Companion to Reinforcement Learning

Rohan Kumar

Contents

0 Notation

0.1 Data

$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_M \end{pmatrix}$: data point corresponding to a column vector of M features

$\overline{\boldsymbol{x}} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ ... \\ x_M \end{pmatrix}$: concatenation of 1 with the vector$\boldsymbol{x}$

$\boldsymbol{X} = \begin{pmatrix} x_{1,1} & ... & x_{1,N} \\ ... & ... & ... \\ x_{M,1} & ... & x_{M,N} \end{pmatrix}$: dataset consisting of N data points and M features

$\overline{\boldsymbol{X}} = \begin{pmatrix} 1 & ... & 1 \\ x_{1,1} & ... & x_{1,N} \\ ... & ... & ... \\ x_{M,1} & ... & x_{M,N} \end{pmatrix}$: concatenation of a vector of 1's with the matrix $\boldsymbol{X}$

$y =$: output target (regression) or label (classification)

$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ ... \\ y_N \end{pmatrix}$: vector of outputs for a dataset of N points

$\boldsymbol{x}_* =$: test input / unknown input

$\boldsymbol{y}_* =$: predicted output

$N =$: Number of data points in the dataset

$M =$: Number of a features in a data point

$\boldsymbol{w} = \begin{pmatrix} w_1 \\ w_2 \\ ... \\ w_M \end{pmatrix}$

$\boldsymbol{w}^T = (w_1, w_2, ..., w_M)$ or $(w_0, w_1, w_2, ..., w_M)$ w_0 multiplies the first entry of $\overline{\boldsymbol{x}}$ (bias)

Note: bold symbols represents a vector

1 Introduction

Reinforcement Learning is an area of machine learning inspired by behavioural psychology, concerned with how software **agents** ought to take **actions** in an **environment** so as to maximize some notion of cumulative **reward**.

2 Markov Processes

Markov processes are important to model environment dynamics and to model this we will use stochastic processes and the two important assumptions we will consider are the Markovian Assumption and the Stationary Assumption.

2.1 Unrolling the Problem

If we consider the problem of reinforcement learning where we take an action based on the state and receive a reward then we can unroll the loop into a sequence of states, actions and rewards:

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2$$

This sequence forms a stochastic process (due to some uncertainty in the dynamics of the process). This is because we don't know the underlying state of s_0 and therefore we don't know the correct action a_0 that will lead to the next optimal state.

2.2 Stochastic Processes

Processes are rarely arbitrary and very often exhibit structure. The laws of the process do not change and the short history is sufficient to predict the future.

2.2.1 Definition

We will define a stochastic process with respect to states. We let S be our set of states and we define the stochastic dynamics to be $P(s_t|s_{t-1}, ..., s_0)$ which in general expresses some conditional distribution over the current state given the past states.

2.2.2 Problem

The problem that arises with this is that if we have a process that is very long then this conditional distribution that produces the state at time step t could depend on a number of states before. Expressing a large number of previous states results in an intractable problem. If the process was infinitely long and every state depended on everything that happened before then the problem cannot be expressed.

The solution to this is to assume that the process is stationary: the dynamics do not change over time. The other assumption we need to make is the Markov assumption which is that the current state depends only on a finite history of past states.

2.3 K-order Markov Process

Here we use the Markov Assumption and assume that the last k states is sufficient. In a first-order markov process we assume that only the last state is relevant and sufficient.

By default a Markov Process refers to a first order process

$$P(s_t|s_{t-1}, s_{t-2}, ..., s_0) = P(s_t|s_{t-1})\forall t$$

and expresses the stationary assumption that

$$P(s_t|s_{t-1}) = P(s_{t'}|s_{t'-1})\forall t'$$

The stationary assumption means that the conditional distribution is going to be the same regardless of which time step we consider.

The advantage of this is we can now specify an entire process with a single concise conditional distribution

$$P(s'|s)$$

2.4 Non-Markovian and Non-Stationary Processes

If the process is not Markovian and/or not stationary then we can add new state components until the dynamics are Markovian and stationary. For example considering the dynamics in robotics we have state $< x, y, z, \theta >$, this is not stationary when velocity varies. The solution is to add velocity to the state description as $< x, y, z, \theta, \dot{x}, \dot{y}, \dot{z}, \dot{\theta} >$

The problem with this solution is that adding components to the state description to force a process to be Markovian and stationary may significantly increase computational complexity. The solution to this would be to try to find the smallest state description that is self-sufficient (Markovian and stationary).

2.5 Inference in Markov Processes

Assuming we have a Markovian Process that is stationary, we want to do inference to predict what will be the value of some future state and the reason this is important is because in a Markov decision process and more generally in reinforcement learning the goal is to select actions that will influence future states and get us into states that will have high rewards. Therefore if we can predict what the future state is going to be then we can select some good actions.

Predicting a state k time steps into the future:

$$P(s_{t+k}|s_t)$$

To compute this we perform the following which takes advantage of the chain rule in probability theory to expand this into a product to sum out all the intermediate states:

$$P(s_{t+k}|s_t) = \sum_{s_{t+1}\dots s_{t+k-1}} \prod_{i=1}^{k} P(s_{t+i}|s_{t+i-1})$$

If we have discrete states (finitely many states) then we can represent the conditional distribution as a matrix and here we use the letter T to indicate a transition matrix. So we let T be a $|S| \times |S|$ matrix representing $P(s_{t+1}|s_t)$. Then $P(s_{t+k}|s_t) = T^k$. What we're doing here is that we are essentially taking the product of that matrix k times. Complexity: $O(k|S|^3)$.

3 Markov Decision Processes

For a Markov Decision Process we will augment the Markov process with Actions and Rewards. Our goal is to select actions that will influence the future states in a good way. The choice of actions depends on the current state. The reward depends on the current state and action, this quantifies how good it is to be in a certain state and execute a certain action.

The current assumptions we need to maintain are it being a stochastic process, sequential process, fully observable states, complete model and the process to be discrete.

3.1 Definition

We define the set of states to be S and the set of actions to be A. We consider a transition model $P(s_t, s_{t-1}, a_{t-1})$ that corresponds to the stochastic process model with the addition of an action. We have the reward model $R(s_t, a_t)$ and its discount factor γ. Finally we have a horizon h, which is the time horizon/number of time steps. The goal is to find an optimal policy, a mapping from states to actions.

3.2 Rewards

The reward is a real number ($r_t \in \mathbb{R}$) that we are going to try and maximize. It is essentially a numerical signal that indicates how good the state and action is at every time step. We can express the reward function as $R(s_t, a_t) = r_t$.

Another common assumption we make here is that the reward function is stationary where $R(s_t, a_t)$ is the same at every time step ($\forall t$). This does not mean the reward is the same at every time step just the function is the same. The exception to this is the terminal reward function is often different (e.g. in a game, 0 reward at each turn but +1/-1 at the end for winning/losing).

The goal is to maximize the sum of the rewards $\sum_t R(s_t, a_t)$.

3.2.1 Discounted/Average Rewards

If process is infinite and my rewards are always positive then $\sum_t R(s_t, a_t)$ approaches infinity. This becomes an issue.

The first solution here is to consider discounted rewards. We introduce a discount factor $0 \leq \gamma \leq 1$. The idea here is that we want to discount the reward function by γ^t where t is the time step. We can represent the finite utility as $\sum_t \gamma^t R(s_t, a_t)$. The rewards further in the future will be discounted more. We can think of γ as an inflation rate of $\frac{1}{\gamma - 1}$. The intuition here is that we prefer utility sooner than later.

The second solution is to average the rewards. in some problems we don't want to discount earning a reward thousands of time steps into the future rather it should have the same reward as earning the utility now. In this case we can average the reward. This is computationally more complicated and will not currently be considered in this literature.

4 Dynamic Programming Algorithms for Solving MDPs

4.1 Policy and Policy Optimization

A policy is a formal definition for how to select actions. We will denote a policy using π. We are mapping states to actions so we can represent this as $\pi(s_t) = a_t$. The goal in reinforcement learning and markov processes is to come up with this policy π. We are going to make the assumption that we have fully observable states, so therefore we can condition the choice of action basely on the current state s_t.

We now define the expected utility as:

$$V^{\pi}(s_0) = \sum_{t=0}^{h} \gamma^t \sum_{s_t} P(s_t|s_0, \pi) R(s_t, \pi(s_t))$$

Here we have a discounted cumulative reward where we have a sum with respect to all time steps from 0 to h (horizon). Then our reward at each time step is $R(s_t, \pi(s_t))$ which depends on the current state. That state has some uncertainty in it and it really depends on the transition dynamics that are stochastic and therefore we have some expectation $P(s_t|s_0, \pi)$.

We represent the optimal policy as π^* which is the policy with the highest expected utility where the following holds:

$$V^{\pi^*}(s_0) \geq V^{\pi}(s_0) \forall \pi$$

There are several classes of algorithms for policy optimizations:

- Value iteration
- Policy iteration
- Linear programming

- Search techniques

Computation may be done offline: before the process starts or online: as the process evolves.

4.2 Value Iteration

The way this algorithm works is by optimizing / selecting actions in reverse order. To find what the best action at every step we can use dynamic programming that will work backwards and optimize the last decision then the second last decision then so on.

We can demonstrate this mathematically as follows:

We start at the last time step h.

$$V(s_h) = max_{a_h} R(s_h, a_h)$$

To optimize the last decision is easy we look at all possible actions and pick the one that yields the highest reward.

Value with one time step left:

$$V\left(s_{h-1}\right) = \max_{a_{h-1}} R\left(s_{h-1}, a_{h-1}\right) + \gamma \sum_{s_h} P(s_h \mid s_{h-1}, a_{h-1}) V\left(s_h\right)$$

Here we are taking into account the reward for the second last time step as well as the last time step and sum the rewards. We have the second last reward $\max_{a_{h-1}} R\left(s_{h-1}, a_{h-1}\right)$ plus the discount factor γ multiplied into the expectation $P(s_h \mid s_{h-1}, a_{h-1})$ with respect to what the last state will be times the value that I can earn in the last state $V(s_h)$. Now we simply select a_{h-1} to maximize this where $V(s_h)$ is already maximized.

Similarly value with two time steps left:

$$V\left(s_{h-2}\right) = \max_{a_{h-2}} R\left(s_{h-2}, a_{h-2}\right) + \gamma \sum_{s_{h-1}} P(s_{h-1} \mid s_{h-2}, a_{h-2}) V\left(s_{h-1}\right)$$

If we do this for the entire planning horizon we can generalize to Bellman's equation:

$$V(s_t) = max_{a_t} R(s_t, a_t) + \gamma \sum_{s_t+1} P(s_{t+1} \mid s_t, a_t) V(s_t + 1)$$

Now we can represent the equation to get the optimal action to construct the optimal policy is:

$$a_t^* = argmax_{a_t} R(s_t, a_t) + \gamma \sum_{s_t+1} P(s_{t+1} \mid s_t, a_t) V(s_t + 1)$$

Finite Horizon When h is finite we have a non-stationary optimal policy where the best action is different at every time step. The intuition here is that the best action varies with the amount of time left.

Infinite Horizon When h is infinite, stationary is the optimal policy. We pick the same best action at each time step. The intuition here is that the same amount of time (infinite) at each time step, therefore we pick the same best action. The main problem with this is that value iteration will do an infinite number of iterations.

The solution is that we can perform value iteration using a finite n since we are using a discount factor γ. After n time steps the rewards are scaled down by γ^n and for a large enough n, the rewards become insignificant since $\gamma^n \to 0$. For this we need to pick a large enough n, run value iteration for n steps and execute policy found at the n^{th} iteration.

4.2.1 Algorithm

Algorithm 1: Value Iteration MDP

$V_0^* \leftarrow max_a R(s,a) \forall s$;
for $t \leftarrow 1$ **to** h **do**
  $V_t^*(s) \leftarrow max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_{t-1}^* \forall s$
end
return V^*

We get the optimal policy π^*:

- $t = 0$; $\pi_0^*(s) \leftarrow argmax_a R(s,a) \forall s$
- $t > 0$; $\pi_t^*(s) \leftarrow argmax_a R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V_{t-1}^*(s') \forall s$

In this case t indicates the number of time steps to go till the end of process and π^* is non stationary.

We can express this in matrix form as follows. Let:

- R^a: $|S| \times 1$ column vector of rewards for a.
- V_t^*: $|S| \times 1$ column vector of state values.
- T^a: $|S| \times |S|$ matrix of transition probabilities for a.

Algorithm 2: Value Iteration MDP

$V_0^* \leftarrow max_a R^a$;
for $t \leftarrow 1$ **to** h **do**
  $V_t^* \leftarrow max_a R^a + \gamma T^a V_{t-1}^*$
end
return V^*

4.3 Policy Evaluation

Policy evaluation computes the value functions for a policy π. If we consider policy evaluation for an infinite horizon where we let $h \to \infty$ then $V_h^\pi \to V_\infty^\pi$ and $V_{h-1}^\pi \to V_\infty^\pi$

Then the equation for policy evaluation becomes:

$$V_{\infty}^{\pi}(s) = R\left(s, \pi_{\infty}(s)\right) + \gamma \sum_{s'} \mathrm{P}\left(s' \mid s, \pi_{\infty}(s)\right) V_{\infty}^{\pi}\left(s'\right) \forall s$$

Bellman's equation:

$$V_{\infty}^{*}(s) = \max_{a} R(s,a) + \gamma \sum_{s'} \mathrm{P}\left(s' \mid s, a\right) V_{\infty}^{*}\left(s'\right)$$

Representing the policy evaluation in matrix form where:

- R: $|S| \times 1$ column vector of state rewards for π.
- V: $|S| \times 1$ column vector of state rewards for π.
- T: $|S| \times |S|$ column vector of state rewards for π.

then we get:

$$V = R + \gamma T V$$

To find the value of the policy we solve the system of linear equations. We can do it using the following methods:

- Gaussian Elimination: $(I - \gamma T)V = R$
- Compute Inverse: $V = (I - \gamma T)^{-1} R$
- Iterative Methods
  - Value Iteration (Richardson Iteration)
  - Repeat $V \leftarrow R + \gamma T V$