Perceptron with one hidden layer

Eric Blohm

October 1, 2024

Here is the Python code for the perception with one hidden layer:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#data has the input in the first two elements and
    output on the third.
def GetInputOutput(data):
    input = []
    output = []
    for row in data:
        input.append([row[0],row[1]])
        output.append(row[2])
    return np.array(input),np.array(output)


def getCSV(file):
    df = pd.read_csv(file,header=None)
    data_list = df.values.tolist()
    input,output = GetInputOutput(data_list)
    return input,output


def init_w_theta(M):
    # between input and hidden
    # 1/2 from the number of inputs
    variance = 1/2
    standard_dev = np.sqrt(variance)
    theta_j = np.zeros(M)
    w_jk = np.random.normal(0, standard_dev, size=(M
        ,2))

    # between hidden and output
    variance_h = 1/M
```

```
32      standard_dev_h = np.sqrt(variance_h)
33      theta = 0
34      w_j = np.random.normal(0, standard_dev_h, size=(M)
            )
35      return w_jk, w_j, theta_j, theta
36
37
38  def compute_hidden_output(w_jk,theta_j,input):
39      b_j = np.dot(w_jk, input.T) - theta_j.T
40      return np.tanh(b_j)
41
42
43  def compute_network_output(w_j,theta,hidden_output,mu)
        :
44      sum = 0
45      for j in range(0,len(w_j)):
46          sum += w_j[j]*hidden_output[mu][j]
47      B_i = sum - theta
48      return np.tanh(B_i)
49
50
51  def back_prop(output_error,w_j,hidden_output,
        hidden_error):
52      for m in range(0,len(w_j)):
53          hidden_error[m] = output_error * w_j[m]* (1-
                hidden_output[m]**2)
54      return hidden_error
55
56
57  def get_delta_w(input,hidden_output,hidden_error,
        output_error,eta, mini_batch,M):
58      delta_w_j = np.zeros(M)
59      #Delta_m, V_n. m = 1 only 1 output per pattern
60      for n in range(0,len(hidden_output[0])):
61          for mu in range(0,mini_batch):
62              delta_w_j[n]+= output_error[mu]*
                    hidden_output[mu][n]
63
64
65      delta_w_jk = np.zeros((M,len(input[0])))
66      #m is every hidden neuron.
67      for m in range(0,len(hidden_error[0])):
68          #n is x_1 and x_2
69          for n in range(0,len(input[0])):
70              for mu in range(0,mini_batch):
71                  delta_w_jk[m][n] += hidden_error[mu][m
```

```
                    ]*input[mu][n]

    return eta*delta_w_jk, eta*delta_w_j


def get_delta_theta(output_error,hidden_error,eta,
    mini_batch,M):
    delta_theta = 0
    #m = 1
    for mu in range(0,mini_batch):
        delta_theta += output_error[mu]

    delta_theta_j = np.zeros(M)
    for m in range(0,len(hidden_error[0])):
        for mu in range(0,mini_batch):
            delta_theta_j[m] += hidden_error[mu][m]

    return -eta*delta_theta_j, -eta*delta_theta


def compute_classification_error(output,target):
    sum = 0
    for mu in range(0,len(target)):
        sum+= np.abs((np.sign(output[mu])-target[mu]))
    return (1/(2*len(target)))*sum


def compute_energy_function(output,target):
    sum = 0
    for mu in range(0,len(target)):
        sum += (target[mu]-output[mu])**2
    return 0.5*sum


def save_values(weights_jk,weights_j,threshold_1,
    threshold_2):
    df = pd.DataFrame(weights_jk)
    df.to_csv('w1.csv', index=False,header=False)

    df = pd.DataFrame(weights_j)
    df.to_csv('w2.csv',index=False, header=False)

    df = pd.DataFrame(threshold_1)
    df.to_csv('t1.csv',index=False, header=False)

    df = pd.DataFrame([threshold_2])
```

```
115     df.to_csv('t2.csv',index=False, header=False)
116
117
118 def main():
119
120     ##### configuration #####
121     M = 10
122     epochsMax = 500
123     batch_size = 64
124     eta = 0.01
125     ##########################
126
127     ### Retrieve data ###
128     input,target = getCSV('training_set.csv')
129     input_validation,target_validation = getCSV('
            validation_set.csv')
130     ####################
131
132     #### Center and normalize data ####
133     input_mean = np.mean(input, axis=0)
134     input_std = np.std(input, axis=0)
135     ## Normalize based in training metrics
136     input = (input - input_mean) / input_std
137     input_validation = (input_validation - input_mean)
            / input_std
138     ##########################################
139
140     ### initialize weights and thresholds ###
141     w_jk,w_j,theta_j,theta = init_w_theta(M)
142     ##########################################
143
144
145     ## used for plotting ##
146     c_train_list = np.zeros(epochsMax)
147     c_validate_list = np.zeros(epochsMax)
148     ########################
149
150     for epoch in range(0,epochsMax):
151         ### Shuffle the input data and targets ###
152         indices = np.arange(len(input))
153         np.random.shuffle(indices)
154         input = input[indices]
155         target = target[indices]
156         ##########################################
157
158         #### Create mini batches #####
```

```
159         for start in range(0, len(input), batch_size):
160             end = start + batch_size
161             mini_batch = input[start:end]
162             target_batch = target[start:end]
163 
164             ### Initialize outputs for the mini-batch
                    ###
165             hidden_output = np.zeros((len(mini_batch),
                    M))
166             output = np.zeros(len(mini_batch))
167             output_error = np.zeros(len(mini_batch))
168             hidden_error = np.zeros((len(mini_batch),M
                    ))
169 
170             #### for each pattern in mini batch  ####
171             for mu in range(0,len(mini_batch)):
172                 #only one layer
173                 ##### Feed forward #####
174                 hidden_output[mu] =
                        compute_hidden_output(w_jk,theta_j,
                        mini_batch[mu])
175                 output[mu] = compute_network_output(
                        w_j,theta,hidden_output,mu)
176                 ########################
177 
178                 ##### back propagation #####
179                 output_error[mu] = (target_batch[mu]-
                        output[mu])*(1-output[mu]**2)
180                 for m in range(0,len(w_j)):
181                     hidden_error[mu][m] = output_error
                            [mu] * w_j[m]* (1-hidden_output
                            [mu][m]**2)
182                 ############################
183 
184             ##### Update weights #####
185             #print(f"\n-Weights_jk before update: {
                    w_jk}, \nWeights_j before update: {w_j
                    }")
186             delta_w_jk, delta_w_j = get_delta_w(
                    mini_batch,hidden_output,hidden_error,
                    output_error,eta, len(mini_batch),M)
187             w_jk+= delta_w_jk
188             w_j += delta_w_j
189 
190             delta_theta_j,delta_theta =
                    get_delta_theta(output_error,
```

```
                hidden_error,eta, len(mini_batch),M)
191             theta_j+= delta_theta_j
192             theta  += delta_theta
193             ############################
194
195         ### validate during training and early stop
                ###
196         hidden_output_validate = np.zeros((len(
                input_validation),M))
197         output_validate = np.zeros(len(
                input_validation))
198         for mu in range(0,len(input_validation)):
199             hidden_output_validate[mu] =
                    compute_hidden_output(w_jk,theta_j,
                    input_validation[mu])
200             output_validate[mu] =
                    compute_network_output(w_j,theta,
                    hidden_output_validate,mu)
201         #
                ################################################


202
203         ## Compute classification error and energy
                function. ##
204         c = compute_classification_error(
                output_validate,target_validation)
205         H_validate = compute_energy_function(
                output_validate,target_validation)
206         print("C:", c*100, ", Energy function: ",
                H_validate)
207         c_validate_list[epoch] = c*100
208         if (c < 0.12):
209             break
210         #
                ################################################


211
212     #if stopped early, retrieve all no negative
            elements
213     c_validate_list = c_validate_list[c_validate_list
            > 0]
214
215     ## create list for plotting ##
216     epochs = np.arange(len(c_validate_list))
217
218     ## plot Validation classification error ##
```

```
219     plt.plot(epochs, c_validate_list, marker='o',
            color='green', markersize=4, linestyle='-')
220     plt.title('Validation Classification Error')
221     plt.xlabel('Epochs')
222     plt.ylabel('c_validate')
223     plt.grid()
224
225     plt.tight_layout()
226     plt.show()
227     #
            ##########################################################

228
229     ### Validate network after training ###
230     hidden_output_validate = np.zeros((len(
            input_validation),M))
231     output_validate = np.zeros(len(input_validation))
232     for mu in range(0,len(input_validation)):
233         hidden_output_validate[mu] =
                compute_hidden_output(w_jk,theta_j,
                input_validation[mu])
234         output_validate[mu] = compute_network_output(
                w_j,theta,hidden_output_validate,mu)
235     c = compute_classification_error(output_validate,
            target_validation)
236     H_validate = compute_energy_function(
            output_validate,target_validation)
237     print("C:", c*100, ", Energy function: ",
            H_validate)
238     ########################################
239
240     ### save the weights and thresholds to csv files
            ###
241     #save_values(w_jk,w_j,theta_j,theta)
242     #
            ##########################################################

243
244
245 if __name__ == "__main__":
246     main()
```