School of Electronics and Computer Science
University of Southampton

COMP6247(2020/21): Reinforcement and Online Learning
Final Assignment [50%]

Issue	26 April 2021
Due	21 May 2021

Objective

There are two aims in this assessed task:

35 points To learn how function approximation is helpful in reinforcement learning to approximate value functions or policies.

15 points To accurately and succinctly summarize a recent scientific publication in the subject of reinforcement learning.

Part I: Radial Basis Functions

This is a warm-up task. The method of Radial Basis Functions (RBF) consists of nonlinear basis functions (usually local functions) and learnable weights. The basis functions are set using some sensible set of rules and the weights are the only parts estimated, thus making the problem linear in parameters.

An RBF model of J basis functions is defined by

$$f(\boldsymbol{x}) \,=\, \sum_{j=1}^{J} w_j \, \phi(||\boldsymbol{x} \,-\, \mathbf{m_j}|| \,/\, \sigma_\mathbf{j})$$

The term "radial" comes from the fact that at each input $\boldsymbol{x}$, we are computing distances to "locations" $\boldsymbol{m}_j$. The nonlinear function $\phi(.)$ is usually a Gaussian function (i.e. $\phi(\alpha) \,=\, \exp(-\alpha^2)$) and σ_j offers the flexibility to control the region of influence of the basis functions. One uses some ad hoc method to set the nonlinear parts of the model (*i.e.* $\boldsymbol{m}_j$, σ_j and $\phi(.)$) and "learns" only the weights on each basis function $w_j, j = 1, ..., J$.

Given a training set, $\{\boldsymbol{x}_n, \, y_n\}_{n=1}^{N}$, the RBF model will construct an $N \times J$ "design matrix" U, whose elements u_{ij} are given by

$$u_{ij} \,=\, \phi(||\boldsymbol{x}_i \,-\, \boldsymbol{x}_j||/\sigma_j), \; i = 1, ..., N, \; j = 1, 2, ..., J.$$

The subequent estimation problem is a linear least squares problem, which could be solved, for example, by $\boldsymbol{w} \,=\, \left(U^T U\right)^{-1} U^T \boldsymbol{y}$, where $\boldsymbol{y}$ is an $N-$dimensional vector of all the targets y_n.

Adaptively estimating $\boldsymbol{w}$ by a gradient search method is the key to the use of the model in a reinforcement learning setting.

10 points To become familiar with the RBF model, solve a regression problem of your choice. Take a small dataset from the UCI repository of machine learning benchmark datasets and

0, 1	[6] on learning to control elevators.
2, 3	[4] on human level performance in Atari games
4, 5	[3] on learning to optimize a neural network
6, 7	[7] on resource allocation in computing
8, 9	[5] on drug design

Table 1: Choice of publication to review

solve it with an RBF model. To help with getting started, snippets of code solving linear and RBF models is given in Appendix A.

Solve the above problem by gradient descent, starting from a random guess of the parameters $\boldsymbol{w}$ and adapting it with random presentation of data (*i.e.* stochastic gradient descent). Draw a graph showing convergence of the error.

15 points Consider the mountain car learning control problem given in Appendix B. where the action value function is discretized and represented in a table. Implement a learning controller that uses an RBF model for approximating the value function in two different ways:

- Obtain the value function learned by the tabular discretization method (given) and build an RBF approximator to it with different numbers of basis functions and explore if a policy derived from the approximation is capable of driving the car to the top of the hill. How does the accuracy of approximation change with the number of basis functions used?
- Learn the weights of the RBF approximation on-line using Q-learning and SARSA update rules and compare the results with the tabular method.

10 points So far, we have fixed the size of the RBF network arbitrarily. Often we might require the size of a model we choose to be adaptive with respect to the complexity of the problem. A particular method to achieve such a constructive model (a growing network) is the pioneering work of John Platt on the Resource Allocation Network (RAN) [1], extended in [2] using the extended Kalman filter. Explain how Platt's RAN can be incorporated into your implementation. You need not correctly implement the algorithm. A description in the form of pseudo-code identifying the computational steps clearly will be sufficient.

Part II: Summarising a recent scientific study (15 points)

For this exercise, we will consider a set of five publications. You are asked to select the paper to study and summarize based on the last digit of your student number as shown in Table 1.

Your summary should be no more than two pages and should contain

- A succinct description of the problem domain and why reinforcement learning was thought to be the right approach.
- Clear identification of states, rewards, state transition and policy models.
- A clear description of the learning paradigm.

- The significance of the results obtained and where you think results in the paper might lead to.

You are encouraged to read background material cited in the paper, as well as look up any followup work via `scholar.google.com`, but please keep this to a minimum in the interest of time available.

Report

Write a report of no more than six pages describing the work you have done.

References

[1] Platt, J. A resource-allocating network for function interpolation, *Neural Computation* **3**(2): 213-225, 1991.

[2] Kadirkamanathan, V. and Niranjan, M. A Function Estimation Approach to Sequential Learning with Neural Networks, *Neural Computation* **5**(6): 954-975, 1993.

[3] Li, K. and Malik, J., Learning to optimize neural nets, `arXiv preprint arXiv:1703.00441`, 2017.

[4] Mnih, V., Kavukcuoglu, K., Silver, D. and others, Human-level control through deep reinforcement learning, *Nature* **518**(7540): 529-533, 2015.

[5] Popova, M., Isayev, O. and Tropsha, A., Deep reinforcement learning for de novo drug design, *Science advances*, **4**(7): eaap7885, 2018.

[6] Yuan, X., Buşoniu, L. and Babuška, R., Reinforcement learning for elevator control, *IFAC Proceedings Volumes* **41**(2): 2212-2217, 2008.

[7] Mao, H., Alizadeh, M., Menache, I. and Kandula, S., Resource management with deep reinforcement learning, *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*: 50-56, 2016.

Mahesan Niranjan April 2021

Appendix A: Regression with Linear and Radial Basis Function Models

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

rawData = # Dataset of your chice
N, pp1 = rawData.shape

# Last column is target
X = np.matrix(rawData[:,0:pp1-1])
y = np.matrix(rawData[:,pp1-1]).T
print(X.shape, y.shape)

# Solve linear regression, plot target and prediction
w = (np.linalg.inv(X.T*X)) * X.T * y
yh_lin = X*w
plt.plot(y, yh_lin, '.', Color='magenta')

# J = 20basis functions obtained by k-means clustering
# sigma set to standard deviation of entire data

from sklearn.cluster import KMeans

J = 20;
kmeans = KMeans(n_clusters=J, random_state=0).fit(X)
sig = np.std(X)

# Construct design matrix
U = np.zeros((N,J))
for i in range(N):
    for j in range(J):
       U[i][j] = np.linalg.norm(X[i] - kmeans.cluster_centers_[j])

# Solve RBF model, predict and plot
w = np.dot((np.linalg.inv(np.dot(U.T,U))), U.T) * y
yh_rbf = np.dot(U,w)
plt.plot(y, yh_rbf, '.', Color='cyan')

print(np.linalg.norm(y-yh_lin), np.linalg.norm(y-yh_rbf))
```

Appendix B: Mountain Car Control Problem, Discrete States

```
import gym
import numpy as np
env_name = "MountainCar-v0"
env = gym.make(env_name)
obs = env.reset()
env.render()

# Some initializations
#
n_states = 40
episodes = 10
initial_lr = 1.0
min_lr = 0.005
gamma = 0.99
max_stps = 300
epsilon = 0.05
env = env.unwrapped
env.seed()
np.random.seed(0)

# Quantize the states
#
def discretization(env, obs):
    env_low = env.observation_space.low
    env_high = env.observation_space.high
    env_den = (env_high - env_low) / n_states
    pos_den = env_den[0]
    vel_den = env_den[1]
    pos_high = env_high[0]
    pos_low = env_low[0]
    vel_high = env_high[1]
    vel_low = env_low[1]
    pos_scaled = int((obs[0] - pos_low) / pos_den)
    vel_scaled = int((obs[1] - vel_low) / vel_den)
    return pos_scaled, vel_scaled

q_table = np.zeros((n_states, n_states, env.action_space.n))
total_steps = 0

for episode in range(episodes):
    print("Episode:", episode)
    obs = env.reset()
    total_reward = 0
    alpha = max(min_lr, initial_lr*(gamma**(episode//100)))
    steps = 0
    while True:
        env.render()
        pos, vel = discretization(env, obs)
        if np.random.uniform(low=0, high=1) < epsilon:
            a = np.random.choice(env.action_space.n)
        else:
            a = np.argmax(q_table[pos][vel])
        obs, reward, terminate,_ = env.step(a)
        total_reward += abs(obs[0]+0.5)
        pos_, vel_ = discretization(env, obs)

        # Q function update
        #
        q_table[pos][vel][a] = (1-alpha)*q_table[pos][vel][a] + alpha*(reward+gamma*np.max(q_table[pos_][vel_]))
        steps += 1
        if terminate:
            break

while True:
    env.render()
```