

$$p\left(X_T | X_{T-1}, X_{T-2}, ..., X_1\right) = p\left(X_T | X_{T-1}\right)$$

Finite-state Discrete Time Markov Chains $|S| < \infty$

- It can be completly specified by the transition matrix.
  - $P = [p_{ij}]$ with $p_{ij} = \mathbb{P}\left[X_t = j | X_{t-1} = i\right]$
- For Irreducible chains, the stationary distribution π is long-term proportion of time that the chain spends in each state.
  - Computed by $\pi = \pi P$.

Outline

The Idea

Draw an i.i.d. set of samples $\left\{ \boldsymbol{x}^{(i)} \right\}_{i=1}^{N}$

From a target density $p\left(\boldsymbol{x}\right)$ on a high-dimensional $\mathcal{X}$.

- The set of possible configurations of a system.

The Idea

Draw an i.i.d. set of samples $\left\{\boldsymbol{x}^{(i)}\right\}_{i=1}^{N}$

From a target density $p(\boldsymbol{x})$ on a high-dimensional $\mathcal{X}$.

- The set of possible configurations of a system.

Thus, we can use those samples to approximate the target density

$$p_N(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\boldsymbol{x}^{(i)}}(\boldsymbol{x})$$

where $\delta_{\boldsymbol{x}^{(i)}}(\boldsymbol{x})$ denotes the Dirac Delta mass located at $\boldsymbol{x}^{(i)}$.

The Idea

Draw an i.i.d. set of samples $\left\{\boldsymbol{x}^{(i)}\right\}_{i=1}^{N}$

From a target density $p(\boldsymbol{x})$ on a high-dimensional $\mathcal{X}$.

- The set of possible configurations of a system.

Thus, we can use those samples to approximate the target density

$$p_N(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\boldsymbol{x}^{(i)}}(\boldsymbol{x})$$

where $\delta_{\boldsymbol{x}^{(i)}}(\boldsymbol{x})$ denotes the Dirac Delta mass located at $\boldsymbol{x}^{(i)}$.

Thus

Given, the Dirac Delta

$$\delta\left(t\right)=\begin{cases}0 & t\neq 0\\ \infty & t=0\end{cases}$$

with

$$\int_{t_1}^{t_2}\delta(t)dt=1$$

Therefore

$$\delta\left(t\right)=\lim_{\sigma\to 0}\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{t^2}{2\sigma^2}}$$

Thus

Given, the Dirac Delta

$$\delta\left(t\right) = \begin{cases} 0 & t \neq 0 \\ \infty & t = 0 \end{cases}$$

with

$$\int_{t_1}^{t_2} \delta(t)dt = 1$$

Therefore

$$\delta\left(t\right) = \lim_{\sigma \to 0} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{t^2}{2\sigma^2}}$$

Consequently

Approximate the integrals

$$I_N(f) = \frac{1}{N}\sum_{i=1}^{N} f\left(\boldsymbol{x}^{(i)}\right) \xrightarrow[N\to\infty]{a.s.} I(f) = \int_{\mathcal{X}} f(\boldsymbol{x})\, p(\boldsymbol{x})\, d\boldsymbol{x}$$

Where Variance of $f(\boldsymbol{x})$

$$\sigma_f^2 = \mathbb{E}_{p(\boldsymbol{x})}\left[f^2(\boldsymbol{x})\right] - I^2(f) < \infty$$

Therefore the variance of the estimator $I_N(f)$

$$E_{I_N(f)} = var\left(I_N(f)\right) = \frac{\sigma_f^2}{N}$$

Consequently

Approximate the integrals

$$I_N(f) = \frac{1}{N}\sum_{i=1}^{N} f\left(\boldsymbol{x}^{(i)}\right) \xrightarrow[N\to\infty]{a.s.} I(f) = \int_{\mathcal{X}} f(\boldsymbol{x})\, p(\boldsymbol{x})\, d\boldsymbol{x}$$

Where Variance of $f(\boldsymbol{x})$

$$\sigma_f^2 = \mathbb{E}_{p(\boldsymbol{x})}\left[f^2(\boldsymbol{x})\right] - I^2(f) < \infty$$

Therefore the variance of the estimator $I_N(f)$

$$E_{I_N(f)} = var\left(I_N(f)\right) = \frac{\sigma_f^2}{N}$$

Consequently

Approximate the integrals

$$I_N(f) = \frac{1}{N}\sum_{i=1}^{N} f\left(\boldsymbol{x}^{(i)}\right) \xrightarrow[N\to\infty]{a.s.} I(f) = \int_{\mathcal{X}} f(\boldsymbol{x})\, p(\boldsymbol{x})\, d\boldsymbol{x}$$

Where Variance of $f(\boldsymbol{x})$

$$\sigma_f^2 = \mathbb{E}_{p(\boldsymbol{x})}\left[f^2(\boldsymbol{x})\right] - I^2(f) < \infty$$

Therefore the variance of the estimator $I_N(f)$

$$E_{I_N(f)} = var\left(I_N(f)\right) = \frac{\sigma_f^2}{N}$$

Then

By Robert & Casella 1999, Section 3.2

$$\sqrt{N}\left(I_N\left(f\right)-I\left(f\right)\right)\underset{N\longrightarrow\infty}{\Longrightarrow}\mathcal{N}\left(0,\sigma_f^2\right)$$

Then

By Robert & Casella 1999, Section 3.2

$$\sqrt{N}\left(I_N\left(f\right)-I\left(f\right)\right)\underset{N\longrightarrow\infty}{\Longrightarrow}\mathcal{N}\left(0,\sigma_f^2\right)$$

Outline

What if we do not wait for G_t

TD methods need to wait only until the next time step

- At time $t+1$, it uses the observed reward R_{t+1} and the estimate $V(S_{t+1})$

The simplest TD method

$$V(S_t) \leftarrow V(S_t) + \gamma\left[R_{t+1} + \alpha V(S_{t+1}) - V(S_t)\right]$$

- Immediately on transition to S_{t+1} and receiving R_{t+1}.

What if we do not wait for G_t

TD methods need to wait only until the next time step

- At time $t+1$, it uses the observed reward R_{t+1} and the estimate $V(S_{t+1})$

The simplest TD method

$$V(S_t) \leftarrow V(S_t) + \gamma\left[R_{t+1} + \alpha V(S_{t+1}) - V(S_t)\right]$$

- Immediately on transition to S_{t+1} and receiving R_{t+1} .

This method is called TD(0) or one-step TD

Actually

- There are TD with n-step TD methods

Differences with the Monte Carlo Methods

- In Monte Carlo update, the target is G_t
- In TD update, the target is $R_t + \alpha V(S_{t+1})$

This method is called TD(0) or one-step TD

Actually

- There are TD with n-step TD methods

Differences with the Monte Carlo Methods

- In Monte Carlo update, the target is G_t
- In TD update, the target is $R_t + \alpha V\left(S_{t+1}\right)$

TD as a Bootstrapping Method

Quite similar to Dynamic

- Additionally, we know that

$$\begin{aligned} v_\pi(s) &= E_\pi\left[G_t|S_t\right] \\ &= E_\pi\left[R_{t+1} + \alpha G_{t+1}|S_t\right] \\ &= E_\pi\left[R_{t+1} + \alpha v_\pi(S_{t+1})\,|S_t\right] \end{aligned}$$

Monte Carlo methods use an estimate of $v_\pi(s) = E_\pi\left[G_t|S_t\right]$

- Instead, Dynamic Programming methods use an estimate of $v_\pi(s) = E_\pi\left[R_{t+1} + \alpha v_\pi(S_{t+1})\,|S_t\right]$.

TD as a Bootstrapping Method

Quite similar to Dynamic

- Additionally, we know that

$$
\begin{aligned}
v_\pi(s) &= E_\pi\left[G_t|S_t\right] \\
&= E_\pi\left[R_{t+1} + \alpha G_{t+1}|S_t\right] \\
&= E_\pi\left[R_{t+1} + \alpha v_\pi\left(S_{t+1}\right)|S_t\right]
\end{aligned}
$$

Monte Carlo methods use an estimate of $v_\pi(s) = E_\pi\left[G_t|S_t\right]$

- Instead, Dynamic Programming methods use an estimate of $v_\pi(s) = E_\pi\left[R_{t+1} + \alpha v_\pi\left(S_{t+1}\right)|S_t\right]$.

Furthermore

The Monte Carlo target is an estimate

- because the expected value in (6.3) is not known, but a sample is used instead

The DP target is an estimate

- because $v_\pi(St+1)$ is not known and the current estimate, $V(S_{t+1})$, is used instead.

Here TD goes beyond and combines both ideas

- It combines the sampling of Monte Carlo with the bootstrapping of DP.

Furthermore

The Monte Carlo target is an estimate

- because the expected value in (6.3) is not known, but a sample is used instead

The DP target is an estimate

- because $v_\pi(St+1)$ is not known and the current estimate,$V(S_{t+1})$, is used instead.

Here TD goes beyond and combines both ideas

- It combines the sampling of Monte Carlo with the bootstrapping of DP.

Furthermore

The Monte Carlo target is an estimate

- because the expected value in (6.3) is not known, but a sample is used instead

The DP target is an estimate

- because $v_\pi(St+1)$ is not known and the current estimate,$V(S_{t+1})$, is used instead.

Here TD goes beyond and combines both ideas

- It combines the sampling of Monte Carlo with the bootstrapping of DP.

As in many methods there is an error

And it is used to improve the policy

$$\delta_t = R_{t+1} + \alpha V\left(S_{t+1}\right) - V\left(S_t\right)$$

Because the TD error depends on the next state and next reward

- It is not actually available until one time step later.

δ_t is the error in $V\left(S_t\right)$

- At time $t+1$

As in many methods there is an error

And it is used to improve the policy

$$\delta_t = R_{t+1} + \alpha V\left(S_{t+1}\right) - V\left(S_t\right)$$

Because the TD error depends on the next state and next reward

- It is not actually available until one time step later.

δ_t is the error in $V\left(S_t\right)$

- At time $t+1$

As in many methods there is an error

And it is used to improve the policy

$$\delta_t = R_{t+1} + \alpha V\left(S_{t+1}\right) - V\left(S_t\right)$$

Because the TD error depends on the next state and next reward

- It is not actually available until one time step later.

δ_t is the error in $V\left(S_t\right)$

- At time $t+1$

Finally

If the array V does not change during the episode,

- The Monte Carlo Error can be written as a sum of TD errors

$$\begin{aligned}
\epsilon = G_t - V(S_t) &= R_{t+1} + \alpha G_{t+1} - V(S_t) + \alpha V(S_{t+1}) - \alpha V(S_{t+1}) \\
&= \delta_t + \alpha \left(G_{t+1} - V(S_{t+1})\right) \\
&= \delta_t + \alpha \delta_{t+!} + \alpha^2 \left(G_{t+2} - V(S_{t+2})\right) \\
&= \sum_{k=t}^{T-1} \alpha^{k-t} \delta_k
\end{aligned}$$

Optimality of TD(0)

Under batch updating

- TD(0) converges deterministically to a single answer independent
  - as long as γ is chosen to be sufficiently small.

$$V(S_t) \leftarrow V(S_t) + \gamma \left[G_t - V(S_t) \right] \text{ and}$$

Outline

Q-learning: Off-Policy TD Control

One of the early breakthroughs in reinforcement learning

- The development of an off-policy TD control algorithm, Q-learning (Watkind, 1989)

It is defined as

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \gamma\left[R_{t+1} + \alpha \max_a Q\left(S_{t+1}, a\right) - Q\left(S_t, A_t\right)\right]$$

Here

- The learned action-value function, Q, directly approximates q_* the optimal action-value function
  - independent of the policy being followed q.

Q-learning: Off-Policy TD Control

One of the early breakthroughs in reinforcement learning

- The development of an off-policy TD control algorithm, Q-learning (Watkind, 1989)

It is defined as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \gamma \left[R_{t+1} + \alpha \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Here

- The learned action-value function, Q, directly approximates q_* the optimal action-value function
  - independent of the policy being followed q.

Q-learning: Off-Policy TD Control

One of the early breakthroughs in reinforcement learning

- The development of an off-policy TD control algorithm, Q-learning (Watkind, 1989)

It is defined as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \gamma \left[R_{t+1} + \alpha \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Here

- The learned action-value function, Q, directly approximates q_* the optimal action-value function
  - independent of the policy being followed q.

Furthermore

The policy still has an effect

- to determine which state-action pairs are visited and updated.

Therefore

- All that is required for correct convergence is that all pairs continue to be updated

Under this assumption and a variant of the usual stochastic approximation conditions

- Q has been shown to converge with probability 1 to q_*

Furthermore

The policy still has an effect

- to determine which state-action pairs are visited and updated.

Therefore

- All that is required for correct convergence is that all pairs continue to be updated

Under this assumption and a variant of the usual stochastic approximation conditions

- Q has been shown to converge with probability 1 to q_*

Furthermore

The policy still has an effect

- to determine which state-action pairs are visited and updated.

Therefore

- All that is required for correct convergence is that all pairs continue to be updated

Under this assumption and a variant of the usual stochastic approximation conditions

- Q has been shown to converge with probability 1 to q_*

Q-learning (Off-Policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters

- $\alpha \in (0, 1]$ and small $\epsilon > 0$

Initialize

- Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}$, arbitrarily except that $Q\,(terminal, \cdot) = 0$

Q-learning (Off-Policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters

- $\alpha \in (0, 1]$ and small $\epsilon > 0$

Initialize

- Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}$, arbitrarily except that $Q\left(terminal, \cdot\right) = 0$

Then

Loop for each episode

1. Initialize S
2. Loop for each step of episode:
3. Choose A from S using policy derived from Q (for example ϵ-greedy)
4. Take action A, observe R, S'
$$Q(S, A) \leftarrow Q(S, A) + \gamma \left[R + \alpha \max_a Q(S', a) - Q(S, A) \right]$$
5. $S \leftarrow S'$
6. Until S is terminal

Outline

One of the earliest successes

Backgammon is a major game

- with numerous tournaments and regular world championship matches.

It is a stochastic game with a full description of the world

One of the earliest successes

Backgammon is a major game

- with numerous tournaments and regular world championship matches.

It is a stochastic game with a full description of the world

What do they used?

TD-Gammon used a nonlinear form of $TD(\lambda)$

- An estimated value $\hat{v}(s, \boldsymbol{w})$ is used to estimate the probability of winning from state s.

Then

- To achieve this, rewards were defined as zero for all time steps except those on which the game is won.

What do they used?

TD-Gammon used a nonlinear form of $TD(\lambda)$

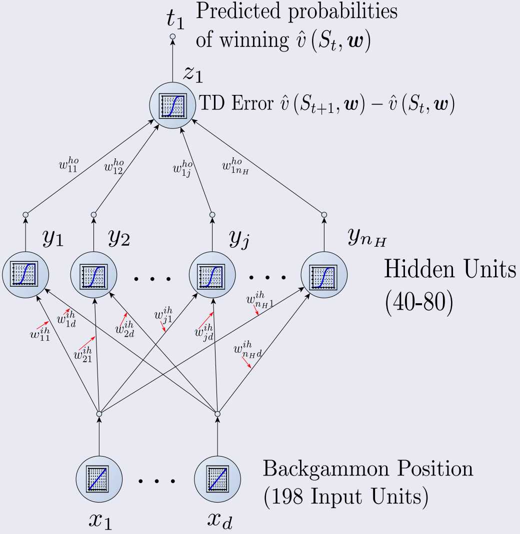
- An estimated value $\hat{v}(s, \boldsymbol{w})$ is used to estimate the probability of winning from state s.

Then

- To achieve this, rewards were defined as zero for all time steps except those on which the game is won.

Implementation of the value function in TD-Gammon

They took the decision to use a standard multilayer ANN



The interesting part

Tesauro's representation

- For each point on the backgammon board
  - Four units indicate the number of white pieces on the point

Basically

- If no white pieces, all four units took zero values,
- if there was one white piece, the first unit takes a one value
- And so on...

Therefore, given four white pieces and another four black pieces at each of the 24 points

- We have 4*24+4*24=192 units

The interesting part

Tesauro's representation

- For each point on the backgammon board
  - Four units indicate the number of white pieces on the point

Basically

- If no white pieces, all four units took zero values,
- if there was one white piece, the first unit takes a one value
- And so on...

Therefore, given four white pieces and another four black pieces at each of the 24 points

- We have 4*24+4*24=192 units

The interesting part

Tesauro's representation

- For each point on the backgammon board
  - Four units indicate the number of white pieces on the point

Basically

- If no white pieces, all four units took zero values,
- if there was one white piece, the first unit takes a one value
- And so on...

Therefore, given four white pieces and another four black pieces at each of the 24 points

- We have 4*24+4*24=192 units

Then

We have that

- Two additional units encoded the number of white and black pieces on the bar (Each took the value $n/2$ with n the number of the bar)
- Two other represent the number of black and white pieces already (these took the value $n/15$)
- two units indicated in a binary fashion the white or black turn.

Outline

TD(λ) by Back-Propagation

TD-Gammon uses a semi-gradient of the $TD(\lambda)$

- $TD(\lambda)$ has the following general update rule:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \gamma \left[R_{t+1} + \alpha \hat{v}\left(S_{t+1}, \boldsymbol{w}_t\right) - \hat{v}\left(S_t, \boldsymbol{w}_t\right) \right] \boldsymbol{z}_t$$

  - with $\boldsymbol{w}_t$ is the vector of all modifiable parameters

$\boldsymbol{z}_t$ is a vector of eligibility traces

- Eligibility traces unify and generalize TD and Monte Carlo methods.
- It works as Long-Short Term Memory

The rough idea is that when a component of $\boldsymbol{w}_t$ produces an estimate

- $\boldsymbol{z}_t$ is bumped out and then begins to fade away.

TD(λ) by Back-Propagation

TD-Gammon uses a semi-gradient of the $TD(\lambda)$

- $TD(\lambda)$ has the following general update rule:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \gamma \left[R_{t+1} + \alpha \hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w}_t) \right] \boldsymbol{z}_t$$

  - with $\boldsymbol{w}_t$ is the vector of all modifiable parameters

$\boldsymbol{z}_t$ is a vector of eligibility traces

- Eligibility traces unify and generalize TD and Monte Carlo methods.
- It works as Long-Short Term Memory

The rough idea is that when a component of $\boldsymbol{w}_t$ produces an estimate

- $\boldsymbol{z}_t$ is bumped out and then begins to fade away.

TD(λ) by Back-Propagation

TD-Gammon uses a semi-gradient of the $TD\left(\lambda\right)$

- $TD\left(\lambda\right)$ has the following general update rule:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \gamma \left[R_{t+1} + \alpha \hat{v}\left(S_{t+1}, \boldsymbol{w}_t\right) - \hat{v}\left(S_t, \boldsymbol{w}_t\right)\right] \boldsymbol{z}_t$$

  - with $\boldsymbol{w}_t$ is the vector of all modifiable parameters

$\boldsymbol{z}_t$ is a vector of eligibility traces

- Eligibility traces unify and generalize TD and Monte Carlo methods.
- It works as Long-Short Term Memory

The rough idea is that when a component of $\boldsymbol{w}_t$ produces an estimate

- $\boldsymbol{z}_t$ is bumped out and then begins to fade away.

How this is done?

Update Rule

$$\boldsymbol{z}_t = \alpha\lambda \boldsymbol{z}_{t-1} + \nabla \hat{v}\left(S_t, \boldsymbol{w}_t\right) \text{ with } \boldsymbol{z}_0 = 0$$

Something Notable

- The gradient in this equation can be computed efficiently by the back-propapagation procedure

For this, we set the error in the ANN

$$\hat{v}\left(S_{t+1}, \boldsymbol{w}_t\right) - \hat{v}\left(S_t, \boldsymbol{w}_t\right)$$

How this is done?

Update Rule

$$\boldsymbol{z}_t = \alpha\lambda\boldsymbol{z}_{t-1} + \nabla\hat{v}\left(S_t, \boldsymbol{w}_t\right) \text{ with } \boldsymbol{z}_0 = 0$$

Something Notable

- The gradient in this equation can be computed efficiently by the back-propapagation procedure

For this, we set the error in the ANN

$$\hat{v}\left(S_{t+1}, \boldsymbol{w}_t\right) - \hat{v}\left(S_t, \boldsymbol{w}_t\right)$$

How this is done?

Update Rule

$$\boldsymbol{z}_t = \alpha\lambda \boldsymbol{z}_{t-1} + \nabla \hat{v}\left(S_t, \boldsymbol{w}_t\right) \text{ with } \boldsymbol{z}_0 = 0$$

Something Notable

- The gradient in this equation can be computed efficiently by the back-propapagation procedure

For this, we set the error in the ANN

$$\hat{v}\left(S_{t+1}, \boldsymbol{w}_t\right) - \hat{v}\left(S_t, \boldsymbol{w}_t\right)$$

Outline

Conclusions

With this new tools as Reinforcement Learning

- The New Artificial Intelligence has a lot to do in the world

Thanks to be in this class

- As they say we have a lot to do...

Conclusions

With this new tools as Reinforcement Learning

- The New Artificial Intelligence has a lot to do in the world

Thanks to be in this class

- As they say we have a lot to do...