Implementation Report-3
Offloading Decision Process using Reinforcement Learning with Neural Network (RL with NN)

Aditya Khune

February 6, 2015

Contents

Abstract

In this implementation I have used Neural Network with Reinforcement Learning to create a decision engine prototype for the offloading process. Reinforcement learning is learning by interacting with an environment.

I have developed a Neural Network code in Python and have demonstrated results produced by the prototype of Reinforcement Learning offloading app engine. I have explained how the offloading decision process can benefit from Reinforcement Learning and also presented different scenarios where it can be applicable in [1].

1 Mechanism

Consider an application for which we want to train the Q function for various instances of following parameters:

- Bandwidth = Speed Low, Speed Normal, Speed High
- WiFi = available, not available
- Data transfered = Data Small, Data Medium, Data Big
- CPU instance = CPU Low, CPU Normal, CPU High

We need to train our Q function in all these conditions. In each such instance our smartphone will have three actions to choose from which are

- Local Processing
- Offload on Local Servers
- Offload on Remote Servers

After choosing any of these actions, the smartphone will record the battery units consumed during the processing of the application. The reinforcements will be assigned depending upon the battery consumption. The action which gave us best performance will be the one which consumed least battery units. So we are minimizing our battery units consumption here.

A function called Q function stores the reinforcement values for each case it encounters, I have explained about it in my previous report [1]. In this implementation my main task was to train the Q function using Neural Network. Some more mathematics about this Q function which is used in RL algorithm is shown here:

1.1 Q Function

The state-action value function is a function of both state and action and its value is a prediction of the expected sum of future reinforcements. We will call the state-action value function Q.

$$Q(s_t, a_t) \approx \sum_{k=0}^{\infty} r_{t+k+1}$$

here s_t = state, a_t = actions, and r_t = reinforcements received. This is usually formulated as the least squares objective:

$$\text{Minimize E} \left(\sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t) \right)^2$$

2 Implementation

I have explained the basic idea behind using Reinforcement Learning in the offloading decision process in detail in [1]. In this section we will see some code and the plots showing some relevant results.

2.1 Code

I have developed a Python code to demonstrate this algorithm. There are three important functions that I am using in this implementation which asre as follows:

```
def reinforcement(s,sn):
    if sn[0] < 4 and sn[1] == -1:
        r = 0
    elif sn[0] > 3 and sn[0] < 7 and sn[1] == 0:
        r = 0
    elif sn[0] > 6 and sn[1] == 1:
        r = 0
    else:
        r = -1
    return r

def initialState():
    initialStates = 0
    process = -1.0
    return np.array([initialStates, process])

def nextState(s,a):
    s = copy.copy(s)
    if s[0] < 10:
        s[0] += 1 #location
        s[1] = a
    return s
```

```
validActions = (-1,0,1)
```

The 'intialState()' function is where we define the starting point of our user, this state of smartphone can contain parameters like bandwidth, data requirement, CPU instance and others. The 'reinforcement(s,sn)' function gives reinforcements for the ideal situations in which parameters are suitable for either offloading or local processing. In the 'validActions' we can see three choices which are as follows:

- -1 for Local processing
- 0 for Offloading on Local Servers
- 1 for Offloading on Remote Servers

2.2 Results

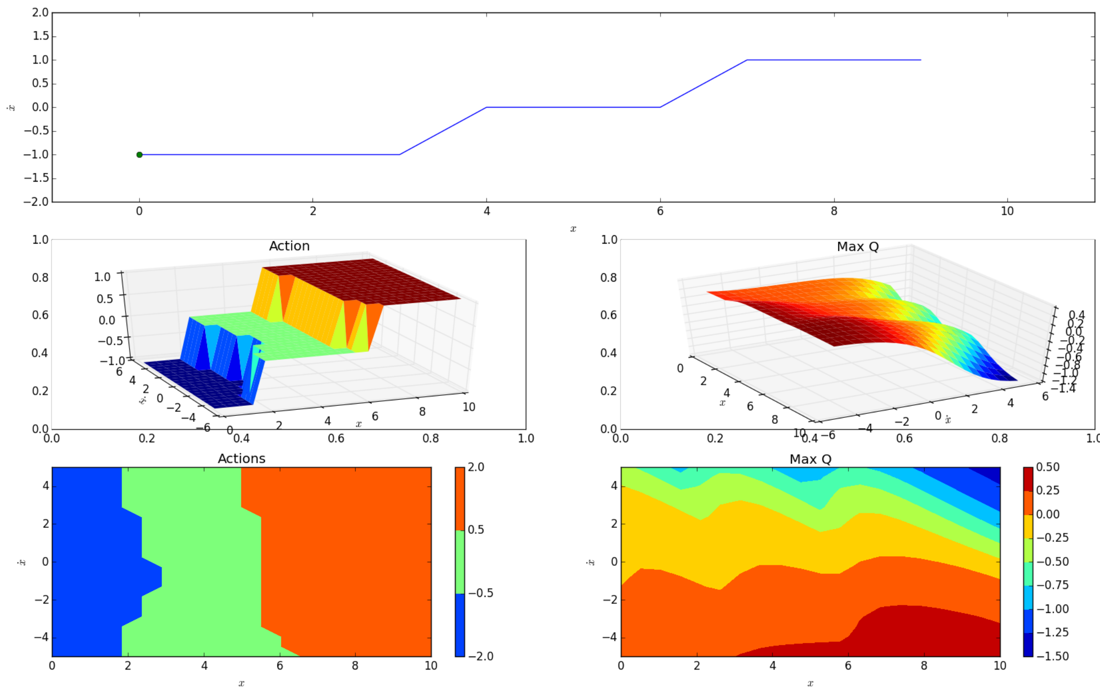

Figure 1: Best Action: Local Processing

In the figure 1 we can see the surface plot of our trained Q function. For these set of results I have customized my Neural Network with no. of hidden layers as (nh) = 5, run for trials (nTrials) = 100 and Steps per trial (nStepsPerTrial) = 10.

in the first plot on $x-axis$ we have different locations where the devices is in and location point varies from $0-10$; on $y-axis$ I have plotted the actions recommended by the Q function depending upon the best scenario to save the battery power. So we can see that for location 0, 2 and 3 Local processing is favored by our Q function. When the user moves to different locations between $4-6$ the Q functions choose to Offload the processing on Local servers whereas for locations $6-10$ we should Offload on remote servers. This decision is based on various parameters values present in that location such as 'bandwidth available'.

In the 'Actions' plot we can see 3-D plot with location and Bandwidth parameters. In the 'Max Q' plot we can see the maximum values that our Q function has for various locations, the Red part is where we got maximum Reinforcement values.

References

[1] A. Khune, "Implementation report-2 using reinforcement learning to create offloading decision engine," Jan 2015.

[2] A. Khune, "Report on implementation (offloading decision engine app for mobile cloud applications)," Dec 2014.

[3] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?," *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[4] H. R. Flores Macario and S. Srirama, "Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning," in *Proceeding of the fourth ACM workshop on Mobile cloud computing and services*, pp. 9–16, ACM, 2013.