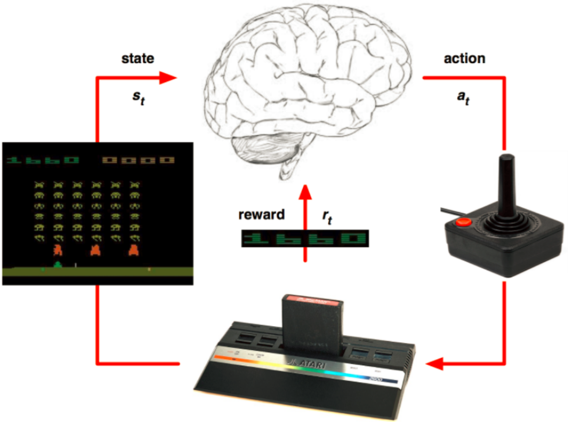
Deep Q-Learning in a nutshell

Bar Hilleli

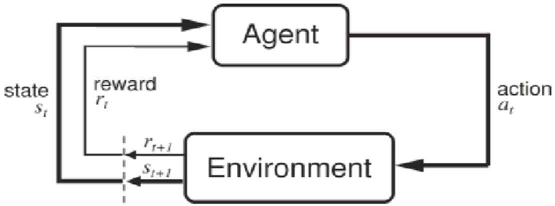
barhilleli5@gmail.com

May 23, 2019

Reinforcement Learning - background (0)



Reinforcement Learning - background (1)



- *Reinforcement Learning* (RL) describes a set of learning problems where an **agent interacts with an environment**.
- RL algorithms seek to find a policy, π, that **maximize the reward received** over time from the environment.

Motivation

Videos

Alpha Go trailer
David Silver

Markov Decision Process - MDP

Definition

The MDP framework consists of four elements: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$

- $\mathcal{S}$ is a discrete set of states
- $\mathcal{A}$ is a discrete set of actions
- $\mathcal{R}$ is a reward model $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a transition probability matrix $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$

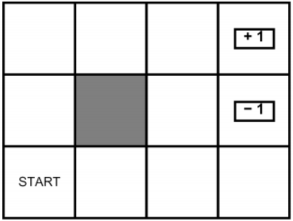
Policy

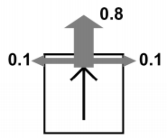
Definition

A **policy** gives the probability of taking action a when in state s:

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$$
$$\pi(a|s) = \mathbb{P}(a_t = a|s_t = s)$$

Grid World Example





- The agent's actions do not always go as planned
- Small "living" reward of 0.1 each step
- Big rewards come at the end
- Following a policy produces sample trajectories:
  $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

Objective

Given an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ we wish to find optimal policy $\boldsymbol{\pi}^\star$ so as to **maximize** the **expected sum of rewards**. Formally,

$$\pi^\star = \underset{\pi}{\operatorname{argmax}}\ \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t r_t \mid \pi\right] \quad \text{with } a_t \sim \pi(\cdot|s_t),\ s_{t+1} \sim p(\cdot|s_t, a_t)$$

where γ is called the *discount factor* in range $[0, 1]$

State-Value Function

Definition

The **state value function** for policy π, denoted by $V^{\pi}(s)$, is the **expected return when starting in state s and following policy π thereafter**, and is formally defined as,

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \middle| s_t = s, \pi\right]$$

Action-Value Function

Definition

Similarly, The **state-action value function** for policy π, denoted by $Q^\pi(s, a)$, is the **expected reward of taking action *a* in state *s* and following policy π thereafter**, and is formally defined as,

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a, \pi\right]$$

Main Idea behind Q-Learning

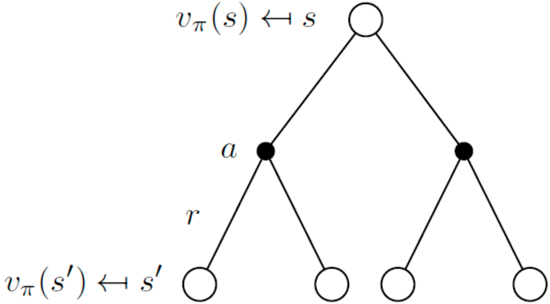
The main idea behind Q-learning is that if we have a function

$$Q^{\pi} : \mathcal{S} \times \mathcal{A} \to \textit{Total Reward}$$

that could tell us what our return would be, if we were to take an action in a given state, then we could easily construct an improved policy that maximizes our rewards (considering the deteministic case):

$$\pi_{new}(s) = \underset{a}{\operatorname{argmax}}\, Q^{\pi}(s, a)$$

v^{π} Backup Diagram



Bellman Expectation Equation for v^π

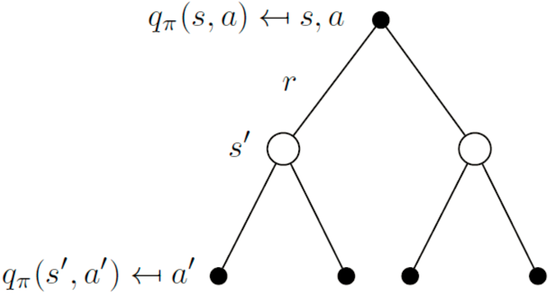
A fundamental property of value functions used throughout reinforcement learning is that they satisfy particular **recursive relationships**, which is also known as the **Bellman expectation equation for** v^π:

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, \pi\right] \\
&= \mathbb{E}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, \pi\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s', \pi\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right] \\
&= \mathbb{E}_{a_t, s_{t+1}} \left[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s\right]
\end{aligned}
$$

Bellman Expectation Equation for Q^{π}

In the same way we can write the **Bellman expectation equation** for Q^{π}:

$$Q^{\pi}(s,a) = \mathbb{E}_{s_{t+1}}\left[r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s, a_t = a\right]$$



Optimal Policy

Considering finite MDPs, an **optimal policy**, $\pi^\star$, is better than all other policies in the sense that its expected return is greater than or equal to that of all other policies for all states. Meaning,

$$V^{\pi^\star}(s) \geq V^{\pi}(s)$$

for all $s \in \mathcal{S}, \pi \in \Pi$.

Optimal State-Value Function

Definition

The value functions of $\pi^\star$ is called **optimal state-value function**, denoted by $\boldsymbol{V^\star}$ and defined as,

$$V^\star(s) = \max_\pi V^\pi(s)$$

for all $s \in \mathcal{S}$.

Optimal Action-Value Function

Definition

Similarly the **optimal action-value function**, denoted $\boldsymbol{Q^{\star}}$, is defined as

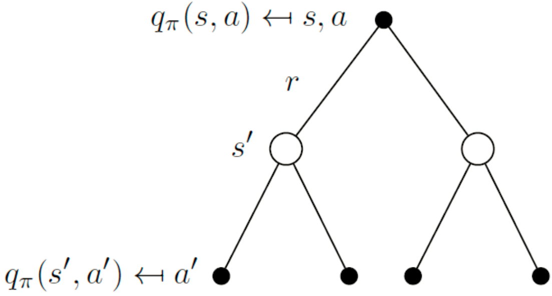
$$Q^{\star}(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$.

$Q^\star$ in terms of $V^\star$

$Q^\star(s,a)$ gives the expected return taking action a in state s and thereafter following $\pi^\star$, therefore we can write,

$$Q^\star(s,a) = \mathbb{E}_{s_{t+1}}\left[r_{t+1} + \gamma V^\star(s_{t+1}) \mid s_t = s, a_t = a\right]$$



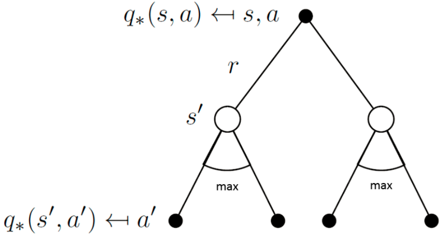
Bellman Optimality Equation for $Q^\star$

Since the value of a state s under $\pi^\star$ must be **equal to the expected return of the best action** from that state we have that,

$$V^\star(s) = \max_{a \in \mathcal{A}} Q^\star(s, a)$$

Therefore, we can write the **Bellman optimality equation** for $Q^\star$,

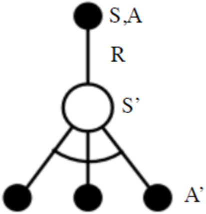
$$Q^\star(s, a) = \mathbb{E}_{s_{t+1}}\left[r_{t+1} + \gamma \max_{a'} Q^\star(s_{t+1}, a') | s_t = s, a_t = a \right]$$



Q-Learning

- The **Q-learning** algorithm aims to find the optimal action-value function.
- The **Q-learning update** is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$



Q-Learning Algorithm

Q-learning: An off-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Very Cool Online Demos

Demos

Demo 1
Demo 2 (karpathy)

Q-Learning Properties

- Model-Free: no knowledge of MDP
- TD(0): bootstrapping - updates a guess towards a guess
- Off-Policy: learn about optimal policy while following exploratory policy

Theorem

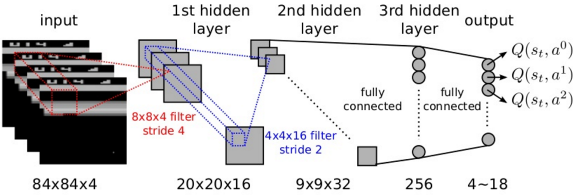
Q-learning control converges to the optimal action-value function, $Q(s,a) \rightarrow Q^{\star}(s,a)$

Approximating $Q^\star(s, a)$ using a Neural Network

- Dealing with a very large state space of images, we **approximate** the optimal action-value function using a **deep Q-network** (DQN) with **parameters $\boldsymbol{\theta}$**:

$$Q_\theta(s, a) \approx Q^\star(s, a).$$

- A deep Q-network is a neural network that for a given state outputs a vector of action values.



DQN Algorithm

The **DQN** algorithm:

- Take action a_t according to ϵ-greedy policy.
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{B}$.
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{B}$.
- Compute Q-learning targets w.r.t. old, fixed parameters $\tilde{\theta}$.
- Optimize MSE between Q-network and Q-learning targets

$$L(\theta) = \mathop{\mathbb{E}}_{s,a,r,s' \sim \mathcal{B}} \left[\left(r + \gamma \max_{a'} Q_{\tilde{\theta}}(s', a') - Q_\theta(s, a) \right)^2 \right]$$

- Using variant of stochastic gradient descent.

Toward Deep Reinforcement Learning without a Simulator: An Autonomous Steering Example

Video Link

Toward Deep Reinforcement Learning without a Simulator: An Autonomous Steering Example

Recommended Reading/Viewing Material

Excellent book

Reinforcement Learning: An Introduction (Richard S. Sutton and Andrew G. Barto)

Excellent video course

RL Course by David Silver

Thank you