Lez09_Hidden_Layers_Representation

November 8, 2022

[1]:
```
##################################################################Imports#############################
import numpy as np
import random

##################################################################Variables###########################
n_1 = 8 #Input dimension, size of first layer
n_2 = 3 #Size of second layer
n_3 = 8 #Size of third layer
learning_rate = 0.01 #Learning rate for actualisation of the weights
training_set_size = 8 #Number of training inputs
training_epochs = 5000 #Number of iterations for training
activation_hidden_layer = 1 # 0 = linear activation, 1 = sigmoid activation
activation_output_layer = 1

##################################################################Define activation␣
↪function#########################################################
def activation(activation_function, 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()


##################################################################Define weight_sum␣
↪function#########################################################

def weight_sum(w, delta, j):
    result = 0.0
    for i in range(n_3):
        result += w[i][j] * delta_k[i]
```

```
    return result

##############################################################Set input data and␣
 ↪target output#################################################################
input_data = np.identity(8)

target_output = np.identity(8)


#############################################################Set arrays for␣
 ↪output#####################################################################
output_hidden_layer = np.zeros((8,3)) # 8 Training iterations times 3 nodes

output_output_layer = np.zeros((8,8))

#############################################################Set arrays for␣
 ↪weights####################################################################
weight_input_hidden = np.zeros((3,8)) # 3 Hidden nodes * 8 connections each

weight_hidden_output = np.zeros((8,3)) #8 Output nodes * 3 connections each


#############################################################Set arrays for weight␣
 ↪updates####################################################################

delta_k = np.arange(0, n_3, 1) #Initialise array with correct size
delta_j = np.arange(0, n_2, 1)
delta_w_kj = np.zeros((8,3)) # 3 connections for each output node
delta_w_ji = np.zeros((3,8)) # 3 Hidden nodes * 8 connections each

################################################Initialise weights with random␣
 ↪values from -0.05 to 0.05#####################################
# To ensure that random numbers stay the same for multiple executions
random.seed(3)

# Initalise weights input-hidden
for i in range(3):
    for j in range(8):
        sign = random.random()
        weight_input_hidden[i][j] = random.random()*0.05
        if(sign < 0.5):
            weight_input_hidden[i][j] *= -1

# Initalise weights hidden-output
for i in range(8):
    for j in range(3):
```

```
        sign = random.random()
        weight_hidden_output[i][j] = random.random()*0.05
        if(sign < 0.5):
            weight_hidden_output[i][j] *= -1


##################################################Backpropagation algorithm for␣
 ↪1 hidden layer#########################################################
for i in range(training_epochs): # Loop over epochs
    for j in range(training_set_size): # Loop over training set
        #Compute output for hidden layer
        for k in range(n_2):
            output_hidden_layer[j][k] = activation(activation_hidden_layer,␣
 ↪input_data[j], weight_input_hidden[k])

        #Compute output for output layer
        for k in range(n_3):
            output_output_layer[j][k] = activation(activation_output_layer,␣
 ↪output_hidden_layer[k], weight_hidden_output[k])

        #Update weights
        #Calculate delta_ks and weight steps for output layer - hidden layer
        for k in range(n_3):
            delta_k[k] = output_output_layer[j][k] * (1 -␣
 ↪output_output_layer[j][k]) * (target_output[j][k] -␣
 ↪output_output_layer[j][k])
            for l in range(3):
                delta_w_kj[k][l] = delta_k[k] * output_hidden_layer[j][l]

        #Calculate delta_js and weight steps for hidden layer - input layer
        for k in range(n_2):
            delta_j[k] = output_hidden_layer[j][k] * (1 -␣
 ↪output_hidden_layer[j][k]) * weight_sum(weight_hidden_output, delta_k, k)
            for l in range(8):
                delta_w_ji[k][l] = delta_j[k] * input_data[j][l]

        #Update weights
        #Update weights input-hidden
        for k in range(3):
            for l in range(8):
                weight_input_hidden[k][l] += learning_rate * delta_w_ji[k][l]

        #Update weights hidden_output
        for k in range(8):
            for l in range(3):
                weight_hidden_output[k][l] += learning_rate * delta_w_kj[k][l]
```

```
##################################################Print out hidden output for␣
 ↪comparison of results#############################################################
print("Output of hidden layer:")
for i in range(training_set_size):
    print("Input Number: ", i + 1)
    for j in range(n_2):
        print(activation(activation_hidden_layer, input_data[i],␣
 ↪weight_input_hidden[j]))
```

```
Output of hidden layer:
Input Number:  1
0.49319755442557917
0.5108488633412298
0.48998997803052363
Input Number:  2
0.4924515730625487
0.5092645877126953
0.4883072981686718
Input Number:  3
0.5008191100077303
0.500800392294165
0.5012181764613372
Input Number:  4
0.48953316579595485
0.5073882069965446
0.49728768983965893
Input Number:  5
0.4970708964951731
0.4996123531842971
0.5054518072644708
Input Number:  6
0.5058780230315958
0.5059090884791648
0.5037627564454114
Input Number:  7
0.505954133639188
0.5109833928505972
0.5048231786247344
Input Number:  8
0.5018826964024803
0.5115116986694116
0.49268709516936254
```

```
[ ]:
```