Reinforcement Learning Tutorial 5, Week 6 — with solutions — Reward Shaping and Gradient Monte Carlo

Pavlos Andreadis[*]

February 2024

Overview: The following tutorial questions relate to material taught in weeks 3 and 5 of the 2023-24 Reinforcement Learning course. They aim at encouraging engagement with the course material and facilitating a deeper understanding.

This week's tutorial continues with the problem from Tutorial 4 and asks us to consider the use of *reward shaping*. This is of course a toy problem in which reward shaping would not have anything to offer if we only ran our procedures a bit more. To simulate situations from more complicated MDPs where training might indeed be too slow (or not converging at all; see *value function approximation*), we are assuming procedures that have stopped prematurely. The effect of reward shaping in our toy problem below is analogous to that of applying it in a more complex problem, especially as concerns the problems, or sacrifices if you prefer, inherent in reward shaping. What is less demonstrable here are the benefits of the approach in helping converge to reasonable solutions where otherwise it might not be practically feasible.

Having hopefully received a better understanding of reward shaping, we move on to an exercise on *value function approximation* and specifically on Gradient Monte Carlo. There is something odd with the samples handed to us here, but you can still compute an updated estimate of your action-value function. We will continue on this problem in the next tutorial.

Problem 1 - Discussion: Reward Shaping

Assume the problem as described in *Problem 1 of Tutorial 4*, but where after evaluating a deterministic policy π_2 (as given in Table 1 below) (e.g. using $TD(0)$), we have the following estimation of the state-value function:

[*]with special thanks to **Ross McKenzie** for introducing a first version of Problem 2

$(s6, \rightarrow, v = 0)$	$(s7, \rightarrow, v = 0)$	$(s8)$
$(s4, \downarrow, v = 6)$	▨	$(s5, \uparrow, v = 10)$
$(s1, \rightarrow, v = 7)$	$(s2, \rightarrow, v = 8)$	$(s3, \uparrow, v = 9)$

Table 1: Suboptimal policy.

We are frustrated that our agent has not learned to "go up" when in state s_4, and decide to, instead of running the process further, apply reward shaping to the model, adding +2 reward to the state visits for states s_6, s_7, hoping that this will help the agent learn to take the shortest route from s_4.

1. What do you think would be the optimal policy for this modified MDP (with rewards for arriving at states s_6 and s_7 of +1)? Would the episodes terminate?
2. If instead of +2 to the above rewards, we instead add +1 (rewards for arriving at states s_6 and s_7 of 0), what would be the optimal policy?
3. In the above two models, would the calculated state-value function, after convergence, be representative of the original problem? Why?
4. When can reward shaping be a useful tool, and how?

Answer:

1. As going to s_8 would terminate the episode, the agent would prefer to move between states s_6 and s_7 accumulating infinite reward (for this non-discounted case). An episode running on this policy would never terminate, and computation of the value function would never converge.
2. The optimal policy would be similar to the one for the original MDP. The main difference being that for s_1 we will always go up (as opposed to both actions being equally good), and s_2 now having two equally good actions. That means, that under some optimal policies, we will end up going left from s_2, which definitely does not lead to s_8 through the shortest path. It's also possible for the optimal policy to go left from s_7 to s_6 and back to s_7 again at no cost for a discount of 1 (still achieve the optimal reward), which also does not lead to s_8 through the shortest path.
3. The optimal value state function for the modified problem will not be the same as for the original problem (though it could happen by coincidence), since the reward function has changed. If the reward translates into some real-world quantity*, then the value function after shaping will no longer be representative of the sum of those quantities. In this case, the original problem ranked saving 10 steps as equally good to reaching the goal. This no longer holds after reward shaping.

   *[*Though consider to what extent this also holds for the return, especially when discounted. If each unit of reward represented an apple, what are* $1 + \gamma * 1$ *apples?*]

4. Reward shaping is a heuristic for faster learning, and allows us to use our understanding of a problem to engineer the model such that the learning process converges faster to what we would expect to be useful behaviour (or brings it closer to learning the optimal behaviour). In problems where we are confident we can compute an optimal policy in reasonable time, reward shaping has little use (though any gains in computational time could be important, even for problems we know we can solve). One good use of reward shaping is to reduce it progressively, such that the final policy computed is for the original problem.

If you are interested in what reward transformations would not change the optimal policy (though, they will change the optimal value function, if that was important), then we refer you to Ng et al. [1999].

One thing alluded to by our problem above is how the sparsity in rewards can inhibit training. See how the evaluation of states 6 and 7 is stuck at 0, as there were no trajectories involving visits to those states and subsequent rewards.

If you want a strong real-world case for reward shaping, consider the game of Chess: it will be much faster to start converging to some reasonable behaviour if we reward good situations (e.g. having more pieces or our pieces covering the board better) rather than just the event of winning or losing the game.

We have spoken about reward shaping in a general sense: "changing the reward function from its original formulation (assumed to be a representation of what we actually want the agent to achieve) to aid learning". In practise we might not always draw an explicit distinction (but we should attempt to do so or be faced with surprising agent behaviour).

Lastly, there is a lot of work out there looking at specific formulations of reward shaping. A good starting place could be this article from Springer's Machine Learning Encyclopedia [Wiewiora, 2010].

Problem 2 - Gradient Monte Carlo

An AI controlled orchard needs to decide when to harvest its trees. To do this it measures the concentration of three chemicals in the air. Each day the orchard can choose to wait or harvest. Waiting costs one credit in operating costs while a harvest ends the process. Once a crop is harvested, packaged and sold, the orchard is told the profit or loss of that harvest. Most experts agree that the function mapping the chemical concentrations to the profit is approximately linear.

The orchard has several samples of the profits from other harvests. as seen in the table below:

Begin to approximate the function that maps the state feature vector to Q(state,

Figure 1: "An AI-controlled orchard"

Concentration of A (ppm)	Concentration of B (ppm)	Concentration of C (ppm)	Profit/Reward (credits)
4	7	1	3
10	6	0	-15
20	1	15	5
4	19	3	21

Table 2: Samples of harvest profits

harvest) using a Monte Carlo target, doing a gradient decent step on each sample (using linear function approximation). Solutions will be provided for a learning rate of 0.01 and initial weights of $\boldsymbol{w}^0 = [w_a^0, w_b^0, w_c^0] = [3, 2, 1]$, but feel free to try any values.

Do you expect our $Q(\text{state, harvest})$ function to have converged?

Answer:

Set up arbitrary weights for $t = 0$: $\boldsymbol{w}^0 = [w_a^0, w_b^0, w_c^0] = [3, 2, 1]$. (Learning rate $\alpha = 0.01$).

We are going to use linear function approximation, making the estimation of the value of harvesting for the first state (which also is the only state in the first sample):

$$\begin{aligned} Q(s_0, harvest) = [w_a^0, w_b^0, w_c^0] \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} &= w_a^0 x_a + w_b^0 x_b + w_c^0 x_c \\ &= 3 \cdot 4 + 2 \cdot 7 + 1 \cdot 1 = 27 \end{aligned}$$

The gradient at $t = 0$ will be $\nabla Q_0 \propto [x_a, x_b, x_c] = [4, 7, 1]$.

Note that each trajectory here is comprised of only a single action-state pair. For Monte Carlo, the target update will be the return after visiting the state. Here, that is simply the reward from taking the action *harvest* at that state.

After the first trajectory, we will have one sample return for state s_0 and the action *harvest*, from which we get the update:

$\boldsymbol{w}^1 = \boldsymbol{w}^0 + \alpha(3 - Q_0)[x_a, x_b, x_c]$
$= [3, 2, 1] + 0.01(3 - 27)[4, 7, 1] = [2.04, 0.32, 0.76]$

Similarly for later trajectories, we have:

$Q(s_1, harvest) = 10 \cdot 2.04 + 6 \cdot 0.32 + 0 \cdot 0.76 = 22.32$
$\boldsymbol{w}^2 = [2.04, 0.32, 0.76] + 0.01(-15 - 22.32)[10, 6, 0] = [-1.69, -1.92, 0.76]$

$Q(s_2, harvest) = 20 \cdot -1.69 + 1 \cdot -1.92 + 15 \cdot 0.76 = -24.32$
$\boldsymbol{w}^3 = [-1.69, -1.92, 0.76] + 0.01(5 + 24.32)[20, 1, 15] = [4.17, -1.63, 5.16]$

$Q(s_3, harvest) = 4 \cdot 4.17 + 19 \cdot -1.63 + 3 \cdot 5.16 = 1.19$
$\boldsymbol{w}^4 = [4.17, -1.63, 5.16] + 0.01(21 - 1.19)[4, 19, 3] = [4.96, 2.13, 5.75]$

Though we have not done so here, for linear function approximation it is generally good practise to include a feature that is always set to the value of 1. The respective weight would then be the bias term for the linear function. This will increase the space of candidate functions for your approximation of the state or action-value function.

[*There are less subtle ways in which adding a bias term might be useful. Consider a state for which all features have a value of* 0. *What would be the value of that state or state-action?*]

[*Of course, one could also consider redesigning the features. For example, with one-hot encoding. How could this help?*]

With a learning rate of $\alpha = 0.01$ and only 4 samples from other harvests (training data points), we would not expect our Q(state, harvest) function to have converged yet. We can see this as our approximated Q-values are still a long way from the targets in the updates, making the updates large in magnitude. (Although remember we do not expect the values to become exact if the true function mapping is not exactly linear, so the updates would never truly converge). We could continue to improve our function approximator by passing through the training data again (another 'epoch' in machine learning terminology).

References

P. Andreadis and S. Rakshit. Reinforcement Learning Tutorial 5, Week 6 Monte Carlo Control / TD Prediction. https://www.learn.ed.ac.uk/bbcswebdav/pid-5766616-dt-content-rid-19263909_1/xid-19263909_1, 2021.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

Eric Wiewiora. Reward shaping. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, pages 863–865. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_731. URL https://doi.org/10.1007/978-0-387-30164-8_731.