Artificial Neural network home problem 3

Purusothaman Seenivasan CID: pursee, Personal number: 19970421-8396

October 2023

1 Self-Organizing Map

In this problem 4 dimensional input data is mapped into 2 dimensional output space. Weights are initiated from a uniform distribution in the interval [0,1]. The plot on the left side is plotted before any weight update that is location of winning neuron before any weight update and right plot is the plot of output after 10 epochs that is location of winning neuron after weight update. In each epochs, the number of times the weighted gets updated is equal to number of rows in input data. The weight is updated using below equation

$\delta\omega_i = \eta(i, io)(x - w_i)$
where
$h(i, io)$ is neighbourhood function which is Gaussian function of euclidean distance r_i, r_{i0} in output array
i_0 is the index of winning neuron

$h(i, io) = \exp\left(-\frac{|r_i - r_{io}|^2}{2\sigma^2}\right)$
where
r_i is position of neuron i in the output array
r_{io} is position of winning neuron in the output array

In each epochs σ and η undergo decay as per below equation.
$\eta = \eta_0 \exp(-d_\eta \text{epoch})$
$\sigma = \sigma_0 \exp(-d_\sigma \text{epoch})$

There are instances where the output nodes, which should represent one specific class, may be found in regions associated with another class. I suspect this occurs because there is a scarcity of data available for training in those particular areas.

1.1 Plot

Legends are given outside the plot
Red - Versicolor
Blue - Sestosa
Green - Virginica

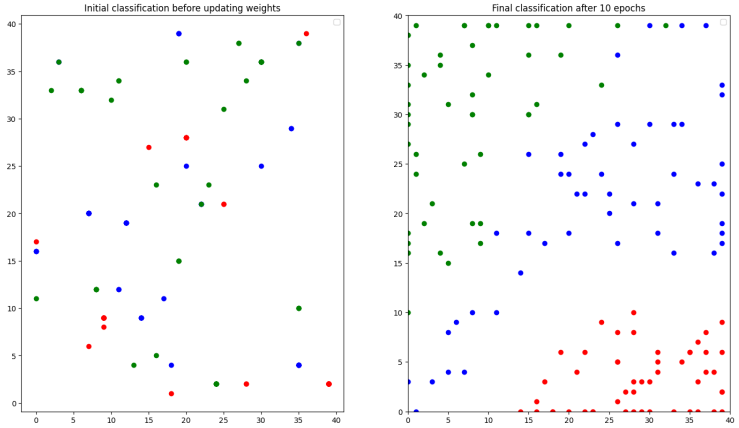

Figure 1: Plot of winning neurons before and after updating the weights

1.2 Program

Listing 1: Python Code

```
import numpy as np
import random
import csv
from matplotlib import pyplot as plt

with open('iris-data.csv', 'r') as f:
  reader = csv.reader(f)
  input = list(reader)

with open('iris-labels.csv', 'r') as f:
  reader = csv.reader(f)
  input_labels = list(reader)

sigma0 = 10
eta0 = 0.1
dsigma = 0.05
deta = 0.01

input = np.array(input)
input = input.astype(float)

max_values = input.max(axis=0)
input_data = input / max_values
```

```
input_size = len(input_data)
print(np.array(input_data).shape)
print(len(input_data))

intial_output = np.zeros((40, 40))
output = np.zeros((40, 40))
output_weight = np.random.rand(40, 40, 4)
distance = np.zeros((40, 40))
min_distance = 100000
(min_i, min_j) = (-1, -1)

intial_norm = np.full((input_size, 1), 1000000)
intial_winner = np.full((input_size, 2), 1000000)
final_norm = np.full((input_size, 1), 1000000)
final_winner = np.full((input_size, 2), 1000000)

print('outpt weight', output_weight[0, 1], output_weight[1, 1])
print('input', input_data)

for i in range(input_size):

  intial_output = np.dot(output_weight, input_data[i])

  min_dis = 100000
  winnerNeuron = (-1, -1)

  for j in range(40):
    for k in range(40):
      d = np.linalg.norm(input_data[i] - output_weight[j, k])

      if d < min_dis:
        winnerNeuron = (j, k)
        min_dis = d
  intial_winner[i, :] = winnerNeuron
  intial_norm[i] = min_dis
  print(intial_norm)

print(intial_winner)

input_dictionary = {"Versicolor": [], "Sestosa": [], 'Virginica': []}
for i in range(input_size):
  if i < 50:
    input_dictionary['Versicolor'].append(i)
  elif i > 49 and i < 100:
    input_dictionary['Sestosa'].append(i)
```

```
  elif i > 99:
    input_dictionary['Virginica'].append(i)
plotnumber = 0

colors = ['red', 'blue', 'green']


def h(d, sigma):

  neigbour = np.exp((d**2) / (-2 * sigma**2))
  return neigbour


for epoch in range(10):
  sigma = sigma0 if epoch == 0 else sigma * np.exp(-0.05 * epoch)
  eta = eta0 if epoch == 0 else eta * np.exp(-0.01 * epoch)
  for n in range(input_size):
    output = np.dot(output_weight, input_data[n])

    min_dis = 100000
    winnerNeuron = (-1, -1)
    for i in range(40):
      for j in range(40):
        d = np.linalg.norm(input_data[n] - output_weight[i, j])

        if d < min_dis:
          winnerNeuron = (i, j)
          min_dis = d
    final_winner[n, :] = winnerNeuron
    final_norm[n] = min_dis

    for i in range(40):
      for j in range(40):
        (win_x, win_y) = final_winner[n]

        euclid_d = np.linalg.norm(np.array([i, j]) - final_winner[n])
        output_weight[i][j] = output_weight[i][j] + eta * h(
            euclid_d, sigma) * (input_data[n] - output_weight[i][j])

legend_labels = ['Versicolor', 'Sestosa', 'Virginica']
fig, (ax1, ax2) = plt.subplots(1, 2)
plt.xlim(0, 40)
plt.ylim(0, 40)
for lab in range(3):
  indices = []  #[int(x) for x in input_labels if int(x)==labels]
  for x in range(150):
```

```
    if int(float(input_labels[x][0])) == lab:
      indices.append(x)
  for index in indices:
    ax1.scatter(intial_winner[index][0],
                intial_winner[index][1],
                c=colors[lab])
    ax2.scatter(final_winner[index][0], final_winner[index][1], c=colors[lab])

ax1.set_title('Initial Winner')
ax2.set_title('Final Winner')

ax1.legend(handles=colors, labels=legend_labels)
ax2.legend(handles=colors, labels=legend_labels)

plt.show()
```