School of Electronics and Computer Science
University of Southampton

COMP6247(2018/19): Reinforcement and Online Learning Lab 2 [10%]

Issue	21 February 2019
Due	11 March 2019

The lab work may be done in any programming language, but Python is recommended. Snippets of code are given to help you get started; these are best viewed as pseudo-code and not as full working programmes.

Labs form part of the assessment for this module. This is worth 10% of the total assessment. Please make sure you understand what is expected during the timetabled lab session and complete the work in your own time.

The aim of **Lab Two** is to learn how function approximation is helpful in reinforcement learning to approximate value functions or policies. We will use the method of Radial Basis Functions (RBF) which consist of nonlinear basis functions (usually local functions) and learnable weights. Thus the estimation problem is in fact 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.

1. 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 solve it with an RBF model. To help with getting started, snippets of code solving linear and RBF models is given in Appendix A. The dataset `housing.data` is available in the module notes pages.

2. 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.

3. Consider the mountain car learning control problem considered in `Lab One` where the action value function was 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 in `Lab One`; build an RBF approximator to it with different numbers of basis functions and see 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 either a Q-learning or SARSA update rule and compare the results with the tabular method.

Report

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

Appendix A: Regression with Linear and Radial Basis Function Models

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
rawData = np.genfromtxt('housing.data')
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))
```