Neelkumar S Bhuva

Neural Networks Project 1

Without momentum :

Choice of learning : Online learning - it is simple and requires minimal memory

With different learning rates the learning seems to be converging with different number of epochs and there is no observed pattern. That is, the number of epochs does not decrease with increasing learning rate (at least not strictly). Overall there seems to be a decrease in the number of epochs. The minimum of the error function seems to lie in a narrow valley and hence the gradient descent follows wide oscillations. The gradient descent oscillates back and forth resulting in larger number of epochs.

The initial weights and bias are initialized randomly and with different initial weights and bias, the number of epochs varies significantly. Learning was repeated with different initial weights.

With Momentum :

The number of epochs observed with momentum also does not follow a pattern. That is, it does not seem to increase or decrease with learning rate and gives different results with different learning rates. Theoretically using momentum helps avoid excessive oscillations of gradient descent and tends to follow one direction. But,from the test results it is clear that is not the case here. The combination of learning rates and momentum rate chosen here is not optimal.

However, increasing the learning rate beyond 0.5 (without momentum) reduces the number of epochs significantly.

Neelkumar S Bhuva

Neural Networks Project 1

Test results without using momentum :

Learning Rate	Number of epochs
0.05	67459
0.1	814628
0.15	31167
0.2	50322
0.25	13063
0.3	120775
0.35	43088
0.4	38863
0.45	30829
0.5	32588

Neelkumar S Bhuva

Neural Networks Project 1

Test results with momentum alpha = 0.9 :

Learning Rate	**Number of epochs**
0.05	22957
0.1	101532
0.15	83153
0.2	418794
0.25	11695
0.3	34178
0.35	97473
0.4	349479
0.45	87967
0.5	46217

Neelkumar S Bhuva

Neural Networks Project 1

```
import itertools
import numpy as np
import math
import random as rd

def initialize_weights():
    weights = []
    for i in range(0,num_units):
        weights.append([])
        for j in range(0,4):
            rand = rd.random()
            flag = rd.choice([0,1])
            if(flag == 0):
                rand = rand * (-1)
            weights[i].append(rand)
    return weights

def initialize_bias():
    weights = []
    for i in range(0,num_units):
        rand = rd.random()
        flag = rd.choice([0,1])
        if(flag == 0):
            rand = rand * (-1)
        weights.append(rand)
    return weights

def get_sigmoid(x,a):
    return 1 / (1 + math.exp((-1) * a * x))
```

Neelkumar S Bhuva
Neural Networks Project 1

```
def get_sigmoid_prime(x,a):
    sig = get_sigmoid(x,a)
    return (a * sig * (1 - sig))

def get_v_of_n(x,w,b):
    v_n = np.dot(x,w)
    v_n = b + v_n
    return v_n

def desiredOutputs(lst):
    des_output = []
    for i in lst:
        s = 0
        s = sum(i)
        if(s % 2 == 0):
            des_output.append(0)
        else:
            des_output.append(1)
    return des_output

def activate(v_n):
    #print("In activate")
    y_n = []
    for i in v_n:
        #print(i)
        y_n.append(get_sigmoid(i,1))
    return y_n

def get_error(d_n,y_n):
    return (d_n - y_n)
```

Neelkumar S Bhuva

Neural Networks Project 1

```
def get_delta_k(error_out,v_n_out):
        return (error_out * get_sigmoid_prime(v_n_out,1))

def
out_weight_update(w_n,b_n,delta_w_n,delta_b_n,learn_rate,error_out,v_n
_out,y_n,alpha):
        w_n_plus_one = []
        b_n_plus_one = 0
        delta_w_n_new = []
        delta_b_n_new = 0
        for i in range(0,len(w_n)):
                w_n_plus_one.append(w_n[i] + learn_rate *
get_delta_k(error_out,v_n_out) * y_n[i] + delta_w_n[i])
                delta_w_n_new.append(alpha * (w_n_plus_one[i] - w_n[i]))
        b_n_plus_one = b_n + learn_rate * get_delta_k(error_out,v_n_out) *
1 + delta_b_n
        delta_b_n_new = alpha * (b_n_plus_one - b_n)
        return (w_n_plus_one,b_n_plus_one,delta_w_n_new,delta_b_n_new)

def initializeWithZero(m,n):
        temp = []
        for i in range(0,m):
                temp.append([])
                for j in range(0,n):
                        temp[i].append(0)
        return temp

def
hidden_weight_update(w_n,b_n,delta_w_n,delta_b_n,learn_rate,v_n,x,delt
a_k,alpha):
        w_n_plus_one = []
        b_n_plus_one = []
```

```
    delta_w_n_new = initializeWithZero(num_hidden_units,len(w_n[0]))
    delta_b_n_new = []
    for i in range(0,num_hidden_units):
        w_n_plus_one.append([])
        for j in range(0,len(w_n[i])):
            w_n_plus_one[i].append(w_n[i][j] + learn_rate *
get_sigmoid_prime(v_n[i],1) * delta_k * w_n[4][i] * x[j] + delta_w_n[i][j])
            delta_w_n_new[i][j] = alpha * (w_n_plus_one[i][j] -
w_n[i][j])
        b_n_plus_one.append(b_n[i] + learn_rate *
get_sigmoid_prime(v_n[i],1) * delta_k * w_n[4][i] * 1 + delta_b_n[i])
        delta_b_n_new.append(alpha * (b_n_plus_one[i] - b_n[i]))
    return(w_n_plus_one,b_n_plus_one,delta_w_n_new,delta_b_n_new)

def isDesiredError(error,stop_error):
    for i in error:
        if(i > stop_error):
            return False
    return True

def initializeWithOne(size):
    temp = []
    for i in range(0,size):
        temp.append(1)
    return temp

def update_weights(weights,b,w_n_plus_one,b_n_plus_one,w,b1):
    for i in range(0,len(w_n_plus_one)):
        weights[4][i] = w_n_plus_one[i]
    b[4] = b_n_plus_one

    for i in range(0,num_hidden_units):
        for j in range(0,4):
            weights[i][j] = w[i][j]
        b[i] = b1[i]
    return (weights,b)

def write_to_file(fd,learn_rate,epochs,error):
    fd.write("Learning Rate : " + str(learn_rate) + "\n")
    fd.write("Number of epochs : " + str(epochs) + "\n")
    fd.write("Error : " + str(error) + "\n")

def learn(train_data,des_output,learn_rate,weights,b,fd,alpha):
    stop_error = 0.05
    size = 16
    error = initializeWithOne(size)
    epochs = 0
    print("Learning Rate : " + str(learn_rate))
    print("alpha : " + str(alpha))
    #print(weights)
    while(not(isDesiredError(error,stop_error))):
        delta_b_n_out = 0
        delta_w_n_out = [0,0,0,0]
        delta_w_n = initializeWithZero(num_hidden_units,4)
        delta_b_n = [0,0,0,0]
        #train_data = rd.sample(train_data,len(train_data))
        #print(train_data)
        for p in range(0,len(train_data)):
            #print("--------------------------Training data : " + str(p))
            v_n = []
            v_n_out = []
            for i in range(0,num_hidden_units):
                v_n.append(get_v_of_n(train_data[p],weights[i],b[i]))
            y_n = activate(v_n)
            #print(y_n)
            for i in range(num_hidden_units,num_hidden_units +
num_out_units):
                v_n_out.append(get_v_of_n(y_n,weights[4],b[4]))
            y_n_out = activate(v_n_out)
            #print(y_n_out)
            error_out = get_error(des_output[p],y_n_out[0])
            error[p] = abs(error_out)

w_n_plus_one,b_n_plus_one,delta_w_n_out,delta_b_n_out =
out_weight_update(weights[4],b[4],delta_w_n_out,delta_b_n_out,learn_rate
,error_out,v_n_out[0],y_n,alpha)

            delta_k = get_delta_k(error_out,v_n_out[0])
            w,b1,delta_w_n,delta_b_n =
hidden_weight_update(weights,b,delta_w_n,delta_b_n,learn_rate,v_n,train
_data[p],delta_k,alpha)

            weights,b =
update_weights(weights,b,w_n_plus_one,b_n_plus_one,w,b1)

        epochs = epochs + 1
        if(epochs % 3000 == 0):
            print(epochs)
        if(epochs % 15000 == 0):
            print("Error : " + str(error))

    write_to_file(fd,learn_rate,epochs,error)
    # print(weights)
    # print(b)

def initializeWith(x,m,n):
    temp = []
    if(n == 0):
        for i in range(0,m):
            temp.append(x[i])
    else:
        for i in range(0,m):
            temp.append([])
            for j in range(0,n):
                temp[i].append(x[i][j])
    return temp

if __name__ == '__main__':
    global num_hidden_units
    num_hidden_units = 4
    global num_out_units
    num_out_units = 1
    global num_units
    num_units = num_out_units + num_hidden_units
    n = 4
    lst = list(itertools.product([0, 1], repeat=n))
    des_output = desiredOutputs(lst)
    #print(lst)
    #print(des_output)
    global initial_weights
    initial_weights = initialize_weights()
    global initial_b
    initial_b = initialize_bias()
    #print(init_b)
    fd = open("Results.txt",'w')
    learn_rate = np.arange(0.05,0.51,0.05)
    alpha = 0
    print(learn_rate)
```

```
fd.write("Initial Weights : " + str(initial_weights))
fd.write("Initial bias : " + str(initial_b))
for i in learn_rate:
        init_weights = initializeWith(initial_weights,num_units,4)
        init_b = initializeWith(initial_b,num_units,0)
        print("Initial Weights : ")
        print(init_weights)
        print("Initial bias : ")
        print(init_b)
        #learn(lst,des_output,i,init_weights,init_b,fd)
        learn(lst,des_output,i,init_weights,init_b,fd,alpha)
fd.close()

fd = open("Results_momentum_3.txt",'w')
fd.write("Initial Weights : " + str(initial_weights))
fd.write("Initial bias : " + str(initial_b))
alpha = 0.9
for i in learn_rate:
        init_weights = initializeWith(initial_weights,num_units,4)
        init_b = initializeWith(initial_b,num_units,0)
        learn(lst,des_output,i,init_weights,init_b,fd,alpha)
fd.close()
```