# Neural Network Assignment 02 ¶

1. Anirudha Pal
2. Mohammad Wasil
3. Wei-Chan Hsu

1. Summary of Chapter 2

- The property that is of primary significance for a neural network is the ability to learn from its environment and to improve its performance through learning.
- Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.
- A prescribed set of well-defined rules for the solution of a learning problem is called a learning algorithm.
- Basic Learning rules:
  - **Error correction learning**

    > - The error signal $e_k(n)$ actuates a control mechanism, the purpose of which is to apply a sequence of corrective adjustments to the synaptic weights of neuron $k$.
    > - The corrective adjustments are designed to make the output signal $y_k(n)$ come closer to the desired response $d_k(n)$ in a step-by-step manner.

  - **Memory based learning**

    > - Store or memorize a set of patterns. We try to memorize the association between the input vector and desired output.
    > - For a new input $x_{test}$, find out from memory which of input $x_i$ is closest to $x_{test}$.
    > - Distance measure, euclidean distance between $x_{test}$ and each input $x_i$.
    > - A variant of the nearest neighbor classifier is the k-nearest neighbor classifier, which proceeds as follows:
    >
    >   > - Identify the $k$ classified patterns that lie nearest to the test vector $X_{test}$ for some integer $k$.
    >   > - Assign $x_{test}$ to the class (hypothesis) that is most frequently represented in the $k$ nearest neighbors to $x_{test}$ (i.e., use a majority vote to make the classification).

  - **Hebbian learning**

- If two neurons on either side of a synapse (connection) are activated simultaneously (i.e. synchronously). then the strength of that synapse is selectively increased.
- If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.
- Key properties that characterize a Hebbian synapse:
    - Time-dependent mechanism.
    - Local mechanism
    - Strongly interactive mechanism.
    - Conjunctional or correlational mechanism

- Classifiaction of synaptic modification
    - Hebbian
    - Anti-Hebbian
    - Non-Hebbian
- Mathematical Models of Hebbian Modifications
    - Hebb's hypothesis
    - Covariance hypothesis

- **Competitive learning**
    - Three basic elements to a competitive learning rule:
        - A set of neurons that are all the same except for some randomly distributed synaptic weights, and which therefore respond differently to a given set of input patterns.
        - A limit imposed on the "strength" of each neuron.
        - A mechanism that permits the neurons to compete for the right to respond to a given subset of inputs, such that only one output neuron, or only one neuron per group, is active (i.e., "on") at a time. The neuron that wins the competition is called a winner-takes-all neuron.
- **Boltzmann learning**

- In a Boltzmann machine the neurons constitute a recurrent structure, and they operate in a binary manner since, for example, they are either in an "on" state denoted by +1 or in an "off" state denoted by -1
- The neurons of a Boltzmann machine partition into two functional groups: visible and hidden.
- The visible neurons provide an interface between the network and the environment in which it operates, whereas the hidden neurons always operate freely.
- There are two modes of operation to be considered:
  - Clamped condition, in which the visible neurons are all clamped onto specific states determined by the environment.
  - Free-running condition, in which all the neurons (visible and hidden) are allowed to operate freely.

- There are two fundamental learning paradigms:
  - Learning with a teacher (**supervised learning**): The network is provided with input-output examples for error-correction learning; the parameters are adjusted under the influence of both the training vector and error signal.
  - Learning without a teacher has two forms:
    - **Reinforcement learning**: It works through continuous interaction with the environment in order to minimize a scalar index of performance.
    - **Unsupervised learning**: The parameters of the network are optimized with respect to a task-independent measure. The competitive learning rule may be used.
- Six learning tasks can be identified:
  - **Pattern association**: Retrieve a particular pattern from a partial pattern
  - **Pattern recognition**: Associate an input signal to a prescribed number of classes
  - **Function approximation**: Approximate a input-output mapping, such as system identification and inverse system
  - **Control**: Maintain a system in a controlled condition
  - **Filtering**: Extract information from noisy data
  - **Beamforming**: Distinguish between spatial properties between target and background noise
- Memory can be divided into short-term and long-term memory based on the retention time.
- **Memory matrix** defines the connectivity between input (key patterns) and output (memorized patterns) layers of the associate memory. The influence of a new pattern on the memory is reduced with increasing number of stored patterns.

$$\mathbf{M} = \sum_{k=1}^{q} \mathbf{W}(k),$$

where $q$ is the number of input-output patterns, and $\mathbf{W}(k)$ is the weight matrix.

- The memory associates perfectly if the key (input) vectors form an orthonormal set.

$$\mathbf{x_k^T x_j} = \begin{cases} 1, & k = j \\ 0, & k \neq j \end{cases}$$

- The number of patterns which can be stored in a **correlation matrix memory** cannot exceed the input space dimensionality.
- Space and time are two fundamental dimensions of learning process.
- A stationary environment can be learned by a network using supervised learning.
- To handle a dynamic environment, a network has to adapt its parameters by **continuous learning**. One way is to identify the time interval in which the process can be viewed as pseudostationary; another way is to build temporal structure into the network which becomes a nonlinear adaptive filter.

3. Do the problem 1.13 (Network architecture) from the previous week's assignment. This time use Python's (sympy) symbolic toolbox. Finally assume the network presented in fig P1.13 is a binary-classifier, please depict how the input space (R2) is classified on a 2D graph using different colors.

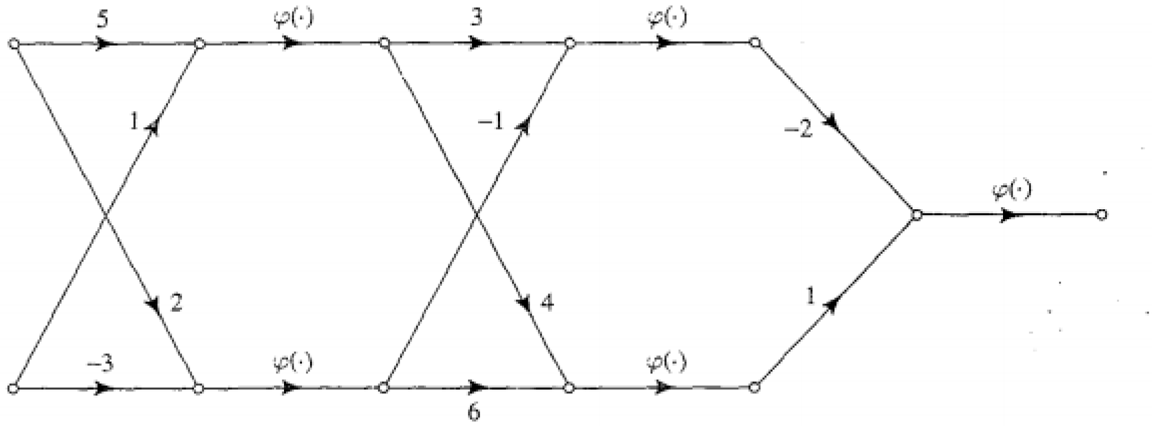



**FIGURE P1.13**

In [3]:

```
import numpy as np
import sklearn
import sklearn.datasets
import sklearn.linear_model
from sympy import *
init_printing(use_latex='true')
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:

```
x_1, x_2, a = symbols('x_1,x_2,a')
u_1, u_2, u_3, u_4, u_5 = symbols('u_1, u_2, u_3, u_4, u_5')
v_1, v_2, v_3, v_4, v_5 = symbols('v_1, v_2, v_3, v_4, v_5')
y_1, y_1, y_2, y_3, y_4, y_5 = symbols('y_1, y_1, y_2, y_3, y_4, y_5')

u_1 = 5*x_1 + 1*x_2
u_2 = 2*x_1 - 3*x_2

v_1 = u_1 #Normally + bias
v_2 = u_2

#Calculate the output using logistic function
y_1 = 1/(1 + exp(-a * v_1))
y_2 = 1/(1 + exp(-a * v_2))

u_3 = 3*y_1 + 4*y_1
u_4 = -1*y_2 + 6*y_2

v_3 = u_3
v_4 = u_4

y_3 = 1/(1 + exp(-a * v_3))
y_4 = 1/(1 + exp(-a * v_4))

u_5 = -2*y_3 + 1*y_4

v_5 = u_5

y_5 = 1/(1 + exp(-a * v_5))
```

In [5]:

```
y_5
```

Out[5]:

$$\frac{1}{1 + e^{-a\cdot\left(- \frac{2}{1 + e^{\frac{-7\cdot a}{1 + e^{-a\cdot(5\cdot x_1 + x_2)}}}} + \frac{1}{1 + e^{\frac{-5\cdot a}{1 + e^{-a\cdot(2\cdot x_1 - 3\cdot x_2)}}}}\right)}}$$

In [6]:

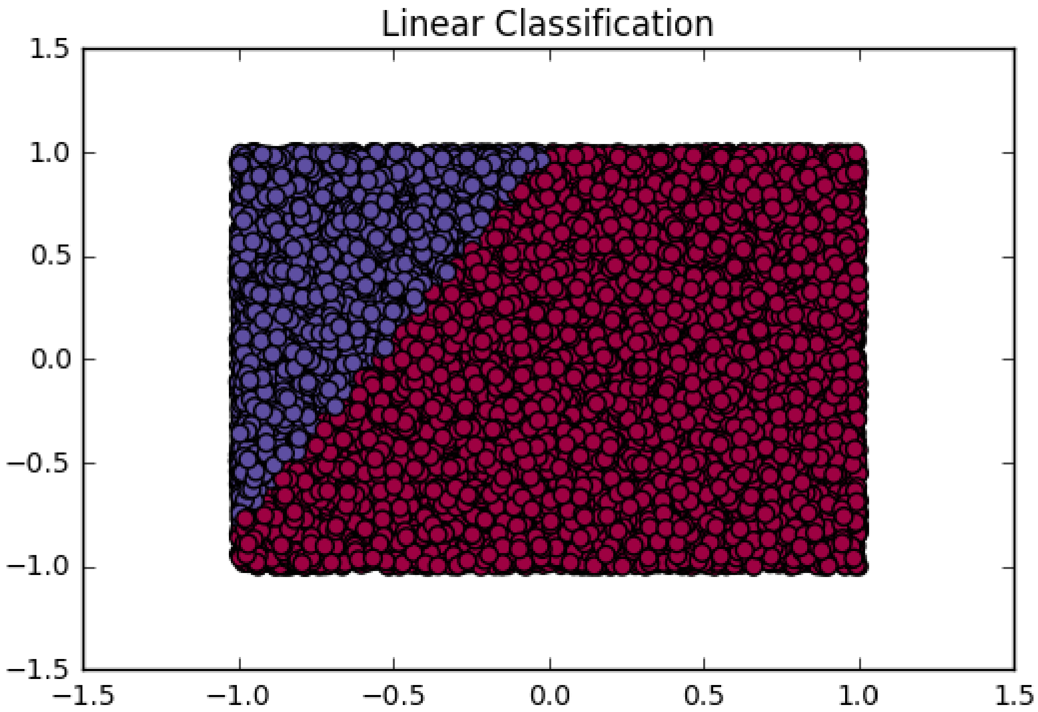
```
import numpy as np
from sympy import *

class classification(object):
    w1 = np.array([[5.,2.],[1.,-3.]])
    w2 = np.array([[3.,4.],[-1.,-6.]])
    w3 = np.array([-2.,1.])
    w3 = w3.reshape(2,1)
    number_inputs = 200
    X = np.zeros((number_inputs,2))

    def sigmoid_function(self, v, derivative=False):
        return 1/(1 + np.exp(-v))

    def training(self):
        self.feed_forward(self.X)

    def test(self, X_test, numb_of_inputs, linear=True):
        hidden_l3 = 0
        if linear==True:
            input_l = X_test
            hidden_l1 = np.dot(input_l, self.w1)
            hidden_l2 = np.dot(hidden_l1, self.w2)
            hidden_l3 = np.array(np.dot(hidden_l2, self.w3))
            hidden_l3 = hidden_l3.reshape(numb_of_inputs,1)
        else:
            input_l = X_test
            hidden_l1 = self.sigmoid_function(np.dot(input_l, self.w1))
            hidden_l2 = self.sigmoid_function(np.dot(hidden_l1, self.w2))
            hidden_l3 = np.array(self.sigmoid_function(np.dot(hidden_l2, self.w3)))
            hidden_l3 = hidden_l3.reshape(numb_of_inputs,1)

        return hidden_l3
```

In [7]:

```
#Linear function
np.random.seed(3)
cls = classification()
#Test
number_of_inputs = 20000
test_inputs = 2*np.random.random((number_of_inputs,2)) - 1

y_test = cls.test(test_inputs, number_of_inputs, linear=True)
y_mean = np.mean(np.abs(y_test))
for i in range (len(y_test)):
    #take the mean as a reference
    if y_test[i] <= y_mean:
        y_test[i] = 0
    else:
        y_test[i] = 1

y_test = y_test.round(1)
y_test = y_test.astype(int)

fig1 = plt.figure()
plt.scatter(test_inputs[:,0], test_inputs[:,1], s=40, c=y_test, cmap=plt.cm.Spectra
plt.title("Linear Classification")
```

Out[7]:

```
<matplotlib.text.Text at 0x7f5c8ea2e210>
```

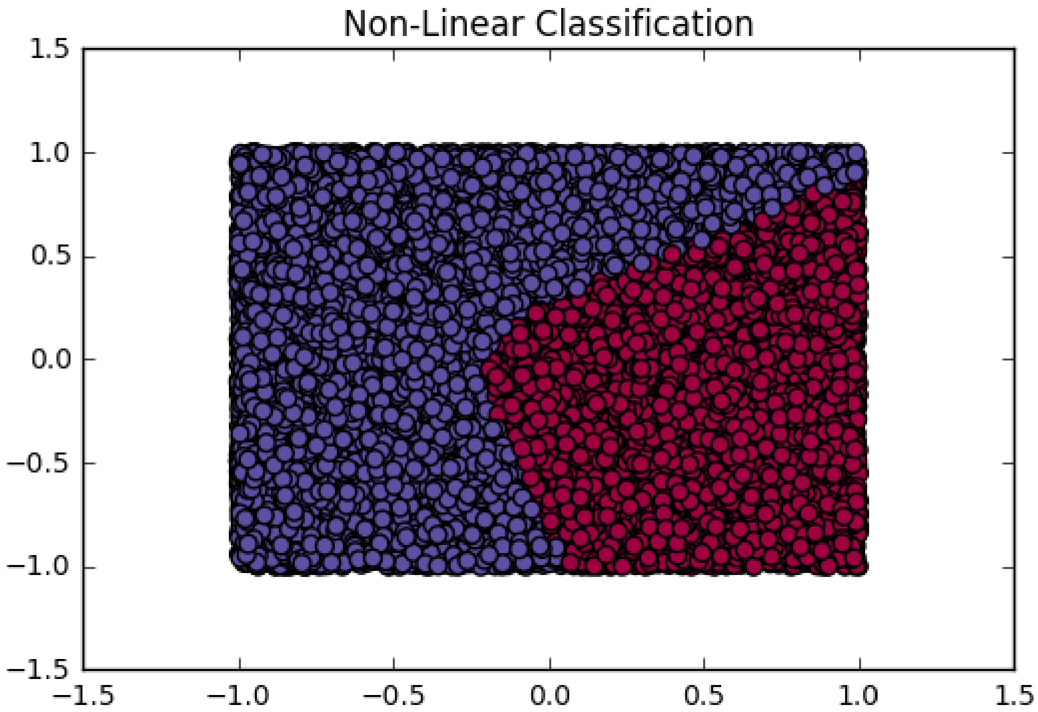


In [8]:

```
#Linear function
np.random.seed(3)
cls = classification()
#Test
number_of_inputs = 20000
test_inputs = 2*np.random.random((number_of_inputs,2)) - 1

y_test = cls.test(test_inputs, number_of_inputs, linear=False)
y_mean = np.mean(np.abs(y_test))
for i in range (len(y_test)):
    #take the mean as a reference
    if y_test[i] <= y_mean:
        y_test[i] = 0
    else:
        y_test[i] = 1

y_test = y_test.round(1)
y_test = y_test.astype(int)

fig1 = plt.figure()
plt.scatter(test_inputs[:,0], test_inputs[:,1], s=40, c=y_test, cmap=plt.cm.Spectra
plt.title("Non-Linear Classification")
```

Out[8]:

<matplotlib.text.Text at 0x7f5c8e8f3c90>




In [ ]:

4. Adjust the data at the "New Classification Example (now *with* bias)" slide, such that a bais becomes necessary (not 0). Validate the perceptron learning algorithm.

Consider a 2D training data set with eight samples. The set is augmented by inserting the first entry fixed to 1 to deal with the bias as extra weight.

$$\begin{aligned} C_1 &= \{(1,1,1),(1,2,1),(1,2,-1),(1,0,1)\} \\ C_2 &= \{(1,0,0),(1,-1,1),(1,-1,-1),(1,1,-1)\} \end{aligned}$$

The initial weight is $w(0) = (1,0,0)$. For each epoch, the output of each sample is calculated and compared with the target. The output $w(n)^T x(n)$ should be $1$ for set $C_1$, and $-1$ for set $C_2$. If the outout of sample $i$ is different from the desired output $d_i$, the weight is updates:

$$w(n+1) = w(n) + \eta * d_i x_i(n)$$

where $\eta$ is the learning rate.

In [3]:

```
# Training data: data = [x0,x1,x2]
data = np.array([[1,1,1],[1,2,1],[1,2,-1],[1,0,1],[1,0,0.5],[1,-1,1],[1,-1,-1],[1,1
target = np.array([1,1,1,1,-1,-1,-1,-1])
# Initial weight
w = np.array([1,0,0])
# Parameters
eta = 0.8
epochs = 5

def perceptron(data,y,w):
    for t in range(epochs):
        for i in range(len(data)):
            # Update weight if ouput (w*x) is different from target
            if (np.dot(data[i],w)*y[i]) < 0:
                w = w + eta*data[i]*y[i]
    return w

w_new = perceptron(data,target,w)
print(w_new)
```

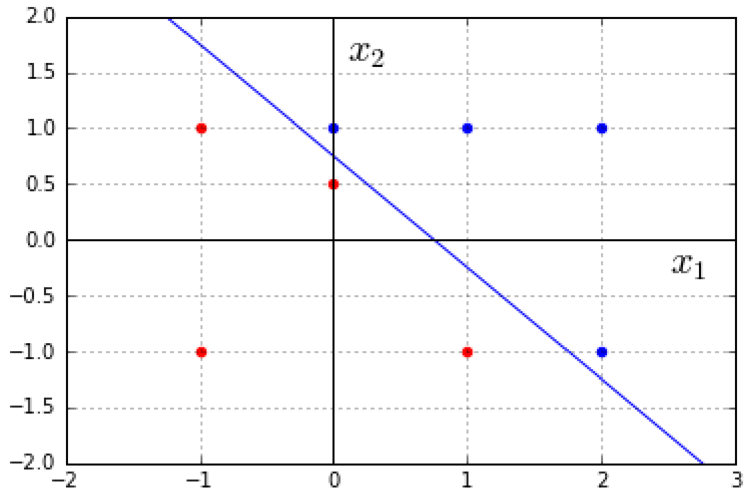
```
[-0.6  0.8  0.8]
```

In [4]:

```
# Plot the classification result
fig, ax = plt.subplots()
for i, p in enumerate(data):
    # Plot the positive samples
    if i < 4:
        ax.scatter(p[1], p[2], color='blue')
    # Plot the negative samples
    else:
        ax.scatter(p[1], p[2], color='red')
        ax.axis([-2,3,-2,2])

x1 = np.linspace(-2,3,50)
x2 = -w_new[1]*x1/w_new[2] - w_new[0]/w_new[2]
ax.plot(x1,x2)
ax.grid()
ax.text(2.5,-0.3,'$x_1$',fontsize=20)
ax.text(0.1,1.6,'$x_2$',fontsize=20)
ax.axhline(y=0, color='k')
ax.axvline(x=0, color='k')
```

Out[4]:

```
<matplotlib.lines.Line2D at 0x7f5199ddbf90>
```




In [ ]:

```
```