Perceptron
&
Multilayer Perceptron

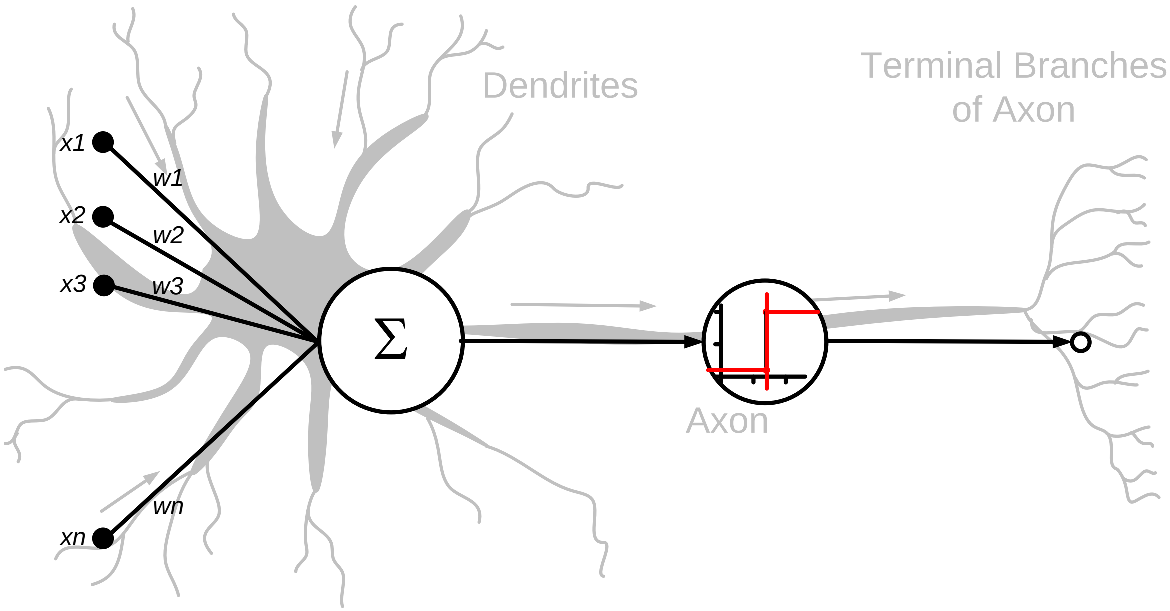
Introduction to Perceptron

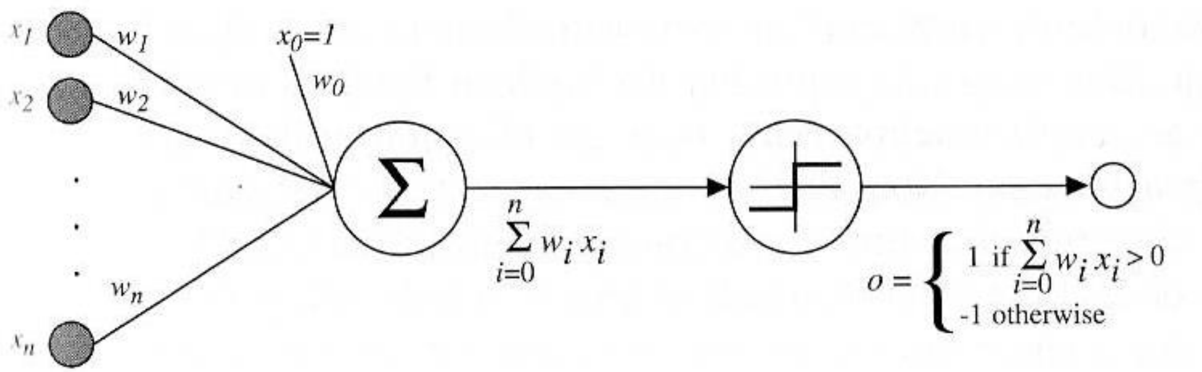
- The aim of the perceptron is to classify a set of inputs into one of two categories (usually 1 or 0).

- If the inputs are in the form of a grid, a perceptron can be used to recognize visual images of shapes.

- The perceptron usually uses a step function, which returns 1 if the weighted sum of inputs exceeds a threshold, and 0 otherwise.

Perceptron structure





Learning Rules

1. Perceptron Learning Rule
   - The perceptron training algorithm for classification tasks.

2. Delta Rules
   - For continuous activation function
   - The aim of the delta rule is to minimize the error over all training patterns

Perceptron Training Algorithm

1. Set initial weights w_1, w_2,…, w_n and threshold θ to random numbers. Normally [-0.5,0.5]

2. Calculate output:

$$X = \sum_{i=1}^{n} w_i x_i$$

3. Apply activation functions

$$Y = \begin{cases} 1 & for \quad X > \theta \\ 0 & for \quad X \leq \theta \end{cases}$$

Perceptron Training Algorithm

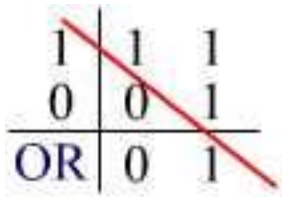
4. An item of training data is presented. If the perceptron mis-classifies it, the weights are modified according to the following:

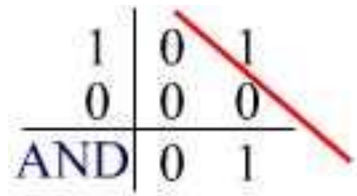
$$w_i \leftarrow w_i + (a \times x_i \times (t - o))$$

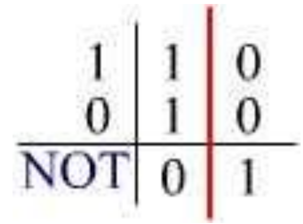
where *t* is the target output for the training example, *o* is the output generated by the perceptron and *a* is the learning rate, between 0 and 1 (usually small such as 0.1)

5. Cycle through training examples until successfully classify all examples

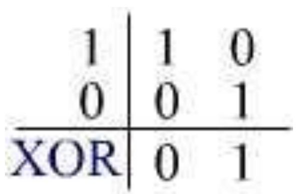
Linear and Non Linear Separable





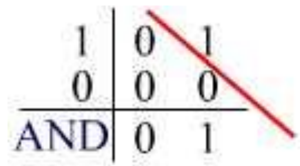


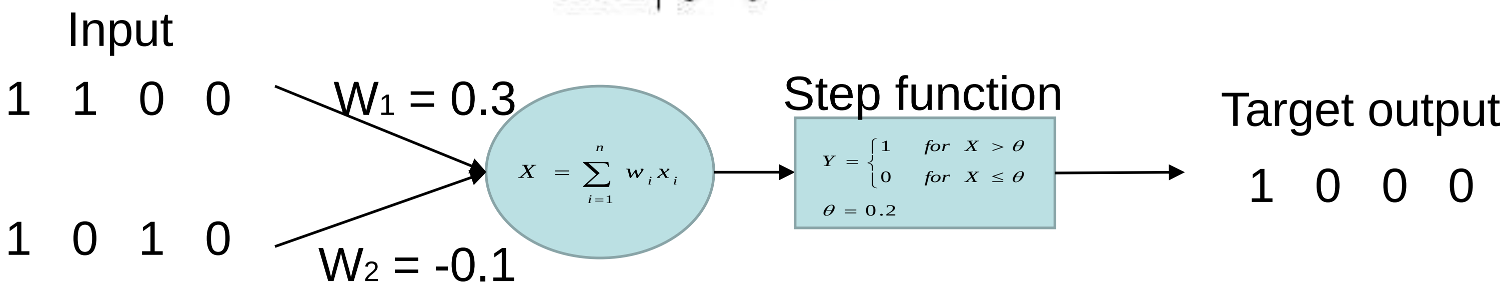
OR, AND and NOT are linearly separable Boolean Functions



XOR is not linearly separable

Perceptron learning example: AND





Pattern 1: (0x0.3)+(0x-0.1) = 0, Less than 0.2, output = 0 (target 0)
Pattern 2: (0x0.3)+(1x-0.1) = -0.1, Less than 0.2, output = 0 (target 0)
Pattern 3: (1x0.3)+(0x-0.1) = 0.3, Greater than 0.2, output = 1 (target 0)

Weight update: $w_i \leftarrow w_i + (a \times x_i \times (t - o))$

w_1 = 0.3 + (0.1 x 1 x (0-1) = 0.2

w_2 = -0.1 + (0.1 x 0 x (0-1) = -0.1

Pattern 4: (1x0.2)+(1x-0.1) = 0.1, Less than 0.2, output = 0 (target 1)

Weight update:…..

Perceptron learning example: AND

Epoch	Inputs x_1	Inputs x_2	Desired output Y_d	Initial weights w_1	Initial weights w_2	Actual output Y	Error e	Final weights w_1	Final weights w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	−1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	−1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	−1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

Lab Exercise 1

Implementation of Perceptron using C/C++/Python

H:\UTM Courses\SCJ3563 Computer Intelligence\2016-2017 Semester 2\01 Perceptron\Perceptron (Release Version).cpp - Dev-C++ 5.11

[*] Perceptron (Release Version).cpp

```
//****************************************************************************************
//Perceptron
//Written by Haza Nuzly, Applied Computing Research Group, Universiti Teknologi Malaysia
//****************************************************************************************

#include <stdio.h>
#include <iostream>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include <conio.h>

//Set parameters
#define iteration   20
#define sample      4
#define feature     2
#define threshold   0.2
#define learnrate   0.1

int     a,b,i=1;
float   sum, testoutput;
float   InputArray[sample][feature],OutputArray[sample],weight [sample],output[sample],testinput[feature];

FILE    *InputFile;
FILE    *OutputFile;
FILE    *TrainingFile;
void    initialize ();
void    readdata ();
void    training ();
void    test ();

int main ()
{
    srand(time(0));     //Reset random number generator
    TrainingFile = fopen("Training.txt","w");

    initialize ();
```

Line: 1   Col: 86   Sel: 0   Lines: 144   Length: 2868   Insert   Done parsing in 1.172 seconds

Perceptron Assignment:

Using Lab Exercise 1 code, solve OR and NOT problem using perceptron. Select your own threshold and learning rate values.

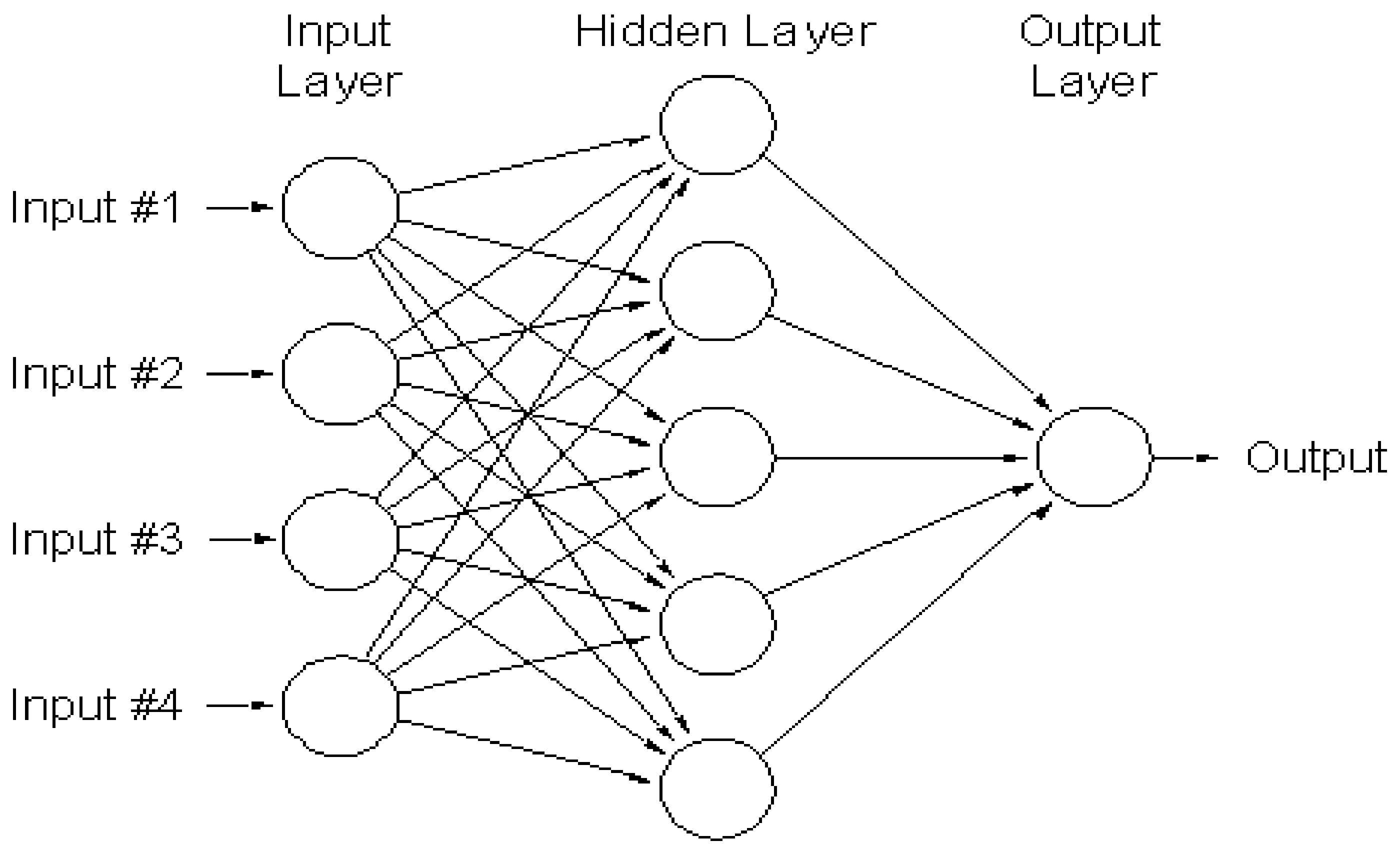
Introduction to Multilayer Perceptron(MLP)

A multilayer perceptron is a feedforward neural network with one or more hidden layers.

The network consists of:
-An input layer of source neurons,
-At least one middle or hidden layer of computational neurons,
-An output layer of computational neurons.

The input signals are propagated in a forward direction on a layer-by-layer basis.

Introduction to MLP



Introduction to MLP

Learning in MLP: Backpropagation

- Learning in a multilayer network proceeds the same way as for a perceptron.
- A training set of input patterns is presented to the network.
- The network computes its output pattern, and if there is an error - or in other words a difference between actual and desired output patterns - the weights are adjusted to reduce this error.

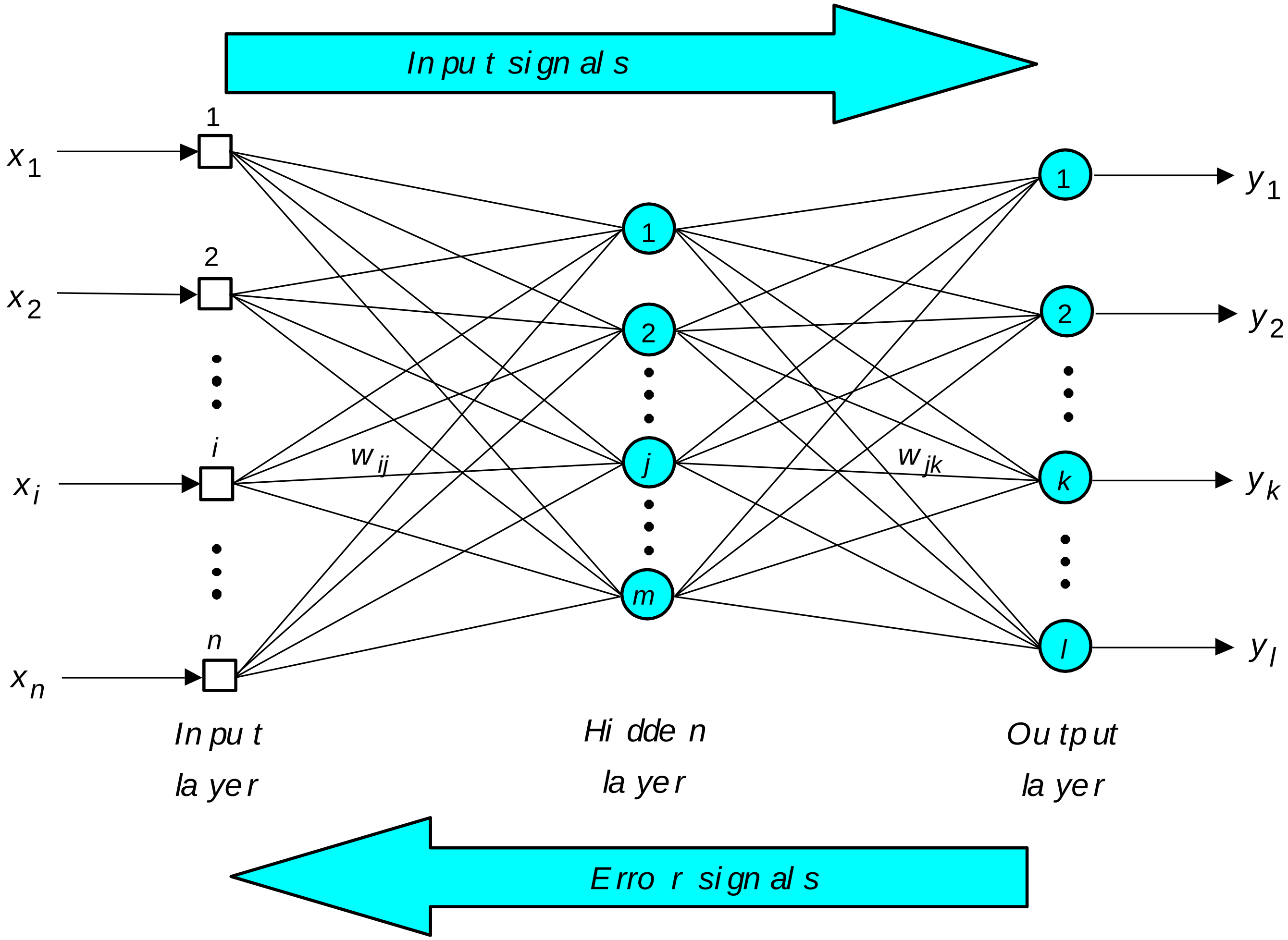
Introduction to MLP

- In a back-propagation neural network, the learning algorithm has two phases.

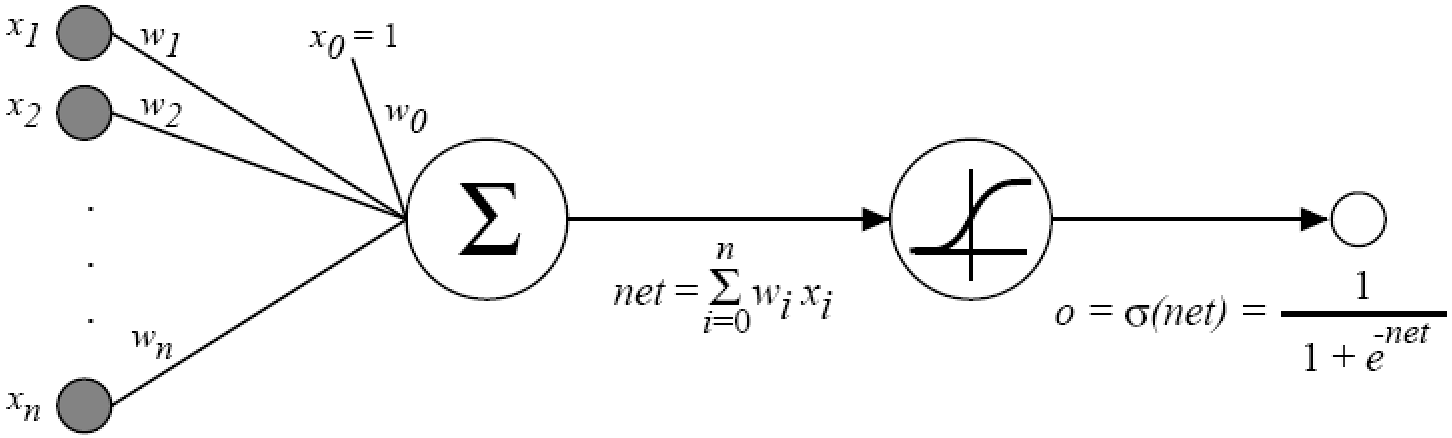
1. A training input pattern is presented to the network input layer. The network propagates the input pattern from layer to layer until the output pattern is generated by the output layer.

2. If this pattern is different from the desired output, an error is calculated and then propagated backwards through the network from the output layer to the input layer. The weights are modified as the error is propagated.

Introduction to MLP



Activation Function

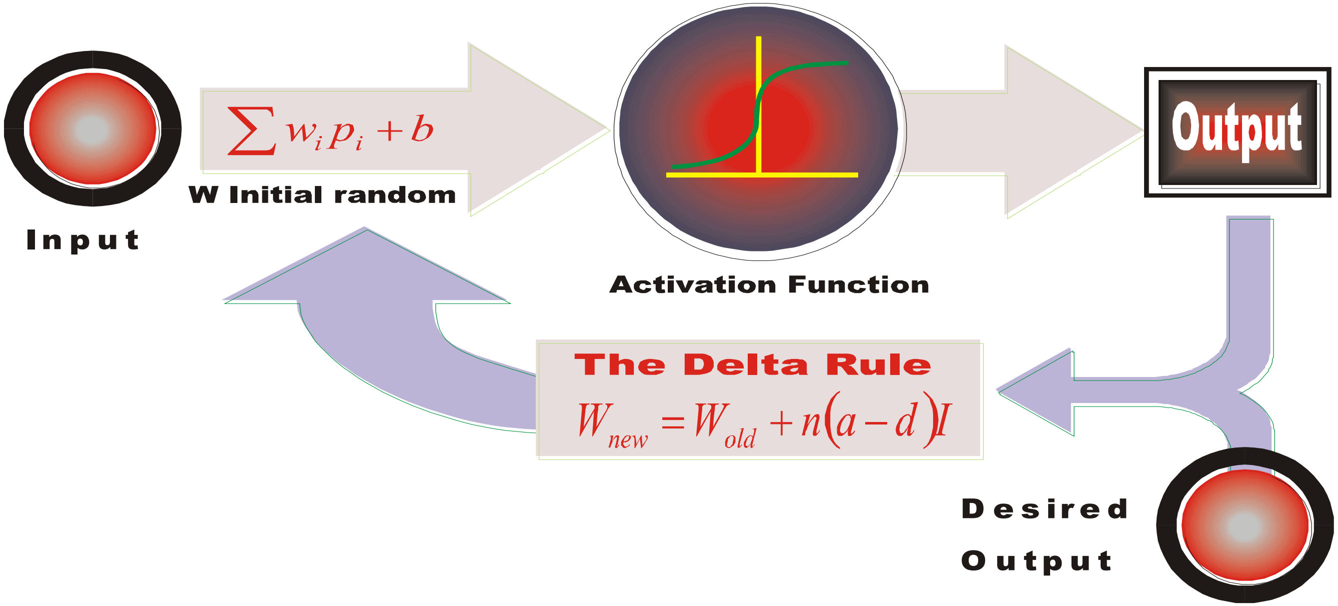


Sigmoid activation function: $f(x) = \frac{1}{1 + e^{-x}}$

Delta Rules

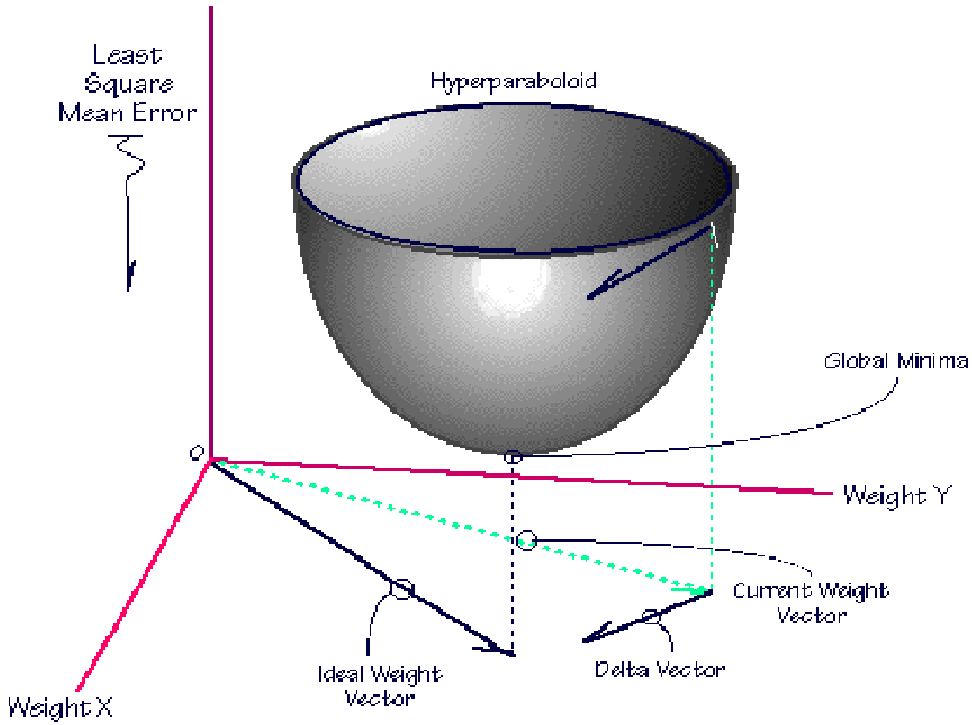
Often utilized by backpropagation

The network sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights.



Delta Rules

Backpropagation performs a gradient descent within the solution's vector space towards a global minimum and avoiding local minimum



Backpropagation Algorithm

1. Calculate the outputs of all neurons in the hidden layer:

$$x = \sum_{i=1}^{n} x_i \times w_i + bias$$

$$O_j = f(x) = \frac{1}{1 + e^{-x}}$$

2. Calculate the outputs of (all) neuron(s) in the output layer:

$$x = \sum_{i=1}^{n} x_i \times w_i + bias$$

$$O_k = f(x) = \frac{1}{1 + e^{-x}}$$

Backpropagation Algorithm

3. Calculate output error:

$$\delta_k = O_k \times (1 - O_k) \times (t - O_k)$$

4. Update weight between hidden-output layer:

$$\Delta w_{(jk)} = \alpha \times O_j \times \delta_k$$

$$w_{(jk)_{(t+1)}} = w_{(jk)_t} + \Delta w_{(jk)}$$

5. Calculate hidden error:

$$\delta_j = O_j \times (1 - O_j) \times (\sum_{k=1}^{n} \delta_k \times w_{(jk)})$$

Backpropagation Algorithm

6. Update weight between input-hidden layer:

$$\Delta w_{(ij)} = \alpha \times x_i \times \delta_j$$

$$w_{(ij)_{(t+1)}} = w_{(ij)_t} + \Delta w_{(ij)}$$

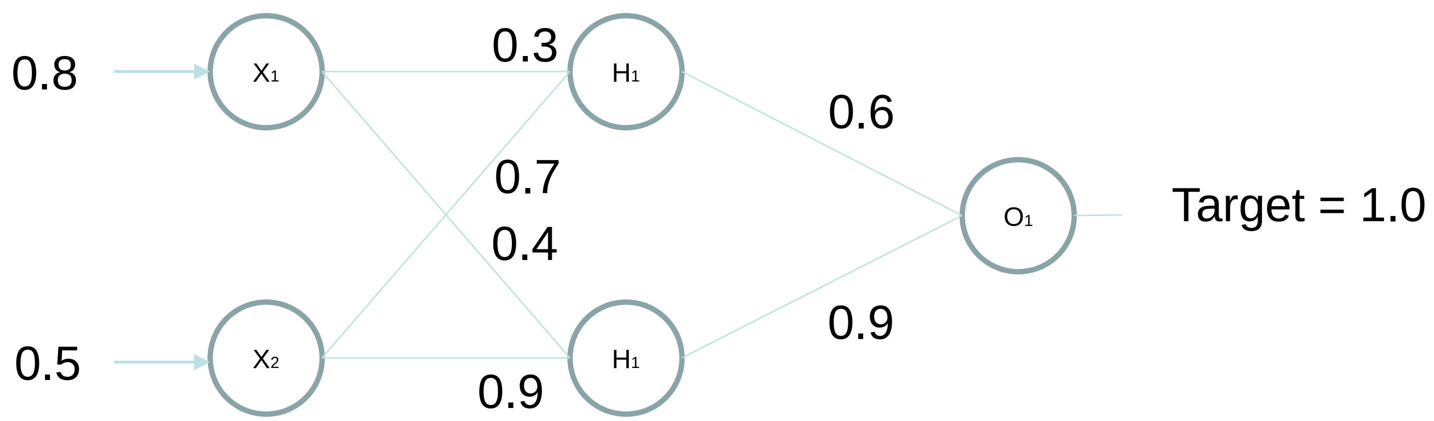
i = Input

j = Hidden

k = Output

Example:

Input (X)	Weight (Input to Hidden)	Weight (Hidden to Output)
X_1 = 0.8	X_1 to H_1 = 0.3	H_1 to O_1 = 0.6
X_2 = 0.5	X_1 to H_2 = 0.4	H_2 to O_1 = 0.9
	X_2 to H_1 = 0.7	
	X_2 to H_2 = 0.9	
Learning rate α = 0.6 **Target Output = 1.0**		



Example

1. Calculate the outputs of all neurons in the hidden layer:

   H_1 = (0.8 x 0.3) + (0.5 x 0.7) = 0.59, Sigmoid = 1/1+2.71828^-0.59 = 0.6434

   H_2 = (0.8 x 0.4) + (0.5 x 0.9) = 0.77, Sigmoid = 0.6835

2. Calculate the outputs of (all) neuron(s) in the output layer:

   O_1 = (0.6434 x 0.6) + (0.6835 x 0.9) = 1.0012, Sigmoid = 0.7313

3. Calculate output error

   Error = 0.7313 x (1-0.7313) x (1-0.7313) = 0.0528

Example

4. Update weight between hidden-output layer :

   $W_{H1\text{-}O1} = 0.6 + (0.6 \times 0.6434 \times 0.0528) = 0.6204$

   $W_{H2\text{-}O1} = 0.9 + (0.6 \times 0.6835 \times 0.0528) = 0.9217$

5. Calculate hidden error

6. Update weight between input-hidden layer:

   Error gradient $H_1 = (0.0528 \times 0.6) \times ((1\text{-}0.6434) \times 0.6434) = 0.0073$

   $W_{X1\text{-}H1} = 0.3 + (0.6 \times 0.8 \times 0.0073) = 0.3035$

   $W_{X2\text{-}H1} = 0.7 + (0.6 \times 0.5 \times 0.0073) = 0.7021$

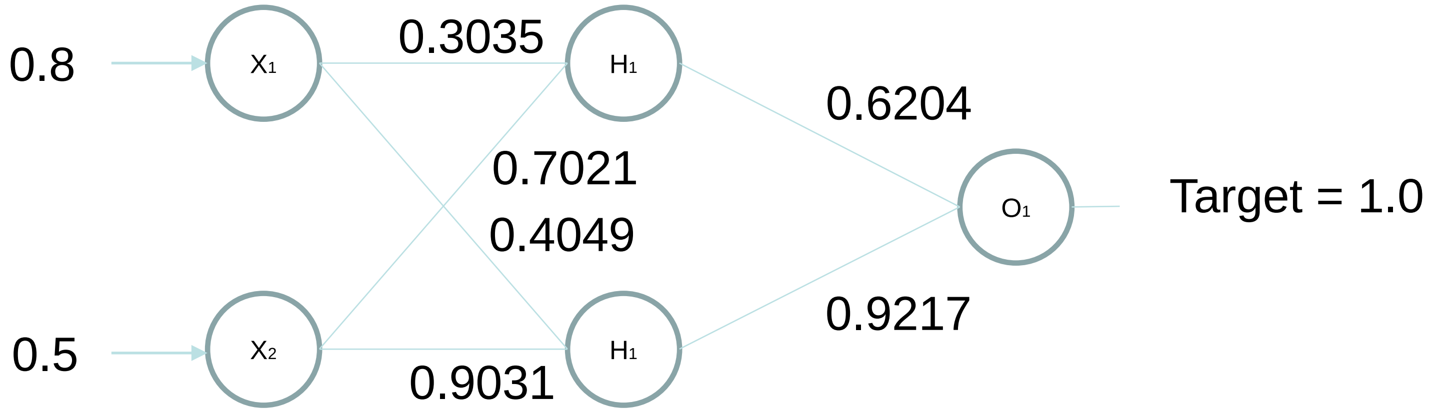
   Error gradient $H_2 = (0.0528 \times 0.9) \times ((1\text{-}0.6835) \times 0.6835) = 0.0103$

   $W_{X1\text{-}H2} = 0.4 + (0.6 \times 0.8 \times 0.0103) = 0.4049$

   $W_{X2\text{-}H2} = 0.9 + (0.6 \times 0.5 \times 0.0103) = 0.9031$

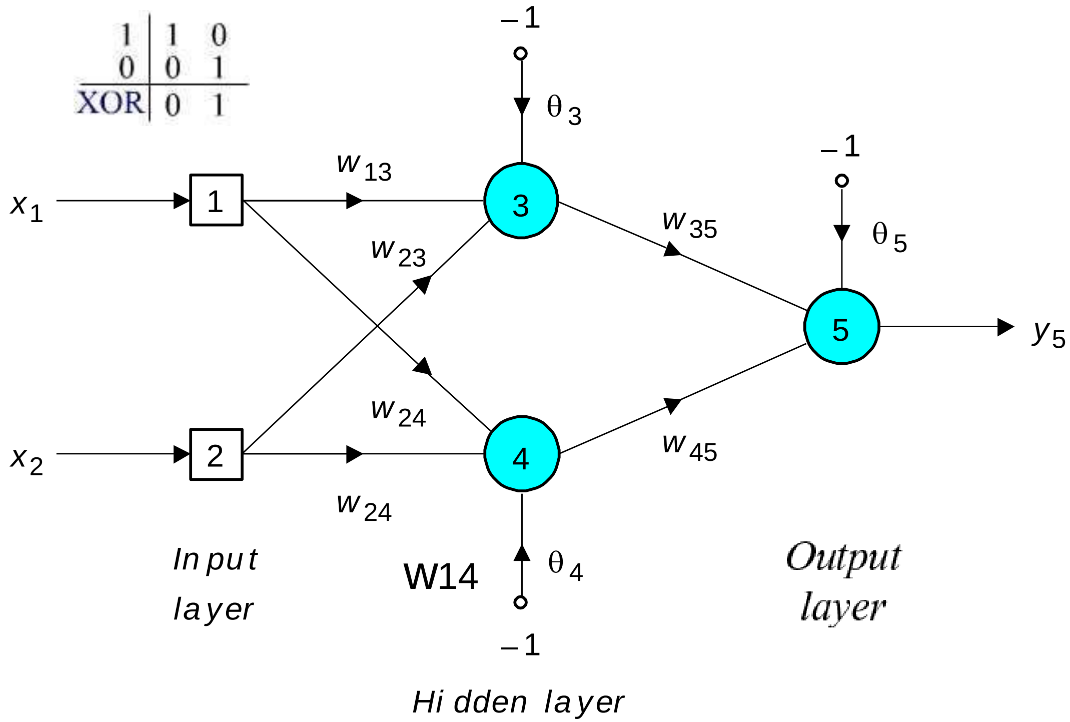
Quiz

New weights:



Execute the feed-forward network to calculate the new output using the updated weights. Discuss the current output compared to the previous output. Perform 2nd iteration backpropagation and feedforward and compare the new result with previous result.

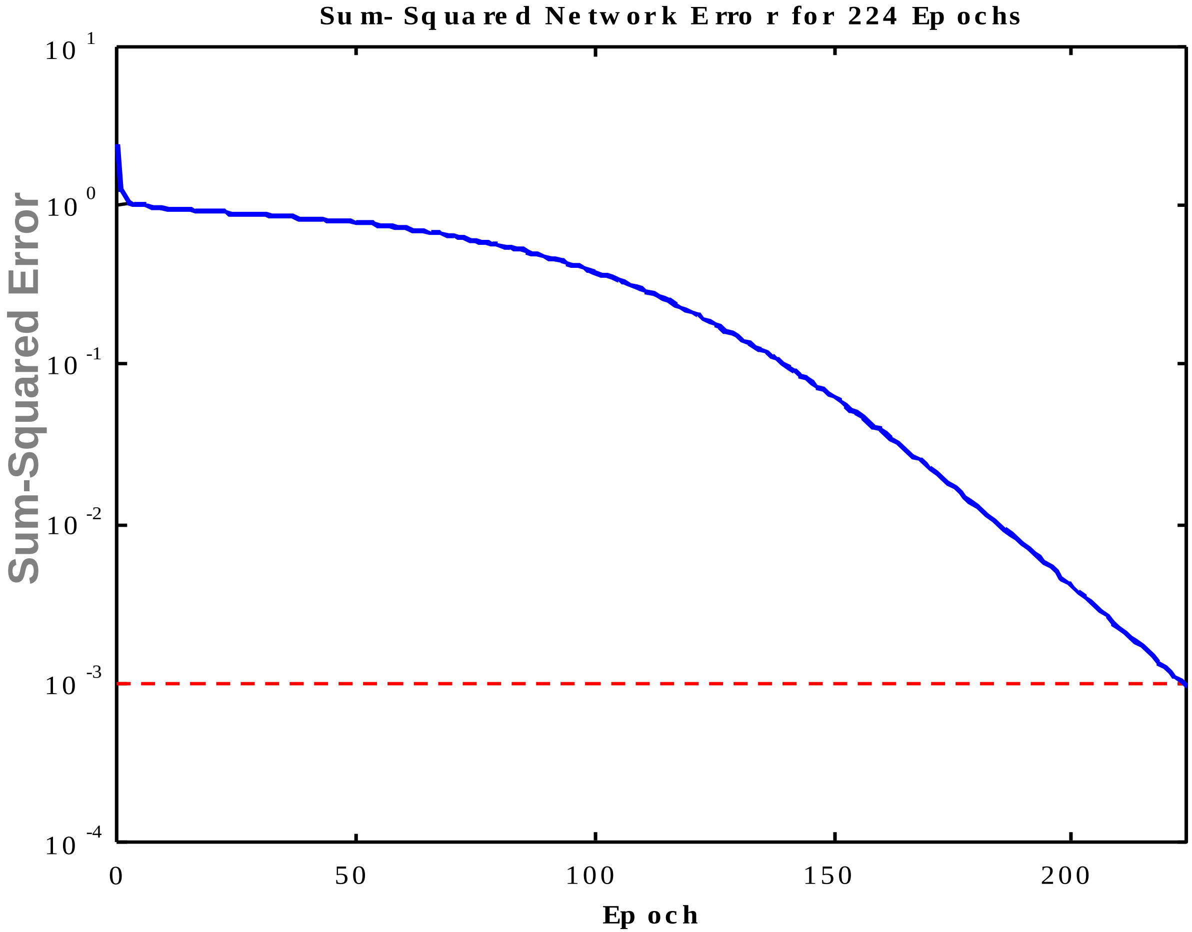
Example: XOR Problem



The initial weights and threshold levels are set randomly as follows:

w13 = 0.5, w14 = 0.9, w23 = 0.4, w24 = 1.0, w35 = -1.2, w45 = 1.1
θ3 = 0.8, θ 4 = -0.1 and θ 5 = 0.3.

Example: XOR Problem

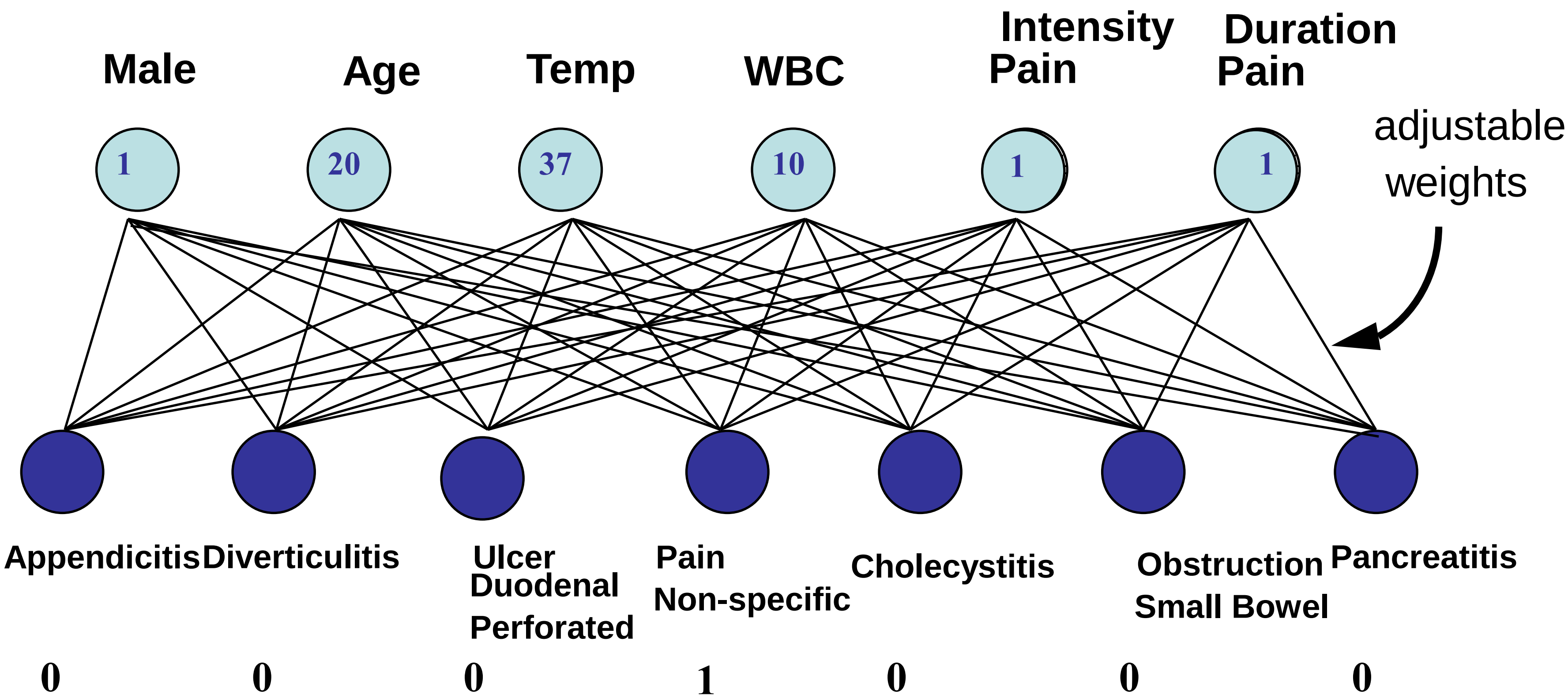


Example: XOR Problem

Inputs		Desired output	Actual output	Error	Sum of squared errors
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	−0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	−0.0175	

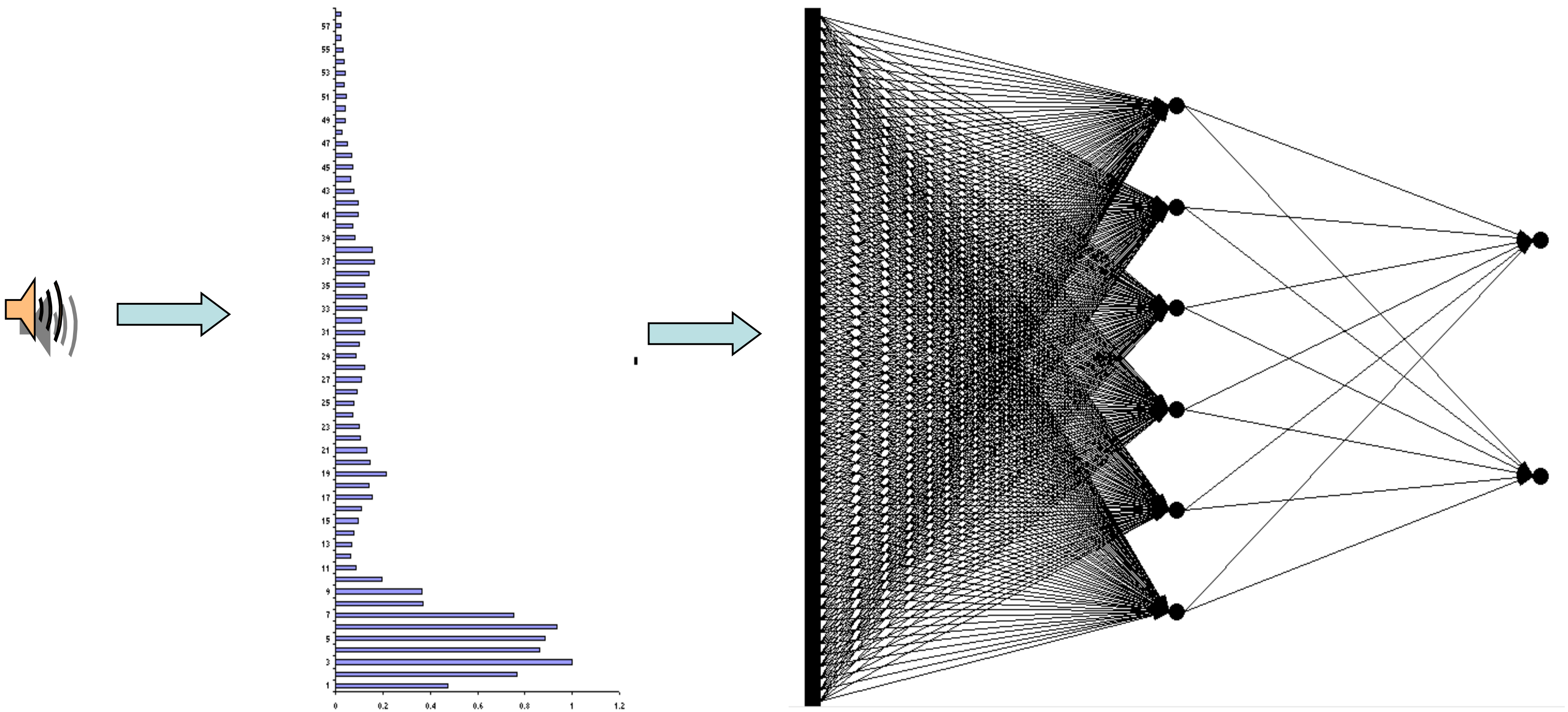
Applications

Abdominal Pain Prediction



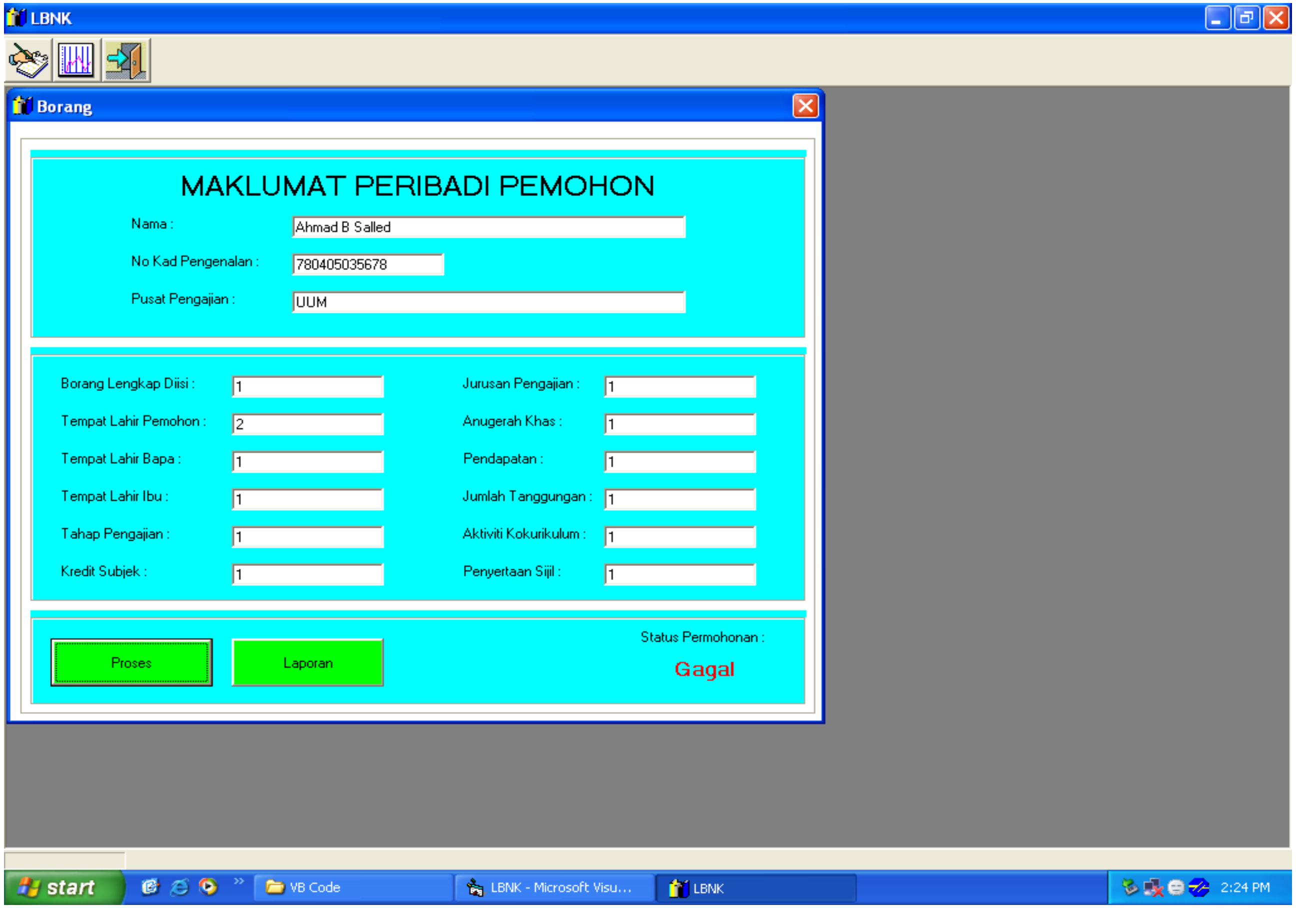
Applications

Voice Recognition



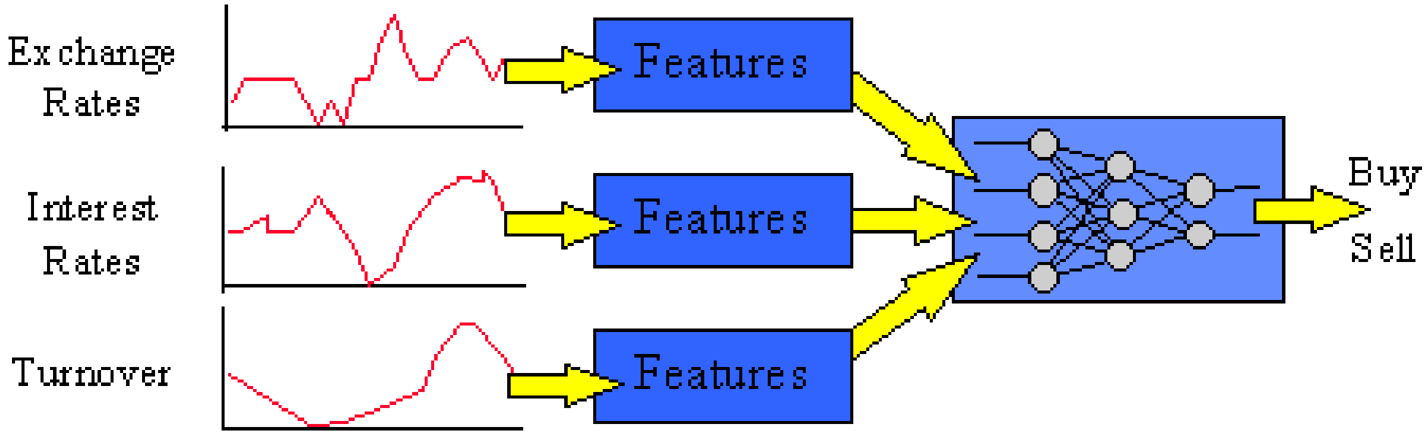
Applications

Educational Loan Forecasting System



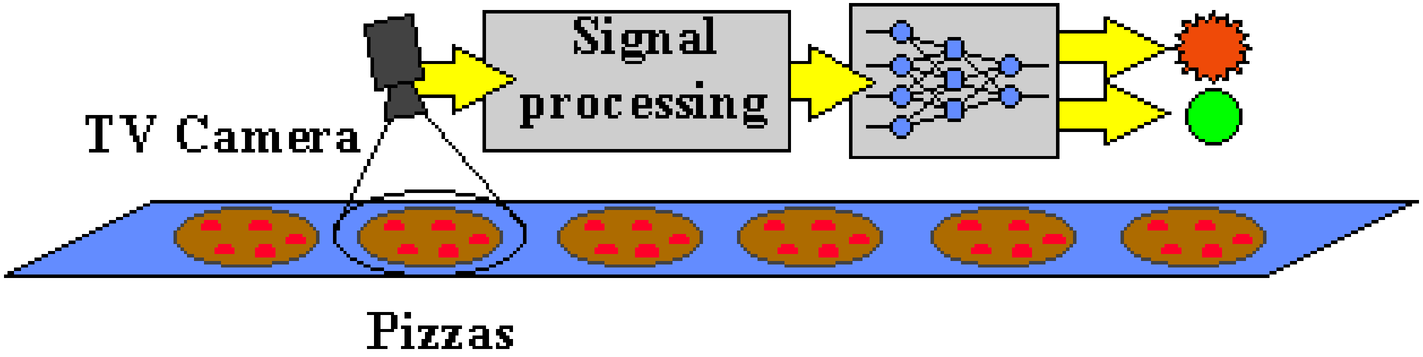
Applications

Stock Market Prediction



Applications

Automated Industrial Inspection



Traditional VS ANN

CHARACTERISTICS	TRADITIONAL COMPUTING (including Expert Systems)	ARTIFICIAL NEURAL NETWORKS
Processing style	Sequential	Parallel
Functions	Logically (left brained) via Rules Concepts Calculations	Gestault (right brained) via Images Pictures Controls
Learning Method	by rules (didactically)	by example (Socratically)
Applications	Accounting, word processing, math, inventory, digital communications	Sensor processing, speech recognition, pattern recognition, text recognition

Lab Exercise 2

Download and install WEKA

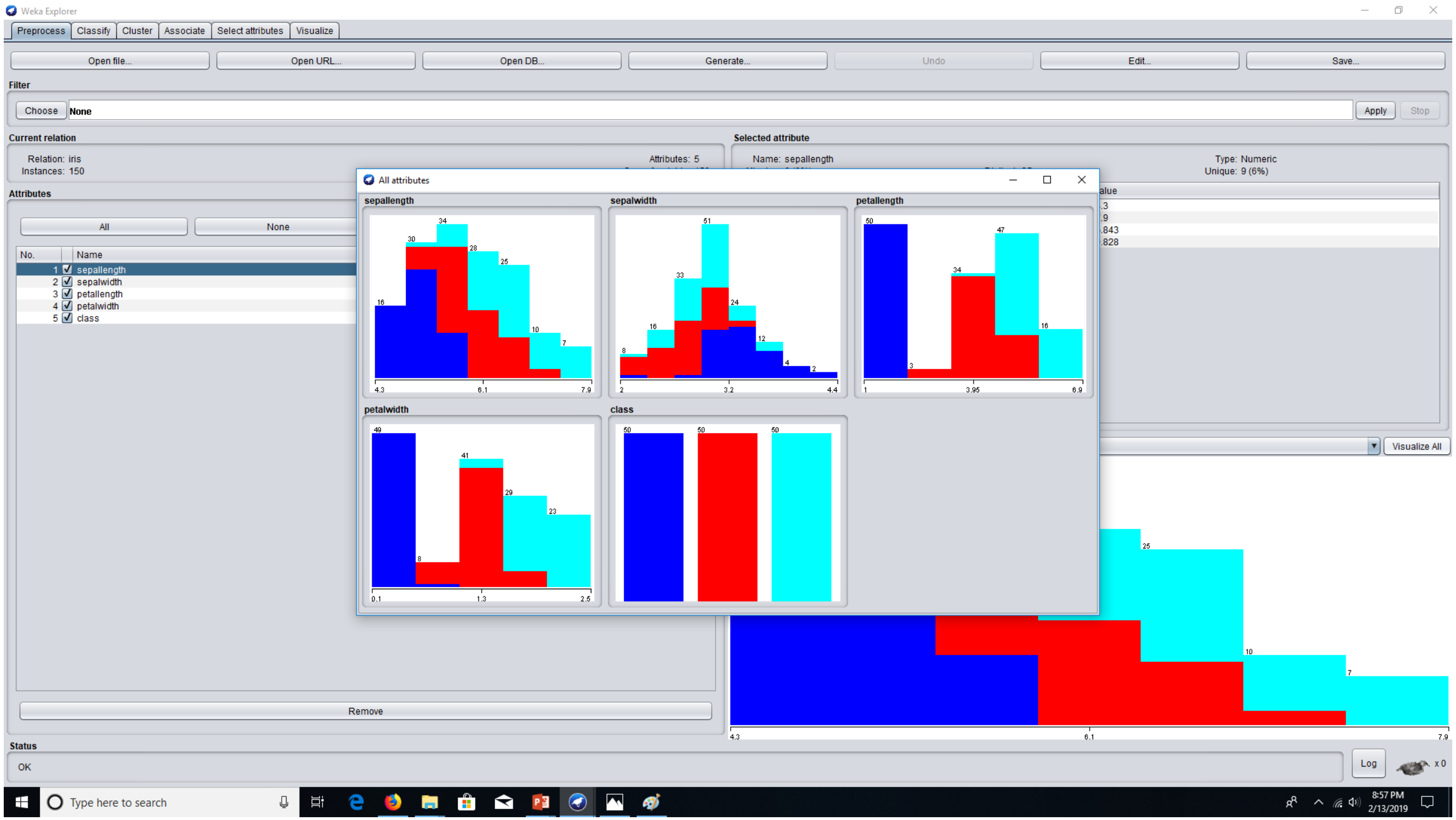
https://www.cs.waikato.ac.nz/ml/weka/

Download dataset from UCI Machine Learning and prepare ARFF format

https://www.cs.waikato.ac.nz/ml/weka/arff.html

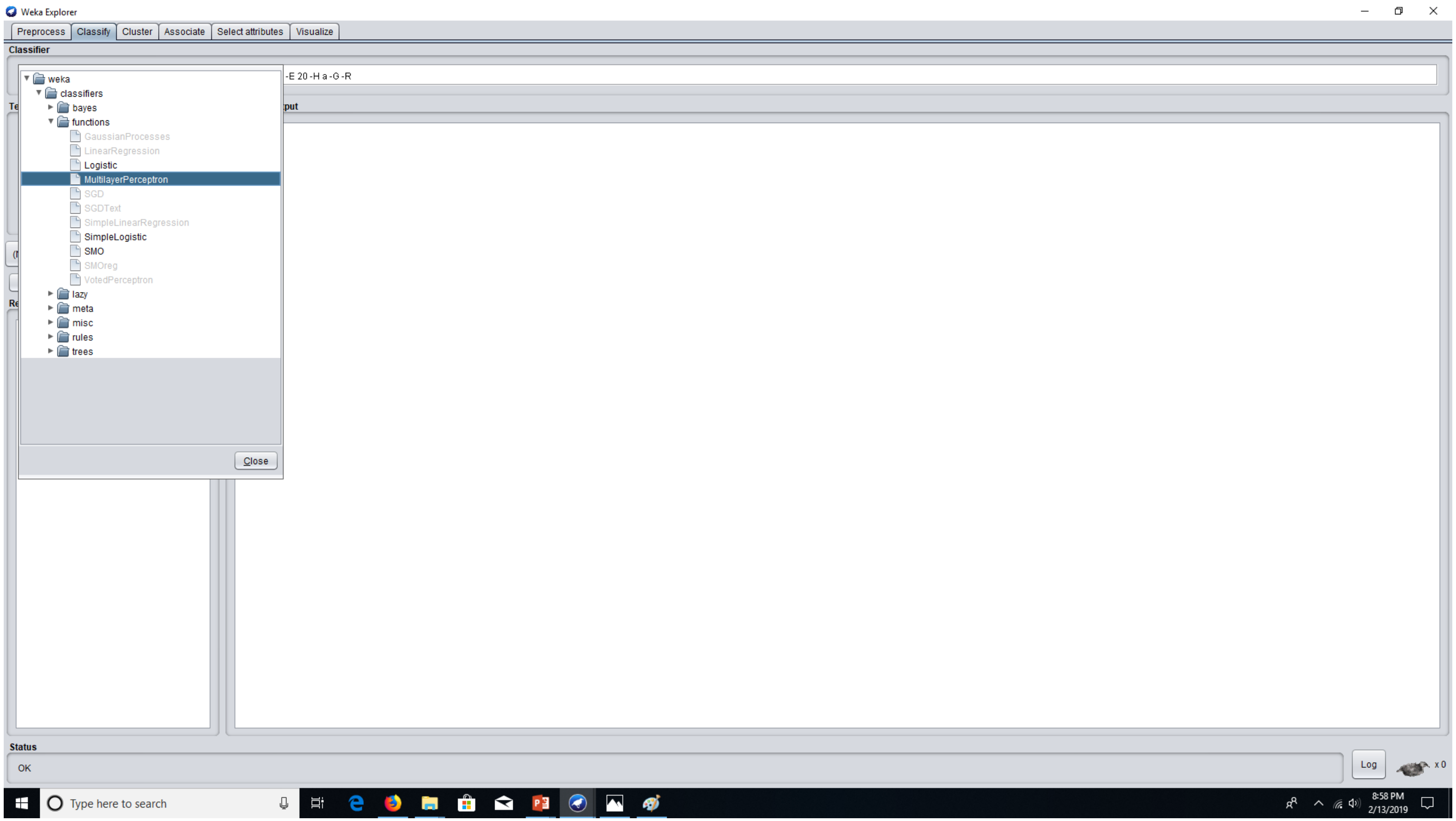
Lab Exercise 2

Data visualization

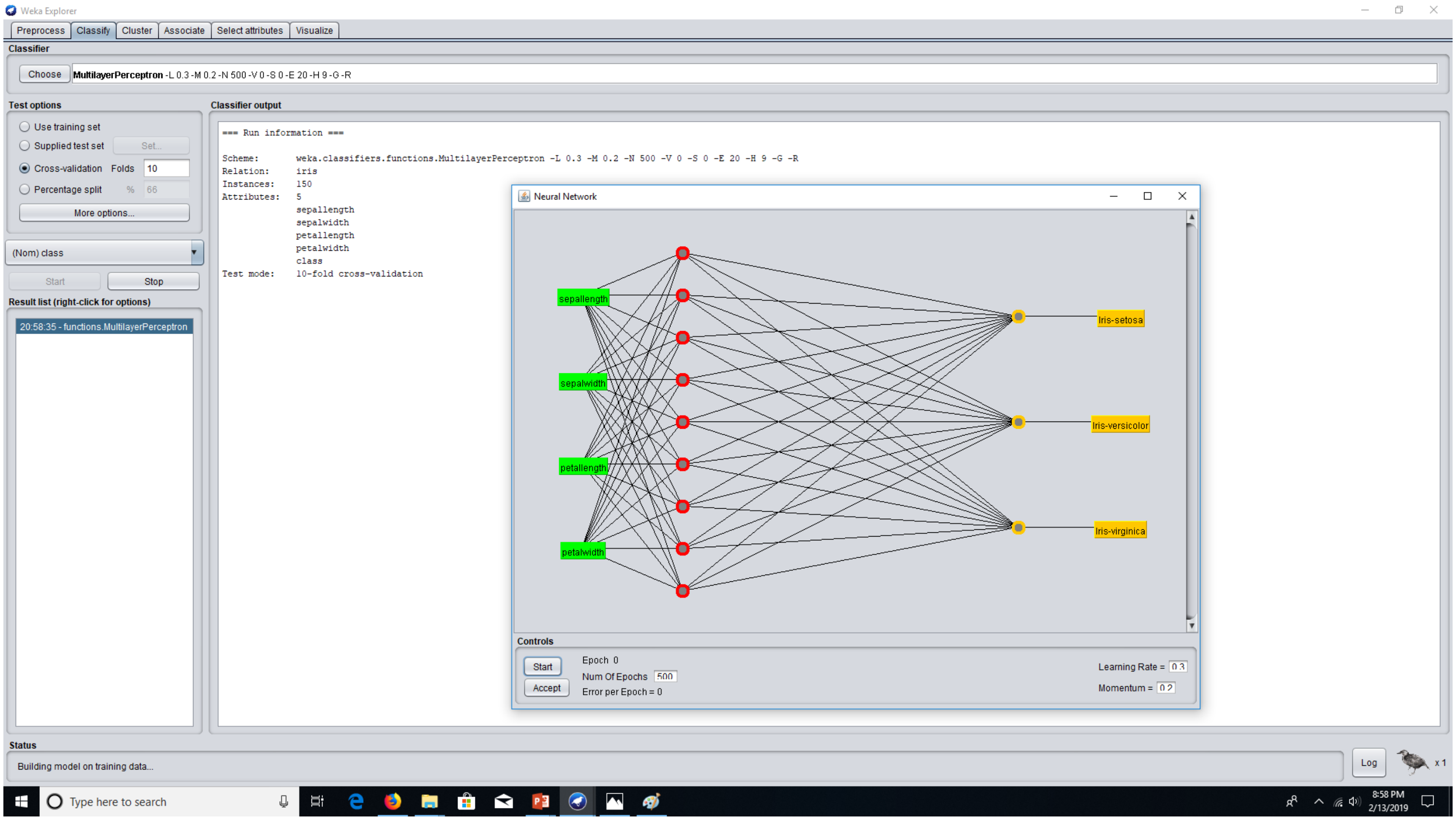


Lab Exercise 2

Classification using MLP



Lab Exercise 2



Lab Exercise 3

Implementation of MLP using C/C++/Python

```
//*******************************************************************************
// Artificial Neural Network with BackPropagation Learning Algorithm
// Written by Dr Haza Nuzly & Prof Siti Mariyam
// Soft Computing Research Group, Universiti Teknologi Malaysia
//*******************************************************************************

#include <stdio.h>
#include <tchar.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>

#define InputUnitNo     11      /* bil. nod pada aras input - aras i       */
#define HiddenUnitNo    23      /* bil. nod pada aras tersembunyi - aras j */
#define OutputUnitNo    4       /* bil. nod pada aras output  - aras k     */
#define TrainPatternNo  180     /* bilangan pola input                     */
#define TestPatternNo   20      /* bilangan input test                     */
#define TargetError     0.05    /* Ralat Maximum                           */
#define Iteration       10000
#define LRate           0.10    /* Training rate                           */
#define MRate           0.90    /* Momentum factor                         */
#define Wmin            -1.0    /* Digunakan dalam fungsi rawak            */
#define Wmax            1.0 /* bagi menjana nilai rawak               */

#define f(x)  1/(1+exp(-(x)))                                    //Sigmoid
#define rnd() ((double)rand()/0x7fff*((Wmax)-(Wmin)) + (Wmin))      //Random

char    Check[30];
double  TrainInputData[TrainPatternNo][InputUnitNo];           /* Output bagi Aras Input - i           */
double  TrainOutputData[TrainPatternNo][OutputUnitNo];
double  TestInputData[TestPatternNo][InputUnitNo];
double  TestOutputData[TestPatternNo][OutputUnitNo];
double  hidOut[HiddenUnitNo];                        /* Output bagi Aras Tersembunyi - j     */
double  outOut[OutputUnitNo];                        /* Output bagi Aras Output - k          */
double  wHidInp[HiddenUnitNo][InputUnitNo];          /* Pemberat dari aras  i kepada aras j  */
```

MLP Assignment 1:

From Lab Exercise 3 code, classify Iris dataset using MLP. Explore best MLP parameter values for Iris classification problem.

MLP Assignment 2:

Discuss other ANN architectures:

1. Hopfield Network
2. Kohonen Network
3. Self-Organizing Map (SOM)

And others….