ECE 449 - Intelligent Systems Engineering

Lab 2 - D42: Neural Networks - Self-Organizing Maps

Lab date: *Thursday, October 14, 2021 -- 2:00 - 4:50 PM*
Room: *ETLC E5-013*
Lab report due: *Wednesday, October 27, 2021 -- 3:50 PM*

1. Objectives

The objective of this lab is to learn the concepts behind self-organizing maps, or Kohonen networks, and apply them to real-world datasets to examine how effective they can be.

2. Expectations

Complete the pre-lab, and hand it in before the lab starts. A formal lab report is required for this lab, which will be the completed version of this notebook. There is a marking guide at the end of the lab manual. If figures are required, label the axes and provide a legend when appropriate. An abstract, introduction, and conclusion are required as well, for which cells are provided at the end of the notebook. The abstract should be a brief description of the topic, the introduction a description of the goals of the lab, and the conclusion a summary of what you learned, what you found difficult, and your own ideas and observations.

3. Pre-lab

1. What is the curse of dimensionality? Why is dimensionality reduction even necessary?

4. Introduction

Self-organizing maps (SOMs), or *Kohonen networks*, are neural networks based on *competitive learning* that employ *unsupervised learning*. This means, that no targets are presented to the network for training and after a competitive process, only one output neuron is activated (*best-matching or winning neuron*). SOMs are usually two-dimensional lattices where the neurons are placed on the nodes, in this way, they map high dimensional input vectors onto a lower dimension map (in this lab two-dimensional), and are commonly used for clustering.
Mathematically, *competitive learning* involves an input vector activating a neuron or *winning neuron* with a weight vector closest to the input vector, determined by Euclidean distance:
x= $[x_1, x_2, \ldots, x_m]^T$
$\mathbf{w}_j = [w_{j1}, w_{j2}, \ldots, w_{jm}]^T$; j=1, 2, ..., *l*
x - input vector selected at random from an *m* input space
w - synaptic weight vector
l - total number of neurons in the network

$$i(\mathbf{x}) = argmin_j ||x - w_j||; \text{ j=1, 2, ..., } l$$

$i(\mathbf{x})$ - neuron that best matches the input vector **x**
i - winning neuron

Once the winning neuron has been identified, a *cooperative process* among neighbor neurons happens. This excites more closer neurons to the winning neuron than those farther away from it, according to a neighborhood function. Finally, an *adaptive process* changes the values of the weight vector in relation to the input vector. The winning neuron's weights are updated to values that are closer to the input vector, according to the following formula:

$$\Delta \mathbf{w}_j = \eta \, h_{j,i(\mathbf{x})}(x - w_j)$$

η - learning rate
$h_{j,i(\mathbf{x})}$ - neighborhood function

Various neighborhood functions can be used, but in general, the further the neuron is from the winning neuron, the smaller its weights shift towards to the input vector.
This process is repeated for each input vector for a large number of iterations, resulting in a map that clusters data in a way that preserves the topology of the inputs, and therefore revealing similarities in the inputs when visualized.
To visualize the results of the training, a *distance map* plots the average distance of the weights and the coordinates of the associated winning neuron. Finally, an *activation map* returns a matrix showing the number of times that each neuron has been activated.

5. Experimental Procedure / Assignment

While doing research or working as an engineer it is often important to get an intuitive grasp on the data that you are working with. With multidimensional data, this is often more complicated than "just plotting everything in one figure" because of the amount of dimensions and our lack of ability to intuitively grasp 4+ dimensional relations. This lab will confront you with 2 multidimensional datasets and require you to develop intuitive understanding for what is happening in those datasets. As a tool for better understanding, we will need some sort of dimensionality reduction - which will be *Self-Organizing maps*.

The two datasets and exercises you will work with:
The first batch of questions (**Exercise 1**) revolve around the Iris dataset again. The Iris Dataset is a set basically every machine learning student works with at some point, usually in the early beginning and revolving around a classification task. This will be the case here. Your overall task will be to use a SOM on this dataset to cluster and to classify, outperforming an alternative classification tool. **Exercise 1** is worth 63 points of the overall assignment and pretty straightforward.
In **Exercise 2** you will work on a more obscure and mixed dataset. You will have more freedom and this exercise set will require a little more creativity and critical thinking. The overarching task here will be pattern recognition and interpretation, while the questions are more open-ended and require more fundamental understanding than in **Exercise 1**. **It is recommended to start with Exercise 1.**

Methodology: The lab will walk you through the fetching of the datasets and supply the basic functions needed for completing each task.
The idea behind this is to minimize the amount of coding and programming, and encourage

experimentation from your side. Each supplied code snipped will be adequately documented, so you can modify it if you feel like it (just please make a safety copy of the function first...).

Run the cells below to install and import the libraries required to complete this lab.

In [1]:
```
%%bash
pip install --user -U minisom
# Make sure to restart the kernel after running this cell, if you are install
```

```
Requirement already satisfied: minisom in /home/jupyter/.local/lib/python3.
9/site-packages (2.2.9)
```

In [1]:
```
'For all the calculations and import stuff'
import os
import numpy as np
import scipy.io as sio              # Loads .mat variables

'To plot beautiful figures'
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from pylab import plot,axis,show,pcolor,colorbar,bone

'The AI modules of this lab'
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn import preprocessing  # Data preprocessing
from minisom import MiniSom        # SOM toolbox
```

Exercise 1: Self-organizing map on the IRIS Dataset

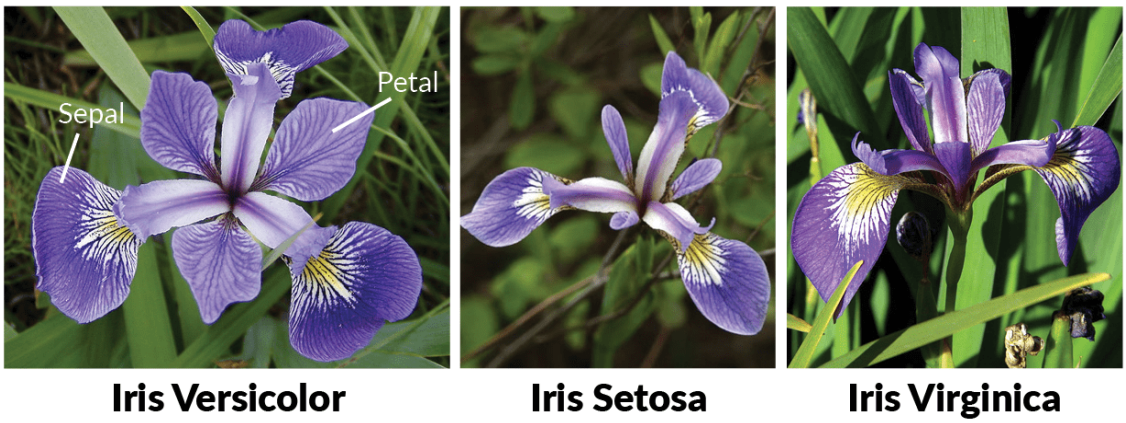
Task 1.1: During this exercise you will train a SOM to cluster the dataset. As a benchmark against classification, your SOM will compete with a simple decision tree. The exercise provides you with most of the functions that you will need. The goal is to develop a baseline understanding what SOMs are good at, and what they might have difficulties with. But let's start slower and at the beginning: Run the following cell to import the dataset and its description. To make sure you can properly interpret the results, you can spend the next couple of minutes reading about plants, or continue on and return when you need to give an explanation for your results.

In [2]:
```
'Use Scikit-learn to import the Iris Dataset, print a little literature on th
from sklearn import datasets
IRIS = datasets.load_iris()
print(IRIS.DESCR)

'What is a petal, what is a sepal?! ... Versicolor/Setosa/Virginica what?! ..
from IPython.display import Image
from IPython.core.display import HTML
Image(url= "https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Machine+
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is tak
en
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field a
nd
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers t
```

```
o a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analy
sis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transact
ions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

Out[2]:



Task 1.2:

Run the next cells to provide the 'naive' plots of the dataset's dimensional relations (created by projecting the data onto a two-dimensional plane):

- SepalLength
- SepalWidth
- PetalLength
- PetalWidth

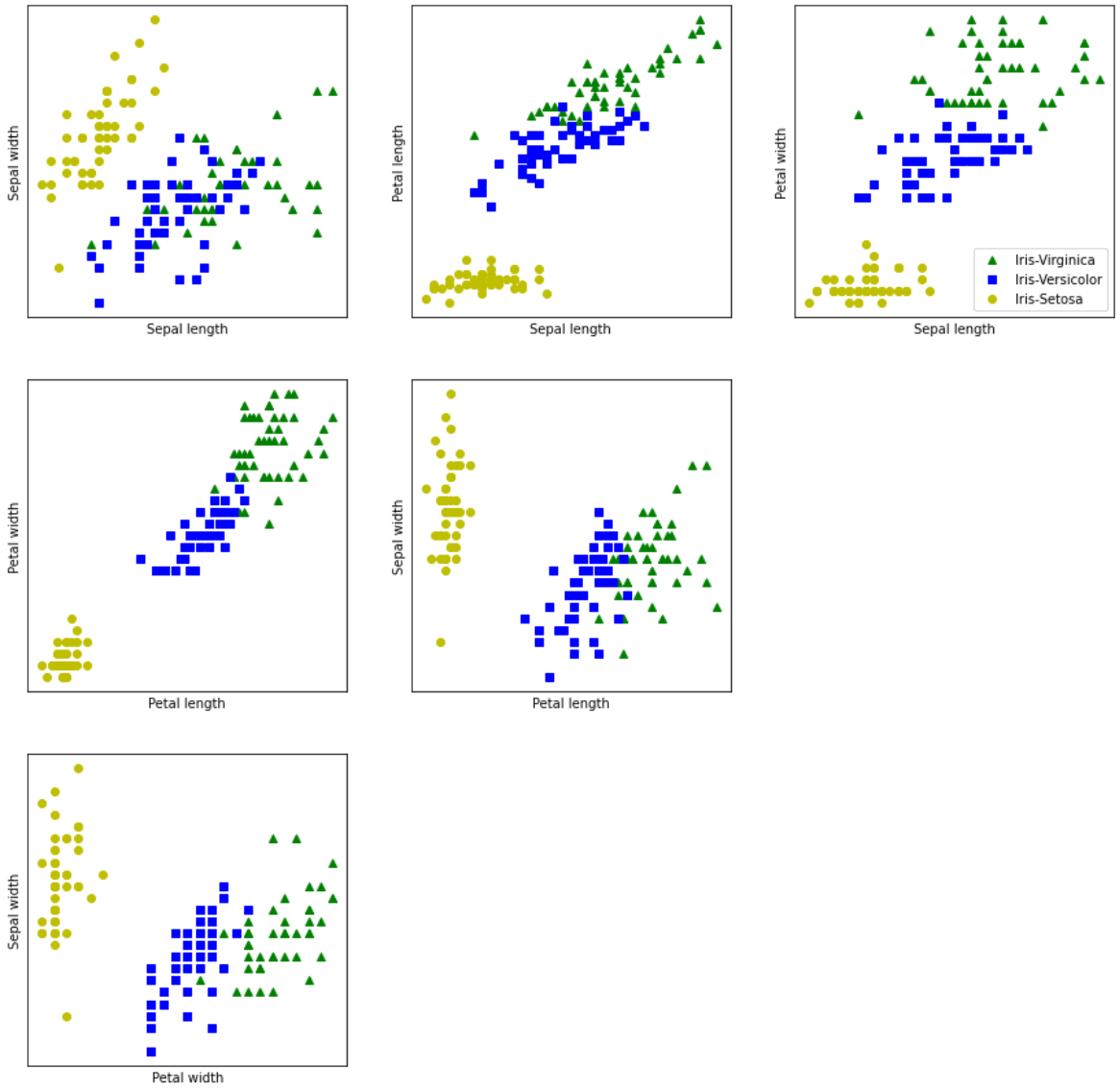
In [3]:

```
'This cell provides some plotting. It also splits the data into SepalLength,

'The following separation is basically for you :-) '
SepalLength = IRIS.data[:,0]
SepalWidth = IRIS.data[:,1]
PetalLength = IRIS.data[:,2]
PetalWidth = IRIS.data[:,3]
y = IRIS.target # 0 == Iris Setosa, 1 == Iris Versicolor, 2 == Iris Virginica
```

In [4]:

```
'All them easy plots'
plt.figure(figsize=(15,15))
plt.subplot(3, 3, 1)
plt.plot(SepalLength[y==2], SepalWidth[y==2], "g^", label="Iris-Virginica")
plt.plot(SepalLength[y==1], SepalWidth[y==1], "bs", label="Iris-Versicolor")
plt.plot(SepalLength[y==0], SepalWidth[y==0], "yo", label="Iris-Setosa")
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xticks(())
plt.yticks(())

plt.subplot(3, 3, 2)
plt.plot(SepalLength[y==2], PetalLength[y==2], "g^", label="Iris-Virginica")
plt.plot(SepalLength[y==1], PetalLength[y==1], "bs", label="Iris-Versicolor")
plt.plot(SepalLength[y==0], PetalLength[y==0], "yo", label="Iris-Setosa")
plt.xlabel('Sepal length')
plt.ylabel('Petal length')
plt.xticks(())
plt.yticks(())

plt.subplot(3, 3, 3)
plt.plot(SepalLength[y==2], PetalWidth[y==2], "g^", label="Iris-Virginica")
plt.plot(SepalLength[y==1], PetalWidth[y==1], "bs", label="Iris-Versicolor")
plt.plot(SepalLength[y==0], PetalWidth[y==0], "yo", label="Iris-Setosa")
plt.xlabel('Sepal length')
plt.ylabel('Petal width')
plt.legend()
plt.xticks(())
plt.yticks(())

plt.subplot(3, 3, 4)
plt.plot(PetalLength[y==2], PetalWidth[y==2], "g^", label="Iris-Virginica")
plt.plot(PetalLength[y==1], PetalWidth[y==1], "bs", label="Iris-Versicolor")
plt.plot(PetalLength[y==0], PetalWidth[y==0], "yo", label="Iris-Setosa")
plt.xlabel('Petal length')
plt.ylabel('Petal width')
plt.xticks(())
plt.yticks(())

plt.subplot(3, 3, 5)
plt.plot(PetalLength[y==2], SepalWidth[y==2], "g^", label="Iris-Virginica")
plt.plot(PetalLength[y==1], SepalWidth[y==1], "bs", label="Iris-Versicolor")
plt.plot(PetalLength[y==0], SepalWidth[y==0], "yo", label="Iris-Setosa")
plt.xlabel('Petal length')
plt.ylabel('Sepal width')
plt.xticks(())
plt.yticks(())

plt.subplot(3, 3, 7)
plt.plot(PetalWidth[y==2], SepalWidth[y==2], "g^", label="Iris-Virginica")
plt.plot(PetalWidth[y==1], SepalWidth[y==1], "bs", label="Iris-Versicolor")
plt.plot(PetalWidth[y==0], SepalWidth[y==0], "yo", label="Iris-Setosa")
plt.xlabel('Petal width')
plt.ylabel('Sepal width')
plt.xticks(())
plt.yticks(())
```

```
plt.show()
```



Task 1.3:

Answer the following questions:

- How does a SOM work in general?
- How can using a SOM help us represent this dataset in a more digestible way?
- How would you see a SOM working here?
- What are the parameters of the SOM that need to be tuned?
- For the parameters you listed, what range do you think the parameters should be in? Provide a brief justification for your answer

A SOM involves competitive learning where each neuron is assigned a weight vector with the same dimensionality as the input space. The input patterns are then compared to each weight vector and the closest neuron (found using Euclidean Distance) wins. This neuron is called the winning neuron. After the winning neuron is identified, the neighbour neurons enter a competitive process which excites more closer neurons to the winning neuron than those farther away from it, according to a neighborhood function. Eventually, an adaptive process changes the values of the weight vector in relation to the input vector i.e.,the winning neuron's weights are updated to values that are closer to the input vector. This entire process is repeated for n iterations.This is how a SOM generally works.
Using a SOM in this case can help us distinguish two Iris type clusters i.e., one type of Iris class separate from the other two classes. As a SOM grid is well suited for data visualization and a SOM works well for data reduction, it shall provide us with two clear clusters distinguishing one Iris type separate from the other two. In this case, a SOM shall classify each cluster as the dominant class in the cluster. The SOM in this case shall be represented as a plotted distance map of the neurons where each square in the plot shall represent one neuron, and the normalized distance from its neighbouring neurons could be represented using color.
The learning rate and the radius of the winning neuron should be tuned using the neighbourhood function. If both of these values are too high, the neurons will constantly drive around without settling down and if values are too low, the analysis will take too long because neurons will move slowly towards their optimal positions. To begin with, we should have the range of these parameters relatively larger and it should be reduced over time because the neighbourhood shall get smaller with time ensuring convergence and specialization of the output nodes. The number of iterations can also be tuned and it is mostly observed that a larger number of iterations provide better data visualization. It is all important to ensure that all the variables used in the process are standardized so that one variable does not outpower another because of different units.

Task 1.4:

The next cell provides a basic function *Train_and_Plot([X,Y], InData, steps)* to train a SOM of dimensionality X times Y to represent the input data after a number of training steps. After execution, it will plot the SOM, comparing it to the Iris labels. The SOM classifies each cluster as the dominant class in the cluster, yielding a classification accuracy.

To represent the SOM, a distance map of the neurons is plotted. Each square in this plot represents one neuron, and the color represents the normalized distance from its neighbouring neurons.

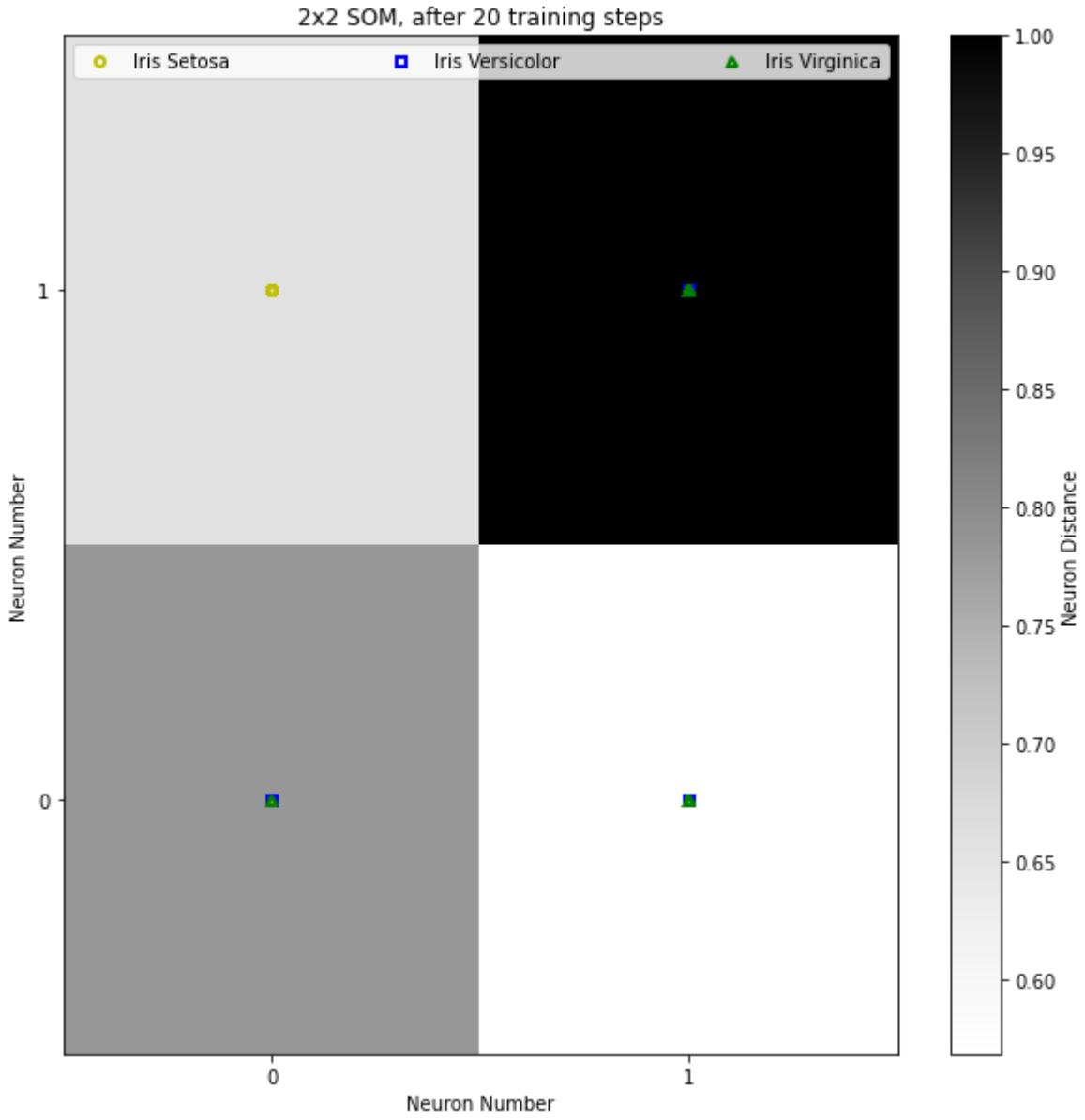
Run the cell to compile the function and the followup cell to call the function for the first time.

In [47]:

```
import math

def plot_legend():
    ''' Plots a legend for the colour scheme
    given by abc_to_rgb. Includes some code adapted
    from http://stackoverflow.com/a/6076050/637562'''

    # Basis vectors for triangle
    basis = np.array([[0.0, 1.0], [-1.5/np.sqrt(3), -0.5],[1.5/np.sqrt(3), -0

    fig = plt.figure()
    ax = fig.add_subplot(111,aspect='equal')

    # Plot points
    a, b, c = np.mgrid[0.0:1.0:50j, 0.0:1.0:50j, 0.0:1.0:50j]
    a, b, c = a.flatten(), b.flatten(), c.flatten()

    abc = np.dstack((a,b,c))[0]
    #abc = filter(lambda x: x[0]+x[1]+x[2]==1, abc) # remove points outside t
    abc = map(lambda x: x/sum(x), abc) # or just make sure points lie inside

    data = np.dot(abc, basis)
    colours = [abc_to_rgb(A=point[0],B=point[1],C=point[2]) for point in abc]

    ax.scatter(data[:,0], data[:,1],marker=',',edgecolors='none',facecolors=c

    # Plot triangle
    ax.plot([basis[_,0] for _ in range(3) + [0,]],[basis[_,1] for _ in range(

    # Plot labels at vertices
    offset = 0.25
    fontsize = 32
    ax.text(basis[0,0]*(1+offset), basis[0,1]*(1+offset), '$A$', horizontalal
            verticalalignment='center', fontsize=fontsize)
    ax.text(basis[1,0]*(1+offset), basis[1,1]*(1+offset), '$B$', horizontalal
            verticalalignment='center', fontsize=fontsize)
    ax.text(basis[2,0]*(1+offset), basis[2,1]*(1+offset), '$C$', horizontalal
            verticalalignment='center', fontsize=fontsize)

    ax.set_frame_on(False)
    ax.set_xticks(())
    ax.set_yticks(())

    plt.show()

def Train_and_Plot_SOM(dimensions, inputData, steps):
    dimensions = np.array(dimensions)
    inputData = np.transpose(inputData)
    # Create SOM
    som = MiniSom(
        int(dimensions[0]),                                  # Number of neuron
        int(dimensions[1]),                                  # Number of neuron
        np.shape(inputData)[1],             # Number of elements in an input
        sigma = .9,                         # Spread of the neighbourhood func
        learning_rate = 0.001,                 # Initial learning rate
        neighborhood_function = 'gaussian'  # Type of neighborhood function
```

```
)

som.random_weights_init(inputData)  # Initialize weights of neurons rando
som.train_random(inputData, steps)  # Train the SOM using the input vecto

# Plot distance map of the SOM
t = IRIS.target
fig = plt.figure(figsize=(10,10))
ax = plt.subplot(1, 1, 1)
plt.bone()
plt.pcolor(som.distance_map().T, cmap='Greys')
bar = plt.colorbar()
bar.set_label('Neuron Distance')

# use different colors and markers for each label
markers = ['o', 's', '^']
colors = ['y', 'b', 'g']
labels = ['Iris Setosa', 'Iris Versicolor', 'Iris Virginica']
counter = [ False, False, False]
w_class = np.zeros([dimensions[0], dimensions[1], 3])
for cnt, xx in enumerate(inputData):
    w = som.winner(xx)  # getting the winner
    w_class[w[0], w[1], t[cnt]] +=1

    # place a marker on the winning position for the sample xx
    if counter[t[cnt]] == False:
        ax.plot(w[0]+.5, w[1]+.5, markers[t[cnt]], markerfacecolor='None'
                markeredgecolor=colors[t[cnt]], markersize=5, markeredge
        counter[t[cnt]] = True
    else:
        ax.plot(w[0]+.5, w[1]+.5, markers[t[cnt]], markerfacecolor='None'
             markeredgecolor=colors[t[cnt]], markersize=5, markeredgewidt
ax.axis([0, dimensions[0], 0, dimensions[1]])
plt.xticks((np.arange(dimensions[0]) + np.ones((1, dimensions[0]))*0.5)[0
plt.yticks((np.arange(dimensions[1]) + np.ones((1, dimensions[1]))*0.5)[0
plt.xlabel('Neuron Number')
plt.ylabel('Neuron Number')
ax.legend(loc=2, mode='expand', numpoints=1, ncol=4, fancybox = True)
plt.title(''.join([str(dimensions[0]), 'x' ,str(dimensions[1]),  ' SOM, a
plt.show()


Class_acc = np.sum(np.amax(w_class, 2)) / np.sum(w_class)
print("Classification Accuracy: ", Class_acc)
```

In [6]:
```
Train_and_Plot_SOM(dimensions=[2, 2],
inputData=[SepalLength, SepalWidth], steps=20)
# inputData can be any combination of SepalLength, SepalWidth, PetalLength, P
```




```
Classification Accuracy:  0.7066666666666667
```

Task 1.5:

In your next task, please:

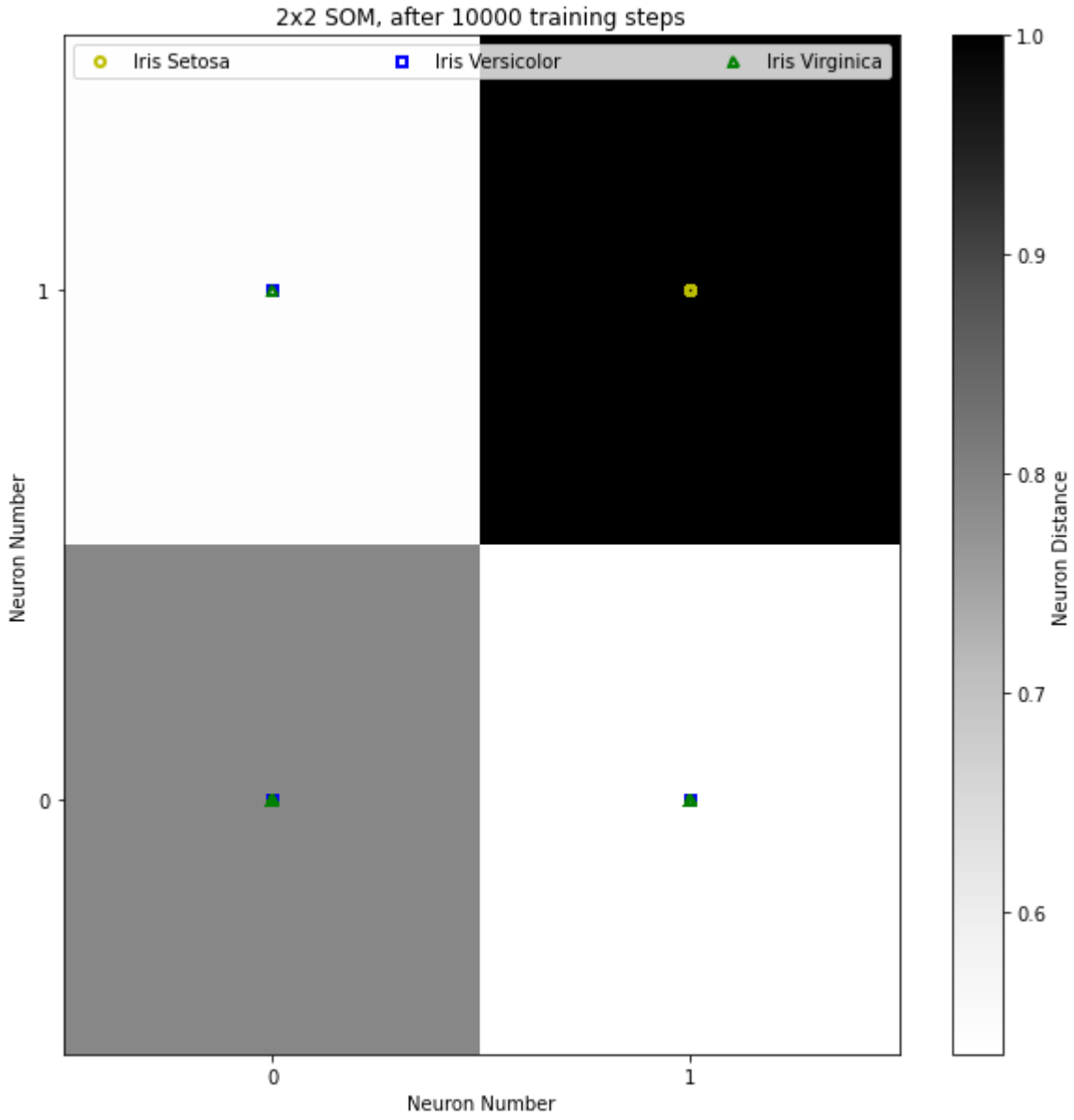
1. Comment on the quality of the above SOM. Does this provide any valuable insight? Why or why

not? What do you need to change?

The SOM obtained above is of a poor quality and does not provide any valuable insight. This can be concluded from two factors - firstly, the classification accuracy obtained is 0.7067 (approx) which implies that it is a poor model. Secondly, the number of iterations is too low to obtain a good visualization of the clusters. Since the number of iterations is very small i.e., 20 we can see that all nodes have weights close to the center of the field, which is near the average of the training set.
This is the result of selecting a large initial neighborhood, so that nearly all nodes are trained at every step. The rate of learning has to be eventually decreased. We need to change the number of iterations, i.e., we have to increase the number of iterations so that the neighbourhood shrinks, and further training provides fine adjustments so that we get a better quality SOM.

2. In the python cell below, experiment on your own with some function calls to properly cluster the samples. What parameters lead to a result that looks 'well clustered' to you?

In [40]:
```
Train_and_Plot_SOM(dimensions=[2, 2],
inputData=[SepalLength,PetalLength], steps=10000)
```




```
Classification Accuracy:  0.8933333333333333
```

Task 1.6:

Provide a few comments about how helpful you find the above SOM. Did anything surprise you?

I played with the inputData, by putting in all four parameters, i.eSepalLength,PetalLength, SepalWidth and PetalWidth and I also used a combination by using two and three parameters. Also, I eventually decreased the learning rate to 0.01 and increased the number of steps to 10000. With the abovementioned learning rate value and the number of steps, I obtained a classification accuracy of 0.893, the highest obtained out of all the combinations I tried. The SOM obtained as a result provided a clear classification showing that the Iris type Setosa is different/separate from Versicolor and Virginica. The SOM obtained is more helpful than the one obtained when the learning rate was 0.9 and steps were 20. This is because we have a good model in our case as the classification accuracy is very high. The model helps us clearly distinguish Setosa from Versiclor and Virginica. In some cases, I noticed something very surprising. Some SOMs did not show Versicolor and Virginica in a single cluster, rather there was an individual cluster for either Virginica and Versicolor along with a cluster that was a combination of both. Since we learnt from the previous lab that Virginica and Versicolor are necessarily similar in some aspects and are hard to differentiate, it was surprising to see a SOM which had a separate cluster for only Virginica or Versicolor.

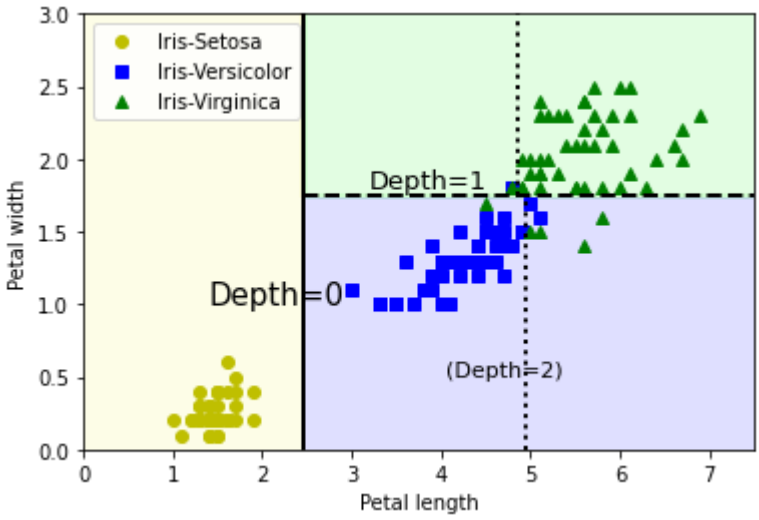
In [ ]:

```
;<b>Task 1.7:</b>
ginning with the SOM from <b>1.5.2</b> try to outperform the decision tree in
al should be to just barely match the performance of the decision tree with as
 cell below to initialize your benchmark decision tree, then see how well SOM
```

In [41]:

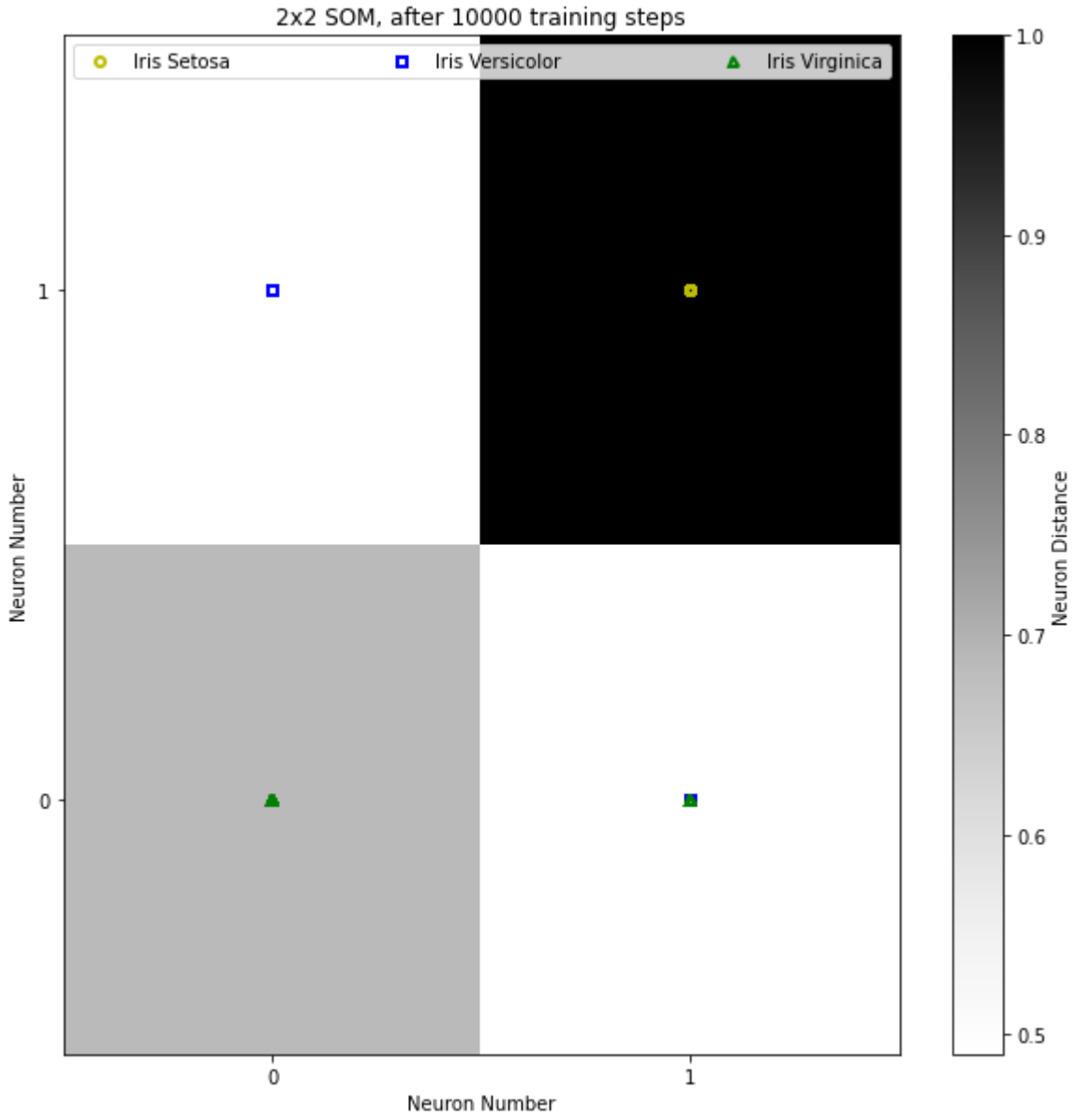
```
'Building a Decision Tree'
Benchmark_tree = DecisionTreeClassifier(max_depth=2, random_state=42)
Benchmark_tree.fit(IRIS.data[:,2:], IRIS.target)
print("This Decision tree's accuracy is: ", Benchmark_tree.score(IRIS.data[:,

'Plotting the Decision boundaries'
from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[0, 7.5, 0, 3]):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])

    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", label="Iris-Setosa")
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", label="Iris-Versicolor")
    plt.plot(X[:, 0][y==2], X[:, 1][y==2], "g^", label="Iris-Virginica")
    plt.axis(axes)
    plt.xlabel("Petal length")
    plt.ylabel("Petal width")

plot_decision_boundary(Benchmark_tree, IRIS.data[:,2:], IRIS.target)
plt.plot([2.45, 2.45], [0, 3], "k-", linewidth=2)
plt.plot([2.45, 7.5], [1.75, 1.75], "k--", linewidth=2)
plt.plot([4.95, 4.95], [0, 1.75], "k:", linewidth=2)
plt.plot([4.85, 4.85], [1.75, 3], "k:", linewidth=2)
plt.text(1.40, 1.0, "Depth=0", fontsize=15)
plt.text(3.2, 1.80, "Depth=1", fontsize=13)
plt.text(4.05, 0.5, "(Depth=2)", fontsize=11)
plt.legend()

plt.show()
```

```
This Decision tree's accuracy is:  0.96
```



Match the performance of the decision tree in the cell below

In [48]:
```
Train_and_Plot_SOM(dimensions=[2, 2],
inputData=[PetalWidth,PetalLength], steps=10000)
```




```
Classification Accuracy:  0.9533333333333334
```

Provide your answer in the cell below:

```
I changed the following parameters to outperform the decision tree - learning
rate to 0.001, inputData as [PetalWidth,PetalLength] and number of steps to
10000. This change in the parameters gave a classification accuracy of 0.953
which is very close to the accuracy of the decision tree i.e., 0.96. The SOM
could almost potentially beat the decision tree. The resulting SOM provided a
clear classification for Virginica, Versicolor and Setosa individually and
also for Virginica and Verscilor separately. It fared very well as a
classifier because in the previous lab, we couldn’t observe a separate
classification for Virginica and Versicolor separately, however through the
obtained SOM we could see their classification both separately and together.
This was indeed surprising compared to the previous lab!
```

Exercise 2: Self-organizing maps with real-world energy data

In this exercise, you will be tasked to explore a provided dataset a little more independently. The dataset that you will be working with is energy consumption data of an industrial park. The park's owner has heard that you are taking an artificial intelligence course and asks you to use 'this machine learning stuff' to help with analyzing patterns in the park's energy consumption. Luckily, you remember that SOMs are considered 'machine learning stuff', and that they could actually be used for such a task. The dataset that the park provided looks like this:

- *Index 0*: Day of the week (0, 1,..., 6)
- *Index 1*: Month (1, 2,..., 12)
- *Index 2*: Holiday (1 = workday, 2 = holiday)
- *Index 3 - 26*: Energy consumed for each hour of the day in kWh (on the order of 10^{14})

Task 2.1:

Run the cells below. The first cell loads the dataset, does some slight preprocessing and packages it all in one array per day, separated into months. The dataset is then organized in the following way: February contains 28 arrays, for the 28 days in February. Each of those days, has in the 0th index the Weekday, the 1st index the Month, the 2nd index the type of day, and the rest is 24 power values for each hour of the day.
The second cell defines the *trainSOM* and *plotSOM* functions, and the third cell calls it for an exemplary dataset.

In addition to the usual distance map, an activation map is plotted, with the color for each neuron indicating how many times a neuron was activated.

In [49]:

```
# Load input data
dataset = sio.loadmat('somdata.mat')
dayInfo = dataset['dayInfo'].astype(float)
rawHourlyEnergy = dataset['rawHourlyEnergy']

# Scale the data to the interval [0, 1] for each dimension
iScaler = preprocessing.MinMaxScaler()
rawHourlyEnergy = iScaler.fit_transform(rawHourlyEnergy)
dayInfo[:,0] = (1.0/np.amax(dayInfo[:,0]))*dayInfo[:,0]
dayInfo[:,1] = (1.0/np.amax(dayInfo[:,1]))*dayInfo[:,1]
dayInfo[:,2] = (1.0/np.amax(dayInfo[:,2]))*dayInfo[:,2]

# Combine variables to obtain full input vector arrays
inputData = np.hstack((dayInfo, rawHourlyEnergy))

# Split the data into months
january = inputData[0:31, :]
february = inputData[31:59, :]
march = inputData[59:90, :]
april = inputData[90:120, :]
may = inputData[120:151, :]
june = inputData[151:181, :]
july = inputData[181:212, :]
august = inputData[212:243, :]
september = inputData[243:273, :]
october = inputData[273:304, :]
november = inputData[304:334, :]
december = inputData[334:, :]
```

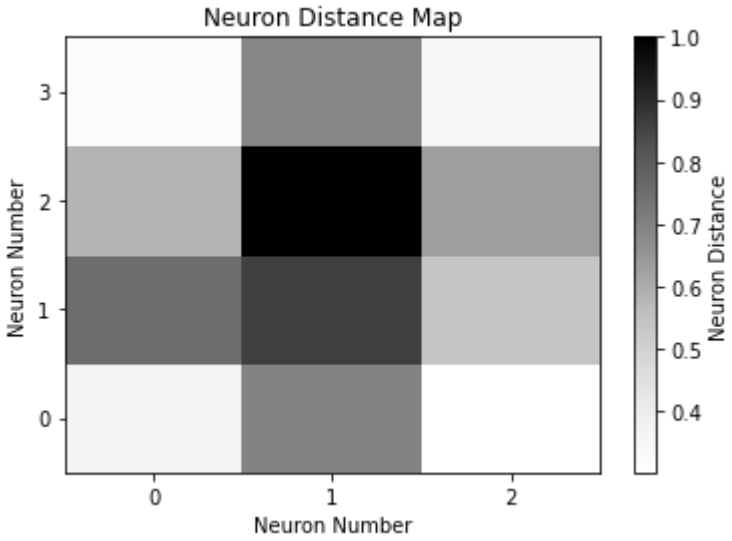
In [50]:

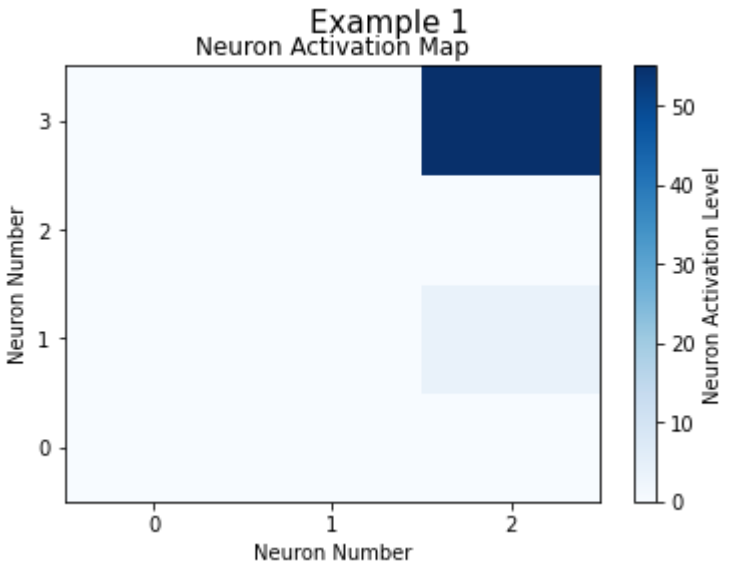
```
'Trains a SOM'
def trainSOM(dimensions, inputData, steps):
    dimensions = np.array(dimensions) # From list to np.array

    'Create SOM'
    som = MiniSom(
        int(dimensions[0]),                    # Number of neurons (x-axis)
        int(dimensions[1]),                    # Number of neurons (y-axis)
        np.shape(inputData)[1],                # Number of elements in an in
        sigma = .5,                            # Spread of the neighbourhood
        learning_rate = 0.5,                   # Initial learning rate
        neighborhood_function = 'gaussian',    # Type of neighborhood functi
        random_seed = 42)                      # Random seed

    'Train SOM'
    som.random_weights_init(inputData)  # Initialize weights of neurons rando
    som.train_random(inputData, steps)  # Train the SOM using the input vecto

    x = som._activation_map.shape[0]
    y = som._activation_map.shape[1]

    'Plot the SOM distance map'
    f = plt.figure()
    ax = f.add_subplot(1, 1, 1)
    im = ax.pcolor(som.distance_map().T, cmap='Greys')
    bar = plt.colorbar(im, ax=ax)
    bar.set_label('Neuron Distance')
    ax.axis([0, x, 0, y])
    ax.set_xticks((np.arange(x) + np.ones((1, x))*0.5)[0])
    ax.set_xticklabels(np.arange(x))
    ax.set_yticks((np.arange(y) + np.ones((1, y))*0.5)[0])
    ax.set_yticklabels(np.arange(y))
    ax.set_xlabel('Neuron Number')
    ax.set_ylabel('Neuron Number')
    ax.set_title("Neuron Distance Map")
    return som

'Plots distance and activation maps'
def plotSOM(som, inputCollection, name):
    'Stacking the input data'
    inputData = np.vstack(inputCollection)

    x = som._activation_map.shape[0]
    y = som._activation_map.shape[1]

    'Plot the SOM activation map'
    f = plt.figure()
    ax2 = f.add_subplot(1, 1, 1)
    im2 = ax2.pcolor(som.activation_response(inputData).T, cmap='Blues')  # P
    bar2 = plt.colorbar(im2, ax=ax2)
    bar2.set_label('Neuron Activation Level')
    ax2.axis([0, x, 0, y])
    ax2.set_xticks((np.arange(x) + np.ones((1, x))*0.5)[0])
    ax2.set_xticklabels(np.arange(x))
    ax2.set_yticks((np.arange(y) + np.ones((1, y))*0.5)[0])
    ax2.set_yticklabels(np.arange(y))
```

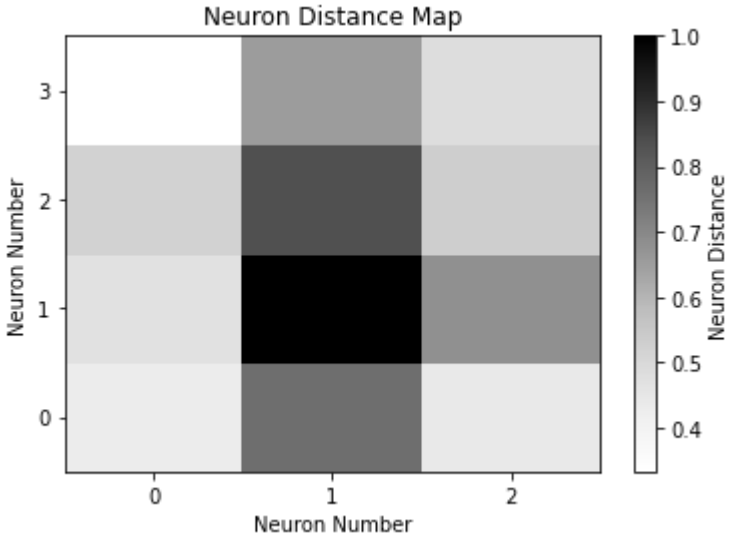
```
    ax2.set_xlabel('Neuron Number')
    ax2.set_ylabel('Neuron Number')
    ax2.set_title("Neuron Activation Map")
    plt.suptitle(name, size=15)
    plt.show()
```

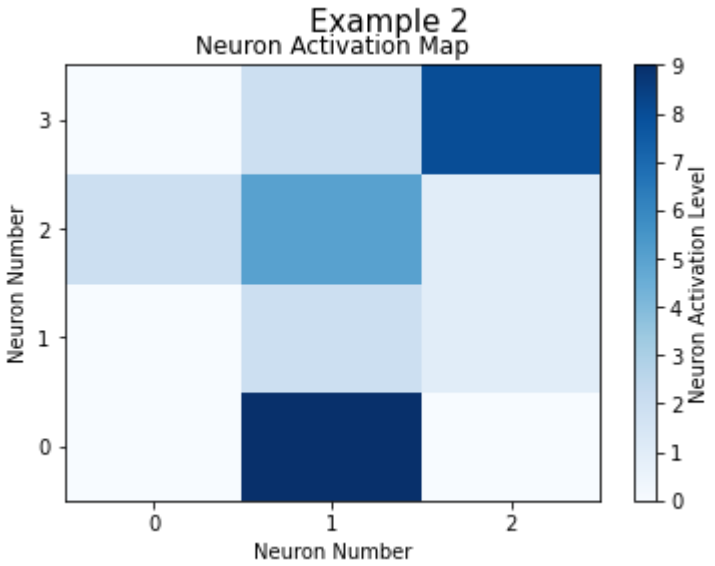
In [51]:

```
som1 = trainSOM([3,4], inputData[:, 1:20], 2000)
plotSOM(som1, [january[:, 1:20],
    february[:, 1:20]], 'Example 1')

som2 = trainSOM([3,4], inputData[151:243, 1:25], 1870)
plotSOM(som2, [june[:, 1:25]], 'Example 2')
```









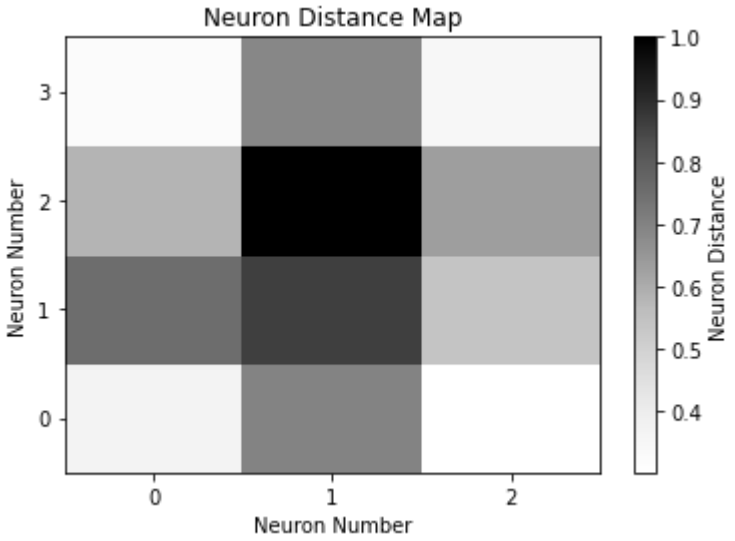
Task 2.2:

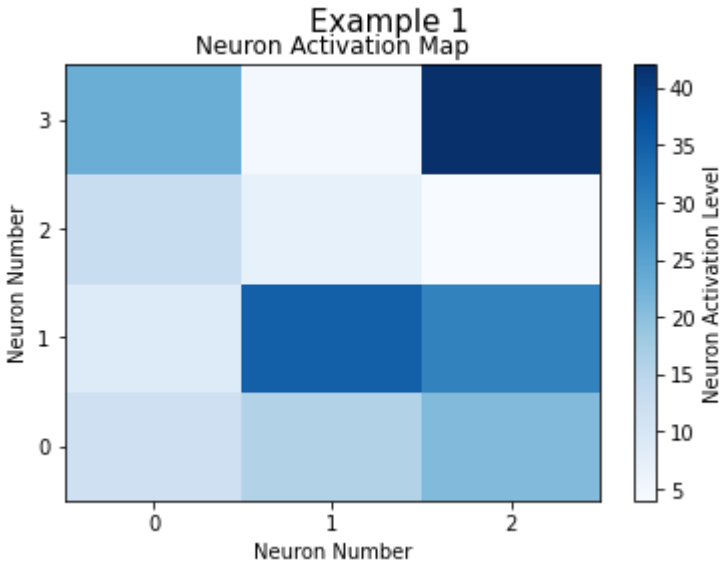
Ok, now it's your turn! You have to find some patterns in the data. This is purposely a more open ended question! Some ideas worth thinking about:

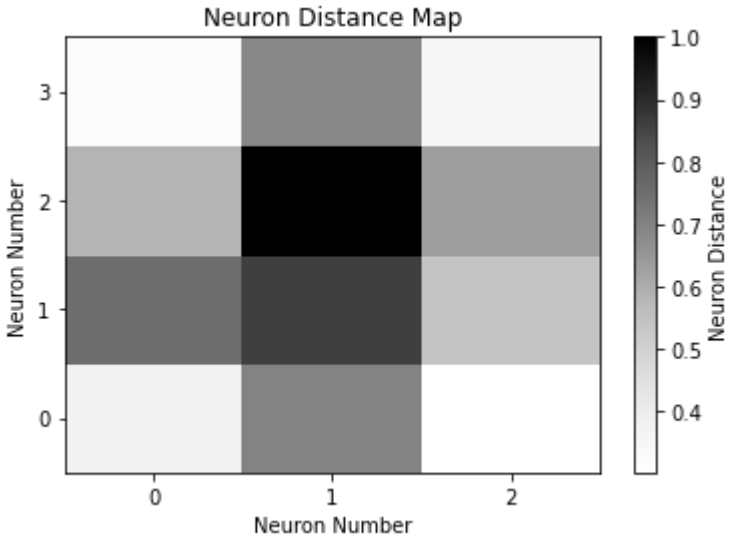
- Look for seasonal dependencies. Maybe January and December look similar, while June will look completely different?
- Maybe it is easier if you omit some data (like the month identifier?) to raise the sensitivity to other data?
- Possibly it makes sense to bundle similar months?
- To start, it could be useful to plot all months by themselves at first?

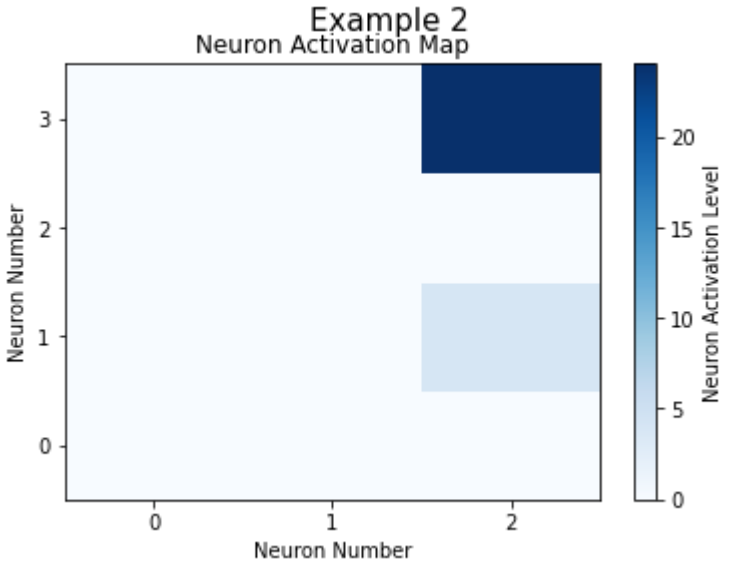
In the cell below, please provide your function calls.

In [72]:
```
som1 = trainSOM([3,4], inputData[:, 1:20], 2000)
plotSOM(som1, [january[:, 1:20], march[:,1:20],
may[:,1:20], july[:,1:20], august[:,1:20],
october[:,1:20],december[:, 1:20]], 'Example 1')

som2 = trainSOM([3,4], inputData[:, 1:20], 2000)
plotSOM(som2, [february[:, 1:20]], 'Example 2')
```









What are the insights you gained (if any) and what was your thought process behind your solution? Remember we cannot read your thoughts, so everything you communicate that sounds like smart, logical engineering behavior is god input :-) (i.e. Deus Ex Machina).

Additionally:
Be critical, what problems do you see with using SOM in such a task? What do you think the SOM learns to represent, etc...

I used multiple combinations of functions some of which were - initially using january and december for Example1 and june for Example2, then all months with 31 days i.e., january, may, july, august, october for Example1 and all days with 30 days i.e., april, june, september and november for example 2. I used different combinations of months with 31 days and 30 days for both the functions respectively. Initially, for the first couple of runs, I could see a clear difference in the two patterns. For instance, when I used january [:, 1:20] and december [:, 1:20] as input parameters for the first function and june[:, 1:20] for the second, I could see a clear difference in the patterns obtained from the two functions. However, as I proceeded with different function calls, the two patterns became indistinguishable. Therefore, I could observe the difference in the patterns in the case mentioned above, but eventually with more and more function calls the patterns were indistinguishable.The insight I gained from this was that SOM was not the best choice for classification in this task.
My thought process was to look for months that have the same number of days, so I used a combination of months with 31 days in the first function and a combination of months with 30 days in the second, i.e., bundled similar months . I also omitted the month identifiers to raise the sensitivity to other data. Eventually I used months with 31 days and no month identifiers for the first function call and the month of february (quite different as it has only 28 days) with no month identifiers for the second. I also kept the number of iterations the same throughout the function calls. In the described function calls, I could not see any difference in the patterns obtained.
There are certain problems in using SOM in this task. A SOM requires necessary and sufficient data in order to develop meaningful clusters. Lack of data or too much data in the weight vectors can add randomness to the groupings. Finding the correct data involves determining which factors are relevant and can be a difficult task for several classification problems. Thus, the ability to determine a good data set is a deciding factor in determining whether to use a SOM or not.
The size, values for the training parameters and topology of the map also need to be determined. We must spend time on the optimization of the mapping even though there is usually a wide range of settings that provide comparable results. It is often difficult to obtain a perfect mapping where groupings are unique. Since, the process of using a SOM has several random components like the initialization and the order of the presentation of the data thus, in many cases, the conclusions drawn from different maps may still be similar like in our case for this task.

Abstract

Self-organizing maps (SOMs) / Kohonen networks are neural networks that employ unsupervised competitive learning. Therefore, no targets are presented to the network for training and after a competitive process, only one output neuron called the winning neuron is activated. SOMs usually map high dimensional input vectors onto a lower dimension map through two-dimensional lattices where the neurons are placed on the nodes. A practical use of SOMs is usually for clustering.

A SOM usually works in the following way - each neuron is assigned a weight vector with the same dimensionality as the input space. The input patterns are then compared to each weight vector and the closest neuron called the winning neuron (found using Euclidean Distance) wins. After the winning neuron is identified, the neighbour neurons enter a competitive process which excites more closer neurons to the winning neuron than those farther away from it, according to a neighborhood function. Eventually, an adaptive process changes the values of the weight vector in relation to the input vector i.e.,the winning neuron's weights are updated to values that are closer to the input vector. This entire process is repeated for n iterations. To visualize the results of the training, a distance map is used which plots the average distance of the weights and the coordinates of the associated winning neuron. An activation map is also used that returns a matrix showing the number of times each neuron was activated.
A SOM requires necessary and sufficient data in order to develop meaningful clusters. Lack of data or too much data in the weight vectors can add randomness to the groupings. The size, values for the training parameters and topology of the map also need to be determined. We must spend time on the optimization of the mapping even though there is usually a wide range of settings that provide comparable results. These factors often become a problem in certain tasks where SOMs don’t necessarily work.

Introduction

In this lab, we used 2 multidimensional datasets that required the development of an intuitive understanding about what was happening in the datasets. Thus, we performed dimensionality reduction using SOMs.
In Exercise 1, we used the Iris dataset like in the previous lab. A SOM was used on the Iris dataset to cluster and to classify different types of Iris, i.e., Setosa, Versicolor and Virginica. We also tried outperforming an alternative classification tool i.e., a decision tree to see how a SOM fares as a classifier.
In Exercise 2, we worked with the energy consumption data of an industrial park for pattern recognition and interpretation.The data consisted of the day of the week, month, holiday and the energy consumed at each hour of the day. In this task we had the independence of finding patterns in the data using our own thought process. We played around with functions trainSOM and plotSOM, by feeding them input parameters of our own choice for pattern recognition. Similar months were bundled together for the pattern recognition and some data was omitted to raise the sensitivity to other data. SOM activation maps were also used in this case.

Conclusion

Two machine learning classification tasks were performed in the lab using 2 different multidimensional datasets: Iris dataset and the energy consumption data of an industrial park. Dimensionality reduction was performed by using Self-organizing maps (SOMs) / Kohonen networks for both the exercises. To visualize the results of the training, a distance map was used in both the exercises while an activation map was also used in the second exercise.

In exercise1, we discovered how a SOM works in general, how it can be used in representing the Iris dataset in a more digestible way and what parameters of the SOM shall be tuned with what possible range. We further plotted a SOM distance map for the Iris dataset which also yielded a classification accuracy. Through this, we could judge how accurately the classification of a SOM works with different parameters values. It was observed that a lower learning rate and a higher number of iterations yield a good classification model, however, a higher learning rate with low number of iterations yields a poor classification model. We were also surprised by seeing how a SOM could classify the Versicolor and Verginica class of Iris both separately and together, as in lab1 we only saw their classification together. We also outperformed the decision tree with a close margin by lowering the value of the learning rate and increasing the number of iterations.

In exercise2, we independently chose the parameters for pattern recognition and interpretation for the energy consumption data of an industrial park. Using our own thought process, we played around with functions trainSOM and plotSOM, by feeding them input parameters of our own choice for pattern recognition. With our thought process, we used a combination of months with 31 days in the first function and a combination of months with 30 days in the second, i.e., bundled similar months. The month identifiers were also omitted to raise the sensitivity to other data. Eventually, months with 31 days and no month identifiers were used for the first function call and the month of february (quite different as it has only 28 days) with no month identifiers was used for the second. The number of iterations were kept the same throughout the function calls. We could not observe any difference in the patterns obtained and it was concluded that SOM was not the best choice as a classifier for the problem as the ability to determine a good data set is a deciding factor in determining whether to use a SOM or not. A SOM requires necessary and sufficient data in order to develop meaningful clusters and a lack of data or too much data in the weight vectors can add randomness to the groupings. Also, the size, values for the training parameters and topology of the map need to be determined in a SOM so, we must spend time on the optimization of the mapping even though there is usually a wide range of settings that provide comparable results. Since, the process of using a SOM has several random components like the initialization and the order of the presentation of the data thus, we observed that the conclusions drawn from the two different maps in exercise 2 were similar.

Lab 4 Marking Guide

Exercise	Item	Total Marks	Earned Marks
	Pre – lab	10	
	Abstract	3	
	Introduction	3	
	Conclusion	4	
1	*Clustering*	63	
2	*Patternrecognition*	17	
	TOTAL	100	