Lez08_Delta_Rule_Optimization

November 8, 2022

[5]:
```
#Imports
import numpy as np
import random

#Define activation function
def activation(x, w):
    #Linear activation function
    if(activation_function == 0):
        return np.dot(x,w)

    #Sigmoid activation function
    elif(activation_function == 1):
        return 1.0/(1.0 + np.exp(-np.dot(x,w)))

    else:
        print("Error: Activation function not specified.")
        exit()

#Variables
n = 5 #Input dimension
training_set_size = 20 #Number of training examples
eta = 0.1 * 1.0/training_set_size # Learning Rate
activation_function = 1 # 0 = linear activation function, 1 = sigmoid␣
 ↪activation function

x = np.mgrid[0.0:training_set_size, 0:n] # Training set, only use x[0] subarray
t = np.arange(0, training_set_size, 1.0) #Target values
w = np.arange(0, n, 1.0) # Weight vector
w_step = np.arange(0, n, 1.0) # Weight step vector
w_grad = np.arange(0, n, 1.0) # Weight gradient
output = np.arange(0, training_set_size, 1.0) # Output vector

number_iterations = 0
max_iterations = 100 # Secondary exit condition for gradient descent algorithm

# To ensure that random numbers stay the same for multiple executions
random.seed(3)
```

```
#Initialise x and t array
for i in range(training_set_size):
    #Initialise target values randomly between -1 and 1
    random_number = random.random()
    t[i] = random.random()*1.0
    if(random_number < 0.5):
        t[i] *= -1

    #Initialise inputs
    for j in range(n):
        random_number = random.random()
        x[0][i][j] = random.random()
        if(random_number < 0.5):
            x[0][i][j] *= -1

#Initialise weight vector
for i in range(n):
    w[i] = random.random()*0.25

print("Weights at the beginning: ")
print(w)

#Gradient descent
while(number_iterations < max_iterations): #Loop until gradient of weights is␣
 ↪(approximately) equal to zero or max iterations are reached
    #Initalise step vector, output vector and gradient
    for i in range(n):
        w_step[i] = 0.0
        w_grad[i] = 0.0
        output[i] = 0.0

    #Loop over training sample
    for i in range(training_set_size):
        #Compute the output
        output[i] = activation(x[0][i], w)

        #Update weight step
        for j in range(n):
            w_step[j] += eta*(t[i] - output[i])*x[0][i][j]

    #Update weights
    for i in range(n):
        w[i] += w_step[i]

    #Calculate gradient of the weights and check if training is finished
    for i in range(n): #Loop over vector entries
```

```
        result = 0.0
        for j in range(training_set_size): # Loop over sample
            result += (t[j] - output[j])*x[0][j][i]

        w_grad[i] = -1.0 * (1.0/training_set_size) *result

    # Check, if weights have reached their optimum
    cancellation = True
    for i in range(n):
        if(abs(w_grad[i]) >= 0.01):
            cancellation = False

    if(cancellation):
        break


print("Algorithm completed. Final weights: ")
print(w)
```

```
Weights at the beginning:
[0.105746   0.118225   0.19412442 0.00045216 0.0137084 ]
Algorithm completed. Final weights:
[-1.23451128  1.06542436 -0.91510931  1.24943678 -1.15528725]
```

[ ]: