Reinforcement Learning

a practical approach

Prof. Renzo Silva, Konstantin Kuchenmeister, Elijah Hall
Department of Mathematics
Columbia University

Mathematics Building
MATH203

Saturday 9$^{\text{th}}$ April, 2022

Outline

Supervised Learning

General Model

Recall that, in supervised learning we have a set up tuples

$$T = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\} \tag{1}$$

and through training we are trying to estimate the underlying map:

$$f(X) + \epsilon \longrightarrow Y \tag{2}$$

with

$$\hat{Y} = \hat{f}(X) \tag{3}$$

where ϵ is the irreducible error we can not model.

Supervised Learning
Algorithms

To model this relationship we have, among others, looked at

1. Regression
   - Linear Regression (Performance, Simplicity)
   - kNN-Regressor (Performance, Simplicity)
   - Decision Tree (Performance, Simplicity
   - Random Forest Regressor (Accuracy)
   - Neural Network (Accuracy)
2. Classification
   - Logistic Regression (Performance, Simplicity)
   - kNN
   - Decision Tree, Random Forest
   - Neural Network (Accuracy)
   - SVM

Transitioning

BUT, in HW3 we have also looked at clustering...

What is the difference?

Definition

Reinforcement Learning

Definition

Reinforcement learning problems involve learning what to do, how to map situations to actions, so as to maximize a numerical reward signal. [...] Moreover, the learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them out[1].

In short: We also do not know the $y's$... Motivation

Markov Decision Process

Definitions

Definition

In our context, a Markov Decision Process (MDP) is a 4-tuple (S, A, P, R) which allows us to formalize and model sequential decision making[2] as used in RL.

Key Terminology

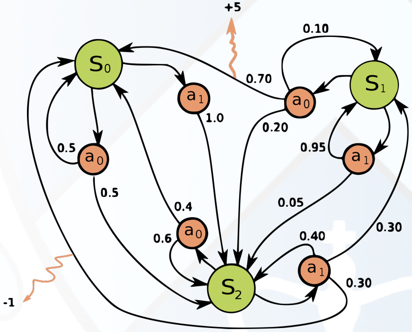
In particular,

- S_i = State of the Environment
- A_i = Set of actions available in state S_i
- $P_i = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$
- R_i = Reward attained from transitioning from state S_i to S_{i+1} under action A_i

Markov Decision Process

Visualized

Figure: Markov Decision Process (Source=Wikipedia)



Markov Decision Process

Episodes

Definition

Naturally, as we go through states through actions, we create a trajectory (similar to the natural filtration in stochastic processes):

$$\tau = (s_0, a_0, s_1, a_1, ...) \tag{4}$$

If $|\tau| < \infty$ (i.e. $\exists$a terminal state T) it is called episode, and is used to describe the agent "playing the game" one time.

Markov Decision Process

Rewards

At every discrete time step we earn a reward:

Lemma

$$R(s) = \mathbf{E}\left[R(t+1) \mid S_t = s'\right] \tag{5}$$

The infinite-horizon discounted return function:

$$G(t) = R(t+1) + \gamma R(t+2) + \dots = \sum_{k=0}^{\infty} \gamma^k R(t+k+1) \tag{6}$$

where γ is the discount factor.

Markov Decision Process

Formulating the problem

Now, what is the optimal action to take at an arbitrary time step t?

$$\mathbf{E}\left[G(t) \mid S = s'\right] \tag{7}$$

Which corresponds to maximizing the sum of expected future rewards after starting from state s.

Markov Decision Process

Formulating the problem - Intro to the Bellman Equation

We can split this up in:

- immediate reward earned (R(t+1))
- future rewards earned

$$
\begin{aligned}
v(s) &= \mathbf{E}\left[R(t+1) + \gamma R(t+2) + \gamma^2 R(t+3) + \ldots \mid S = s'\right] \\
&= \mathbf{E}\left[R(t+1) + \gamma(R(t+2) + \gamma R(t+3) + \ldots) \mid S = s'\right] \\
&= \mathbf{E}\left[R(t+1) + \gamma(G(t+1)) \mid S = s'\right] \\
&= \mathbf{E}\left[R(t+1) + \gamma(v(S_{t+1})) \mid S = s'\right]
\end{aligned}
$$

Which corresponds to maximizing the sum of expected future rewards after starting from state s. You enter a state, collect the immediate reward and average all the possible future rewards discounted.

Markov Decision Process

Policies

Definition

A policy π is a distribution over actions given states,

$$\pi(a \mid s) = \Pr(A_t = a \mid S_t = s) \tag{8}$$

"If were in state s, the agent is going to choose action a with probability $\pi(a \mid s)$." Defines the behavior of the agent.

Markov Decision Process

Policies

Definition

The state-value function $v_\pi(s)$ is the expected total return from starting from state s and following policy π thereafter:

$$v_\pi(s) = \mathbf{E}_\pi \left[G(t) \mid S_t = s \right] \tag{9}$$

Definition

The action-value function $q_\pi(s)$ is the expected total return from starting from state s, taking action a and following policy π thereafter:

$$q_\pi(s, a) = \mathbf{E}_\pi \left[G(t) \mid S_t = s, A_t = a \right] \tag{10}$$

Markov Decision Process

Policies

And again, we can decompose them into:

Lemma

Theorem

$$v_\pi(s) = \mathbf{E}_\pi \left[R(t+1) + \gamma(v(S_{t+1}) \mid S_t = s\right]$$

Lemma

Theorem

$$q_\pi(s,a) = \mathbf{E}_\pi \left[R(t+1) + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

Markov Decision Process

Relation between v and q

Remark

$$v_\pi(s) = \sum_{a \in A} \pi(a \mid s) q_\pi(s,a) \tag{11}$$

$$q_\pi(s) = R_s^a + \gamma \sum_{s \in S} P_{ss}^a \sum_{a \in A} \pi(a \mid s) q_\pi(s,a) \tag{12}$$

Markov Decision Process

Putting it all together - Optimality 1

So what is now the optimal way to act?

Definition

$$v_*(s) = \max_\pi v_\pi(s) \tag{13}$$

$$q_*(s) = \max_\pi q_\pi(s, a) \tag{14}$$

The optimal is the maximum value of the respective function over all possible policies.

Markov Decision Process

Putting it all together - Optimality 2

Definition

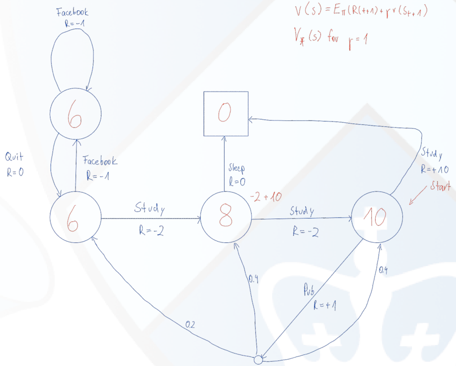
For any MDP, a policy $\pi > \pi'$ iff $v_\pi(s) > v_{\pi'}(s)$ and all optimal policies fulfill the previously defined optimal state-value $v_*(s)$ and action-value $q_*(s)$ functions.

Definition

$$\pi_* = 1_{\max_{a \in A} q_*(a,s)} \tag{15}$$

Markov Decision Process
Visualized

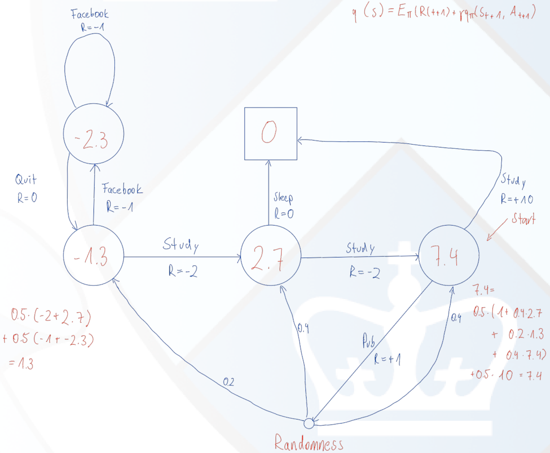
Figure: Markov Decision Process (Credits=David Silver, DeepMind)



Markov Decision Process

Visualized

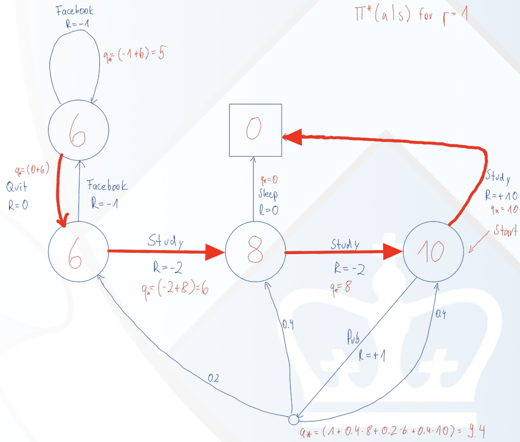
Figure: Markov Decision Process (Credits=David Silver, DeepMind)



Markov Decision Process

Visualized

Figure: Markov Decision Process (Credits=David Silver, DeepMind)



Definition

... this Bellman-Optimality equation where we take max q_* can be solved through Q-learning (which is off-policy).

Definition

For any finite Markov decision process (FMDP), Q-learning finds an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state. Q-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy.

Q-learning formula

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Bigg)}^{\text{temporal difference}} \quad (16)$$

Epsilon-Greedy strategy

At each time step, our agent can either **explore** or **exploit**.

Carl Example

At lunchtime, Carl can either go exploit (go to a place where he knows the food is good, he will get a great reward) or explore (go to a new place with the goal of finding a place with even higher reward)

Epsilon-Greedy strategy

Hence, we start with an ϵ (exploration rate, the probability that our agent will explore rather than exploit) of 1 which gradually decreases after each episode.

Pseudo-Code

$if(random.generate(0,1) > epsilon):$

// exploit, choose action with highest Q-value

$else:$

// explore, choose a random action and see what happens

Numerical Example

Suppose the reward Matrix R is given by

state / action	A	B	C	D	E	F
A	-	-	-	-	0	-
B	-	-	-	0	-	100
C	-	-	-	0	-	-
D	-	0	0	-	0	-
E	0	-	-	0	-	100
F	-	0	-	-	0	100

Numerical Example

And we initialize the Q-table...

	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

... where F is the goal state.

Numerical Example

Now we are starting in state B and randomly go to F:
We can update the Q-table in the following way:

Update Q-values

$$Q^{new}(B,F) \leftarrow \underbrace{Q(B,F)}_{\text{old value}} + \underbrace{0.1}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{\underbrace{R(B,F)}_{\text{reward}} + \underbrace{0.8}_{\text{discount factor}} \cdot \underbrace{\max_a(Q(F,B),Q(F,E),Q(F,F))}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(B,F)}_{\text{old value}} \Big)}^{\text{temporal difference}} \quad (17)$$

$$Q^{new}(B,F) \leftarrow \underbrace{0}_{\text{old value}} + \underbrace{0.1}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{\underbrace{100}_{\text{reward}} + \underbrace{0.8}_{\text{discount factor}} \cdot \underbrace{\max_a(0,0,0)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{0}_{\text{old value}} \Big)}^{\text{temporal difference}} \quad (18)$$

$$Q^{new}(B,F) = 0 + 0.1 \cdot (100 + 0.8 \cdot 0 - 0) = 10 \quad (19)$$

Numerical Example

And we update our Q-table

	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	10
C	0	0	0	0	0	0
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

... and the episode is finished, since F is the goal state.

Deep reinforcement learning motivation

Problems with vanilla Q-learning:

1. Storing the entries Q(s,a) for all possible States and actions A (in our matrix) is not always feasible:
   In classical board games: $|S| = b^d$
   where b is the mean number of allowed moves for a given board position and d is the depth, the typical number of moves per game.
   Chess: b = 35, d = 80
2. Transition probability in MDP's is often unknown or can only be approximated

$\longrightarrow$ Approximate the Q-table with a Neural Net

Policies, Targets and TD's

- Looking back at the formula for the Q-Value: We can see we have a Q-Value for Q(s,a) at time t, but then we can recalculate this value at t+1 with new information.
- This value at t+1 (after the action was taken) - the original value is the temporal difference observed in the formula.
- Naturally, the closer this value gets to zero, the closer we are in converging to the true value.

Policies, Targets and TD's

We can formalize this problem through the MSE:

$$L(\Theta) = \mathbf{E}_a \left\| Y - Q(S, A, \Theta) \right\|_2^2 \tag{20}$$

where y_i is the temporal difference and θ_i are the weights and biases of the network.

Policies, Targets and TD's

And in order to not play cat and mouse, we introduce a policy and a target network:

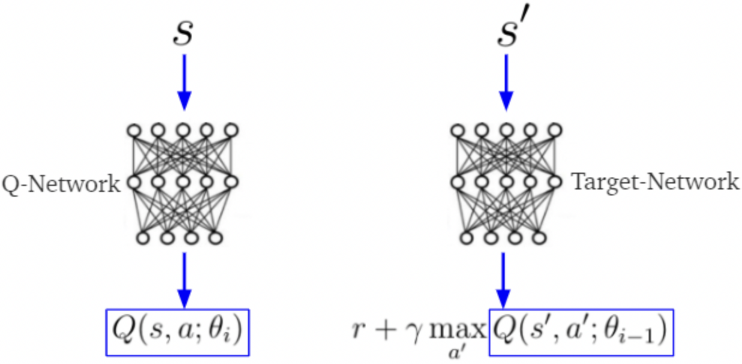


Figure: Policy and target networks in DQN's

Experience replay

- In order to get better at estimate of the target Q-values we store our trajectory (S,A,R,S') at each step.
- We then randomly sample from this replay and train the model based on these results.

DQN Trading Bot

See uploaded Jupyter Notebook

Sutton, Richard S. and Andrew G. Barto. *Reinforcement learning: an introduction.* Adaptive computation and machine learning. Cambridge, Mass: MIT Press, 1998. 322 pp. ISBN: 9780262193986.

White, Chelsea C. “A survey of solution techniques for the partially observed Markov decision process”. In: *Annals of Operations Research* 32.1 (Dec. 1, 1991), pp. 215–230. ISSN: 1572-9338. DOI: 10.1007/BF02204836. URL: https://doi.org/10.1007/BF02204836 (visited on 03/13/2022).

Reinforcement Learning

a practical approach

Prof. Renzo Silva, Konstantin Kuchenmeister, Elijah Hall
Department of Mathematics
Columbia University

Mathematics Building
MATH203

Saturday 9$^{\text{th}}$ April, 2022