CS786: Computational Cognitive Science
Assignment 1

Nayan Das
18111044
M.Tech,CSE

ANN for Boolean function

January 28, 2019

0.1 (a) Takiing any three variable Boolean function as input and genarating the truth table

```
In [2]: import numpy as np
        truth_table=[]
        for p in False, True:
            for q in False, True:
                for r in False, True:
                    truth_table.append([int(p),int(q), int(r)])
```

0.2 b)Generating five boolean function

```
In [3]: l=len(truth_table)
        y_or=[]
        y_and=[]
        y_xor=[]
        y_nand=[]
        y_nor=[]
        y_list=[]
        for i in range(l):
            yt1=truth_table[i][0] or truth_table[i][1]or truth_table[i][2]
            yt2=truth_table[i][0] and truth_table[i][1] and truth_table[i][2]
            yt3=truth_table[i][0] ^ truth_table[i][1] ^ truth_table[i][2]
            yt4=not yt2
            yt5=not yt1
            y_or.append(yt1)
            y_and.append(yt2)
            y_xor.append(yt3)
            y_nand.append(yt4)
            y_nor.append(yt5)
        y_or=np.array(y_or)
        y_and=np.array(y_and)
        y_xor=np.array(y_xor)
        y_nand=np.array(y_nand)
        y_nor=np.array(y_nor)
        y_list.append(y_or)
        y_list.append(y_and)
        y_list.append(y_xor)
```

```
        y_list.append(y_nand)
        y_list.append(y_nor)
        fn_list=["OR","AND","XOR","NAND","NOR"]
```

0.3 Different Activation Function

```
In [6]: def sigmoid(x):
            return 1.0/(1.0 + np.exp(-x))

        def sigmoid_prime(x):
            return sigmoid(x)*(1.0-sigmoid(x))

        def tanh(x):
            return np.tanh(x)

        def tanh_prime(x):
            return 1.0 - x**2
```

Neural Network Implementation

```
In [9]: class NeuralNetwork:

            def __init__(self, layers, activation='tanh'):
                if activation == 'sigmoid':
                    self.activation = sigmoid
                    self.activation_prime = sigmoid_prime
                elif activation == 'tanh':
                    self.activation = tanh
                    self.activation_prime = tanh_prime

                self.weights = []
                for i in range(1, len(layers) - 1):
                    r = 2*np.random.random((layers[i-1] + 1, layers[i] + 1)) -1
                    self.weights.append(r)
                r = 2*np.random.random( (layers[i] + 1, layers[i+1])) - 1
                self.weights.append(r)

            def fit(self, X, y, learning_rate=0.2, epochs=100000):
                ones = np.atleast_2d(np.ones(X.shape[0]))
                X = np.concatenate((ones.T, X), axis=1)

                for k in range(epochs):
                    #if k % 10000 == 0: print('epochs:', k)

                    i = np.random.randint(X.shape[0])
                    a = [X[i]]

                    for l in range(len(self.weights)):
```

```
                dot_value = np.dot(a[l], self.weights[l])
                activation = self.activation(dot_value)
                a.append(activation)
            # output layer
            error = y[i] - a[-1]
            deltas = [error * self.activation_prime(a[-1])]

            for l in range(len(a) - 2, 0, -1):
                deltas.append(deltas[-1].dot(self.weights[l].T)*self.activation_prime(a
```
```
            deltas.reverse()
            for i in range(len(self.weights)):
                layer = np.atleast_2d(a[i])
                delta = np.atleast_2d(deltas[i])
                self.weights[i] += learning_rate * layer.T.dot(delta)

    def predict(self, x):
        a = np.concatenate((np.ones(1).T, np.array(x)))
        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return a
```

0.4 c)d) Traning and Verifying five boolean function

In [10]:
```
X=np.array(truth_table)
nn = NeuralNetwork([3,3,1])
for i in range(len(y_list)):
    nn.fit(X, y_list[i])
    print("Function Name",fn_list[i])
    for e in X:
        p=nn.predict(e)
        if p > 0.50:
            print(e,p,1)
        else:
            print(e,p,0)
```

```
Function Name OR
[0 0 0] [7.09817703e-06] 0
[0 0 1] [0.99733063] 1
[0 1 0] [0.996947] 1
[0 1 1] [0.99951977] 1
[1 0 0] [0.99711943] 1
```

```
[1 0 1] [0.99962003] 1
[1 1 0] [0.99955787] 1
[1 1 1] [0.99972434] 1
Function Name AND
[0 0 0] [0.0001482] 0
[0 0 1] [0.00015319] 0
[0 1 0] [7.24907019e-05] 0
[0 1 1] [0.00047451] 0
[1 0 0] [0.00017896] 0
[1 0 1] [0.00064648] 0
[1 1 0] [0.00035616] 0
[1 1 1] [0.99313259] 1
Function Name XOR
[0 0 0] [0.02054847] 0
[0 0 1] [0.99561807] 1
[0 1 0] [0.99535562] 1
[0 1 1] [0.00284612] 0
[1 0 0] [0.99554909] 1
[1 0 1] [0.00238422] 0
[1 1 0] [-0.00134827] 0
[1 1 1] [0.99163479] 1
Function Name NAND
[0 0 0] [0.99921373] 1
[0 0 1] [0.99996664] 1
[0 1 0] [0.9999641] 1
[0 1 1] [0.99909302] 1
[1 0 0] [0.9999661] 1
[1 0 1] [0.99896553] 1
[1 1 0] [0.9990262] 1
[1 1 1] [1.34080823e-05] 0
Function Name NOR
[0 0 0] [0.99578573] 1
[0 0 1] [-0.00042567] 0
[0 1 0] [0.00035315] 0
[0 1 1] [4.27031609e-05] 0
[1 0 0] [0.00027225] 0
[1 0 1] [0.00023193] 0
[1 1 0] [-0.00045874] 0
[1 1 1] [0.00467176] 0
```

In [ ]:

In [ ]:

Qlearning

January 28, 2019

0.1 ****(a) Generated a random instance of the frozen lake scenario given two inputs - the size of the lake (N)=4 assuming its square, and the number of holes (M)=4****

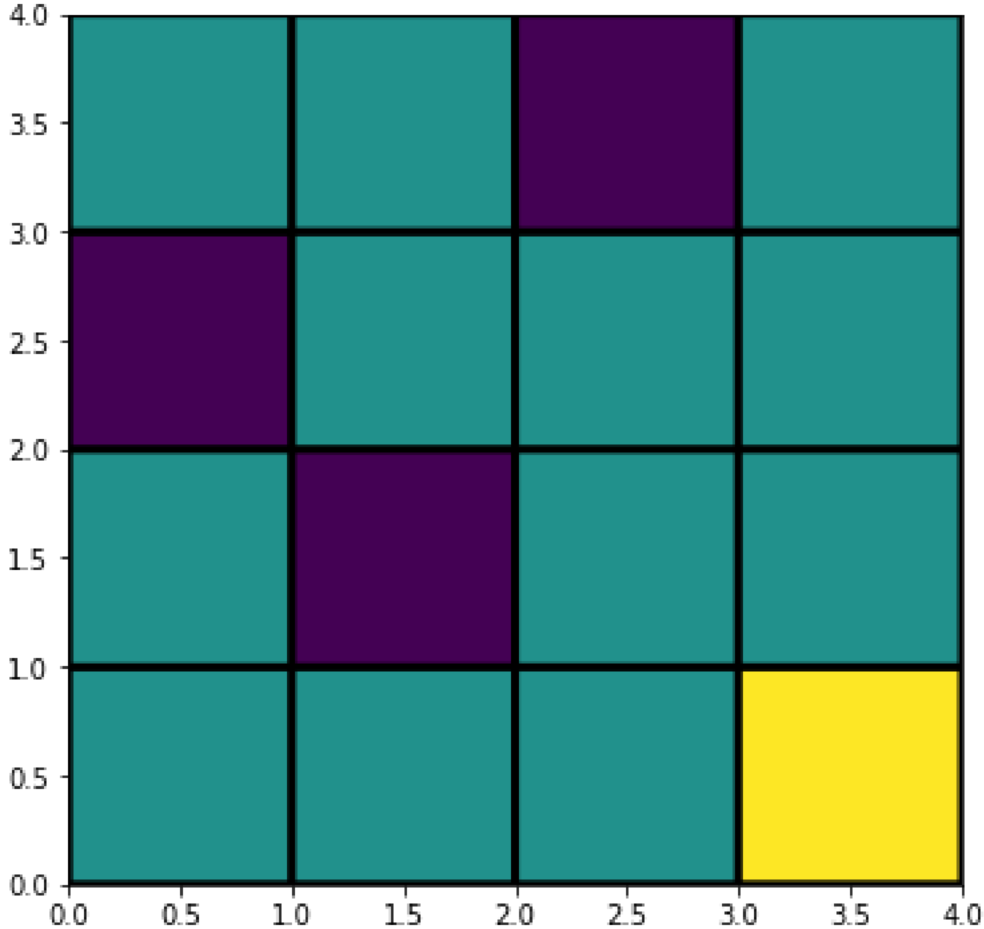
```
In [2]: import numpy as np
        import random
        import matplotlib.pyplot as plt
```

```
In [3]: def generate_Reward_table(n,m):
            #n=4
            #m=4
            R=np.zeros((n,n))
            #print(R)
            i=np.random.choice(n, m,replace=False)
            j=np.random.choice(n, m,replace=False)
            #print(i,j)

            for k in range(len(i)):
                i1=i[k]
                i2=j[k]
                R[i1][i2]=-100

            R[n-1][n-1]=100
            R[0][0]=0
            #print(R)
            F_R=R.flatten()
            #print(F_R)
            q_learning_table = np.zeros([n*n+1,4])
            return R,F_R,q_learning_table
```

```
In [4]: R,F_R,q_learning_table=generate_Reward_table(4,4)
        print(R)
        plt.figure(figsize=(6,6))
        plt.pcolor(R[::-1],edgecolors='k', linewidths=3)
        plt.show()
```

```
[[   0.    0. -100.    0.]
 [-100.    0.    0.    0.]
```

```
 [   0. -100.    0.    0.]
 [   0.    0.    0.  100.]]
```



0.2 ****Function to define available action****

```
In [5]: def available_action(curr,n):
            moves=[]
            if (curr%n)!=1:
                #moves.append(curr-1)
                moves.append(0)
            if (curr%n)!=0:
                #moves.append(curr+1)
                moves.append(1)
            if curr-n>0:
                #moves.append(curr-n)
                moves.append(2)
            if curr+n<=(n**2):
```

```
            #moves.append(curr+n)
            moves.append(3)
        return moves
```

0.3 Calculate Next State and reward

```
In [6]: def next_state_and_reward(curr_state,action,F_R):
            #print("inside next state",curr_state,action)
            if action==0:
                next_st=curr_state-1
            if action==1:
                next_st=curr_state+1
            if action==2:
                next_st=curr_state-n
            if action==3:
                next_st=curr_state+n

            reward=F_R[next_st-1]
            #print(i,j,reward)
            return next_st,reward
```

0.4 Get Sample Action

```
In [7]: def sample_next_action(available_state_range):
            next_action=int(np.random.choice(available_state_range,1))
            return next_action
```

0.5 b) Qlearning Function

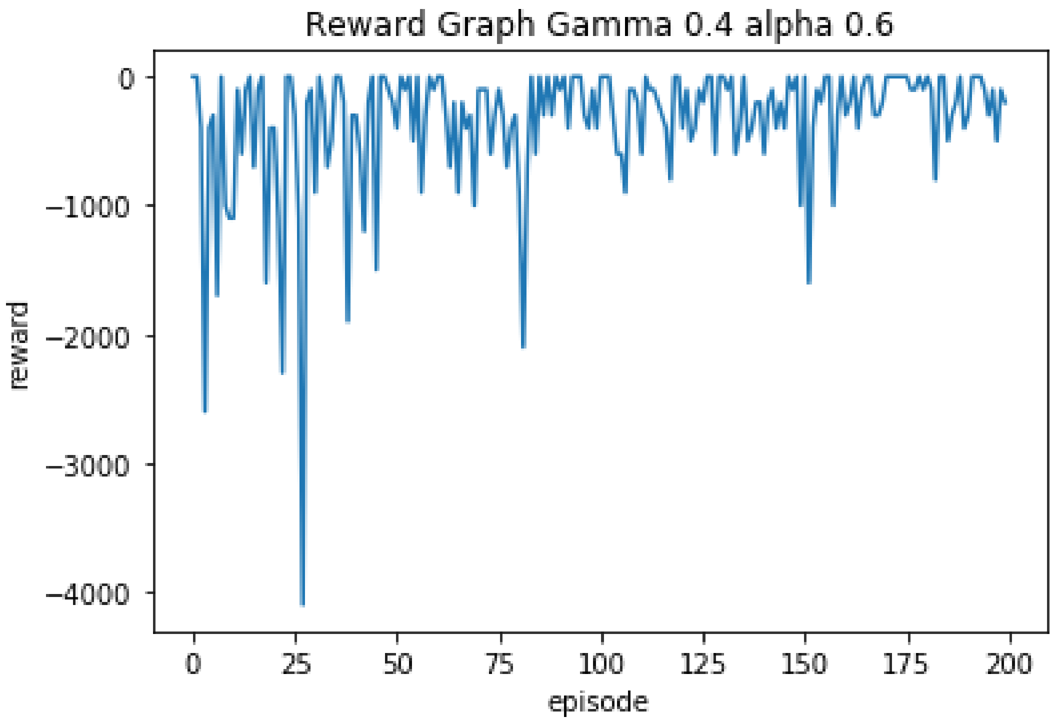
```
In [53]: def QLearning(n,F_R,gamma,alpha,epsilon,q_learning_table,ran):
             state=1
             rlist = []

             for i in range(ran):
                 rAll=0
                 state=1
                 while state <n*n:
                     a=0
                     available_act=available_action(state,n)
                     if random.uniform(0,1) > epsilon:
                         max_v=0
                         for x in available_act:
                             #print(x)
                             if q_learning_table[state][x] >=max_v:
                                 max_v=q_learning_table[state][x]
                                 a=x
                     else:
```

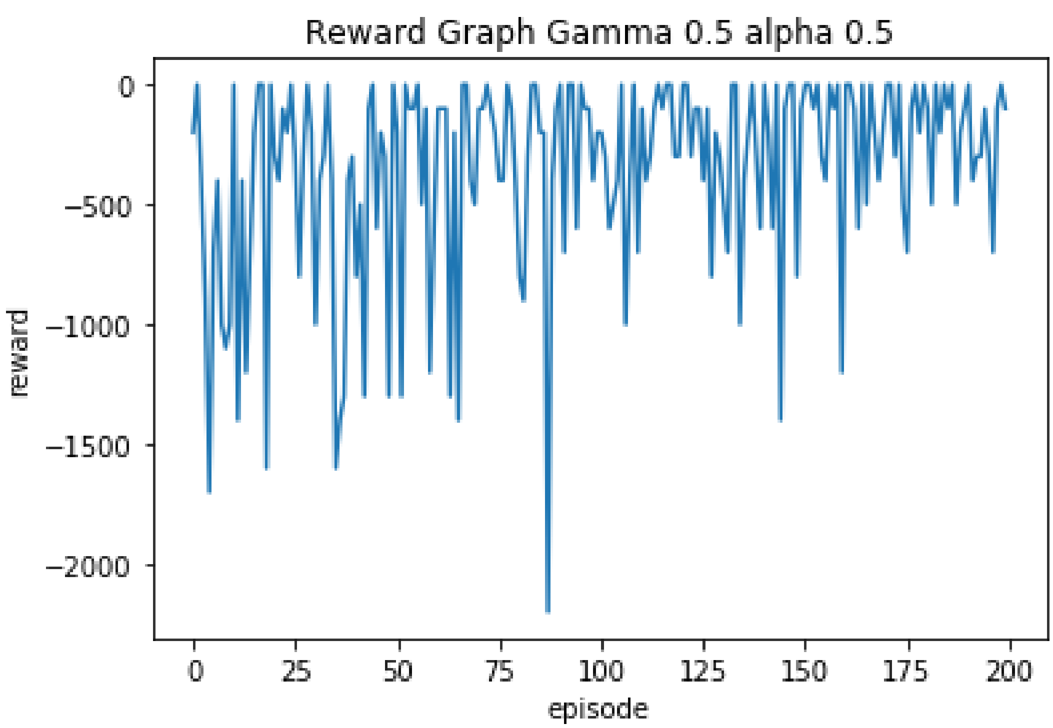
```
                    a=sample_next_action(available_act)


                next_st,reward=next_state_and_reward(state,a,F_R)
                q_learning_table[state,a] = (1-alpha)*q_learning_table[state,a] + (alpha 
                state=next_st
                rAll += reward
                #print(state)
            epsilon = min_epsilon + (max_epsilon - min_epsilon)*np.exp(-decay_rate*i)
#           if i%200 == 0:
#               print('Trial {0}; reward : {1}'.format(i, rAll))
            rlist.append(rAll)
        #print(q_learning_table)
        return q_learning_table,rlist
```

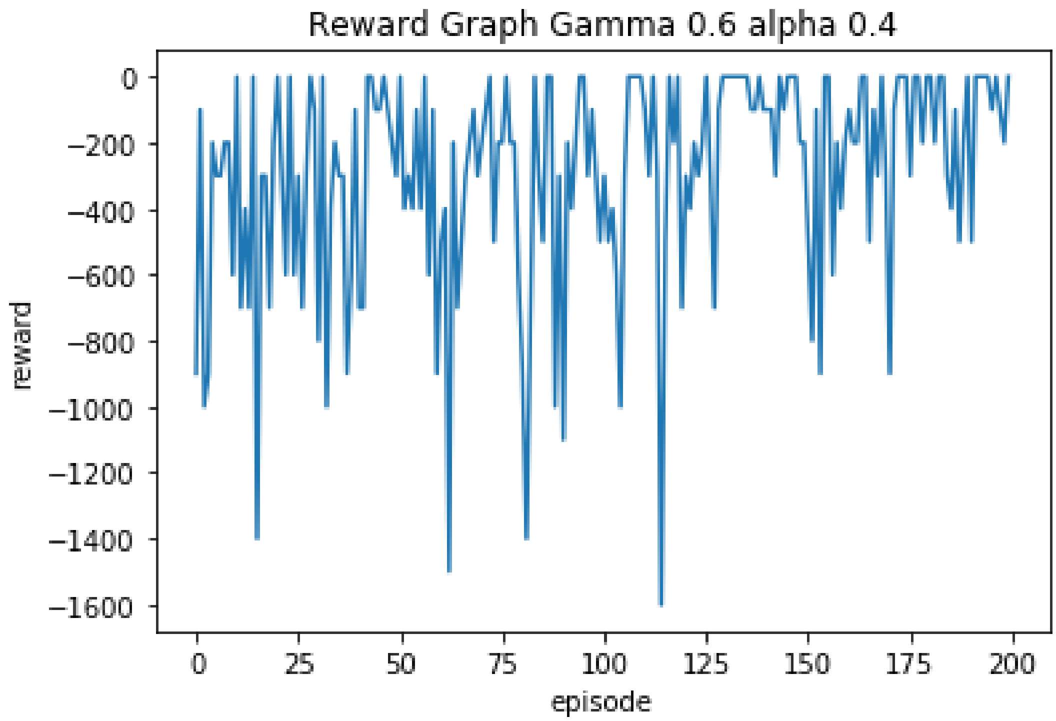
0.6 Plotting Reward with respect to episode for different parameter

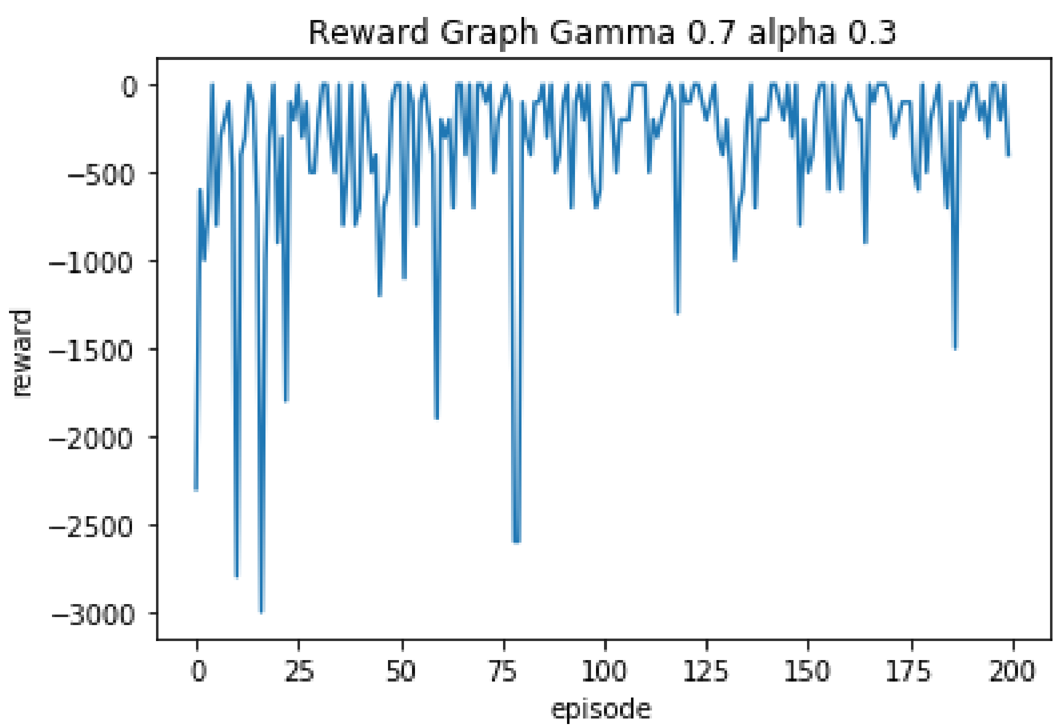
we Can see below the effect of changing alpha value and gamma.aplpha value represent learing rate so small alpha it will take more time to converge. Gamma is the value of future reward. If it is equal to one, the agent values future reward as current reward. when it is reduced it will give discount to the future reward.

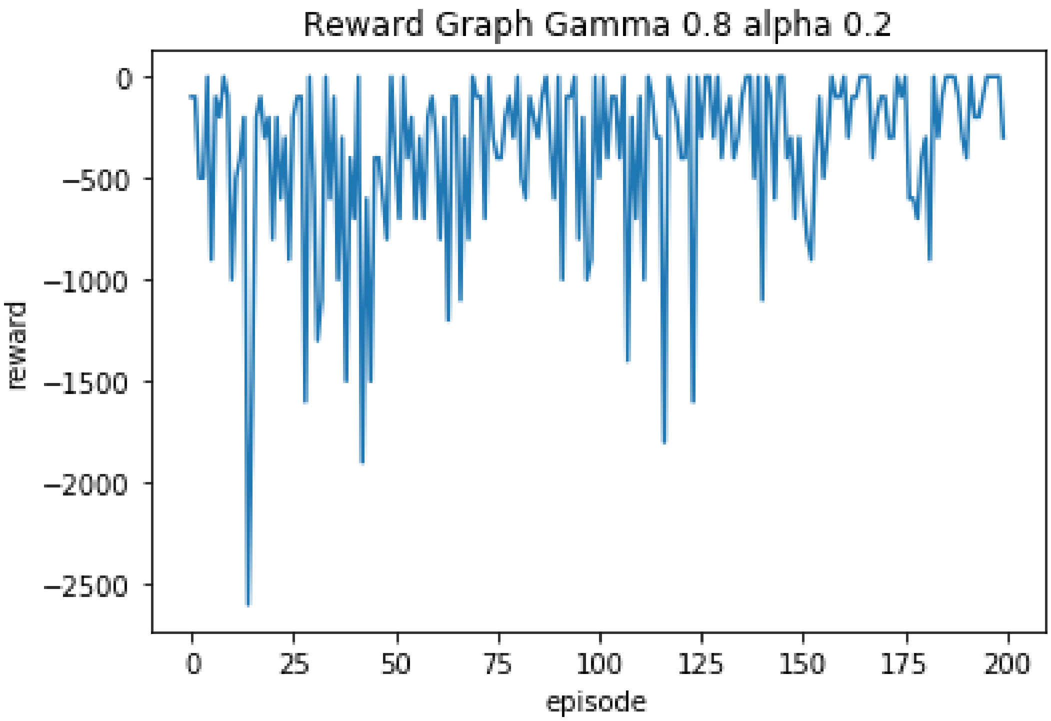
```
In [55]: gamma_l=[0.4,0.5,0.6,0.7,0.8]
         alpha_l=[0.6,0.5,0.4,0.3,0.2]
         epsilon = 1.0
         decay_rate = 0.005
         max_epsilon=1.0
         min_epsilon=0.01
         x_axis=range(200)
         n=4
         ran=200
         for i in range(5):
             gamma=gamma_l[i]
             alpha=alpha_l[i]
             q_learning_table,rlist=QLearning(n,F_R,gamma,alpha,epsilon,q_learning_table,ran)
             plt.plot(x_axis, rlist, label = "r list")
             plt.xlabel('episode')
             plt.ylabel('reward')
             plt.title('Reward Graph Gamma '+str(gamma)+ " alpha "+str(alpha))
             plt.show()
         #rlist
```


Reward Graph Gamma 0.4 alpha 0.6
0
−1000
−2000
−3000
−4000
reward
0
25
50
75
100
125
150
175
200
episode


Reward Graph Gamma 0.5 alpha 0.5
0
−500
−1000
−1500
−2000
reward
0
25
50
75
100
125
150
175
200
episode


Reward Graph Gamma 0.6 alpha 0.4
reward
episode

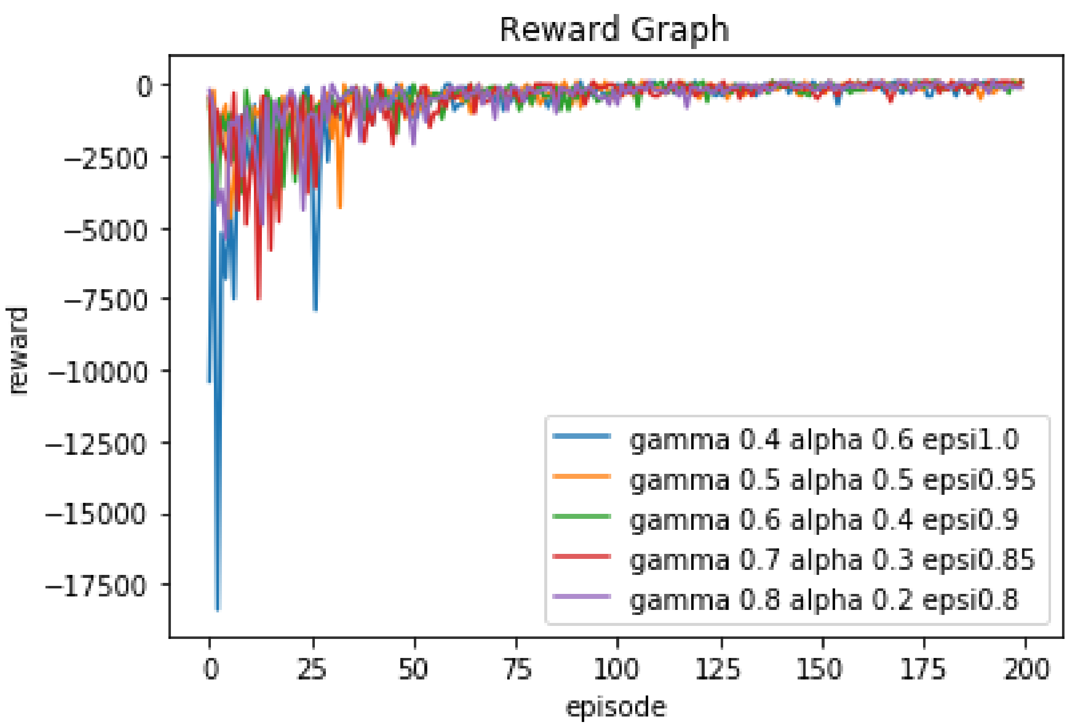

Reward Graph Gamma 0.7 alpha 0.3
reward
episode

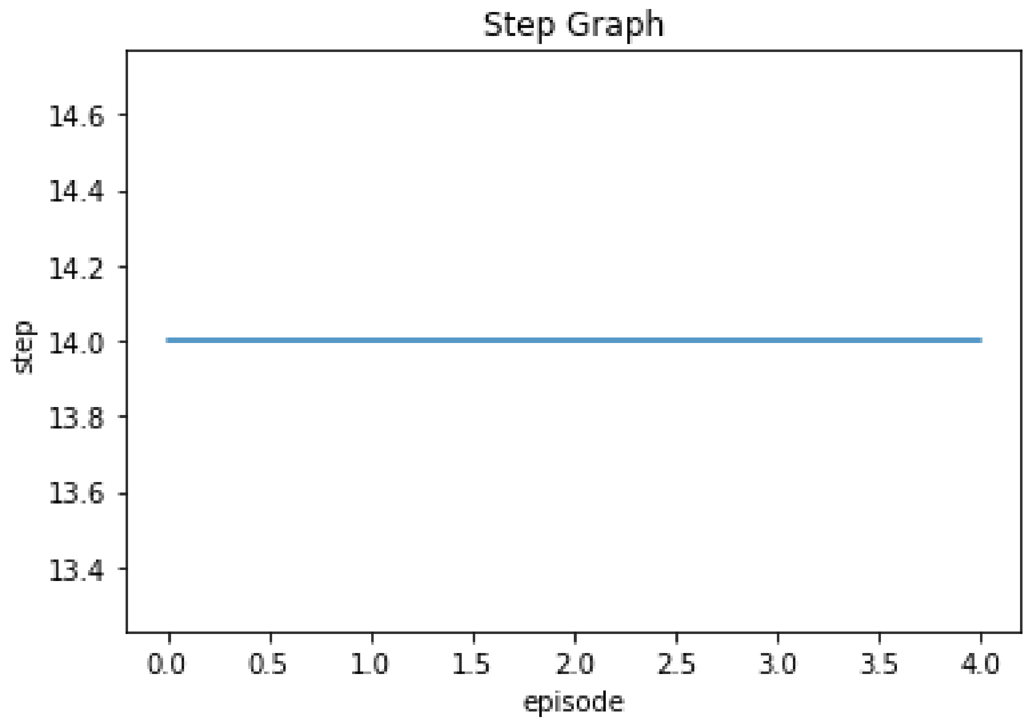



```
In [56]: def calculate_step(n,F_R,q_learning_table):
             #for episode in range(5):
             state=1
             step=0
             while state <n*n:
                     available_act=available_action(state,n)
                     max_v=0
                     for x in available_act:
                         if q_learning_table[state][x] >=max_v:
                             max_v=q_learning_table[state][x]
                             a=x
                     next_st,reward=next_state_and_reward(state,a,F_R)
                     #print(next_st)
                     step+=1
                     state = next_st
                 #print("Steps",step)
             return step
```

```
In [61]: step_l=[]
         epsilon_l=[1.0,0.95,0.9,0.85,0.8]
         x_axis1=range(200)
         ran=200
         for i in range(5):
             gamma=gamma_l[i]
```

```
        alpha=alpha_l[i]
        epsilon=epsilon_l[i]
        q_learning_table,rlist=QLearning(n,F_R,gamma,alpha,epsilon,q_learning_table,ran)
        step=calculate_step(n,F_R,q_learning_table)
        step_l.append(step)
        plt.plot(x_axis1, rlist, label = "gamma "+ str(gamma)+ " alpha "+str(alpha) +" eps
        plt.xlabel('episode')
        plt.ylabel('reward')
        plt.title('Reward Graph')
    plt.legend()
    plt.show()
    x_axis=range(5)
    plt.plot(x_axis, step_l, label = "step")
    plt.xlabel('episode')
    plt.ylabel('step')
    plt.title('Step Graph')
    plt.show()
```



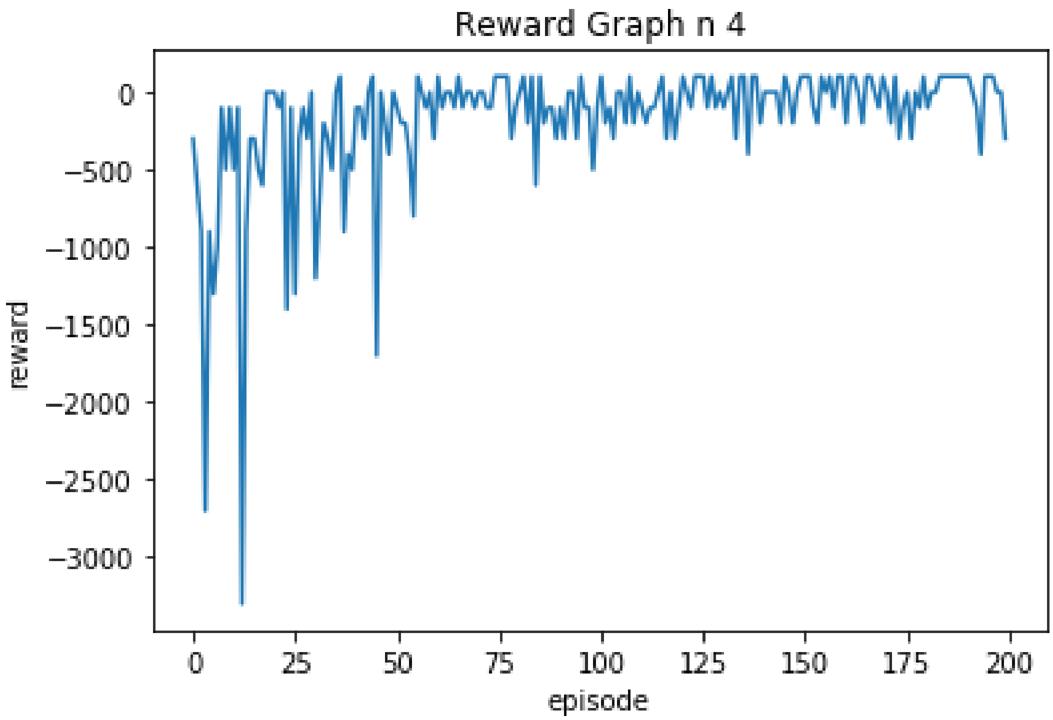


1 # Plotting Reward for different value of n

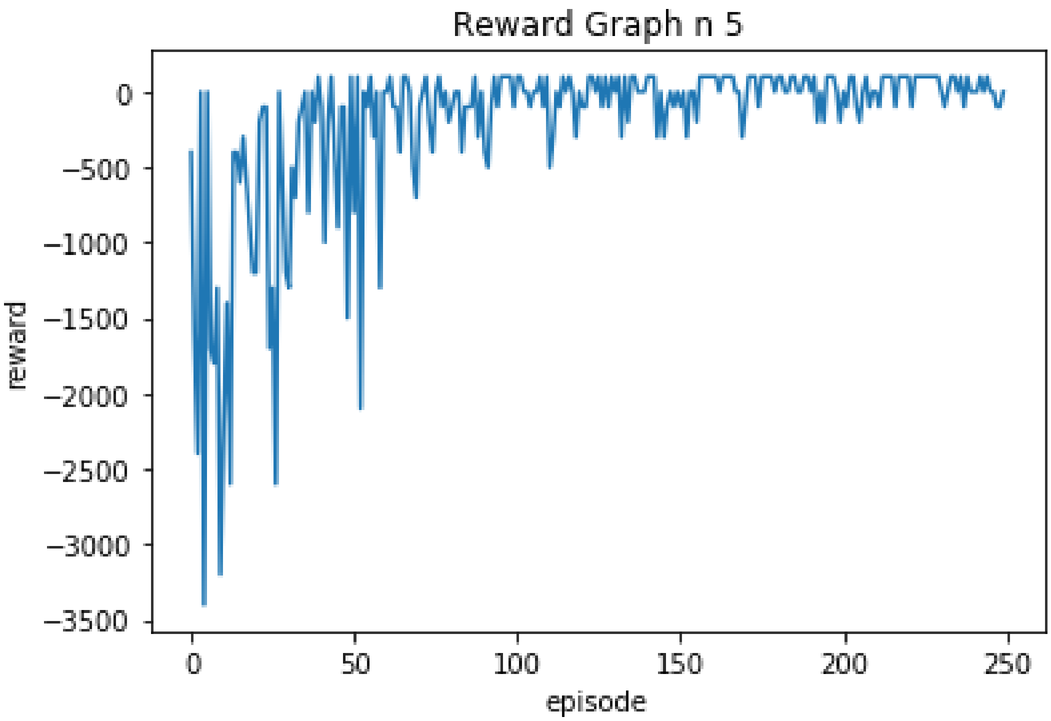
we can see the effect of differnt n and m in below graph

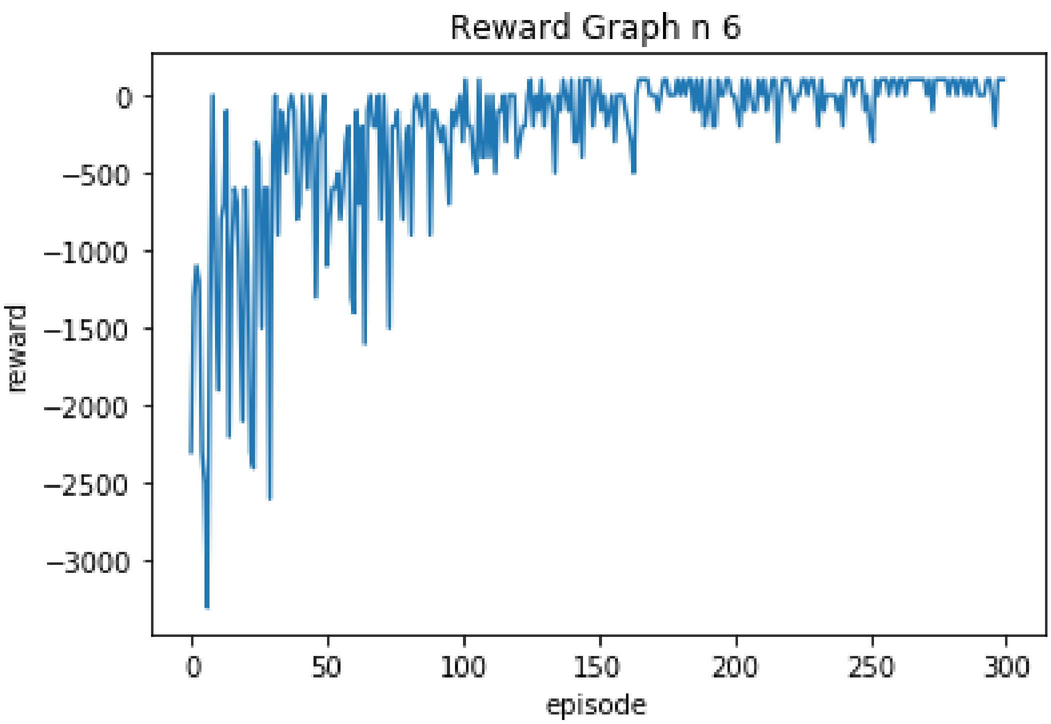
```
In [64]: gamma=0.5
         alpha=0.5
         epsilon = 1.0
         decay_rate = 0.005
         max_epsilon=1.0
         min_epsilon=0.01
         for n in range(4,9):
             R,F_R,q_learning_table=generate_Reward_table(n,n)
         #     print(R)
         #     plt.figure(figsize=(6,6))
         #     plt.pcolor(R[::-1],edgecolors='k', linewidths=3)
         #     plt.show()
             ran=n*50
             x_axis=range(n*50)
             q_learning_table,rlist=QLearning(n,F_R,gamma,alpha,epsilon,q_learning_table,ran)
             plt.plot(x_axis, rlist, label = "r list")
             plt.xlabel('episode')
             plt.ylabel('reward')
             plt.title('Reward Graph n '+str(n))
```

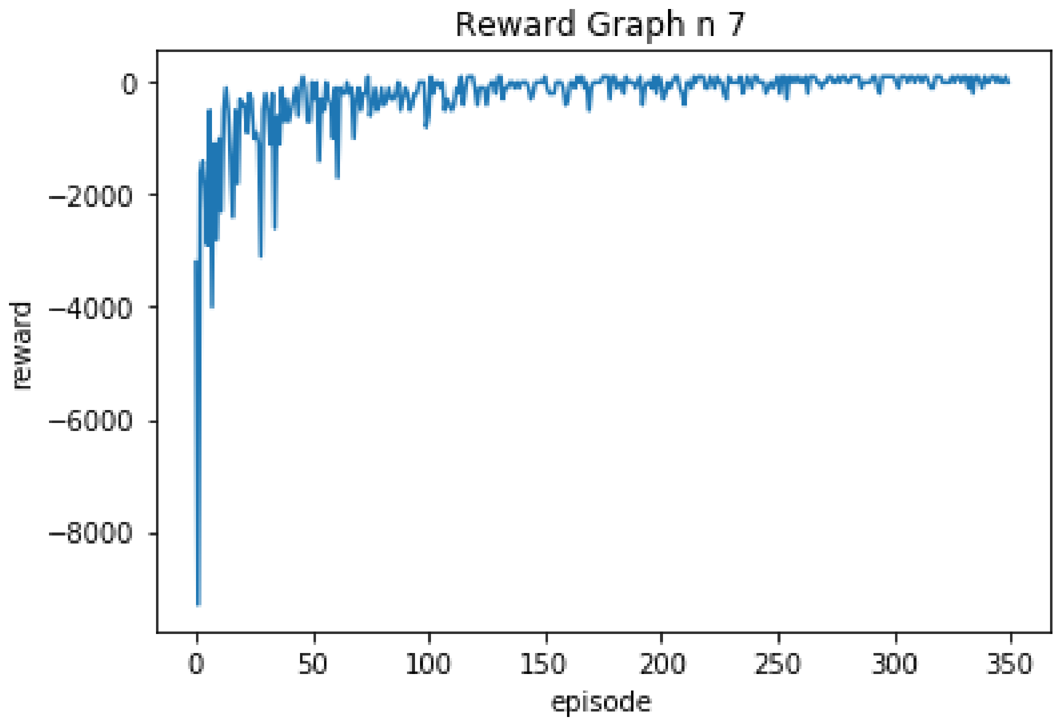
```
plt.show()
```

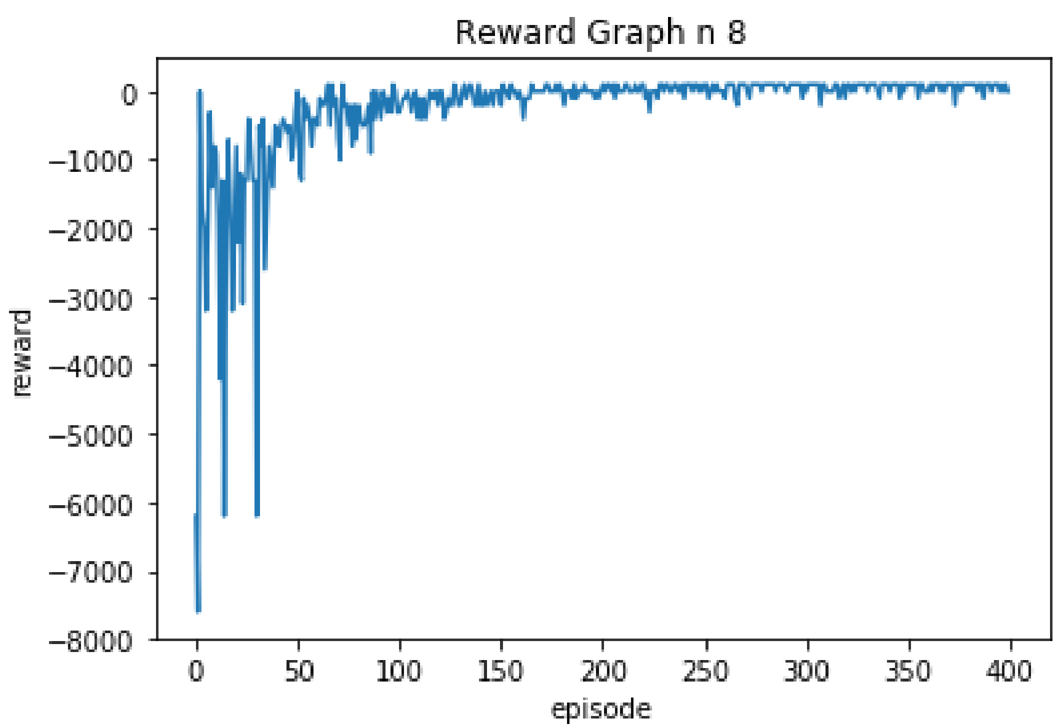


Reward Graph n 4
0
−500
−1000
−1500
−2000
−2500
−3000
reward
0
25
50
75
100
125
150
175
200
episode


Reward Graph n 5
reward
0
−500
−1000
−1500
−2000
−2500
−3000
−3500
0
50
100
150
200
250
episode


Reward Graph n 6
reward
0
−500
−1000
−1500
−2000
−2500
−3000
0
50
100
150
200
250
300
episode


Reward Graph n 7
reward
0
−2000
−4000
−6000
−8000
episode
0
50
100
150
200
250
300
350


Reward Graph n 8
reward
0
−1000
−2000
−3000
−4000
−5000
−6000
−7000
−8000
episode
0
50
100
150
200
250
300
350
400

```
In [ ]:
```

Rulkov

January 29, 2019

0.1 Rulkov map

```
In [49]: def funX(x,y,alpha):
             x_n1=0
             if x <= 0:
                 x_n1=(alpha/(1-x))+y
             elif x > 0 and x < alpha+y:
                 x_n1=alpha+y
             elif x >= alpha+y:
                 x_n1=-1
             return x_n1
```

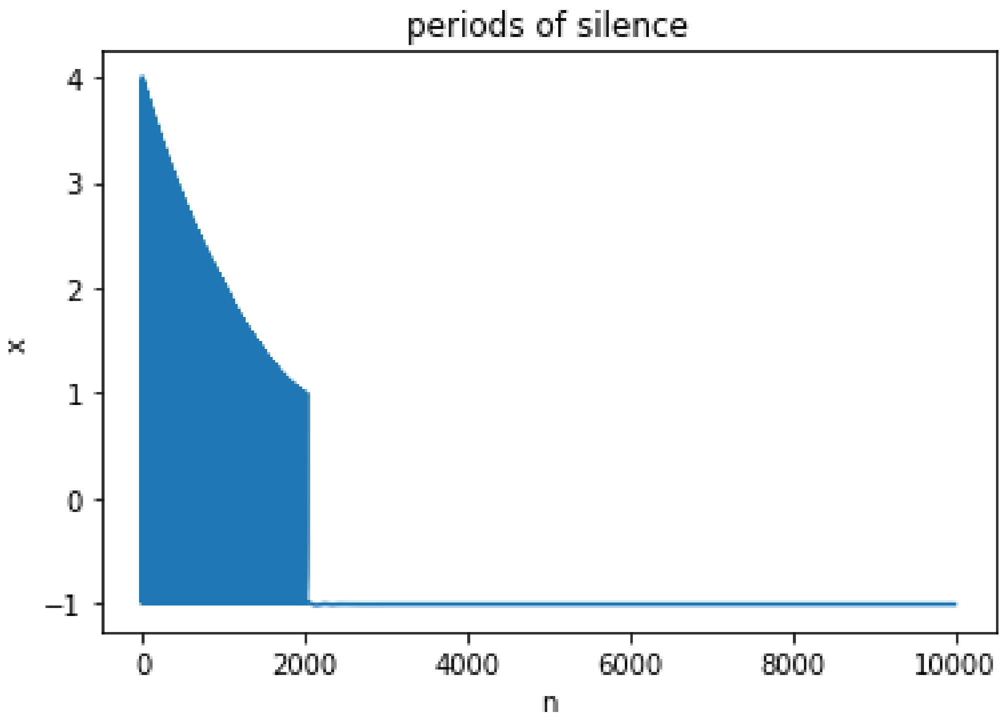
```
In [50]: def funY(x,y,mu,sigma):
             return (y-mu*(x+1)+mu*sigma)
```

0.2 Function to generate spike

```
In [65]: import matplotlib.pyplot as plt
         def generate_spike(alpha,mu,sigma,title):
             x_axis=range(10000)
             l=[]
             x=0
             y=0
             for i in range(10000):
                 x_t=funX(x,y,alpha)
                 x=x_t
                 l.append(x)
                 y_t=funY(x,y,mu,sigma)
                 y=y_t
             plt.plot(x_axis, l, label = "spike")
             plt.xlabel('n')
             plt.ylabel('x')
             plt.title(title)
             plt.show()
```

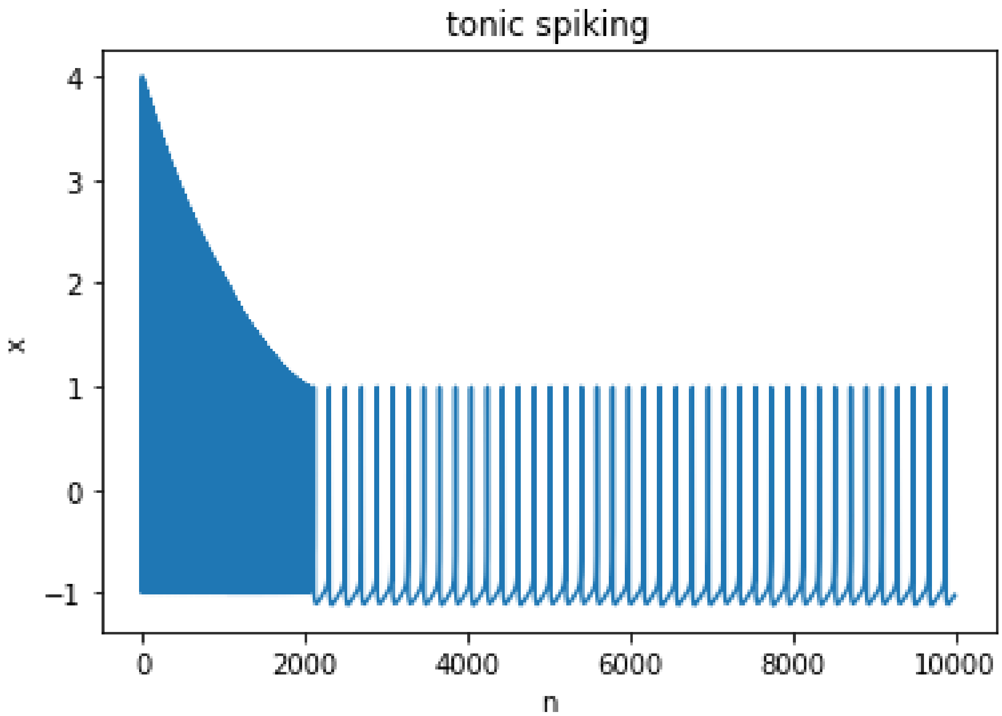
0.3 a) Periods of Silence Plot

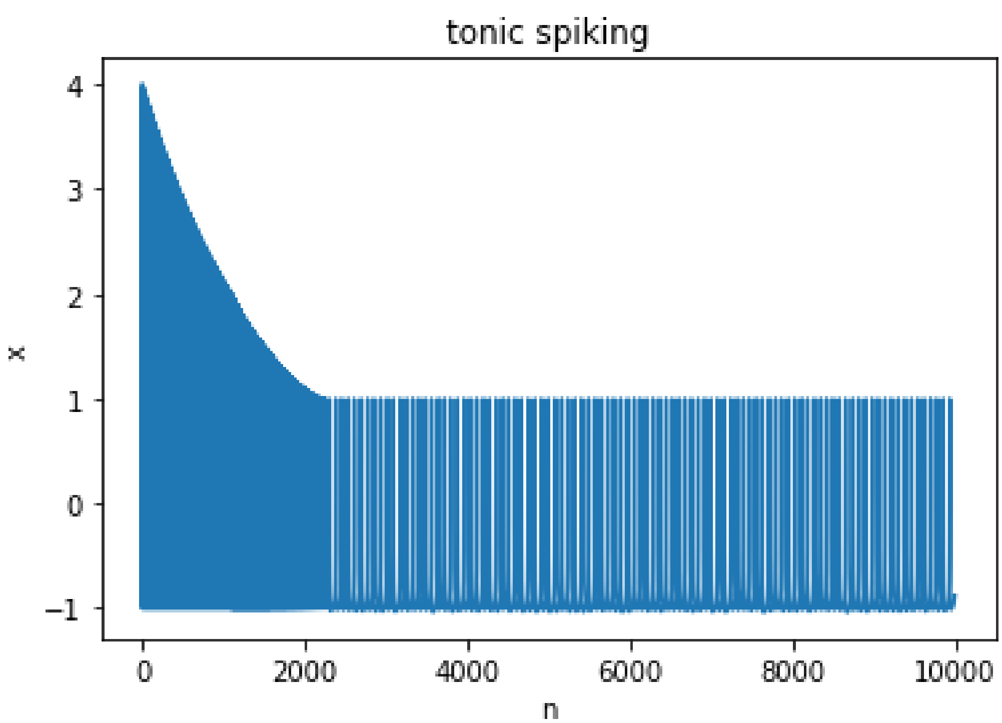
```
In [67]: title="periods of silence"
         alpha=4
         mu=0.001
         sigma=-0.01
         generate_spike(alpha,mu,sigma,title)
```



0.4 b) Tonic spiking plot

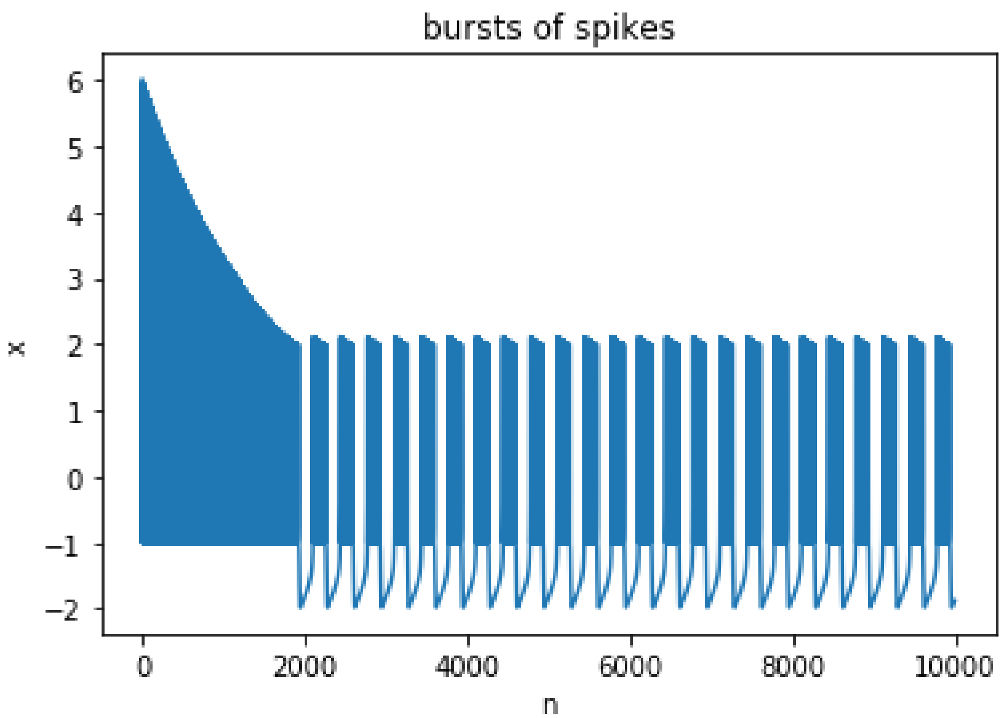
```
In [71]: title="tonic spiking"
         alpha=4
         mu=0.001
         sigma=0.01
         generate_spike(alpha,mu,sigma,title)
         sigma=0.1
         generate_spike(alpha,mu,sigma,title)
```

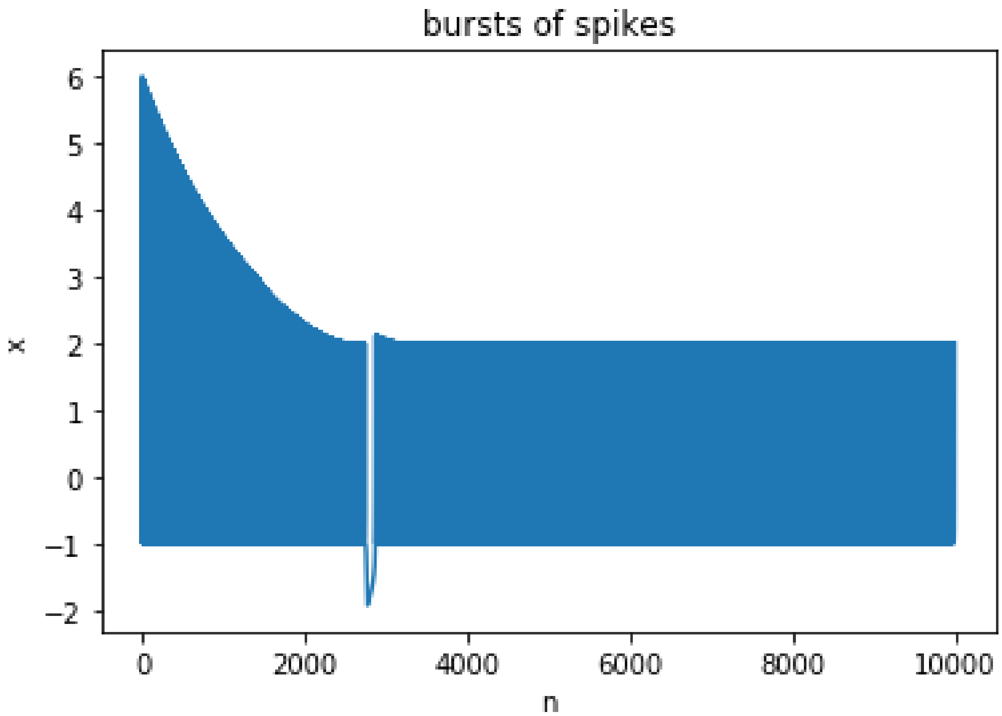

tonic spiking
x
n
4
3
2
1
0
−1
0
2000
4000
6000
8000
10000


tonic spiking
x
n
4
3
2
1
0
−1
0
2000
4000
6000
8000
10000

0.5 c) Bursts of spikes plot

In [70]:
```
title="bursts of spikes"
alpha=6
mu=0.001
sigma=-0.01
generate_spike(alpha,mu,sigma,title)
sigma=0.386
generate_spike(alpha,mu,sigma,title)
```




bursts of spikes
x
6
5
4
3
2
1
0
−1
−2
0
2000
4000
6000
8000
10000
n