Symmetries and Structure in Reinforcement Learning Problems

Dominik Meyer
dominik.meyer@tum.de

May 12, 2016

Introduction

In the Reinforcement Learning (RL) framework, the learning environment is generally described as a Markov Decision Process (MDP). It consists of a tuple of $(\mathcal{S}, \mathcal{A}, R, \gamma, P)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, that can be executed in each state, $R : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a reward function that gives reward on state transition after taking an action $a \in \mathcal{A}$, which is denoted as $R(s, a, s')$ with $s \in \mathcal{S}$ and $s' \in \mathcal{S}$ the state transited to from s. In the framework $\gamma \in [0, 1]$ is a discount factor, that discounts (gives less weight) to rewards occurring in the distant future of the current state and $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability for transiting from state s to state s' upon taking action a with probability $P(s, a, s') \in [0, 1]$.

One very important subgoal in RL is to calculate a so-called value function (VF), which is defined as the expected future return (which again is the sum of discounted rewards) starting from state s and following a fixed policy π thereafter

$$\begin{aligned} V_\pi(s) :=& \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ =& \mathbb{E}_\pi \left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + \gamma^{k+1} R_{t+k+1} \mid S_t = s \right] . \end{aligned} \tag{1}$$

Let us explain equation (1) a little bit further: First a policy π assigns to each state and action pair (s, a) a probability of taking that action in that state $\pi(s, a) \in [0, 1]$. We assume that a specific policy, serving some specific need, has already been found and provided. In general, for the whole RL problem, where we aim at solving a control problem, of course a policy will be derived from the VF, but we will leave this aspect aside for now.

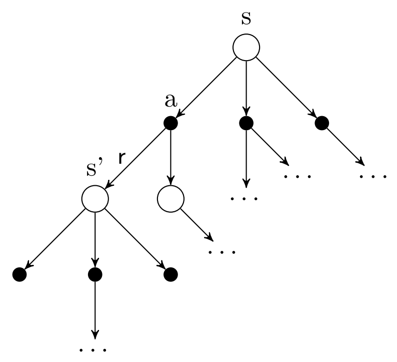
The so-called return $G_t := R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + \gamma^{k+1} R_{t+k+1}$ consists of the discounted rewards starting from state in timestep t and continuing thereafter following policy π until the end of the episode, which consists of k timesteps. We will only consider episodic tasks in this document.

As you may have noted, we denote random variables with upper case letters, like S_t and a specific realization of such a random variable with lower case letters, such as s.

The Example Problem

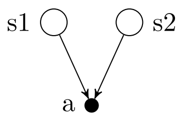
If now the reinforcement framework is to be applied to specific problems, it can happen that in certain circumstances very distinct symmetries and, resulting from that, a kind of state equivalences arises. This means, that we can go through our temporal sequence of states and arrive at a specific state, where we have the choice of a very specific action we would prefer to execute, but we could have also arrived at the point where we would be very likely to execute the same action arriving from a very similar sequence of states. These two sequences have a the common feature, that they are – how we call it from now – on *symmetric*. This means, that not only a physical symmetry, like in a two dimensional environment by rotation would arise, but any symmetry, that could be present in the problem resulting in the same sort of constellation.

Let us explain this fact, by going through how the RL agent will interact with the environment, by using simple graphs, where a white non-filled node will represent a specific state s, the solid black nodes will represent an action a and denoted on the transition line from one action to the resulting state will be the reward.



This is a very general scheme of how the situation could unfold, while starting at a state s_t in the top non-filled node and then having the possibility to choose between three actions (depicted by the solid nodes). After this, the environment will transition to one of several possible states s_{t+1} and will provide us with a reward r_t. In general, since this scheme will go on as long the episode is not yet finished, we can call state s_t just s and the state transited into s', while we also will drop the time index on the reward r. Therefore, an interaction at some timestep t can be summarized as the tuple (s, r, s'), which implicitly belongs together.

Now it could happen, that we are in two states and according to a magic oracle, we know that this single action would be beneficial for us in both of the states. This insight can come from either the fact, that the two states are in fact the same state, but at a different point in time, or that the states are equivalent with respect of gaining as much reward as possible due to symmetry in the underlying problem.



To make it even more illustrative, let us consider the game of tic-tac-toe: In this game two players play against each other and they can place alternatingly either crosses or circles onto a game board consisting of a 3×3 grid. The player which first has three of her symbols in a row (or a column, or on a diagonal) wins the round.

Naturally, in this game there exist a lot of possible symmetries and thereby equivalent states.

As a first example take the fact, that one player can play either as crosses or as circles. From this follows, that all the configurations where some pattern of crosses (and of course the circles placed by the player herself) are present on the board, while the player is playing circles, all the states where the same pattern of circles (instead of crosses) are present on the board, while the player is playing crosses. Therefore, no matter what the player is playing, all configurations with crosses and circles exchanged are symmetric to each other (in the above sense) and can be regarded as equivalent.

A second example would be, when there is just a single cross placed in the center (which is a very popular starting move), and the player places one of her circles in one of the corners. Since the whole rest of the board is uninhabitated, it makes no difference in which direction we would rotate the board, since those configurations would be equivalent (and of course additionally all the configurations, where crosses and circles are interchanged).

The question is now, do we want to account for such a symmetry in the problem, or is this really a symmetry. For the example of the tic-tac-toe game, it could be argued both ways.

For once it could be not relevant in which corner we position ourselves in, as we could rotate the whole playboard if no other field was occupied (or only the center, or all corners with the same stones, ...). We always play our strategy according to those equivalent states, since it only interests us which of these we are in, since it is not relevant, how the board is rotated in order to win.

It could be alternatively argued, that it is indeed relevant to account for the symmetries in the game, since they are not really symmetries with regard to the play strategies of our opponent. It could be, that – like usually humans have – the opponent has a very specific view on the game board and in her mind it indeed makes a difference for choosing her strategy depending on how the board is rotated, because in her mind the opponent does not recognize all the symmetries possible on the board. Therefore, the strategy depends heavily on how the board is rotated, even if the positions are totally equivalent from an abstract point of view.

However, we will assume, that we play against an opponent, who can recognize symmetries and plays a strategy, that is taking those symmetries into account. Therefore, the strategy of the opponent will always be the same, regardless of the rotation of the board towards the two players.

Set States

Naturally, the first thing that comes to mind for coping with the problem of equivalent states, would be to define a new class of states, called *set states*. These set states would each represent one equivalence class of symmetric states.

The most trivial starting position for this would be to define for each of the existiting states a new equivalence class of states q, containing only one state

$$\forall s_i \in \mathcal{S}.\ q_i := \{s_i\}\,. \tag{2}$$

We call the new set of set states then $\mathcal{Q}$ and the MDP would be defined accordingly on the tuple $(\mathcal{Q}, \mathcal{A}, R_q, \gamma, P_q)$, with $R_q : \mathcal{Q} \times \mathcal{A} \to [0,1]$ and $P_q : \mathcal{Q} \times \mathcal{A} \times \mathcal{Q} \to [0,1]$.

Now, if we know the problem, we can model the equivalence classes and assign the right states to them. This, however, is not possible in any case of application or even undesirable, since coping with many states and many symmetries can be an exhausting task.

A simple approach would be to generate all possible combinations of set states and check for each if those, in fact, represent the symmetries present in the underlying problem. This again would require knowledge about the problem and it is not clear how to check if two states are equivalent without having a model that exactly tells us that.

To get around this problem, one could again generate all possible combinations of set states and then check the transition probabilities P_q and reward probabilities R_q, if they are indeed the same (in the equivalence sense) to the original ones P and R. It is not clear on how to do this programmatically since most of the times P and R are unknowns.

Maybe one solution would be to roll out a lot of simulation runs to estimate P, R, P_q and R_q and then compare their numerical identity within a certain tolerance interval. This would still computationally not be feasible, even for small problems, since a lot of set state combinations can be generated and a lot of rollouts have to be executed in order to get a good approximation of the transition and reward probabilities.

Value Function Approximation

As seen above the approach where the equivalent states get grouped into set states is computationally very complex, if even infeasible. Therefore, very probably only suitable for very small problems or theoretical studies.

In RL, however, there exists the very successful concept of value function approximation. This was developed to cope with the so called *curse of dimensionality*, where problems with a lot of states cannot be handled by keeping a table of discrete states anymore.

The general idea here is not to write down a single value for each state in the environment, but instead try to represent the value function with a parametrized linear or non-linear function

$$\hat{V}_\theta(s) := f_\theta(s), \tag{3}$$

where θ are the weights for the parametrized function f. If now the function space is rich enough, the whole prediction problem of RL (calculating the value function) can be solved.

What does this approach help us with equivalent states? Since the approximation function is an arbitrary function, it could be specially designed to exactly reflect this fact that there are symmetries in the underlying problem. This could be handled in two ways.

The first possibility is to adapt the approximation function itself, such that it can be tailored to reflect exact the same symmetries that are there in the environment. This, however, seems again as hard as finding all the equivalence sets among the states. The advantage we gain by doing this is that we are now able to handle bigger state spaces (think of continuous state spaces with infinitely many states) and are able to use all the benefits, that are already established among the RL literature.

The second possibility is to build up a layered approximation framework, where the states are first transformed using so called features, that enable the learning algorithm to have certain guarantees on the representation of the states.

Generalizing- and Specializing Features

The question now is, what are those features? Basically, they are any additional transformation applied on top of the state representation. The transformed state is then denoted as $\phi(s_t)$, or short ϕ_t. In general those features should be diverse enough to cover the whole state space in order to enable the RL algorithm to converge to the correct solution and they should trade off between a generalization and specialization, which we will talk about in a bit.

The most popular way of doing value function approximation with features in RL, is assuming a linear approximation. This means the value function is represented as

$$\hat{V}_\theta(s) := \theta^\top \phi(s). \tag{4}$$

Here θ is a parameter vector, that linearly combines all the components of the features. The big advantage of this form of approximation is its simplicity, computational as well as theoretical. Therefore, a whole bunch of bounds in performance, learning speed and approximation quality can be derived.

The fist straight-forward way to now cope with symmetries, would be to design the features to mask the symmetries out from the learning algorithm. We will call that *generalization*. It can be illustrated again with the example of tic-tac-toe: If we now again place a cross in any of the corners with the rest of the field empty, it does not matter which corner we placed it in. We could therefore construct a *corner* feature, that exactly represents this, namely a cross in any corner has been placed. We therefore generalize from the case of a special corner to the more general notion of *corner is occupied*.

The other extreme to generalization would be *specialization*, where the feature exactly represents all the rotational symmetries as different state representations. Why could that be useful?

The specialized features could be useful in two different ways. First, it could – as we already argued – indeed make a difference in which way the game board is rotated and therefore we need all the specialized rotation information in order to derive a good play strategy. Second, the combination of generalizing and specializing features could be especially beneficial. Lets say, we provide the approximator and the learning algorithm with both types of features, then the hope is, that this algorithm is capable of adjusting the weights on each in such a way that it basically selects which set of features is relevant in the present situation. In some situations, general play is desired, in some very specialized strategies are required.

Outlook

We have established three different ways to possibly handle symmetries in underlying RL problems. First the set theoretic one, which probably is most relevant for theoretical studies, then the general tailored function approximator approach and a very specialized linear case, where generalizing and specializing features are used.

Now still a lot of investigation remains to do. First it has to be established, if indeed there exist problems where symmetry plays a central role and should be handled separately. Second, a full theoretical framework for the set state approach should be established and extensive studies should be carried out therein. And third, it should be evaluated, if the last approach of linear generalizing and specializing features cannot already handle relevant cases. It would be also interesting to see, if results from the set theoretic approach could be adapted to give results for the linear function approximation case. Still one very big issue remains without a concrete proposal for a solution strategy: detecting symmetry and identifying equivalent states. This is one vital part of all the approaches and up to now only the human expert is able to sufficiently understand complex problems in order to identify symmetries. With bigger problems an automated approach still remains a requirement.

Literature

A very accessible introduction to reinforcement learning and linear value function approximation can be found in the book “Reinforcement Learning: An Introduction” by Sutton & Barto. A deep theoretical treatment of dynamic programming and reinforcement learning can be found in the two volume book “Dynamic Programming and Optimal Control” by Bertsekas. Especially, convergence of approximated algorithms and clustering of states into set states are extensively studied here.