Q-FUNCTION APPROXIMATION WITH RADIAL BASIS NETWORK IN REINFORCEMENT LEARNING

Sameen Islam
University of Southampton
England, United Kingdom
si1u19@soton.ac.uk

1 PRELIMINARIES

A Markov Decision Process (MDP) is the framework under which we tackle the reinforcement learning problem. A set of states $\boldsymbol{S}$, actions $\boldsymbol{A}$ and rewards $\boldsymbol{R}$ with finite elements form the main components of the framework. At time step t, the environment has a state S_t and the agent selects action A_t based on its observation. The environment now changes to a new state S_{t+1} and the agent receives R_{t+1}. This continues for $t = 0, 1, \dots, T$ where the terminal state occurs at time T. Due to finite elements of state and reward, we can establish a probability distribution over them based on the previous state and the action taken from there shown in equation (1).

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

From present time t, the sum of future rewards until time T is the return. Thus, the agent's goal is to maximise the expected reward. Due to uncertainty in the long term future, we wish our agent to value immediate future rewards, therefore a discount rate, γ where $0 \leq \gamma \leq 1$ is applied so that even if the episode is infinitely long, the agent will receive finite reward given by equation (2).

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

The policy function $\pi(a|s)$ maps a state to a probability distribution of possible actions, determining the agent's behaviour. The value of an action a in state s under policy π is the expected return from starting at s at t taking a and following π thereafter shown by the following relationship in (3).

$$q_\pi(s, a) = E_\pi\left[G_t \mid S_t = s, A_t = a\right] \tag{3}$$

The optimal policy $\pi*$ gives the largest expected return achieveable by any π for each (s, a). With this notion we can satisfy the Bellman optimality condition which states that for any (s, a) the expected return starting from s and selecting a and following $\pi*$ thereafter is going to be the expected reward we receive from a in s resulting in R_{t+1} and the maximum expected discounted return that can be achieved from any possible next (s', a').

$$Q_{\pi*}(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q_{\pi*}(s', a')] \tag{4}$$

Q-learning iterates through policies by minimising the loss $L = Q_{\pi*}(s, a) - Q_\pi(s, a)$. A Q-Table as exemplified in figure 1 (right) represents the value function learned by the agent. It converges to optimality using a dynamic programming (DP) update equation with a learning rate α in range $[0, 1]$.

$$Q_{\pi*}(s, a) = (1 - \alpha) Q_\pi(s, a) + \alpha(R_{t+1} + \gamma \max_{a'} Q_\pi(s', a')) \tag{5}$$

Q-Learning in (5) learns $\pi*$ with the help of ϵ-greedy policy. Since it does not use its current policy to update $Q(s, a)$ after moving from s to s' but rather uses the best possible action $max_{a'} Q_\pi(s', a')$

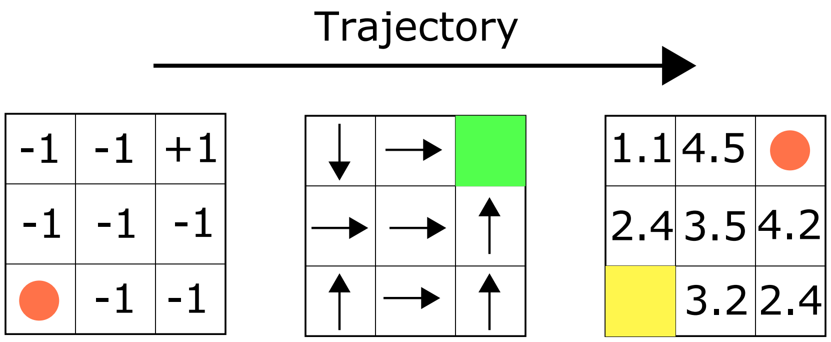


Figure 1: Agent (orange) given a goal (green) from a starting point (yellow). As the agent moves, the state changes and these snapshots together form the state-action trajectory. The agent accumulates a reward of -1 at each step until it reaches goal. The centre grid shows an example policy that could be optimal for achieving goal. Right grid shows Q-Values obtained with Q-Learning.

from the new state, it is called an off-policy method. Instead, if the agent updates its policy based on an action according to its current policy, we call it on-policy learning. SARSA is an on-policy Temporal-Difference (TD) learning algorithm which is updated after every transition in the trajectory.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{6}$$

2 Results

We apply an RBF model to a regression problem to establish a benchmark. We use the dataset created by Cortez & Silva (2008) which contains 58 features and 395 rows to predict the number of days a student will be absent from school . Features include age, sex, weekly study time, health status, whether the student takes part in extra-curricular activities and more. Figure 2a shows the target we try to predict with both least squares linear regression and RBF model. The RBF model we apply uses a Gaussian kernel with J basis functions.

$$f(x) = \sum_{j=1}^{J} \phi(\frac{||x - m_j||}{\sigma_j}) \tag{7}$$

The predictions made by the RBF model is given by multiplying the design matrix with the weight, $\boldsymbol{w}$. The closed form solution of these weights can be found with the Moore-Penrose Pseudo Inverse.

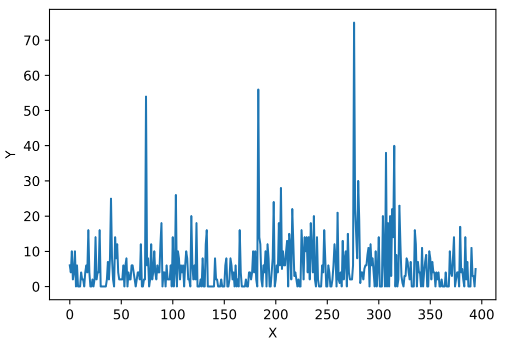
$$\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y} = \Sigma_N^{-1} \sum_{i=1}^{N} (x_i y_i) \tag{8}$$

Another alternative is to approximate the weights using gradient methods. We consider two such methods: Gradient Descent and Stochastic Gradient Descent.

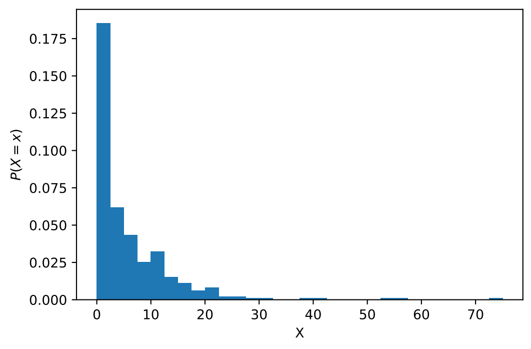
$$\mathbf{w}^{(n+1)} = \mathbf{w}^{(n)} - \eta \nabla_{\mathbf{w}} E \tag{9}$$

$$\mathbf{w}^{n+1} = \mathbf{w}^{n} - \eta(y_n - \mathbf{w}^T \mathbf{x}_n)\mathbf{x}_n \tag{10}$$

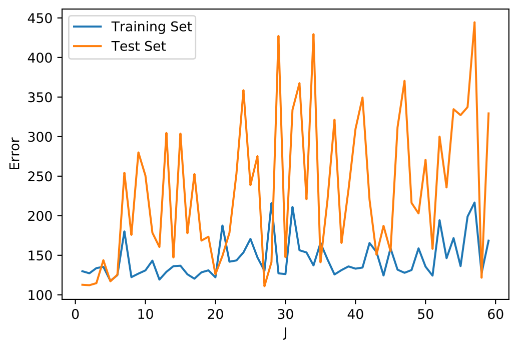
The mountain car problem as formulated by Moore (1990) is shown in figure 5. To successfully achieve the goal, an agent must learn to use momentum to drive a weakly powered car up a steep hill. The goal is indicated by a flag on the rightmost side of the hill. The state, s, consists of a vector containing continuous values of position and velocity. The finite set of action, A have three



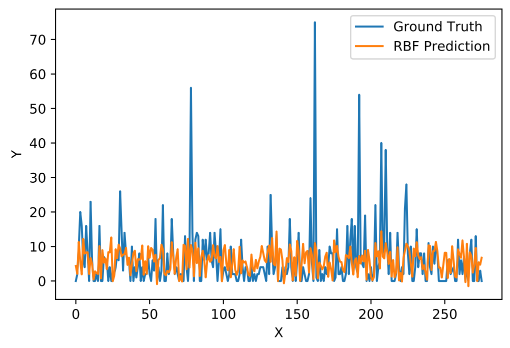
(a) Target series



(b) Target Distribution

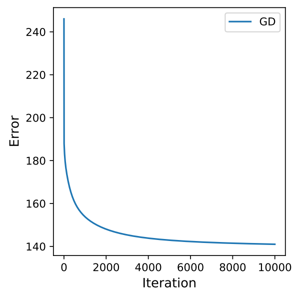


(c) Error as a function of J

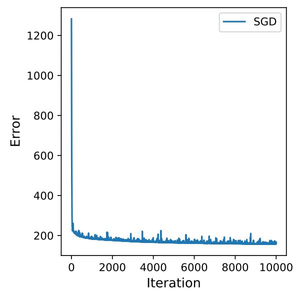


(d) Prediction with $J = 20$

Figure 2: Top row shows student absences and data distribution. Bottom row shows ground truth against model prediction.



(a) $\eta = 0.000001$



(b) $\eta = 0.0001$

Figure 3: Loss minimisation profile using Gradient Descent(GD) and Stochastic Gradient Descent (SGD).

elements: push left, push right and do nothing. To make the problem tractable, we discretise the raw position and velocity into 40 states. Using Q-Learning (5) with an ϵ-greedy exploration-exploration strategy, our agent can learn the optimal policy function $\pi*$ using a Q-Table of (s, a) pairs and the corresponding Q-Value. This results in the Q-Table being of dimension $40 \times 40 \times 4$. Visualisation of this table is shown in figure 5.

Once we have obtained the Q-Function (figure 5), we can approximate it with any universal function approximator. Since the RBF model used previously on the student performance dataset meets the criteria of being a function approximator, we can use it to approximate the optimal Q-Function.

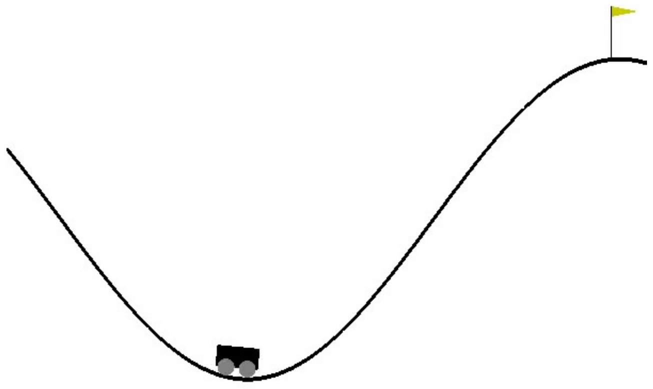


Figure 4: The mountain car control problem.

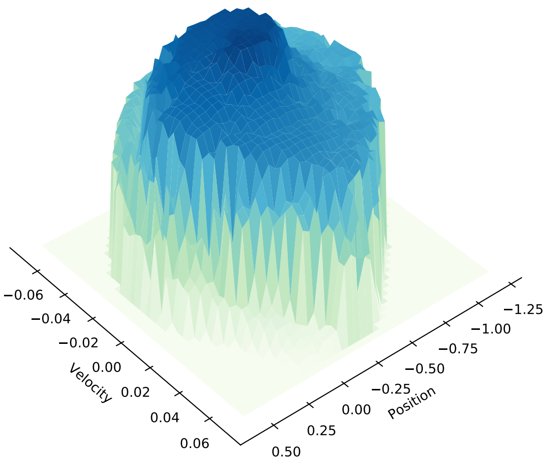


Figure 5: A visualisation of the optimal control function learned by agent with Q-Learning. Z-axis represents arbitrarily increasing Q-Values.

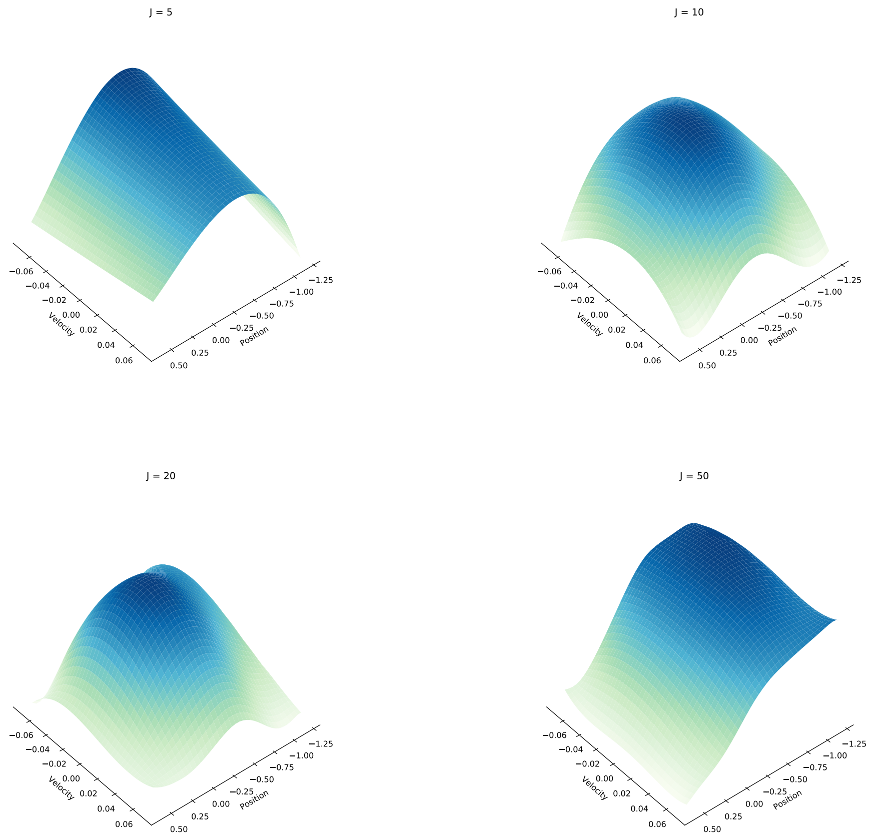


Figure 6: RBF approximation using $J = 5$, $J = 10$, $J = 20$ and $J = 50$ with Q-Learning (from top left, across).

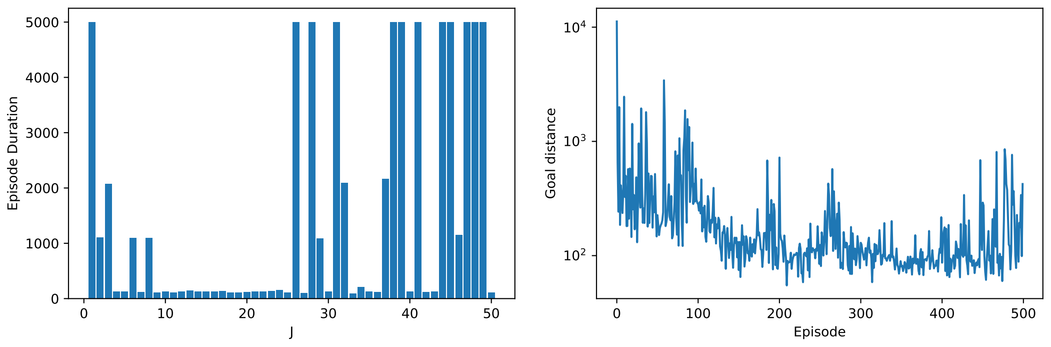


Figure 7: Episode duration is the shortest with approximately 10 to 25 basis functions (left). Policy iteration reduces agent's distance from goal as each training episode passes (right).

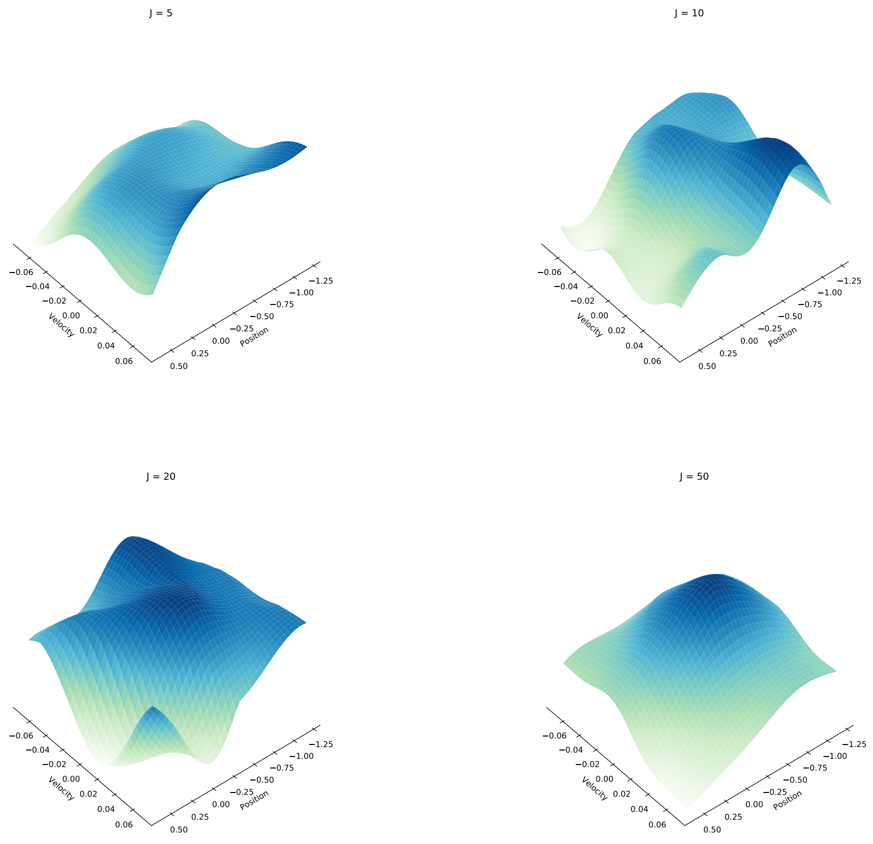


Figure 8: RBF approximation using $J = 5$, $J = 10$, $J = 20$ and $J = 50$ with on-line SARSA (from top left, across).

The basis functions are approximated using Monte Carlo sampling with random Fourier features. Episode duration can be used as a proxy for accuracy as a shorter episode means the agent was able to achieve the goal quickly. Figure 7 shows the accuracy of approximation change with number of basis functions used.

Next, we apply the RBF optimal function approximation method in an on-line learning setting with the on-policy method SARSA as discussed in (6). This allows our agent to approximate the value function during each episode and improve it based on its experience. Figure 8 shows the on-line approximation using SARSA learning. For comparison, we also perform on-line Q-Learning shown in figure 9.

Based on our experiments, we conclude that the dynamic programming based Q-Learning provides the most optimal solution. Following that, we found an RBF approximation of this off-policy method was best found with $J = 20$ basis functions. We also performed RBF approximation of on-policy SARSA and Q-Learning and found its value function was slightly worse than previous methods discussed.

3 Lift Control

Skyscrapers make heavy use of lifts. In particular, some organisations prefer to optimise their lifts in such a way so that workers have to wait for the minimum amount of time. Traditional approaches to optimise this control problem has been to rely on predetermined heuristics which are sub-optimal.

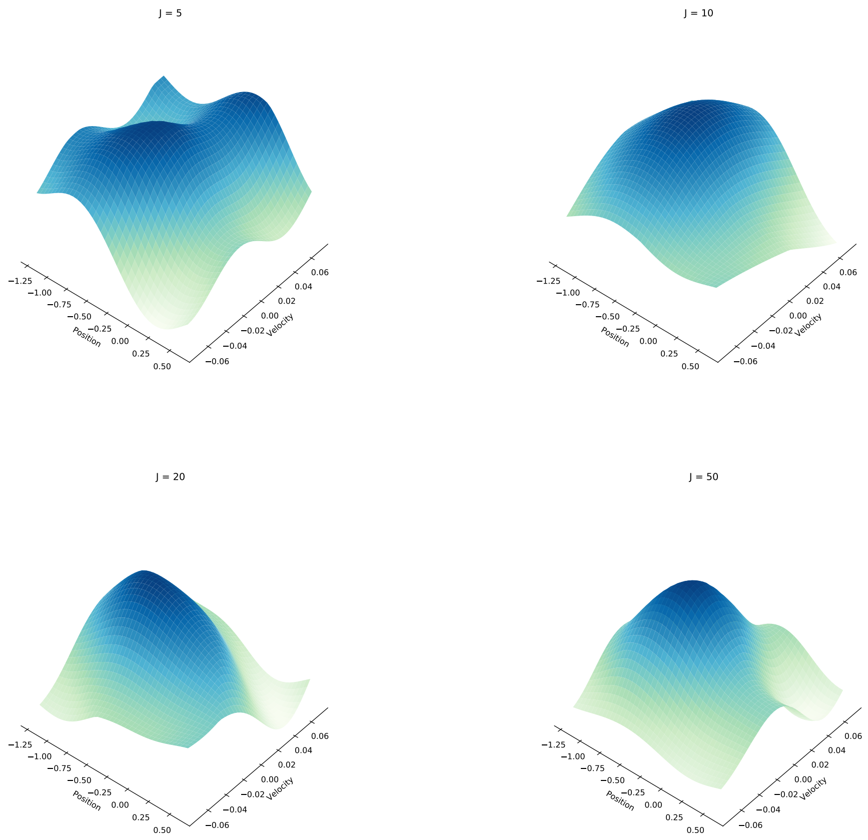


Figure 9: RBF approximation using $J = 5$, $J = 10$, $J = 20$ and $J = 50$ with on-line Q-Learning (from top left, across).

For instance, if we model each lift as an agent and the average waiting time as its reward signal then we have a task whose optimal policy can be learned using reinforcement learning.

Yuan et al. (2008) proposed a lift control system that operates over a discrete and finite state space to learn an optimal policy for the down-peak traffic pattern control problem. There are 5 floors and passengers arrive according to a discrete probability distribution. The state space is defined by

$$S = \{c_1, c_2, c_3, c_4, p, v, o\} \tag{11}$$

where $c_i \in i = 1, 2, 3, 4$ is an indicator representing a call flag (no call from ground floor), p is the disrete position (floor number), v is the discrete vertical velocity from set $\{-3, 0, 3\}$ and o is the discrete lift occupancy from set $\{0, 1, 2, 3, 4\}$. The discrete action space

$$A = \{-1, 0, 1\} \tag{12}$$

accelerates the lift down, stops and accelerates it up, respectively. The reward is given by

$$\rho(x) = -\sum_{i=1}^{4} c_i - o \tag{13}$$

thus the agent constantly receives negative rewards unless no passenger is waiting, so the optimal policy will achieve this state. The authors apply on-line Q-Learning with a discount rate $\gamma = 0.99$

and a constant learning rate $\alpha = 0.38$. ϵ-greedy policy was used for exploration with $\epsilon = 0.8$ with simulated annealing with a temperature decay rate $\tau_d = 0.998$. The Q-Learning algorithm runs for 50 episodes with stochastic passenger arrivals and the algorithm displayed a decaying waiting time over all 50 episodes, converging to 6.32 seconds. To compare against a random agent for baseline, it took 70 seconds.

In many ways, the lift control problem is similar to the mountain car control problem. Both agents in both environment try to learn an optimal policy while they encounter negative return at each step. Both have discrete, finite state action spaces and both problems were tractable using the Q-Learning algorithm as their state-action space were sufficiently small.

References

Paulo Cortez and Alice Maria Gonçalves Silva. Using data mining to predict secondary school student performance. 2008.

Andrew William Moore. Efficient memory-based learning for robot control. 1990.

Xu Yuan, Lucian Buşoniu, and Robert Babuška. Reinforcement learning for elevator control. *IFAC Proceedings Volumes*, 41(2):2212–2217, 2008. ISSN 1474-6670. doi: https://doi.org/10.3182/20080706-5-KR-1001.00373. URL `https://www.sciencedirect.com/science/article/pii/S1474667016392783`. 17th IFAC World Congress.