Lecture 1

Introduction to Reinforcement Learning

Catherine Laflamme[1,2]

[1]Fraunhofer Austria Research GmbH

[2]RL Community
AI Austria

Reinforcement Learning Bootcamp, 25-27.09.2024

What are we trying to do?

Sequential decision-making problems in dynamic environments

- An agent must make a series of decisions over time, with each decision potentially affecting future outcomes.
- The environment is dynamic, meaning it can change in response to the agent's actions
- The goal is to learn a policy that maximizes cumulative rewards (or minimizes costs) over time

AI AUSTRIA RL COMMUNITY AI AUSTRIA

What are we trying to do?

Sequential decision-making problems in dynamic environments

- Job shop scheduling
- Energy management systems
- Routing
- Autonomous driving
- Robotics
- Resource management
- ...

Conceptual idea of RL

Reinforcment learning is a computational approach to learning from interaction.

Conceptual idea of RL

Learning how to map situations to actions so as to maximize a numerical reward signal characterised by:

- Trial and error search
- Delayed reward

Bandit Problems

As a warm-up, let's consider a so-called "Bandit" Problem:

- Non associative: There is only one possible state, and it remains constant
- You are faced repeatedly with a choice among k different actions Each action has an expected or mean reward when that action is selected: this is the action *value*
- The value for an action a is: $q(a) = \mathbb{E}[R_t | A_t = a]$
- If we know this value for each action, the problem is solved.
- Estimated reward is given when that action is selected: $Q_t(a) = \mathbb{E}[R_t | A_t = a]$

Bandit Problems

- Estimated reward is given when that action is selected: $Q_t(a)$
- Exploitation (Greedy): Choosing the action whose estimated value is greatest
- Exploration (Non-greedy): enables you to improve your estimate of the nongreedy action's value.
- In general it is not too important to take exploration/exploitation into account in a sophisticated way, just in some way!

Bandit Problems: Action-value methods

- One way to estimate the action value is by averaging the rewards that have already been received:

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

- Greedy action: $A_t = \operatorname{argmax} Q_t(a)$
- $\epsilon-$Greedy: With a probability ϵ select an action randomly.

Bandit Problems: Incremental Implementations

- Need a way to keep track of the action value function in a computationally efficient manner, with constant memory and constant per-time-step computation

$$Q_{n+1} = Q_n + \frac{1}{N}\left(R_n - Q_n\right)$$

- General form update rule:

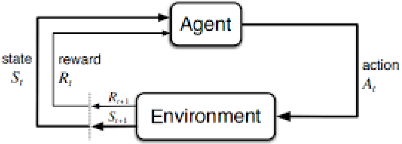
$$NewEstimate \leftarrow OldEstimate + StepSize\left(Target - OldEstimate\right)$$

Summary

- Action Values
- Balancing exploration and exploitation
- Greedy and ϵ-greedy approaches
- Incremental implementation

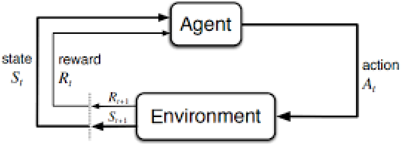
Markov Decision Process

- Mathematically idealized form of the reinforcement learning problem
- Involves evaluative feedback (as in bandits) but also includes an associative aspect - choosing different actions in different situations.
- Trade-off between mathematical tractability and applicability



Markov Decision Process

- Agent: Selects actions to take
- Environment: Responds to these actions and presents new situations to the agent. Gives rise to rewards.
- The agent and environment interact at each of a sequence of discrete time steps. At each time step:
  - The agent receives a representation of the environment's state $S_t \in \mathcal{S}$
  - The agent selects an action $A_t \in \mathcal{A}$
  - The numerical reward from it's previous action $R_t \in \mathcal{R}$



AI AUSTRIA RL COMMUNITY AI AUSTRIA

Markov Decision Process

This gives rise to a trajectory

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...$$

The dynamics of the MDP are defined by the probability of being in some state s' with reward r

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

Markov Decision Process

- Markov property: The probabilities given by p completely charaterise the environment's dynamics.
- The probability of each possible value for S_t, R_t only depend on the immediatley preceding state and action S_{t-1}, A_{t-1}
- "Memoryless"

Markov Decision Process: Example

- Assume that a container contains two red balls and one green ball.
- One ball was drawn yesterday, one ball was drawn today, and the final ball will be drawn tomorrow. All of the draws are without replacement.
- Suppose you know that today's ball was red then the situation is:
-

Day	Outcome 1	Outcome 2
Yesterday	Red	Green
Today	Red	Red
Tomorrow	Green	Red

Markov Decision Process: Example

Day	Outcome 1	Outcome 2
Yesterday	Red	Green
Today	Red	Red
Tomorrow	Green	Red

- Case 1: You have no information about yesterday's ball. The chance that tomorrow's ball will be red is 1/2, both red and green are possible. On the other hand, if you know that both today and yesterday's balls were red, then you are guaranteed to get a green ball tomorrow.

Markov Decision Process: Example

Day	Outcome 1	Outcome 2
Yesterday	Red	Green
Today	Red	Red
Tomorrow	Green	Red

- Case 2: Using the same experiment above, if sampling "without replacement" is changed to sampling "with replacement," the process of observed colors will have the Markov property
- Note: To be truly Markov, the state space dimension must remain constant (ie. you cannot keep logging the previous state information to the next state).

Markov Decision Process

The **expected return** is what we are trying to maximise:

$$G_t = R_{t+1} + \gamma R_{t+2}\ \gamma^2 R_{t+3} + \ldots$$

The discount rate γ determines the present value of future rewards.

Policy and Value Functions

- RL algorithms involve estimating the functions that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state).
- "How good" is defined in terms of the expected return.
- Value functions are defined in terms of **policies**: The expected return is dependent on what actions the agent will take in the future.
- Policy $\pi(a|s)$ is the probability that the agent will take action a given state s.

State-Value Function

The expected return when starting in s and following π thereafter.

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

Action-Value Function

The expected return when starting in s and following π thereafter.

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

AI AUSTRIA RL COMMUNITY AI AUSTRIA

Value Functions

- The value functions can be estimated from experience
- If separate averages are kept for eachaction taken in each state, then these averages will converge to the values. (Monte Carlo Methods)
- If there are many states, it may not be practical to keep separate averages for each state individually.
- Instead, the values can be kept as parameterised functions (with fewer parameters than states), which can produce accurate estimates. (Function Approximator Methods)

Bellman's Equations

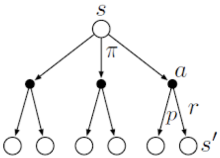
- Fundamental property of value functions is that they satisfy recursive relationships.
- This means, the value function for one state can be written in terms of the value function of a different state.

Bellman's Equations

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

Bellman's Equations

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



Backup diagram for v_π

Bellman's Equations

- Simple case of a policy where only one action is taken from a given state and deterministic environment
- Then, Bellman's equation simplifies to the very intuitive statement: The value function of the state you are in is the reward you get moving to the next state, plus the discounted value of the value function evaluated at that next state.

$$v_\pi(s) \quad = \quad [r + \gamma v_\pi(s')]$$

Bellman's Equations

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

Optimal Policies and Optimal Value Functions

- Solving reinforcement learning means finding a policy that maximises the reward over the entire episode.

Optimal State Value Function:

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

- Once you have the optimal action-value function, you can choose an action by acting greedily.

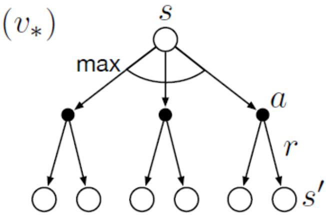
Optimal Policies and Optimal Value Functions

- Optimal value functions still satisfy recursive relationships.

$$
\begin{aligned}
v_*(s) &= \mathrm{max}_a q_{\pi_*}(s,a) \\
&= \mathrm{max}_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\
&= \mathrm{max}_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\
&= \mathrm{max}_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\
&= \mathrm{max}_a \sum_{s'} \sum_{r} p(s', r | s, a)[r + \gamma v_*(s')]
\end{aligned}
$$

Optimal Policies and Optimal Value Functions

$$v_*(s) \quad = \quad \max_a \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma v_*(s')]$$



Solving Bellman's Equations

- Bellman's optimality equation is a set of n equations (for n total number of states), with n unknowns.
- If the probability function (environment) is known, you can just solve these equations with any methods for systems of nonlinear equations.
- Once v_* is known, use a greedy policy and you will have solved your problem!
- Note: Acting "greedily" usually favors short term consequences, but the beauty of v_* is that it includes the long-term consequences of future behaviour.

Solution Dimensions

- Explicitly solving the optimality equation is rarely useful - otherwise we wouldn't be using reinforcement learning :)
- This solution relies on:
  1. The dynamics of the environment are accurately known
  2. Computational resources are sufficient to complete the calculation
  3. The Markov property holds.
- Instead, we try to approximate the solution in some way which allows us to solve the problem as best as possible.

Solution Dimensions

- Explicitly solving the optimality equation is rarely useful - otherwise we wouldn't be using reinforcement learning :)
- This solution relies on:
  1. The dynamics of the environment are accurately known
  2. Computational resources are sufficient to complete the calculation
  3. The Markov property holds.
- Instead, we try to approximate the solution in some way which allows us to solve the problem as best as possible.
- There are many ways to approximate the solution, each has advantages and disadvantages, many of which are still being researched.

Solution Dimensions

Do we...?

1. Use a model or is the update model-free?
   - Do we have invormation about hte model in advance?
   - Do we acquire knowledge about the model that we use for learning?
2. Use sample or expected updates?
   - Do we consider all possible events that can occur (based on a distribution model) or du we consider a single sample of what coupld happen?
3. Use just one step forward or run until termination?
   - Depth of update, degree of bootstrapping.

Solution Dimensions

Do we...?

1. Update on-policy or off-policy?
   - Is the policy we update the one that we use to make a step?
2. Use a function approximator?
   - Does our state space require a function approximator or can we directly implement a tabular solution?

Solution Categories

Dynamic Programming

- One step solutions
- Exact model is known and expected updates are used
- On or Off policy

Monte Carlo Solutions

- Full return solutions
- Model-Free
- On or Off Policy

TD Learning

- N-Step Solutions
- Model Free
- On policy (SARSA) or Off Policy (Q-Learning)

Solution Categories