ECE 449 - Intelligent Systems Engineering

Lab 2 - D41: Neural Networks - Self-Organizing Maps

Lab date: *Thursday, October 7, 2021 -- 2:00 - 4:50 PM*
Room: *ETLC E5-013*
Lab report due: *Wednesday, October 20, 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(**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 installing the library for the first time!
```

```
Collecting minisom
  Downloading MiniSom-2.2.9.tar.gz (8.1 kB)
Building wheels for collected packages: minisom
  Building wheel for minisom (setup.py): started
  Building wheel for minisom (setup.py): finished with status 'done'
  Created wheel for minisom: filename=MiniSom-2.2.9-py3-none-any.whl size=8594 sha256=b7205bb8151b318168bc3777f69
0ef2fcf1c11264721ff7f6f495db36851ed27
  Stored in directory: /home/jupyter/.cache/pip/wheels/f7/c1/13/a89711e5641aedaacbfe71023fba586b9b45d848d3b02192f
8
Successfully built minisom
Installing collected packages: minisom
Successfully installed minisom-2.2.9
```

In [6]:
```
'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

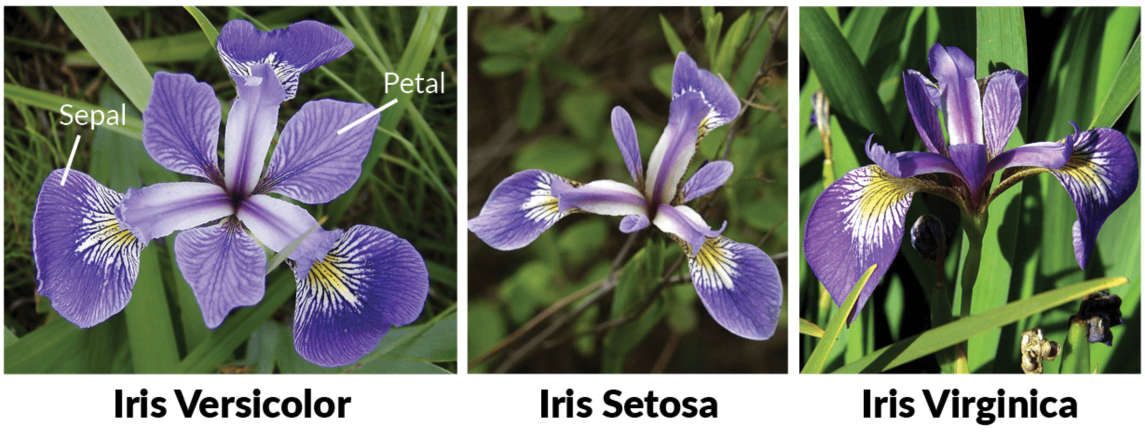
Task 1.1: During this exercise you will train a SOM to cluster the dataset. As a benchmark against classification, your SOM will compete with a simple decision tree. The exercise provides you with most of the functions that you will need. The goal is to develop a baseline understanding what SOMs are 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 [7]:

```python
'Use Scikit-learn to import the Iris Dataset, print a little literature on the set'
from sklearn import datasets
IRIS = datasets.load_iris()
print(IRIS.DESCR)

'What is a petal, what is a sepal?! ... Versicolor/Setosa/Virginica what?! ... What is an Iris flower???? '
from IPython.display import Image
from IPython.core.display import HTML
Image(url= "https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Machine+Learning+R/iris-machinelearning.png")
```

```
.. _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 taken
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 and
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 to 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 Analysis.
     (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 Transactions
     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[7]:



Task 1.2:

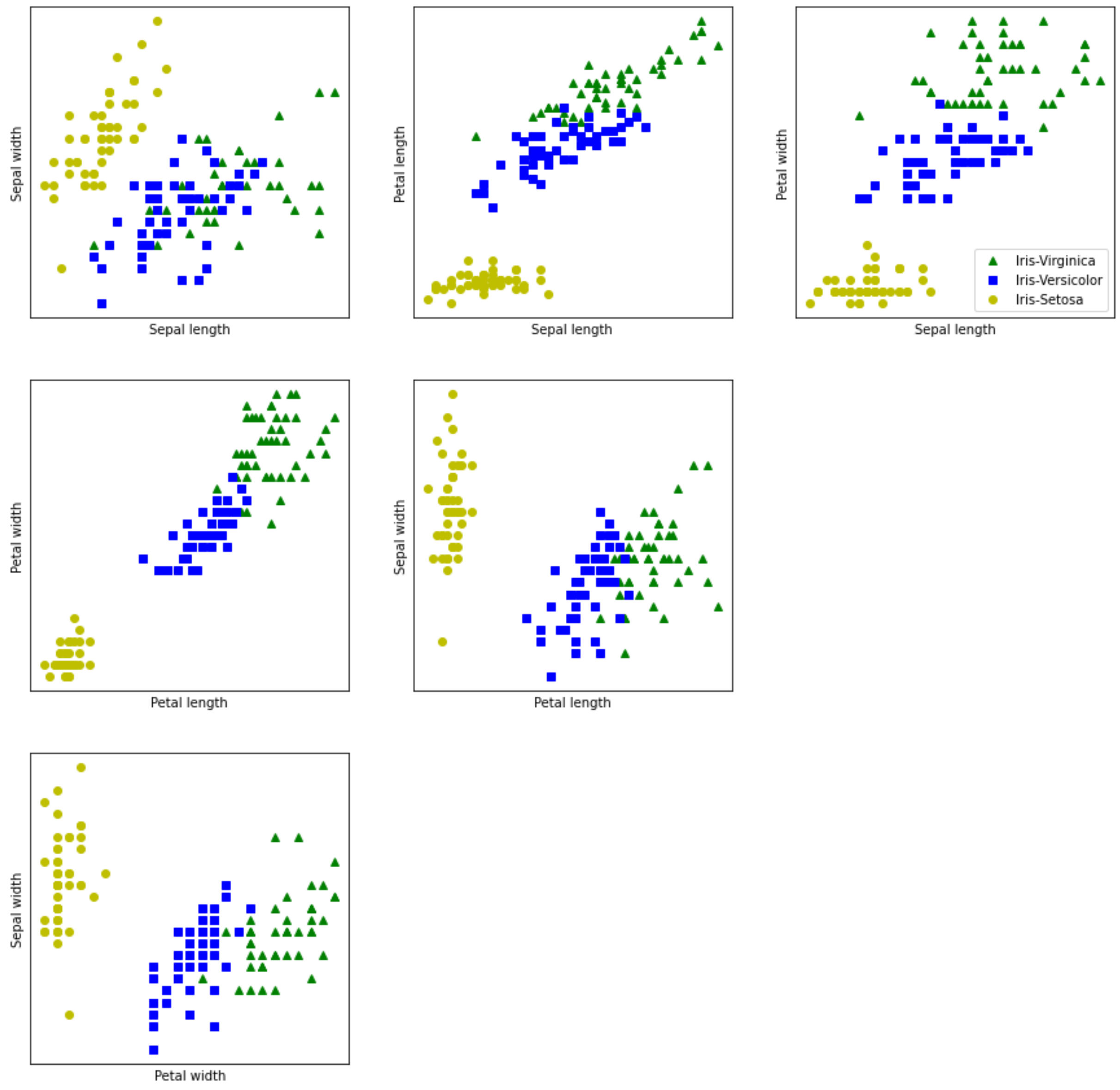
Run the next cells to provide the 'naive' plots of the dataset's dimensional relations (created by projecting the data onto a two-dimensional plane):

- SepalLength
- SepalWidth
- PetalLength
- PetalWidth

In [8]:

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

'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 [9]:
```
'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

```
Self-organizing map is one of the many neural networks and it uses competitive learning to train. It is a form of
unsupervised learning which means that when inputs are provided, no corresponding targets are provided for training.
Due to the competitive learning method that it follows, only one output (winning) neuron is active per iteration.
Via a neighbourhood function, closer neurons to the winning neuron are excited. The winning neuron's weights are re-
adjusted such that they are closer to the input vector. The exact same procedure is repeated for a number of
iterations. In the end, a matrix is returned showing the number of times each neuron has been activated.
```

```
Self-organizing maps (SOM) can help us represent this dataset in a more digestable way, since the dataset is multi-
dimensional one, an SOM can help via dimensionality reduction. This dimension reduction is done by mapping
(projecting) the multi-dimensional data onto a 2-d topological space (SOM grid).
The SOM would work here since it can be used to classify and cluster the provided Iris data-set into one of the
three Iris classes. This is possible since if we look at the above graphs the data is separable (not linearly for
all), therefore classification is possible.
The parameters that will be tuned are: the learning rate, the neighborhood function, the intialized weights of the
neurons, dimension X of the SOM, dimension Y of the SOM, the number of training steps (iterations) and the
combination of input data provided to the SOM.
For the parameters listed:
- dimensions X and Y shouldnt be too large since we are only classifying 3 iris species.
- the number of training steps shouldnt be too great since the data is visually already separable from the 'naive'
plots above, we shouldnt need a great number of steps to separate the data.
- the learning rate shouldnt be too high as it would lead to unstable outcomes (divergence) and not too low such
that learning takes too many iterations.
- the neighbourhood function used should be a gaussian function since closer neurons to the winning neuron should be
excited and the gaussian is a radial based function which is what we need for SOMs.
- the combination of input data should be decided on using the 'naive' plots, we should use the ones where the
different types of markers are most separated such as Petal Width-Petal Length.
```

Task 1.4:

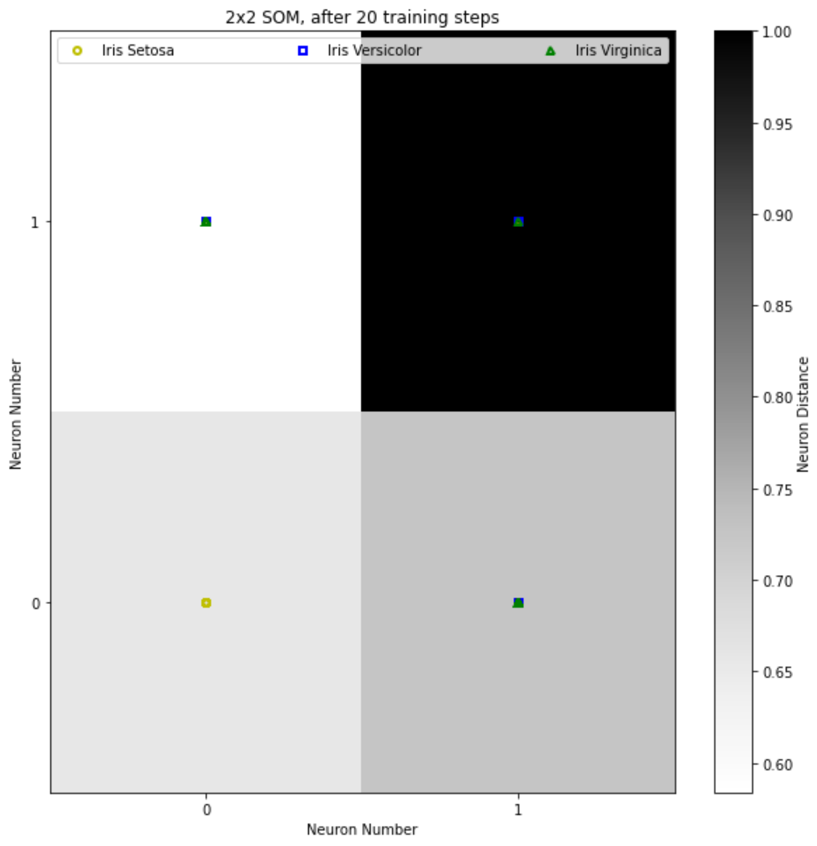
The next cell provides a basic function *Train_and_Plot([X,Y], InData, steps)* to train a SOM of dimensionality X times Y to represent the input data after a number of training steps. After execution, it will plot the SOM, comparing it to the Iris labels. The SOM classifies each cluster as the dominant 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 [10]:
```python
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), -0.5]])

    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 triangle
    abc = map(lambda x: x/sum(x), abc) # or just make sure points lie inside triangle ...

    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=colours)

    # Plot triangle
    ax.plot([basis[_,0] for _ in range(3) + [0,]],[basis[_,1] for _ in range(3) + [0,]],**{'color':'black','linewi

    # Plot labels at vertices
    offset = 0.25
    fontsize = 32
    ax.text(basis[0,0]*(1+offset), basis[0,1]*(1+offset), '$A$', horizontalalignment='center',
            verticalalignment='center', fontsize=fontsize)
    ax.text(basis[1,0]*(1+offset), basis[1,1]*(1+offset), '$B$', horizontalalignment='center',
            verticalalignment='center', fontsize=fontsize)
    ax.text(basis[2,0]*(1+offset), basis[2,1]*(1+offset), '$C$', horizontalalignment='center',
            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 neurons (x-axis)
        int(dimensions[1]),                                # Number of neurons (y-axis)
        np.shape(inputData)[1],              # Number of elements in an input vector
        sigma = .9,                         # Spread of the neighbourhood function
        learning_rate = 0.5,                 # Initial learning rate
        neighborhood_function = 'gaussian'   # Type of neighborhood function
    )

    som.random_weights_init(inputData)  # Initialize weights of neurons randomly
    som.train_random(inputData, steps)  # Train the SOM using the input vectors in a random order (i.e. not batch

    # 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, markeredgewidth=2, label=labels[t[cnt]])
            counter[t[cnt]] = True
        else:
            ax.plot(w[0]+.5, w[1]+.5, markers[t[cnt]], markerfacecolor='None',
                 markeredgecolor=colors[t[cnt]], markersize=5, markeredgewidth=2)
    ax.axis([0, dimensions[0], 0, dimensions[1]])
    plt.xticks((np.arange(dimensions[0]) + np.ones((1, dimensions[0]))*0.5)[0], np.arange(dimensions[0]))
    plt.yticks((np.arange(dimensions[1]) + np.ones((1, dimensions[1]))*0.5)[0], np.arange(dimensions[1]))
    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, after ' , str(steps) ,' training steps
    plt.show()


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

In [18]:
```
Train_and_Plot_SOM(dimensions=[2, 2], inputData=[SepalLength, SepalWidth], steps=20)
# inputData can be any combination of SepalLength, SepalWidth, PetalLength, PetalWidth
```



```
Classification Accuracy:  0.64
```

Task 1.5:

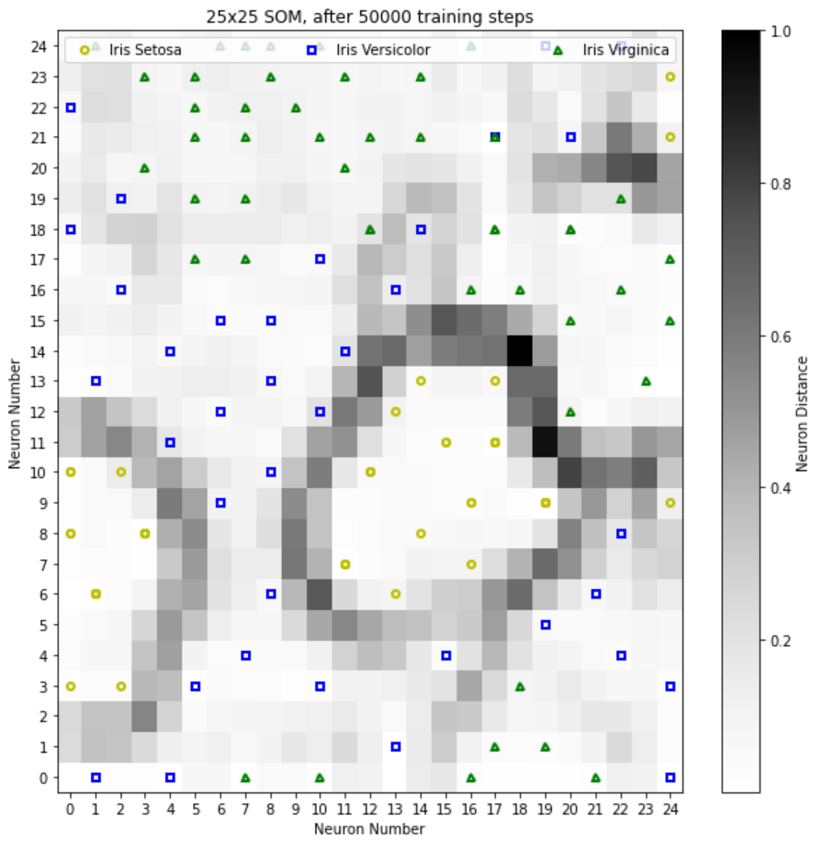
In your next task, please:

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 produced above from using a 2x2 SOM and 20 steps was only able to identify the Iris Setosa specie from the
dataset. Other than that, it is not providing much valuable insight since it is clustering Iris Versicolor-Iris
Virginica together due to the blue square and green triangle overlapping. Upon consecutive runs, it seems to be only
able to classify Iris Setosa (sometimes it can't even do that) and the accuracy on this run is 64% which is not
great at all for this classification task. For this SOM classifier to be more useful, some of the properties such as
dimension X and Y of the SOM grid need to be increased and the number of steps (iterations) can also be increased.
The combination of input data provided can be changed, in this case Sepal Length and Sepal Width are provided as
input data, however, when looking at the previous ('naive') plot of of Sepal Width against Sepal Length we can see
that there isnt a great separation of the data points of the three iris species for classification, a better set of
input data would be for example Petal Width-Sepal Length as from the plot of Petal Width against Sepal length there
is good separation between the data points of different species.
```

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 [50]:
```python
# Change input data set combination, better
# Train_and_Plot_SOM(dimensions=[2, 2], inputData=[PetalWidth, SepalLength], steps=20)

# Increase number of training steps, better
# Train_and_Plot_SOM(dimensions=[2, 2], inputData=[PetalWidth, SepalLength], steps=100)

# Increase dimension of SOM grid
# Train_and_Plot_SOM(dimensions=[25, 25], inputData=[PetalWidth, SepalLength], steps=100)

# Increase number of steps
# Train_and_Plot_SOM(dimensions=[25, 25], inputData=[PetalWidth, SepalLength], steps=10000)

# Increase number of steps
# Train_and_Plot_SOM(dimensions=[25, 25], inputData=[PetalWidth, SepalLength], steps=30000)

# Change input data set combination
# Train_and_Plot_SOM(dimensions=[25, 25], inputData=[PetalWidth, PetalLength], steps=30000)

# Increase number of steps, the below parameters lead to a well clustered SOM
Train_and_Plot_SOM(dimensions=[25, 25], inputData=[PetalWidth, PetalLength], steps=50000)
```



```
Classification Accuracy:  0.9933333333333333
```

Task 1.6:

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

The above SOM was helpful and useful for this classification task after altering the parameters such as the X and Y dimensions of the SOM grid, the input data combination and the number of learning steps (iterations). After changing the X-Y dimensions of the SOM from 2x2 to 25x25, changing the number of steps from 20 steps to 50000 steps and the input data combination from Sepal Length-Sepal Width to Petal Width-Petal Length, I was able to obtain a classification accuracy of 99.3% which is an increase from the original 64.0%. From the diagram above we can see that all Iris species have been correctly classified since there are no markers overlapping on the grid. An interesting observation was how some of the classified data was clustered via regions of increasing neuron distances which are darker in color in the above SOM (look at the 'circular' dark region around the classified Iris Setosa points in the middle on the above SOM grid). What's surprising is that this SOM still classfies Iris Setosa well, however, there is one area (marker) of the SOM grid where the triangle and square still overlap. This could be because from the start in this lab, from the output of line "print(IRIS.DESCR)", we were told that one class is linearly separable from the other two while the two are not linearly separable, the separable class could be Iris Setosa while the other two could be the non-linearly separable.

Task 1.7:

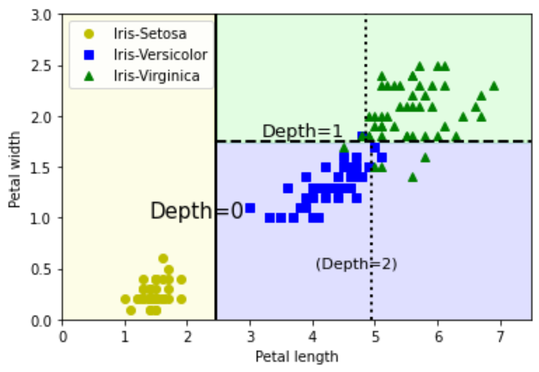
Now, beginning with the SOM from **1.5.2** try to outperform the decision tree in the cell below. The decision tree is probably the simplest way to classify a dataset.

Your goal should be to just barely match the performance of the decision tree with as few parameters as possible! Run the cell below to initialize your benchmark decision tree, then see how well SOM fares as a classifier, and finally comment on your resulting SOM. Is it smaller or larger than the SOM from **1.5.2**, did that surprise you?

In [30]:

```python
'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[:,2:], IRIS.target))

'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()
```

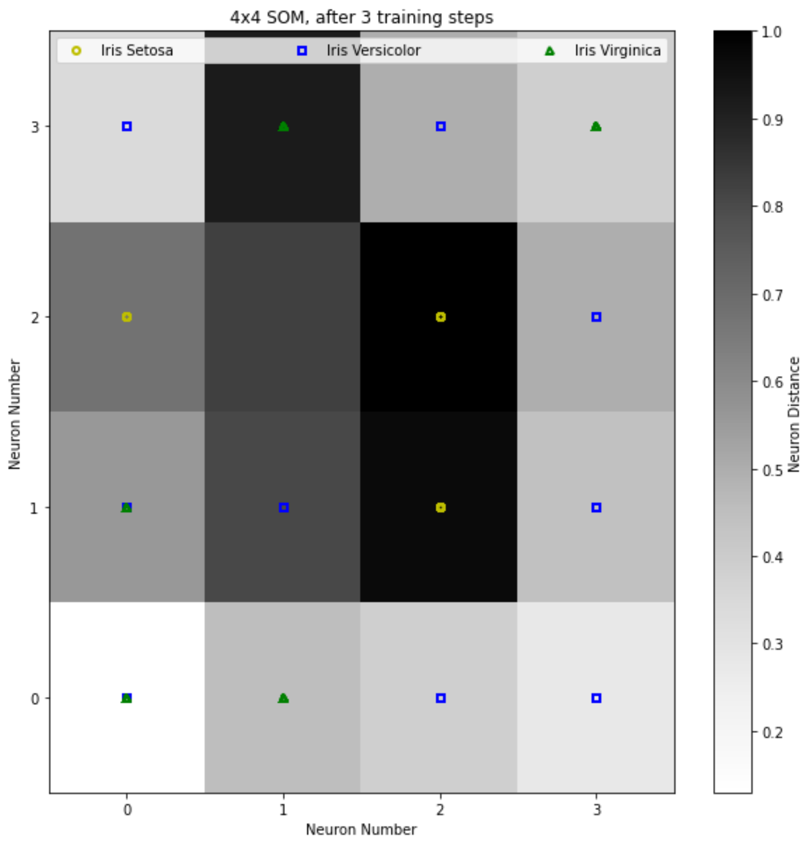
```
This Decision tree's accuracy is:  0.96
```



Match the performance of the decision tree in the cell below

In [70]:

```
# Train_and_Plot_SOM(dimensions=[25, 25], inputData=[PetalWidth, PetalLength], steps=50000)
# Reduce # of steps: 98% Accuracy
# Train_and_Plot_SOM(dimensions=[4, 4], inputData=[PetalWidth, PetalLength], steps=50)
# Reduce # of steps further: 98% Accuracy
Train_and_Plot_SOM(dimensions=[4, 4], inputData=[PetalWidth, PetalLength], steps=3)
```



```
Classification Accuracy:  0.98
```

Provide your answer in the cell below:

```
The SOM's parameters from 1.5.2 were reduced from 25x25 to 4x4 for the X and Y dimensions and the number of steps
were changed from 50000 to 3 steps and the classification accuracy was 98%. The above SOM was significantly smaller
than that in 1.5.2. This accuracy was better than that of the decision tree which was 96%. The fact that this SOM
was smaller than that in 1.5.2 is unsurprising. What surprised me was that such a small number of steps could
generate such good classification accuracy.
```

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:

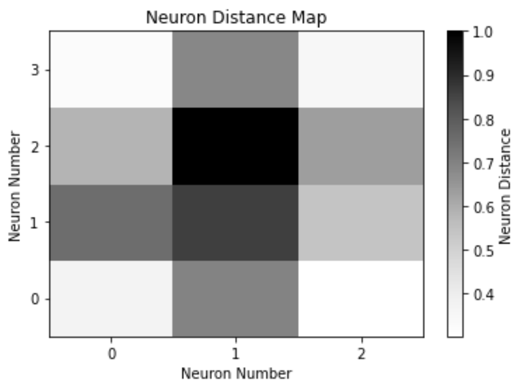
Run the cells below. The first cell loads the dataset, does some slight preprocessing and packages it all in one array per day, separated into months. The dataset is then organized in the following way: February contains 28 arrays, for the 28 days in February. Each of those days, has in the 0th index 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.

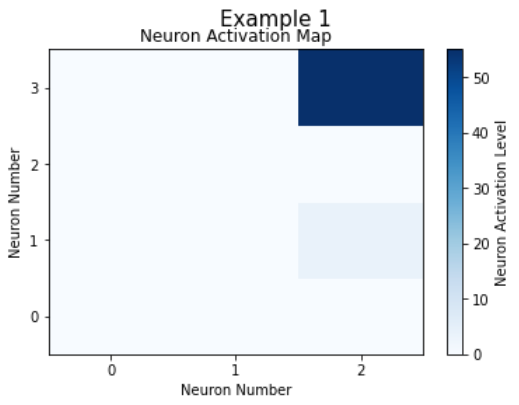
In addition to the usual distance map, an activation map is plotted, with the color for each neuron indicating how many times a neuron was activated.

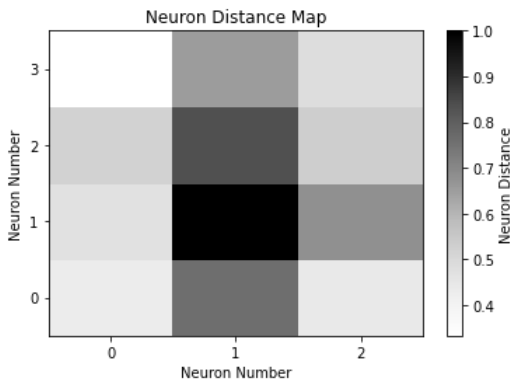
In [143]:
```python
# Load input data
dataset = sio.loadmat('somdata.mat')
dayInfo = dataset['dayInfo'].astype(float)
rawHourlyEnergy = dataset['rawHourlyEnergy']

# 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:, :]
```

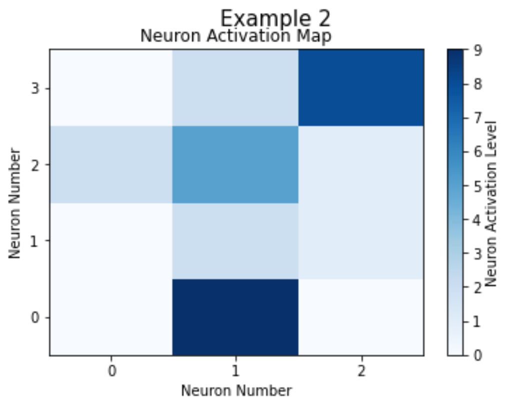
```
In [86]:  'Trains a SOM'
           def trainSOM(dimensions, inputData, steps):
               dimensions = np.array(dimensions) # From list to np.array

               '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 input vector
                   sigma = .5,                             # Spread of the neighbourhood function
                   learning_rate = 0.5,                    # Initial learning rate
                   neighborhood_function = 'gaussian',     # Type of neighborhood function
                   random_seed = 42)                       # Random seed

               'Train SOM'
               som.random_weights_init(inputData)  # Initialize weights of neurons randomly
               som.train_random(inputData, steps)  # Train the SOM using the input vectors in a random order (i.e. not batch

               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')  # Plot the number of hits on each neuron
               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 [75]:
```
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')
```









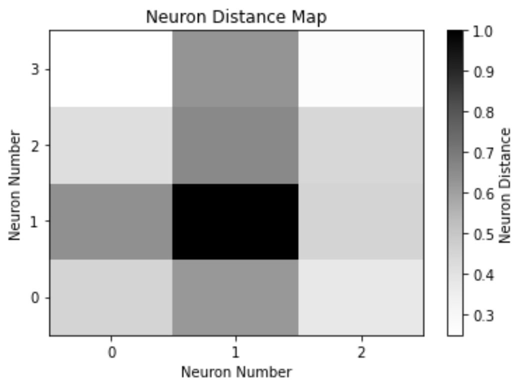
Task 2.2:
Ok, now it's your turn! You have to find some patterns in the data. This is purposely a more open ended question! Some ideas worth thinking about:

- 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?

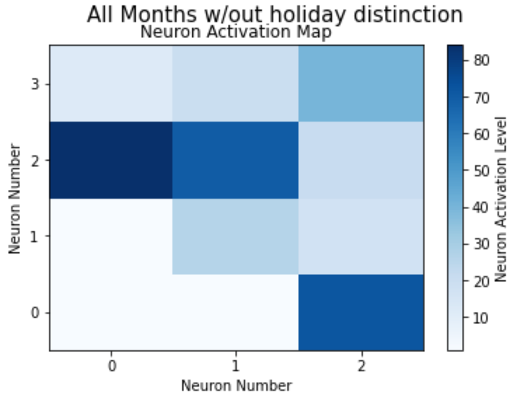
In the cell below, please provide your function calls.

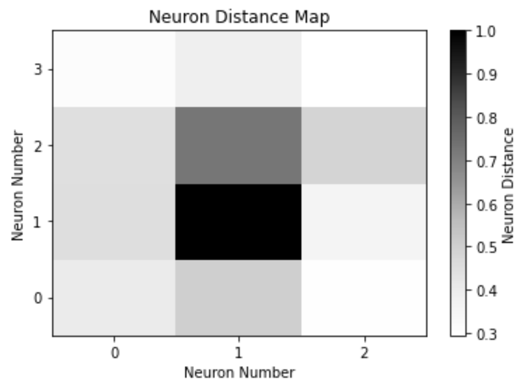
[illegible]

In [141]:
```
# Training without holiday distinction - all months
som = trainSOM([3,4], inputData[:,3:],20000)
plotSOM(som, [inputData[:,3:]], "All Months w/out holiday distinction")

# Training with holiday distinction - all months
som = trainSOM([3,4], inputData[:,2:],20000)
plotSOM(som, [inputData[:,2:]], "All Months with holiday distinction")

# Training without holiday distinction - monthly
som = trainSOM([3,4], inputData[:,3:],20000)
# Look for similarities across all months
plotSOM(som, [january[:,3:]], "January w/out holiday distinction")
plotSOM(som, [february[:,3:]], "February w/out holiday distinction")
plotSOM(som, [march[:,3:]], "March w/out holiday distinction")
plotSOM(som, [april[:,3:]], "April w/out holiday distinction")
plotSOM(som, [may[:,3:]], "May w/out holiday distinction")
plotSOM(som, [june[:,3:]], "June w/out holiday distinction")
plotSOM(som, [july[:,3:]], "July w/out holiday distinction")
plotSOM(som, [august[:,3:]], "August w/out holiday distinction")
plotSOM(som, [september[:,3:]], "September w/out holiday distinction")
plotSOM(som, [october[:,3:]], "October w/out holiday distinction")
plotSOM(som, [november[:,3:]], "November w/out holiday distinction")
plotSOM(som, [december[:,3:]], "December w/out holiday distinction")

# Training with holiday distinction - monthly
som = trainSOM([3,4], inputData[:,2:],20000)
plotSOM(som, [january[:,2:]], "January")
plotSOM(som, [february[:,2:]], "February")
plotSOM(som, [march[:,2:]], "March")
plotSOM(som, [april[:,2:]], "April")
plotSOM(som, [may[:,2:]], "May")
plotSOM(som, [june[:,2:]], "June")
plotSOM(som, [july[:,2:]], "July")
plotSOM(som, [august[:,2:]], "August")
plotSOM(som, [september[:,2:]], "September")
plotSOM(som, [october[:,2:]], "October")
plotSOM(som, [november[:,2:]], "November")
plotSOM(som, [december[:,2:]], "December")

# Training without holiday distinction - seasonal
som = trainSOM([3,4], inputData[:,3:],20000)
plotSOM(som, [january[:,3:],february[:,3:],december[:,3:]],
"Winter w/out holiday distinction")
plotSOM(som, [june[:,3:],july[:,3:],august[:,3:]],
"Summer w/out holiday distinction")

# Training with holiday distinction - seasonal
som = trainSOM([3,4], inputData[:,2:],20000)
plotSOM(som, [january[:,2:],february[:,2:],december[:,2:]],
"Winter w/out holiday distinction")
plotSOM(som, [june[:,2:],july[:,2:],august[:,2:]],
"Summer w/out holiday distinction")
```

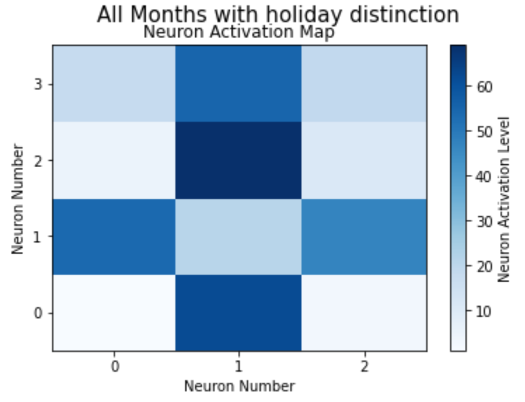


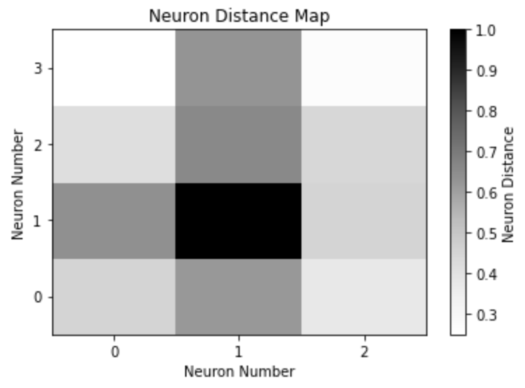
All Months w/out holiday distinction



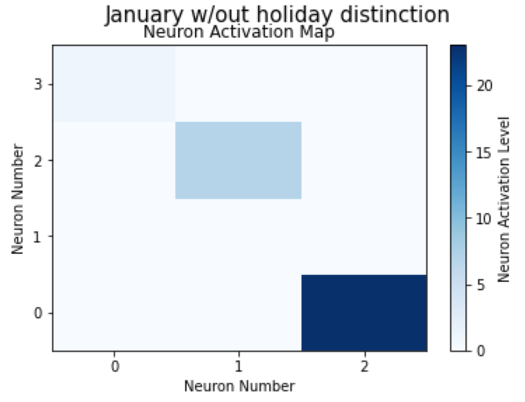


All Months with holiday distinction

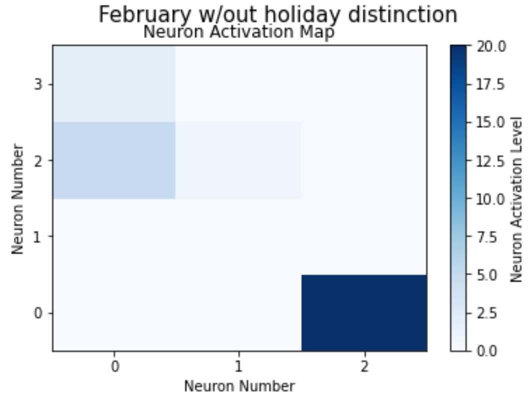


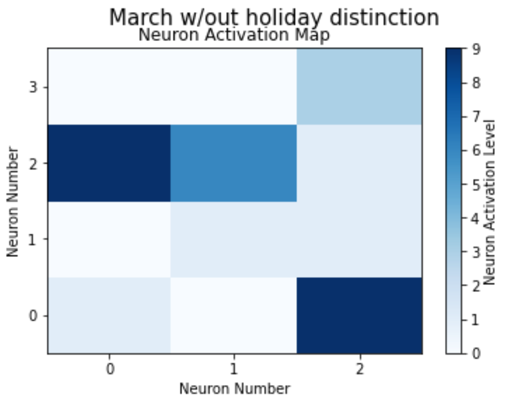


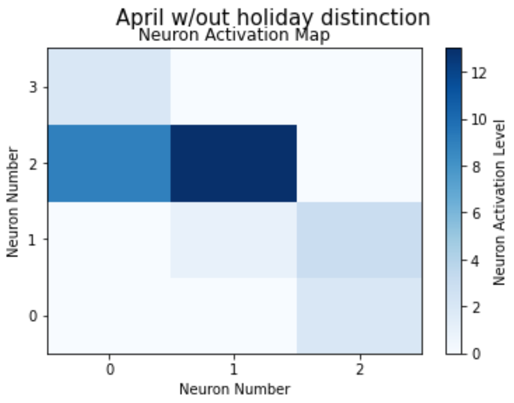
January w/out holiday distinction

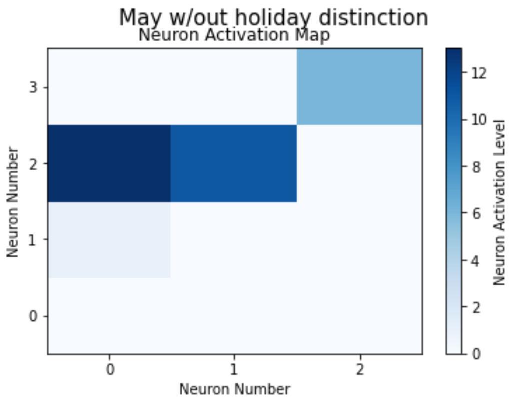


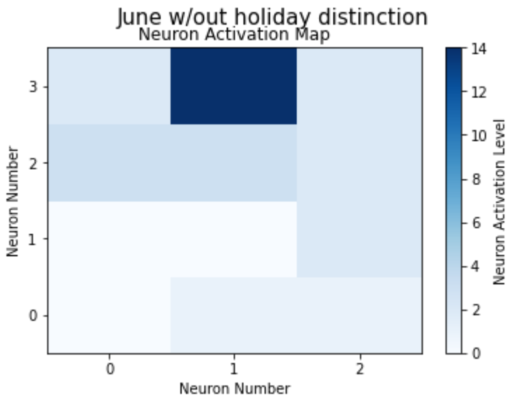
February w/out holiday distinction

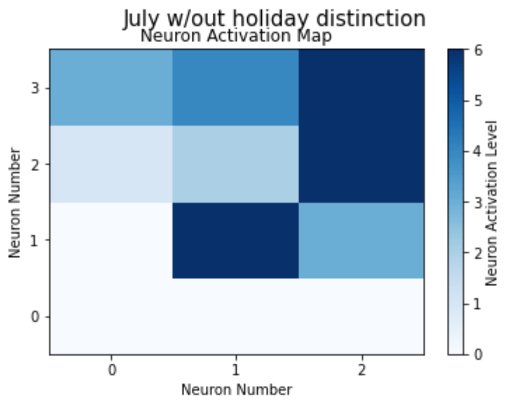


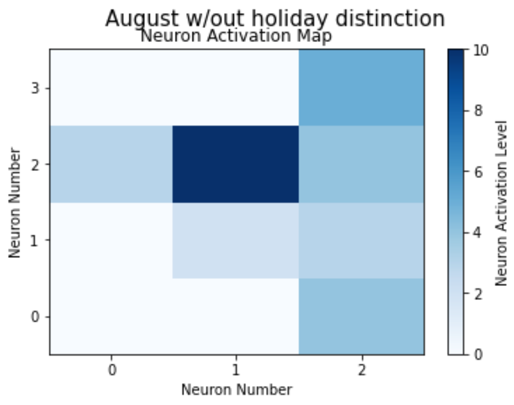

March w/out holiday distinction
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level
0
1
2
3
0
1
2
3
4
5
6
7
8
9

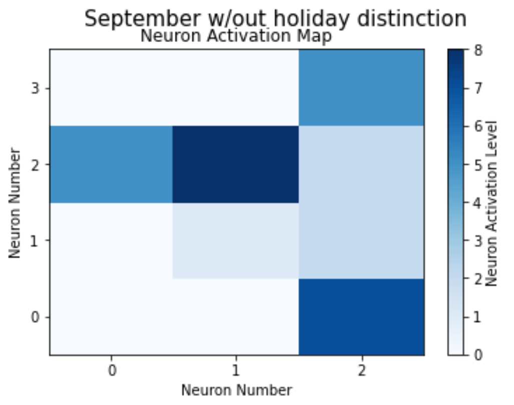

April w/out holiday distinction
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level
0
2
4
6
8
10
12

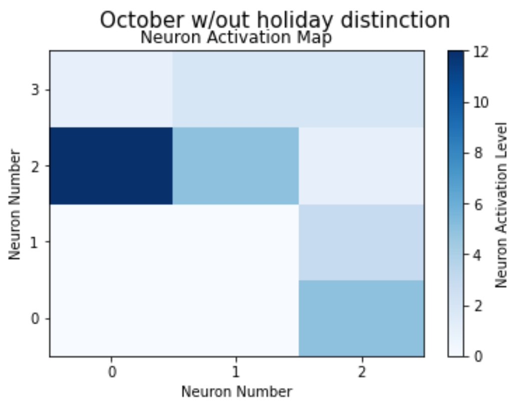

May w/out holiday distinction
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level
0
2
4
6
8
10
12


June w/out holiday distinction
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level
0
2
4
6
8
10
12
14


July w/out holiday distinction
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level

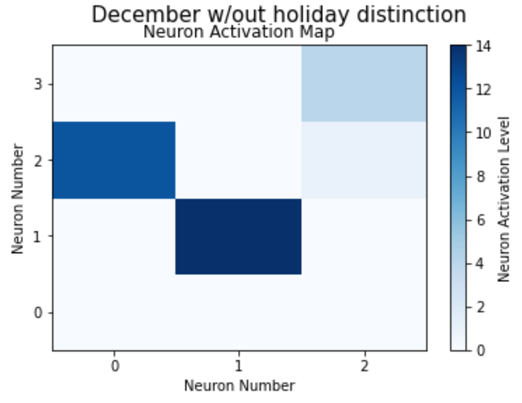

August w/out holiday distinction
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level

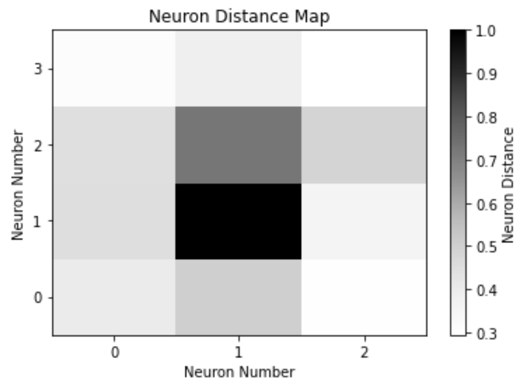

September w/out holiday distinction
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level


October w/out holiday distinction
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level

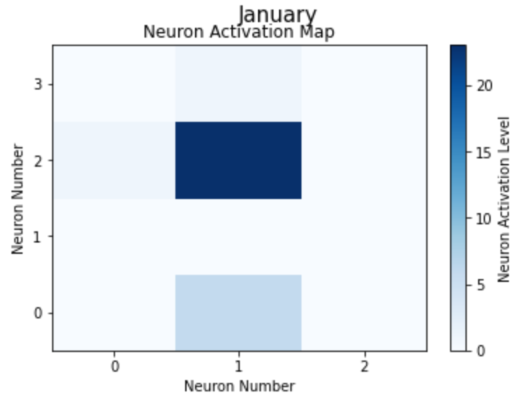
November w/out holiday distinction

December w/out holiday distinction

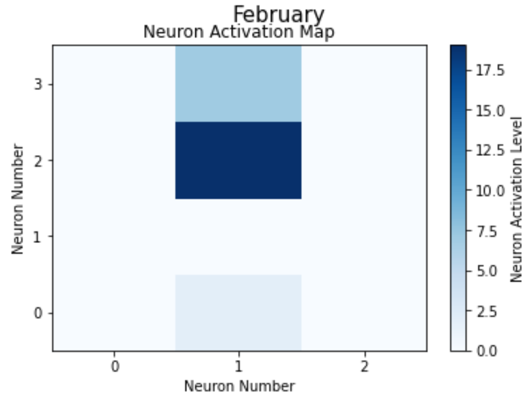


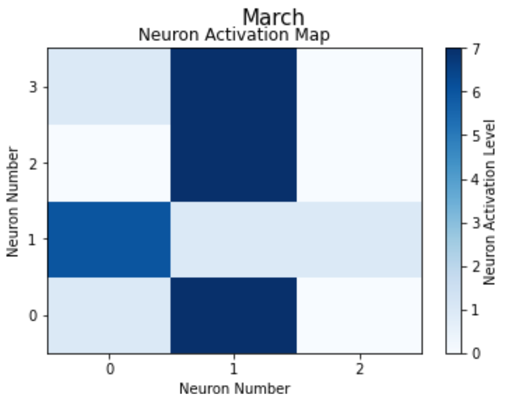


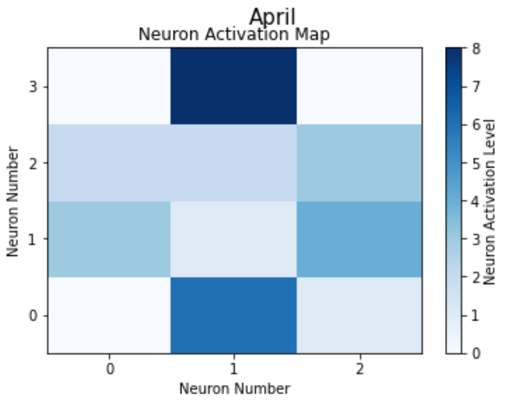
January

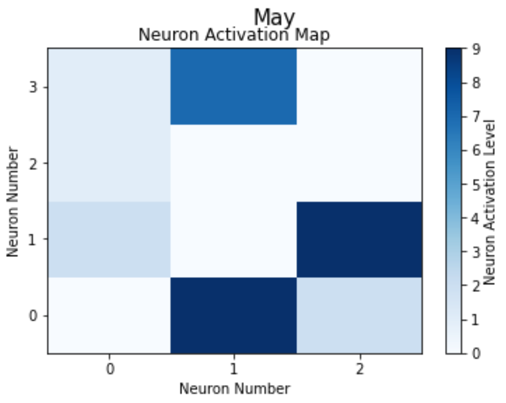


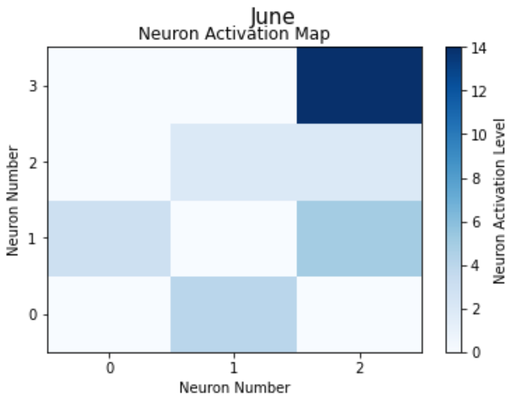
February

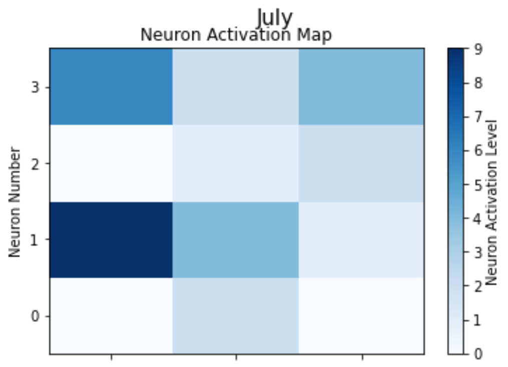


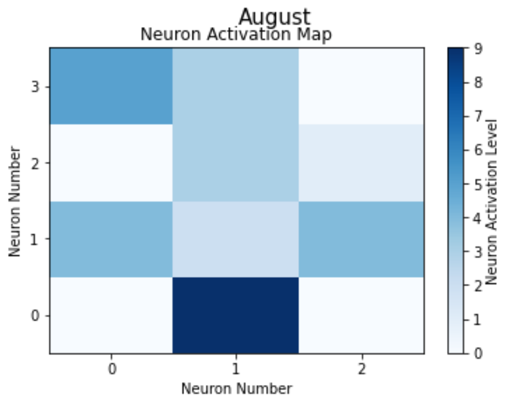

March
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level

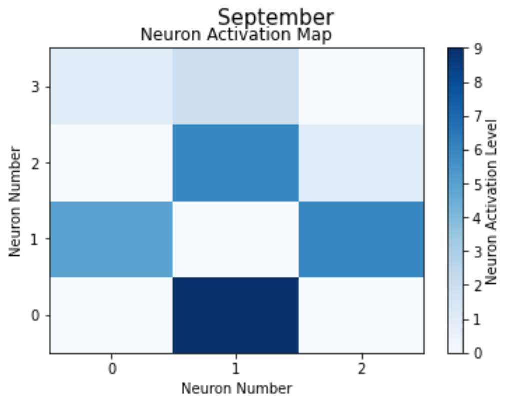

April
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level

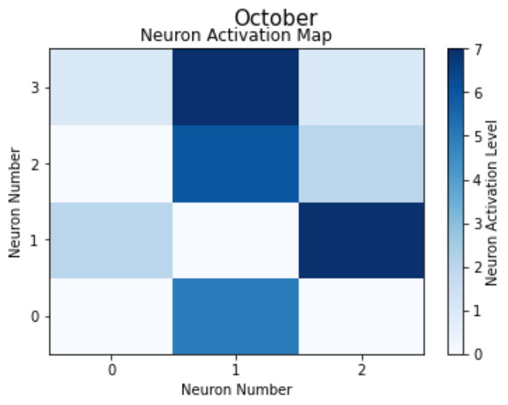

May
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level

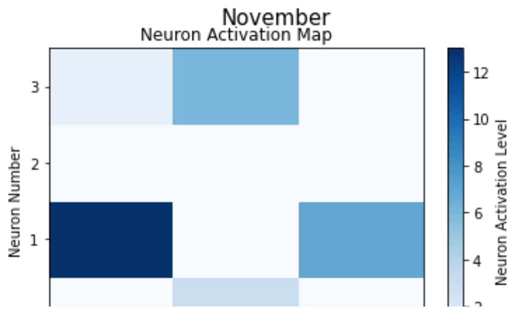

June
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level

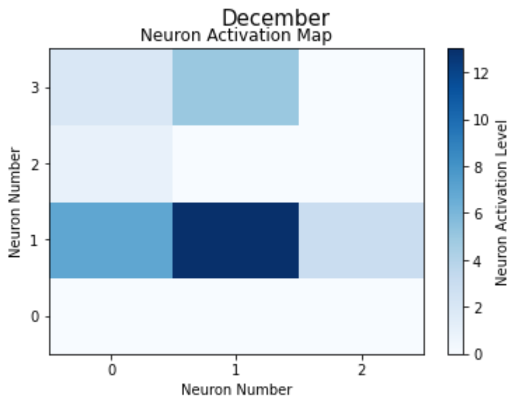

July
Neuron Activation Map
Neuron Number
Neuron Activation Level
0
1
2
3
4
5
6
7
8
9

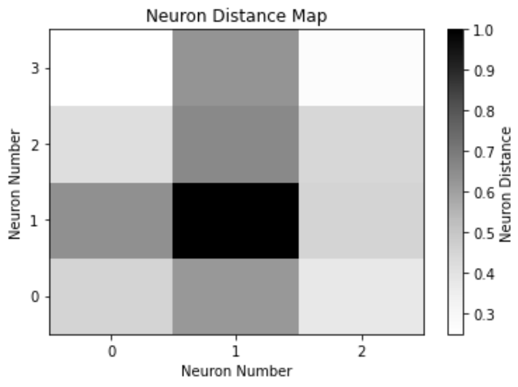

August
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level
0
1
2
3
4
5
6
7
8
9

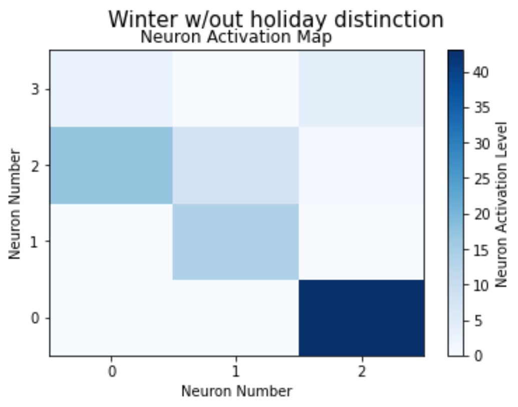

September
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level
0
1
2
3
4
5
6
7
8
9

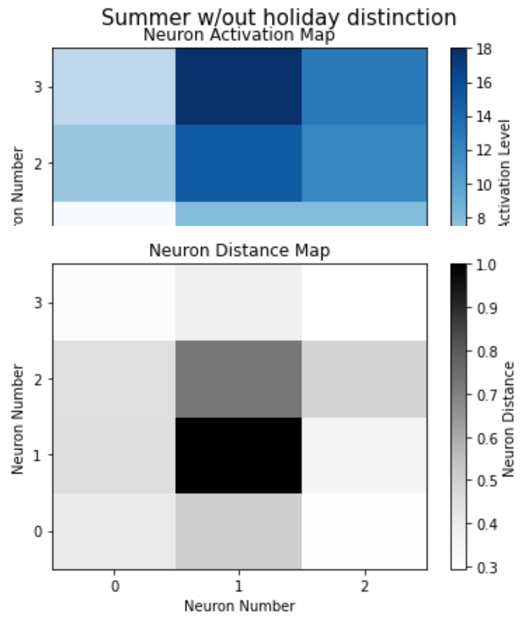

October
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level
0
1
2
3
4
5
6
7

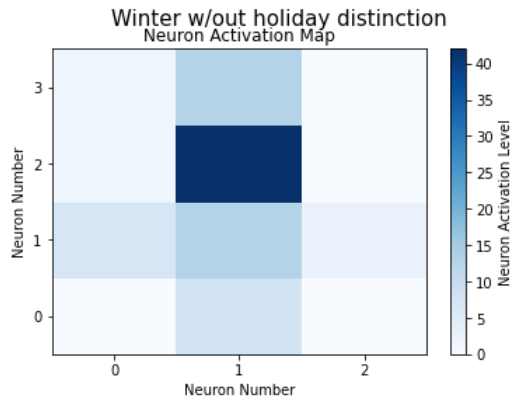

November
Neuron Activation Map
Neuron Number
Neuron Activation Level
3
2
1
12
10
8
6
4

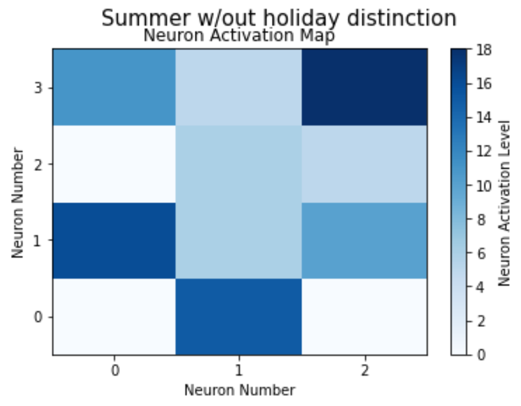

December
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level
0
1
2
3
0
2
4
6
8
10
12


Neuron Distance Map
Neuron Number
Neuron Number
Neuron Distance
0
1
2
3
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


Winter w/out holiday distinction
Neuron Activation Map
Neuron Number
Neuron Number
Neuron Activation Level
0
1
2
3
0
5
10
15
20
25
30
35
40







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...

When comparing the activation maps for the months of January, February and December regardless of including holidays or not, it seems that they are very similar in terms of degree of neuron activation and specifically which neuron has been activated. Moreover, August and September's activation map when not including holidays are very similar, however, when including holidays August, September and October are very similar. When comparing a month that is in the summer like July against a month that is in the Winter like January, from the activation map, we can see July has a greater deal of neuron activation while January has a lesser extent of neuron activation, regardless if holidays are included or not. This observation lead to me delving deeper and generating the activation maps for the winter and summer seasons. From them, we can see that again, regardless of holiday inclusion there is a greater deal of neuron activation in the summer months and less neuron activation during the winter months. The greater degree of activation during the summer months could stem from the fact that the SOM has realised that energy consumption during these times at the industrial park is lower since usually more breaks/vacations are taken during these times. When comparing neuron activation map for all months of the year not including holidays to the neuron activation map for all months including holidays, we see more neuron activation in the one including holidays which makes sense since the SOM is perhaps learning seeing that during holidays energy consumption is lower. Using an SOM for this task may not be such a great idea since the groupings in these neuron activation maps were not greatly informative (hard to interpret). However, it is a good idea to use SOMs here due to the great degree of dimensionality of the input data.

Abstract

In this lab we performed two tasks, both of which utilized Self Organizing Maps (SOM) to better understand the two sets of multi-dimensional data that we have. For the first task, we worked with the infamous Iris data set and explored the use of Self Organizing Maps to perform classification. Using the 4 properties, Sepal Width, Sepal Length, Petal Length and Petal Width, we began by making 6 different 'naive' plots with all the data classified with different markers representing each class. Afterwards, we trained and plotted a 2x2 dimensional SOM with input data Sepal Length-Sepal Width and 20 steps (iterations), this resulted in an accuracy of 64%. This SOM was only able to classify the Iris Setosa specie well from the dataset, while the other two Iris species were not classified well. We proceeded to tune the parameters of the SOM to obtain better accuracy. The result after many trials was a 25x25 dimensional SOM with input data Petal Width-Petal Length with 50000 steps which obtained an accuracy of 99.3 % accuracy. After this, a decision tree was built and fit and tasked to perform this classification, it obtained an accuracy of 96%. We were tasked with beating this using the SOM with as few parameters as possible. The result was a 4x4 dimensional SOM with input data of the form Petal Width-Petal Length and 3 steps and this obtained an accuracy of 98%. The second (more open-ended) task involved exploring a multi-dimensional energy consumption dataset using SOMs to recognize patterns. For this task, there were a great deal of ways to interpret the data, for example, the obtained activation maps for the Summer and Winter seasons showed that the SOM has recognized certain patterns such as lower energy consumption during the summer months as more neurons were activated on the grid. Using an SOM for this task was not the best idea was the neuron activation maps were not greatly informative (hard to interpret), however, there is large dimensionality in the input data, therefore, SOMs are good for this task.

Introduction

This lab was separated into two tasks where both tasks used Self Organizing Maps (SOM) to perform a certain functionality on two different sets of multi-dimensional data (one data-set per task). For the first task of this lab, we performed classification on the Iris data set using SOMs. Classification involves the use of properties provided from data to categorize that data into group(s). In this task, we use Self Organizing Maps (also known as Kohonen Networks) which are one of many neural networks. Self Organizing Maps employ competitve learning methods to train and it is a form of unsupervised learning since inputs are provided but no corresponding targets are provided. We also proceed to try to beat a decision tree's accuracy for classifying the 3 different Iris species using the SOM by fine-tuning the SOM's parameters to be minimal. For the second task, we utilized SOMs to interpret and recognize patterns in energy consumption of industrial parks. There were many ways of training and plotting the SOM. For example, the SOM can train all the days of the year not including holidays and including holidays. Afterwards, the SOM grid can be plotted for: all months including holidays, all months not including holidays, separate months including holidays, separate months not including holidays, seasons including holidays, seasons not including holidays and many more combinations.

Conclusion

In conclusion, with the right parameters the SOM can perform well for classifying the Iris data set. For the first trial, using an SOM with a 2x2 grid, 20 steps and Sepal Length-Sepal Width as the parameters, the SOM was only able to obtain a classification accuracy of 64% with it only being able to classify Iris Setosa well and the other two species were not classified well as indicated by the overlapping green traingle and blue square markers in the produced SOM grid. After tuning the parameters multiple times, a 25x25 sized SOM, with 50000 steps and Petal Width and Petal Length as the input parameters was the final result with a 99.3% classification accuracy. When trying to match the classification accuracy of the decision tree, which was 96% (for the same Iris data) using an SOM with minimal parameters, a 4x4 SOM with 3 steps and input data Petal Width-Petal Length was used. The classification accuracy of this SOM was 98% and this SOM was significantly smaller than the one we used that resulted in a 99.3% accuracy. The second task which was more open-ended had us make use of SOMs to perform pattern recognition on multi-dimensional data which represented the energy consumption of an industrial park. The SOM maps obtained for summer months like July had a greater extent of neuron activation while winter months had a lesser degree of neuron activation. Therefore, I produced the SOM grids for the Winter and Summer seasons (separately) and what we notice is that regardless of holidays there is a greater deal of neuron activation for the Summer season. The SOM had noticed that during the Summer months there is a lesser extent of energy consumption as normally individuals are taking more vacations during those times. For all the months combined SOM grids, the SOM notices that during holidays there is less power consumption as indicated by the greater degree of neuron activation. Employing an SOM for this task was advantageous due to the great degree of dimensionality in the data (SOMs can help in the reduction of the dimensionality), however, the generated neuron activation maps were hard to interpret.

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	