Reinforcement Learning

Mario Dohr

November 2, 2022

Contents

1 Introduction

What is Reinforcement Learning? RL is the computational approach to learning from interactions. RL problems involvle learning "what to do":

- by mapping states to actions
- while maximizing total reward

Three central Characteristics:

- Closed-loop problems: actions of agents influence its later observations
- No direct instructions: the agent is not told which action to take.
- No initial knowledge: the agent has to figure out the consquences of its actions.

Differences to Supervised- and Unsupervised learning

- Difference to Suppervised learning: 1. Agent must learn from own experience. 2. No supervisor tell what to do
- Difference to unsupervised learning: 1. No predefined data. 2. Agent tries to maximize reward.

Reinforcement Maximize reward.

- **Given:** an environment and a reward signal
- **Find:** a behavior that maximizes total reward over time

The agent takes action in the environment and gets back a state and a reward. Challenge is finding actions by exploring, exploit knowledge about good actions to maximize reward.

Challenges in RL

- How maximize reward? We need to exploit our knowledge about the past.
- How find highly rewarded actions? We need to explore the environment.

Dilemma: We have a trade-off between **Explore** and **Exploit**:

- to obtain reward an agent must favor actions that have proven to be benificial in the past (Exploitation).
- to discover such actions, an agent has to try actions that it has not selected before (Exploration).

1.1 Examples

- a animal learn to walk.
- playing chess
- trash collection robots: has to decide search more trash or go to recharge.
- Computer games
- OpenAI Gym is a python library for reinforcement learning

Commonalities

- interaction between agent and environment
- agents seek to achieve a goal in their environment, despite uncertainty
- actions affect the future state of the environment
- has to take into to account indirect, delayed consequences of actions
- consequences of actions can't be fully predicted
- agents use experience
- interaction with environment is essential

1.2 Elements of Reinforcement Learning

- **Environment**: for example: the universe, one street, coffee machine, chess board ...
- **States**: represents the environments current condition: a position, temperature, etc.
- **Actions**: for each state there is a set of actions. E.g. turning steering wheel, move in a particular direction, ...
- **Policy**: A policy completely determines its behavior. It defines which action an agent can take in a given state. E.g. if car leaves street, seer in the other direction, if pressure to high, decrease current to heating element, move to a position with high expected future reward. Is a property of the agent. An agent has a specific policy at a specific time.
- **Reward signal**: Is given out from the environment and encodes how good the agent is doing currently. Given from environment, not the agents knowledge. It is the primary basis for altering the policy. E.g.: winning or loosing the game, car stays on the street, does not crash.
- **Value functions**: is compressed knowledge about the future, it encodes an agent's experience. E.g.: eat cake, feel good. Turn steering wheel D degrees avoid a crash.

Policy The behavior of an agent is called a policy. The policy:

- maps a state to a probability distribution over actions
- samples from this distribution to select an action

We can say an RL agent follows a policy (behaves a certain way) and updates its policy (to try and maximize reward). The policy completely determines its behavior, deciding which action to take at a given state. The policy is that, what the agent should learn???

Policy implementation Is a mapping from states to actions. Could be

- a lookup table
- a simple function
- a search process
- a DNN
- a combination

Reward Signal . Defines the goal in a RL problem. The reward:

- is a single real number
- is perceived by the agent at each time step
- defines what is good and what is bad
- is immediate and defines features of the problem
- primary basis for changing the policy

Value Function Defines what is good in the long run.

- is the total amount of reward an agent can expect to accumulate in the future starting from that state
- usually an estimate
- often used to choose an action

Reward vs. Value

- rewards are immediate, part of environment
- values are predictions of future reward, part of the agent
- rewards are used, in order to estimate value
- the only purpose of estimation values is to get more reward in the long run
- most RL algos focus on efficient value estimation
- a state might yield a low immediate reward but still have a high value because it is usually followed by benificial states.

Model environment Model-based methods vs model-free methods.

Boundaries Boundary between Agent and Environment? Fill summary of Example with the fish this is temporal-difference-learning.

1.3 Example: Gridworld Environment

- **States:** we have 11 states: where is the fish?
- **Actions:** In all states we have the same actions. The fish can move. When he moves left in the left bottom the state doesn't change.
- **Loop:** the agent takes an action and receives a new state and a reward.
- **How get food and survive?** We set up a table with the states and the values for the value function.

Lookup gridworld learning

2 Multi-armed Bandits

Here we are in a so called *nonassociative* setting, this is we only have one situation or state.

2.1 k-armed bandits

Is inspired by the "slot machines" in casinos. The problem is the following: we have k different options or choices for an action and receive a reward from a probability distribution. We try to maximize the return over time. Each of the action we can take have an expected reward:

$$q_*(a) = \mathbb{E}[R_t|A_t = a]$$

If we would know this, we always would choose the action with the highest reward. But we don't know this, wie only have the estimates $Q_t(a)$, which is the estimated value of action a at time step t. We want $Q_t(a)$ close to $q_*(a)$.

At every time step, their is one action a with the highest value, this is the greedy action. If we choose this action, we are *exploiting* our knowledge about the values of the actions, but we don't gain new knowledge. If we choose a non-greedy action, we are exploring and gain knowledge about the values of action. There is always a trade-off between *explore* and *exploit*. The need to balance exploration and exploitation is a distinctive challenge that arises in reinforcement learning.

2.2 Action-value Methods

We are looking for methods to estimate the value of actions. The true value of an action is the mean reward when selected. We can estimate this with:

$$Q_t(a) = \frac{\sum_{i_1}^{t-1} R_t \; 1_{A_t=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}}$$

By the law of large numbers, $Q_t(a)$ converges to $q_*(a)$. The greedy action selection method is

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

This is called the *sample-average* method. This is maybe not the best method. Instead of this you could randomly select another method to explore new knowledge and therefore maybe get higher rewards in the future. This is called the ϵ-*greedy* method: With probability $(1-\epsilon)$ the agent take the greedy action and with probability ϵ it select an action randomly. This method also ensures then $Q_t(a) \to q_*(a)$ for all a and the probability of selecting the optimal solution converges to greater than $1-\epsilon$.

Implementation Q_{t+1} can be computed from Q_t:

$$Q_{t+1} = \frac{1}{t}\sum_{i=1}^{t} R_i$$

$$= \frac{1}{t}(R_t + (t-1)\frac{1}{(t-1)}\sum_{i=1}^{t-1} R_i)$$

$$= Q_t + \frac{1}{t}(R_t + Q_t)$$

This update rule occurs often in RL. The general form is

$$newEstimate = oldEstimate + stepSize[target - old]$$

2.3 Optimistic initial values

All the methods we have discussed so far depend to some extent on the initial values. We can use any prior knowledge that may exist to achieve better results. If we have no prior knowledge, but set high (optimistic) values (higher than the highest possible possible reward), then the algorithm starts looking for the best possible solution. This is because when an action is performed, the reward is lower than the current given value, so the value of the action is reduced and thus it is lower than other actions and the algorithm thereby selects another action in the next step.

2.4 Upper-Confident bound action selection

ϵ-greedy selection selects a non-greedy action but randomly with no preference. It would be better to select among the non-greedy actions according to their potential of being optimal.

$$A_t = \arg\max_a \Big[Q_t(a) + \sqrt{\frac{\ln t}{N_t(a)}}\Big]$$

The square root term represents something like a variance or uncertainty in the estimates of $a's$ values. If a is chosen often, this term gets reduced. The ln ensures, that the increases get smaller over time, if a is not selected, but is still unbounded.

2.5 Non-Stationary problems

The problems discussed so far are so-called stationary problems, i.e., the parameters of the probability distribution for the rewards do not change. In non-stationary problems, they change over time. It makes sense here to give more weight to recently received rewards. One approach would be:

$$Q_{n+1} = Q_n + \alpha[R_n - Q_n]$$

$$= (1-\alpha)Q_n + \alpha R_n$$

$$= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$$

where $\alpha \in (0, 1]$ is constant.

2.6 Gradient Bandit Algorithms

In the previous methods, we looked at functions that help us estimate the value of an action. Now we'll look at how we can learn a numerical preference for an action. The higher the preference, the more often an action is selected. What counts for the preference is not the absolute value but the relative preference over another action. The preference at time t is denoted as $H_t(a)$. The probabilities for selecting an action are determined according to the soft-max distribution:

$$Pr\{A_t = a\} = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} = \pi_t(a)$$

The initial preferences are set to 0.

The learning algorithm is based on stochastic gradient descent. The update rules are

$$H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t), \text{ and}$$

$$H_{t+1}(a) = H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a) \text{ for all } a \neq A_t$$

Beschreibung/Herleitung der Update-Rule einfügen

3 Finite Markov Decission Processes

Beispiele in Slides durchgehen

Until now, we have always considered only one status, or rather, we have always had the same actions available at each point in time. Now we consider different situations, i.e. the environment is in a certain state and depending on this state certain actions are allowed. This can be considered as a consequence of bandit problems. We make the following assumptions:

- We know the current state S_t
- The possible actions depends on the current state
- The decision on an action influences the next state, but the agent can't control how.
- Can be seen as a sequence of bandit problems
- want to maximize the reward over time.

Stochastische Prozesse wiederholen

3.1 Formal Framework for Reinforcement Learning

Agent-Environment The agent learns and makes decisions. The agent interacts with the environment. The environment is everything that is outside of the control of the agent. Based on the current state S_t the agent decides on the action A_t according to its current policy: $\pi_t(a|s) = Pr\{A_t = a|S_t = s)\}$. After taking an action, the action receives a reward R_{t+1} and the new state S_{t+1}. When the agent is in state s and takes action a then, there is a probability for the next state s' and the reward r

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

This is called the *dynamics*. From the dynamics, anything else someone want to know can be computed. For example the state transition probability:

$$p(s', s, a) = \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

The expected reward from *state-action* pairs:

$$r(s, a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$$

and the expected reward for *state-action-next-state*:

$$r(s, a, s') = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a}{p(s'|s, a)}$$

Dynamics You can interpret the dynamics as a model of the laws of nature of the given environment. The dynamics is not always available. If we do have access, we can use it for planning. If not, we can either chose to learn a model of the dynamics or use a **model-free** method.

3.2 Goals and Rewards

The goal of the agent is formalized as a special signal from the environment, the reward, which is a real number. After each action the agent receive such a reward. The goal is, to maximize this reward over the long run and not to maximize the immediate reward. The reward must be chosen appropriately to the goal of the agent.

3.3 Returns and Episodes

How do we define the goal of maximizing the reward formally? If the sequence of rewards after time step t is denoted $R_{t+1}, R_{t+2}, \cdots$, then we try to maximize the expected value of this sequence.

$$G_t = R_{t+1} + R_{t+1} + \cdots + R_T$$

we try to maximize $\mathbb{E}[G_t]$. Here step T is a final time step. If we have such final step, then the agent-environment interactions breaks down in subsequences called *episodes*. If $T = \infty$ this formulation is problematic, since the reward would be infinite.

To tackle this problem we could use a concept called *discounting*.

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

where $0 \leq \gamma \leq 1$. So a reward which is received after k steps is only worth γ^{k-1} times of an immediately reward. One important fact is, that successive time steps are related to each other:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_t{+}2 + \gamma R_{t+3} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

This works for all time steps $t < T$, even if termination occurs at $t + 1$, if we set $G_T = 0$. If we set $R_t = 0$ for all $t > T$ we get a single notion for terminated and unterminated problems.

$$G_t = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} R_k$$

3.4 Policies and Value Functions

Value functions estimates, how good it is to be in a certain state or how good it is to select a specific action in a certain state. How good means, expected future rewards. A value function gives the expected future reward based on a policy π.

The value function of a state s is under a policy π, denoted as $v_\pi(s)$ is the expected reward when starting from state s.

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

Similarly we define the value of taking action a in state s under policy π.

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

The value functions v_π and q_π can be estimated from experience.

Exercise If the current state is S_t, and actions are selected accordingly to policy π, what is the expected value of R_{t+1} in terms of the *dynamics* and π?
Solution:

$$\begin{aligned}\mathbb{E}[R_{t+1}|S_t = s] &= \sum_a \mathbb{E}[R_{t+1}|S_t = s, A_t = a]\pi(a|s) \\ &= \sum_a \pi(a|s) \sum_r r \sum_{s'} p(s', r|s, a)\end{aligned}$$

Exercise Give an equation for v_π in terms of q_π and π.
Solution:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \sum_a q_\pi(s,a)\pi(a|s)$$

Exercise Give an equation for q_π in terms of v_π and the *dynamics*.
Solution: Prüfen, vermutlich falsch. Seite 78

$$\begin{aligned}q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] &= \sum_{s'} \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\, p(s'|s,a) \\ &= \sum_{s'} v_\pi(s') \sum_r p(s', r|s, a)\end{aligned}$$

A fundamental property of value-functions used throughout reinforcement learning and dynamic programming is, that they satisfy recursive relationships.

$$\begin{aligned}v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\Big[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\Big] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big],\ \forall s \in \mathcal{S}\end{aligned}$$

This comes from the following facts:

$$\mathbb{E}_\pi[R_{t+1}|S_t = s] = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a),\ \text{and}$$

$$\mathbb{E}_\pi[G_{t+1}|S_t = s] = \sum_{s'} \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\, p(s'|s)$$

To calculate $p(s'|s)$ we use the total probability:

$$p(s'|s) = \sum_a p(s'|s,a)p(a|s) = \sum_a p(s'|s,a)\pi(a|s) = \sum_a \pi(a|s) \sum_r p(s',r|s,a)$$

Beispiele im Kapitel 3.5 durchgehen, eventuell implementieren, Optimality equations

The equation

$$\sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]$$

is called the **Bellman equation** for v_π. It can be seen as an expectation

$$\sum_a \sum_{s',r} \underbrace{\pi(a|s)p(s',r|s,a)}_{probability of triple(a,s',r)} \Big[\underbrace{r + \gamma v_\pi(s')}_{outcome}\Big]$$

3.5 Optimal policies

- solving a reinforcement learning problem means finding the **optimal policy**
- value functions define a partial ordering over policies.

$$\pi \geq \pi', \text{ iff } v_\pi(s) \geq v_{\pi'}(s)\ \forall s \in \mathcal{S}$$

- there is always one policy that is greater or equal all other policies, that is the **optimal policy**
- even there might be more then one optimal policy we denote them π_*.
- all optimal policies share the same value function called the **optimal value function**

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

- all optimal policies share the same action-value function, called the **optimal action value function**

$$q_*(s,a) = \max_\pi q_\pi(s,a),\ \forall s \in \mathcal{S}, \forall a \mathcal{A}$$

3.6 Summary

RL is about learning from interactions to achieve a goal. The *agents* interacts with the *environment* via *actions*. The agent receives *rewards* and situations, the *states* from the environment. The actions depends on the states. Everything inside the agent is controlled by the agent. Everything outside the agent is the environment. The environment is maybe known, but not controllable. Zusammenfassung vervollständigen

4 Dynamic Programming