FFR135 HP1

Self organising map

1 Introduction and Method

The goal of this assignment is to create a self organising map, that clusters three different Iris flowers of the Iris-Flower data set [**UCI**].
After loading the data, the first step is to normalise it.

$$data = \frac{data}{max\{data\}}$$

The initial weights are set in a uniform distribution in the range $[0, 1]$. The output is set to a 40×40 array.
The learning rate η is depending on the epoch number, the same with the width σ.

$$\eta = \eta_0 exp(-d_\eta \times epoch) \tag{1}$$

$$\sigma = \sigma_0 exp(-d_\sigma \times epoch) \tag{2}$$

Here the initial values are $\eta_0 = 0.1$, $d_\eta = 0.01$, $\sigma_0 = 0.01$ and $d_\sigma = 0.05$.
Now the network is trained. For every epoch, firstly the updated *learning rate* and *width* are calculated. Then one loops over the input data, and finds the winning neuron, which is the one with the weight vector closest to the input $\mathbf{x}$. With the found winning neuron the weights are updated with

$$\delta \mathbf{w}_i = \eta h(i, i_0)(\mathbf{x} - \mathbf{w}_i) \tag{3}$$

After updating all weights the trained network is evaluated with the input data and the new weights. The result is shown in the next section.

2 Results

When executing the before described code, the following *Self-Organising map* can be found 1. The left side shows the untrained network, the right the trained one. Clustering seems successful, the only visual outlier lies at point $(30, 9)$.

3 Cooperation

I cooperated with Martina Gatti.

4 Python code

The code is attached on the following pages.

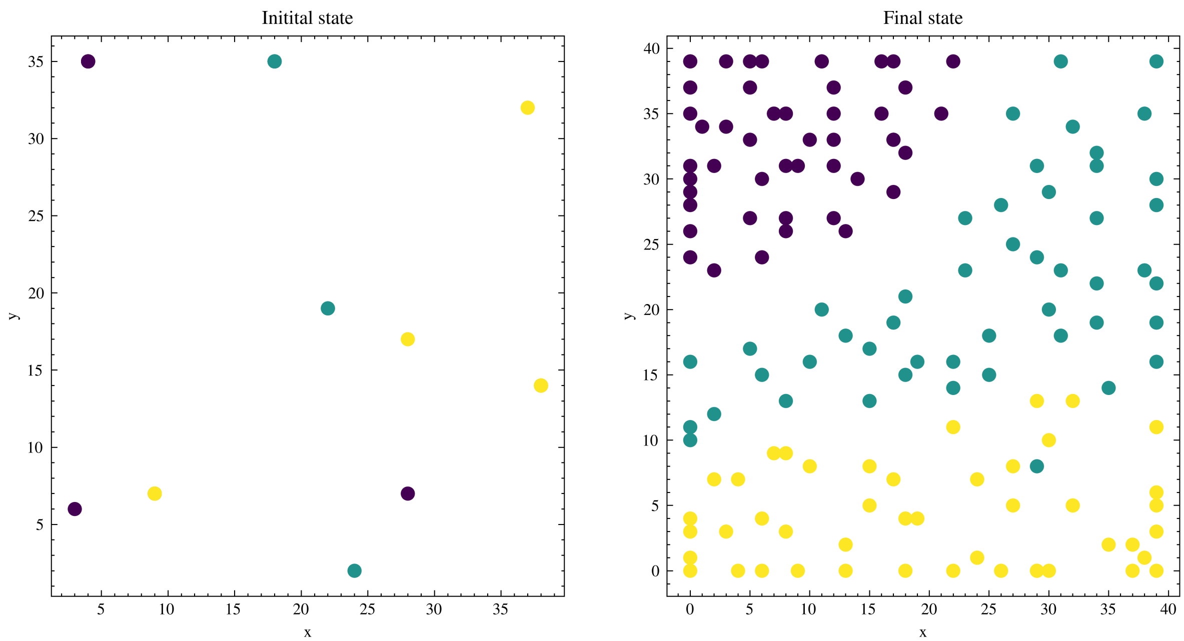

Figure 1: Trained self organising map, for the three Iris-Flowers. Left side of the figure shows the untrained network, where many data points lay above each other. The right side shows the output of the trained network

FFR 135 HW 3

Chaotic time-series prediction 2023

In [30]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [31]:
```
# parameters of the code
k = 0.01
inputNeurons = 3
reservoirNeurons = 500
tMax = 500
```

In [32]:
```
# load the training and test set
trainingSet = np.genfromtxt("training_set.csv", delimiter=",")
testSet = np.genfromtxt("test_set.csv", delimiter=",")

# print the shapes of the training and test set
print("trainingSet.shape: ", trainingSet.shape)
print("testSet.shape: ", testSet.shape)

# init the weights
inputWeights = np.random.normal(loc=0.0, scale= np.sqrt(0.002), size=(reservoirNeurons, inputNeurons
reservoirWeights = np.random.normal(loc=0.0, scale= np.sqrt(2/reservoirNeurons), size=(reservoirNeu

# print the shapes
print("inputWeights.shape: ", inputWeights.shape)
print("reservoirWeights.shape: ", reservoirWeights.shape)

# init the reservoir
X = np.zeros((trainingSet.shape[1], reservoirNeurons))
print("X.shape: ", X.shape)

# loop over all training examples
for i in range(trainingSet.shape[1]):
    ri_t1 = np.zeros((reservoirNeurons, 1))
    tmp1 = np.matmul(reservoirWeights, ri_t1) # shape=(500,1)
    tmp2 = np.matmul(inputWeights, trainingSet[:,i]) # shape=(500,)
    # shapes
    #print("tmp1.shape: ", tmp1.shape)
    #print("tmp2.shape: ", tmp2.shape)
    tmp2 = np.reshape(tmp2, (reservoirNeurons, 1)) # shape=(500,1)
    ri_t1 = np.tanh(tmp1 + tmp2) # shape=(500,1)
    X[i,:] = ri_t1[:,0] # shape=(500,)
```

```
trainingSet.shape:  (3, 19900)
testSet.shape:  (3, 100)
inputWeights.shape:  (500, 3)
reservoirWeights.shape:  (500, 500)
X.shape:  (19900, 500)
```

In [33]:
```
# update the reservoir weights
X = X[0:-1,:] # shape=(19899, 500)
Y = trainingSet[:,1:]

ridgeMatrix = k * np.eye(reservoirNeurons) # shape=(500,500)
XTX = np.matmul(X.T, X) # shape=(500,500)
XTX_inv = np.linalg.inv(XTX + ridgeMatrix) # shape=(500,500)
XTY = np.matmul(X.T, Y.T) # shape=(500,3)
outputWeights = np.matmul(XTX_inv, XTY) # shape=(500,3)
outputWeights = outputWeights.T # shape=(3,500)
```

In [34]:

```
# Output testset
newSize = testSet.shape[1] + tMax
testX = np.zeros((newSize, reservoirNeurons)) # shape=(600,500)
outputX = np.zeros((inputNeurons, newSize)) # shape=(3,600)
print("testX.shape: ", testX.shape)
print("outputX.shape: ", outputX.shape)

# loop over all test examples
for i in range(newSize):
    ri_t1 = np.zeros((reservoirNeurons, 1))
    if i < testSet.shape[1]:
        tmp1 = np.matmul(reservoirWeights, ri_t1) # shape=(500,1)
        tmp2 = np.matmul(inputWeights, testSet[:,i]) # shape=(500,)
        tmp2 = np.reshape(tmp2, (reservoirNeurons, 1)) # shape=(500,1)
        ri_t1 = np.tanh(tmp1 + tmp2) # shape=(500,1)
        testX[i,:] = ri_t1[:,0] # shape=(500,)

        output = np.matmul(outputWeights, ri_t1) # shape=(3,1)
        outputX[:,i] = output[:,0] # shape=(3,)
    else:
        tmp1 = np.matmul(reservoirWeights, ri_t1) # shape=(500,1)
        tmp2 = np.matmul(inputWeights, output) # shape=(500,)
        tmp2 = np.reshape(tmp2, (reservoirNeurons, 1)) # shape=(500,1)
        ri_t1 = np.tanh(tmp1 + tmp2)
        testX[i,:] = ri_t1[:,0]

        output = np.matmul(outputWeights, ri_t1) # shape=(3,1)
        outputX[:,i] = output[:,0]

predictedOutput = outputX[:,100:]
timePred = predictedOutput[1,:]
prediction = np.reshape(timePred, (1, predictedOutput.shape[1]))
```

```
testX.shape:  (600, 500)
outputX.shape:  (3, 600)
```

In [36]:

```
np.savetxt("prediction_1.csv", prediction, delimiter=",")
```

In [ ]:
```
# FFR135 HW3
## Self organising map
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import scienceplots
plt.style.use(['science','ieee'])
```

In [ ]:
```
# load data
inputData = np.loadtxt('/Users/felixwaldschock/Library/CloudStorage/OneDrive-Chalmers/02_Courses/02_
labesData = np.loadtxt('/Users/felixwaldschock/Library/CloudStorage/OneDrive-Chalmers/02_Courses/02_
```

In [ ]:
```
# normalize data
inputData /= np.max(inputData)
# define parameters
input_size = inputData.shape[1]
output_size = 40
epochs = 30
batch_size = len(inputData)
eta_0 = 0.1
d_eta = 0.01
sigma_0 = 10
d_sigma = 0.05
labels = ['Class 1', 'Class 2', 'Class 3']
```

In [ ]:
```
# init the weights with random gaussian
weights = np.random.uniform(0, 1, (output_size, output_size, input_size))

def neighbourhood_function(r_i, r_i0, sigma): # openTA
    return np.exp(-0.5 * (np.linalg.norm(r_i - r_i0)**2) / sigma**2)

def get_winning_neuron(data, weights):
    distances = np.linalg.norm(weights - data, axis=2)
    # print(distances.shape)
    # print(np.argmin(distances))
    # print(np.unravel_index(np.argmin(distances), distances.shape))
    return np.unravel_index(np.argmin(distances), distances.shape)
```

In [ ]:
```

bestInitNeuron = np.zeros((len(inputData), 2))
for i in range(len(inputData)):
    bestInitNeuron[i] = get_winning_neuron(inputData[i], weights)


# train the network
for e in range(epochs):
    eta = eta_0 * np.exp(-e * d_eta)
    sigma = sigma_0 * np.exp(-e * d_sigma)

    for i in range(len(inputData)):
        # get the winning neuron
        winning_neuron = get_winning_neuron(inputData[i], weights)
        # update the weights
        for j in range(output_size):
            for k in range(output_size):
                h = neighbourhood_function(np.array([j, k]), np.array(winning_neuron), sigma)
                weights[j, k, :] += eta * h * (inputData[i] - weights[j, k, :])

bestFinalNeuron = np.zeros((len(inputData), 2))
for i in range(len(inputData)):
    bestFinalNeuron[i] = get_winning_neuron(inputData[i], weights)
```

In [ ]:

```
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
# plot the initial state
ax[0].scatter(bestInitNeuron[:, 0], bestInitNeuron[:, 1], c=labesData, cmap='viridis', label=labels
ax[0].set_title('Initital state')
ax[0].set_xlabel('x')
ax[0].set_ylabel('y')



# plot
ax[1].scatter(bestFinalNeuron[:, 0], bestFinalNeuron[:, 1], c=labesData, cmap='viridis', label=labe
ax[1].set_title('Final state')
ax[1].set_xlabel('x')
ax[1].set_ylabel('y')

plt.show()
```