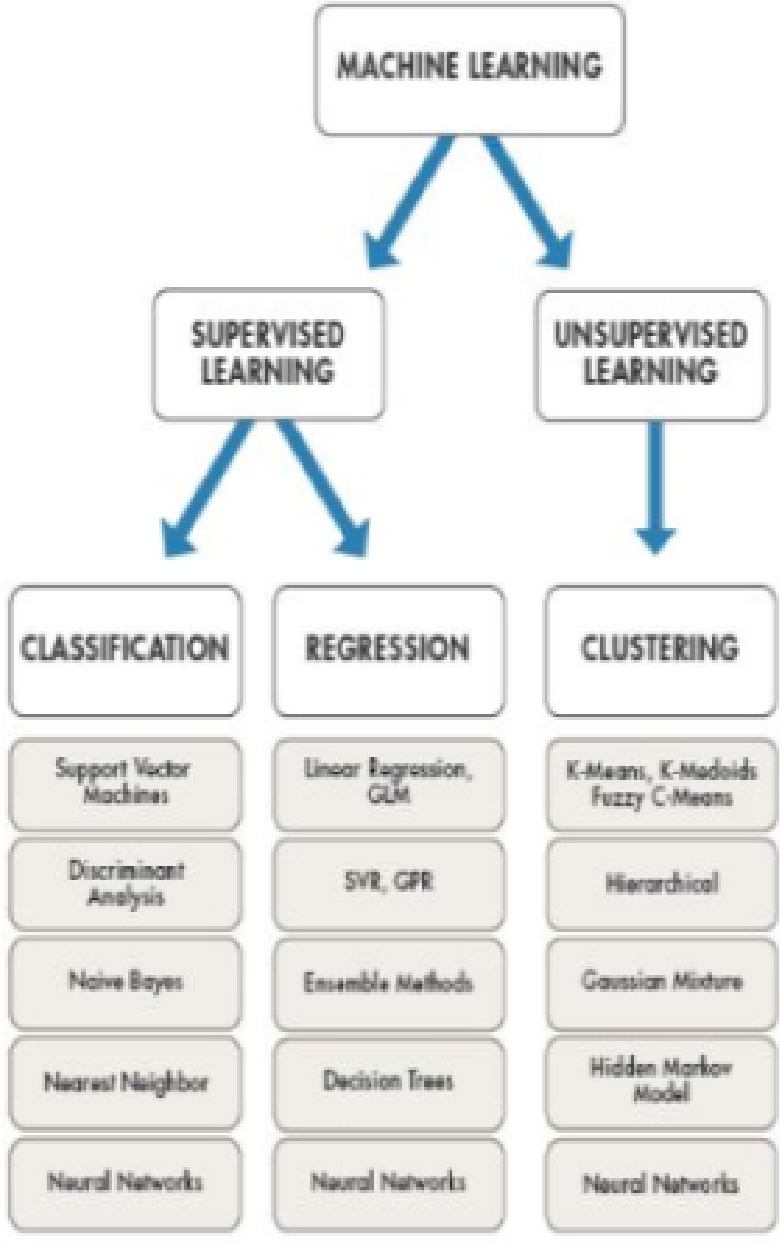
Computational Intelligence Techniques

Busra Corak

Izmir Katip Celebi University

November 2019


MACHINE LEARNING
SUPERVISED LEARNING
UNSUPERVISED LEARNING
CLASSIFICATION
REGRESSION
CLUSTERING
Support Vector Machines
Discriminant Analysis
Naive Bayes
Nearest Neighbor
Neural Networks
Linear Regression, GLM
SVR, GPR
Ensemble Methods
Decision Trees
Neural Networks
K-Means, K-Medoids Fuzzy C-Means
Hierarchical
Gaussian Mixture
Hidden Markov Model
Neural Networks

Overview

- Artificial Neural Network
- Hidden Markov Model
- Fuzzy Sets and Fuzzy Logic
- Support Vector Machines
- Hybrid Systems

Artificial Neural Network

- The Perceptron/ADALINE-MADALINE Units
- Multilayer Perceptrons
- Hopfield Neural Networks
- Kohonen's Self-Organizing Map
- LVQ(Linear Vector Qantization) Networks
- ART(Adaptive Resonance Theory) Networks
- Counter-Propagation Network
- Cognitron and Neo-cognitron Network

- Studies on artificial neural networks have started with the perceptrons.
- The most important feature of these sensors is that they divide the problem space into classes with a line or a plane.
- The inputs of the problem are multiplied by the weights. The class of input is then determined according to whether the value obtained is greater than or less than a threshold value.Classes are represented by the numbers 1 or –1 (sometimes 1 and 0).
- The most important problem of single-layer sensors is that they cannot learn nonlinear events.

The Perceptron/ADALINE-MADALINE Units

- In the perceptron, weights are changed, while the inputs are multiplied by a constant called the learning coefficient λ and added or subtracted to the weights.Based on the inputs presented to the network, weights are increased or decreased according to the value of the output produced.
- In the ADALINE unit, changing weights is based on the difference between the expected output and the actual output (error) The new weight values are determined by adding the value obtained by multiplying the inputs with a learning coefficient λ of the error to the old weights.

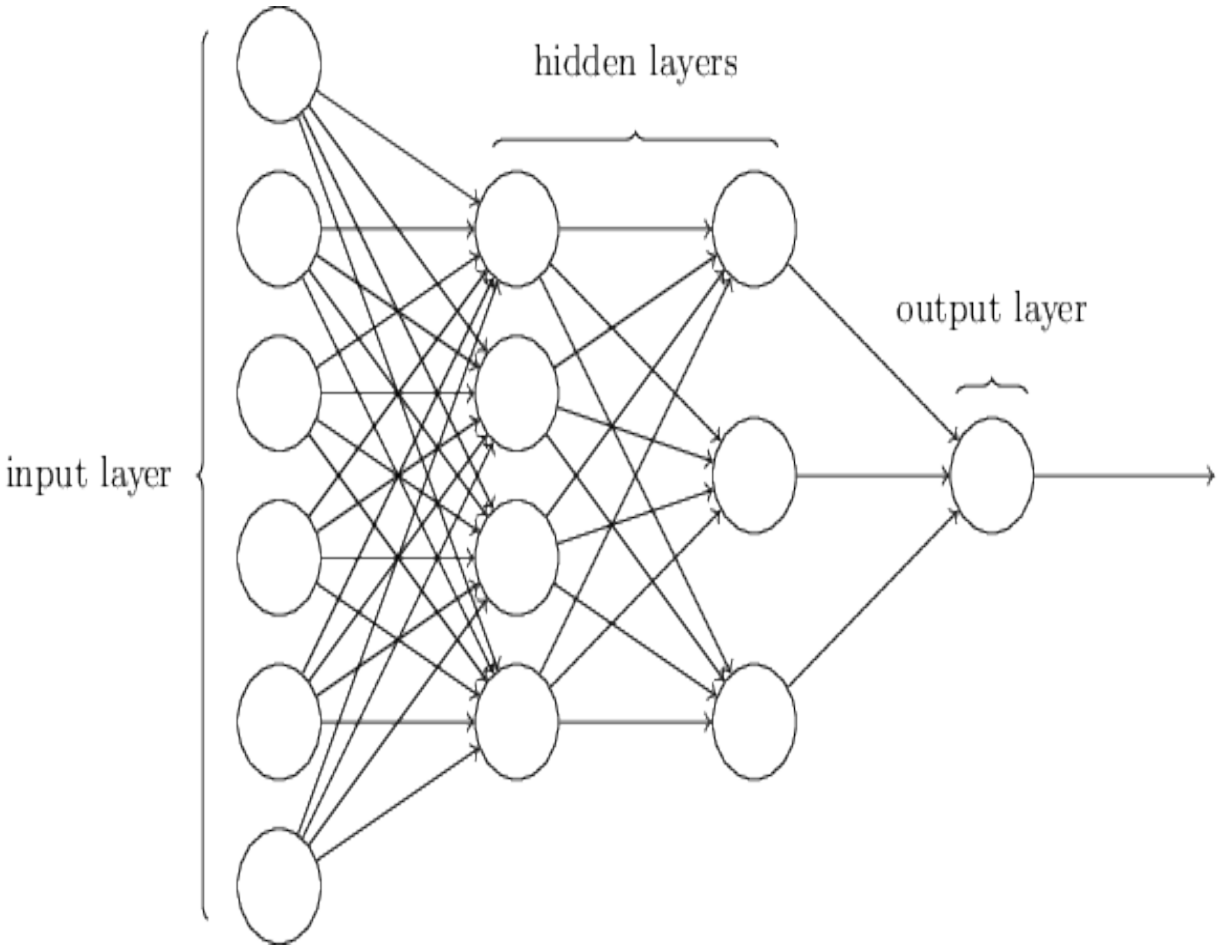
$$W_i^{t+1} = [W_i^t - \lambda(y_i - f(u_i))X_i] \tag{1}$$

Multilayer Perceptrons

The most widely used model of artificial neural networks today is multilayer perceptrons.Multilayer perceptrons have emerged as a result of studies to solve the XOR problem.These perceptrons consist of three layers:

- Input layer: Takes information from the outside world.In this layer there is no information processing.
- Inter-Layer (Hidden layer) : Processes information from the input layer.It is possible to solve many problems with one hidden layer.If the relationship between the input / output of the problem to be learned by the network is not linear and the complexity increases, it can be used in more than one intermediate layer.
- Output layer:It processes the information from the intermediate layer and finds the output produced by the network for the input presented to the network. This output is transmitted to the outside world.

Multilayer Perceptron



Multilayer Perceptron

- Training of the MLP network is based on the “generalized delta rule.MLP network uses supervised learning strategies.During the training, the inputs and the outputs that the network must produce in response to those inputs are shown to the network.
- The difference between the outputs produced by the network and the outputs it should produce represented as ERROR.
- During learning, the inputs are first presented to the network to produce outputs corresponding to those inputs. This process is called forward calculation.Then the produced output is compared with the expected output,weights are changed by distributing the error backwards. This is called backward calculation.

Factors affecting the learning of MLP networks are:

- Selection of samples
- Networking of inputs and outputs
- Numerical representation of inputs and outputs
- Assigning initial values for weights
- Determination of learning coefficient and momentum coefficients
- Networking of samples
- Changing weights times
- Scaling of inputs and outputs
- Determination of stop criterion
- Growing and pruning networks

Applications of MLP networks in engineering problems:

- Classification
- Prediction
- Recognition
- Interpretation
- Diagnosis

LVQ (Linear Vector Quantization) Model

- LVQ networks use the reinforcement learning . During the training, only the inputs that are wanted to be learned are given to the network and the network is asked to produce the output itself.

LVQ network consists of 3 layers:

- Input Layer
- Kohonen Layer
- Output Layer

- Each element in the input layer connected to each element in the Kohonen layer.The weights of the connections from the input layer, to the Kohonen layer form a reference vector.Only the values (weight values) of these reference vectors are changed during learning.Only the values of a single vector are changed at each iteration.
- Each of the elements in the Kohonen layer is connected to only one element in the output layer. The weights between the Kohonen layer and the output layer are constant and their value is 1.
- Each element in the Kohonen layer represents a reference vector and competes with each other. The member with the shortest Euclidean distance wins the competition. The winner of the competition gets 1 output and the other is 0.

- The winning output element shows the class of the corresponding input.If the input is correctly classified, the corresponding reference vector is approximated to the input vector. Otherwise, it is removed.Approximation and removing based on the learning coefficient.
- The most important problem of the LVQ network is that the same vector gains very often and the learning performance of the network is poor.

ART(Adaptive Resonance Theory) Networks

ART networks use the unsupervised learning.

- The most important feature of these networks is that they can work in real time and learn online.ART networks are very powerful to adapt to new situations.
- An ART network consists of two layers, the display layer F1 and the category layer F2.

- Output values from the F2 layer are calculated by using inputs which are presented to the network from the F1 layer and upward weights.
- The process element that creates the highest output value in the F2 layer takes the value 1 as the winning process element and the others 0. The weights connected to this element are changed.In order for the winning element to show the class of the corresponding input vector, the vector in the memory connected to it must be similar to the input vector.
- This similarity is decided by a coefficient called the similarity coefficient.If two vectors are found similar, the input vector is considered an element of that class.If there is no similarity, then the orientation system creates a new class for that input vector. Therefore, the number of classes in an ART network can be as many as the number of instances.

Hopfield Networks

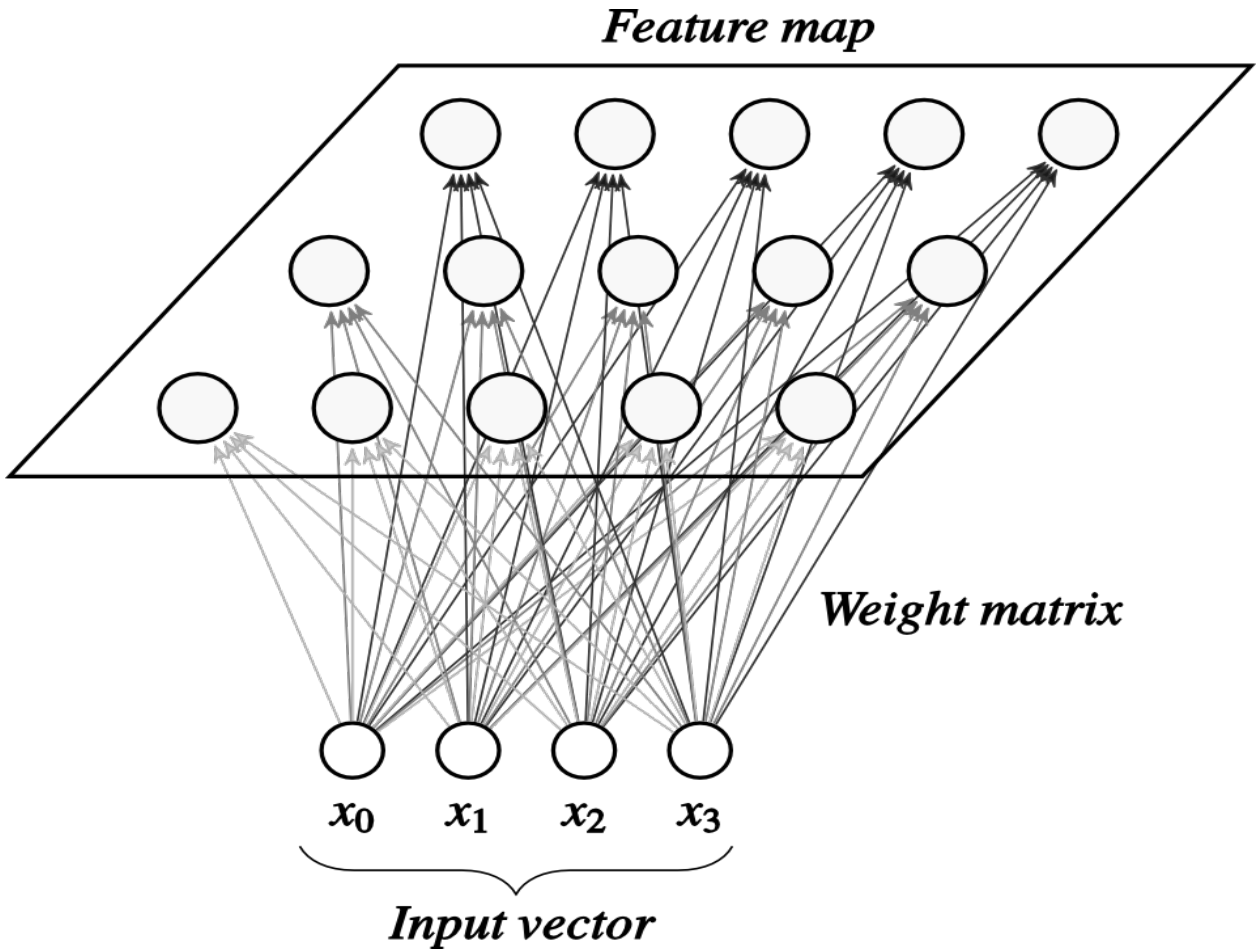
- Hopfield network consists of a single layer which contains one or more fully connected recurrent neurons.The Hopfield network is commonly used for auto-association and optimization tasks.
- Hopfield networks are associated with the concept of simulating human memory through pattern recognition and storage.

Self Orginizing Map

- A self-organizing map (SOM) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map, and is therefore a method to do dimensionality reduction.
- Self-organizing maps differ from other artificial neural networks as they apply competitive learning as opposed to error-correction learning (such as backpropagation with gradient descent), and in the sense that they use a neighborhood function to preserve the topological properties of the input space.

What really happens in SOM ?

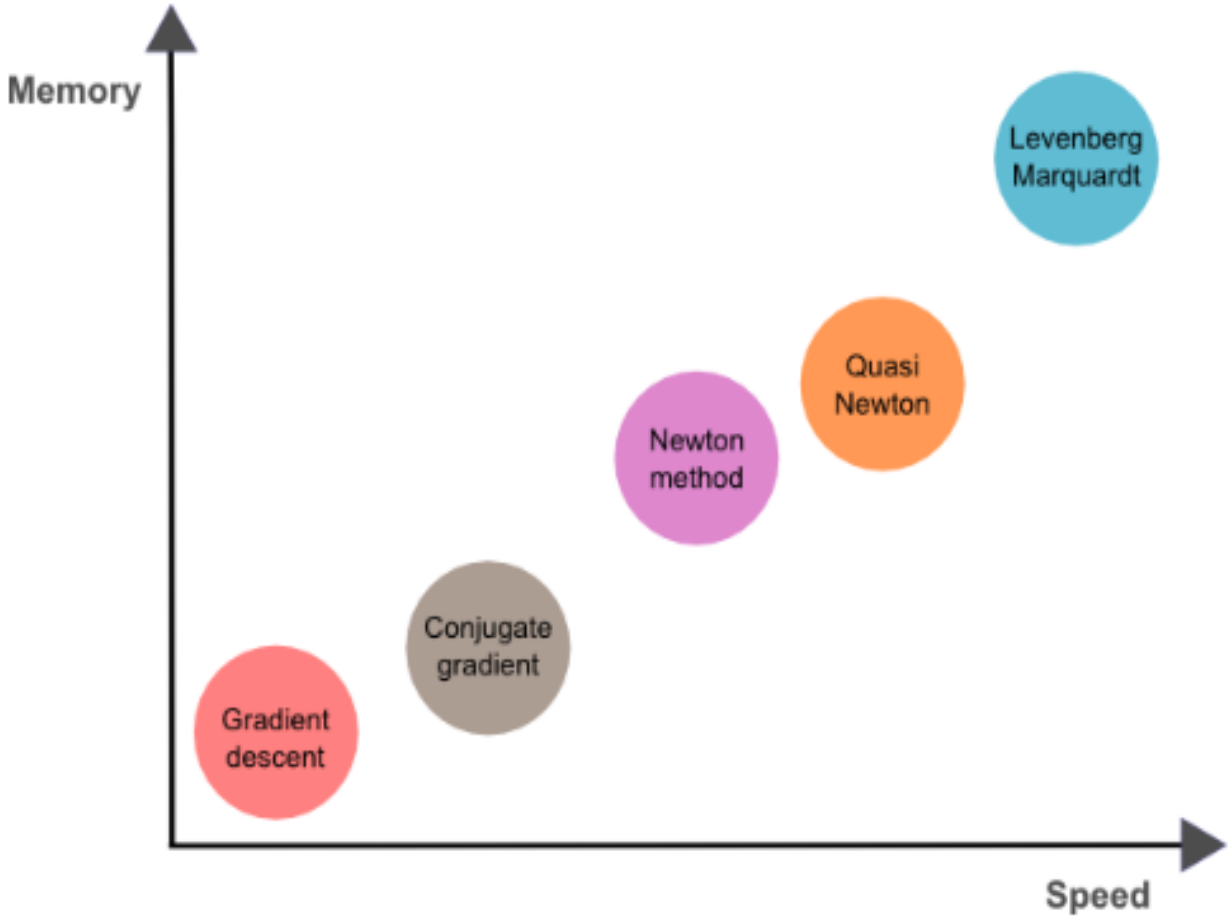
- Each data point in the data set recognizes themselves by competeting for representation.
- SOM mapping steps starts from initializing the weight vectors. From there a sample vector is selected randomly and the map of weight vectors is searched to find which weight best represents that sample.
- Each weight vector has neighboring weights that are close to it.
- The weight that is chosen is rewarded by being able to become more like that randomly selected sample vector.
- The neighbors of that weight are also rewarded by being able to become more like the chosen sample vector. This allows the map to grow and form different shapes. Most generally, they form square/rectangular/hexagonal/L shapes in 2D feature space.


Feature map
Weight matrix
x_0
x_1
x_2
x_3
Input vector

Training Algorithms

5 algorithms to train a neural network:

- Gradient descent
- Newton method
- Conjugate gradient
- Quasi-Newton method
- Levenberg-Marquardt algorithm


Memory
Levenberg Marquardt
Quasi Newton
Newton method
Conjugate gradient
Gradient descent
Speed

Creating Simple Neural Network in Phyton

- The beginning of the program just defines libraries and the values of the parameters, and creates a list which contains the values of the weights that will be modified.
- Create a function which defines the work of the output neuron. It takes 3 parameters (the 2 values of the neurons and the expected output). “outputP” is the variable corresponding to the output given by the Perceptron. Then we calculate the error, used to modify the weights of every connections to the output neuron right after.

Creating Simple Neural Network in Phyton

- Create a loop that makes the neural network repeat every situation several times. This part is the learning phase. The number of iteration is chosen according to the precision we want. However, be aware that too much iterations could lead the network to over-fitting, which causes it to focus too much on the treated examples, so it couldn't get a right output on case it didn't see during its learning phase.
- Finally, we can ask the user to enter himself the values to check if the Perceptron is working. This is the testing phase.The activation function Heaviside is interesting to use in this case, since it takes back all values to exactly 0 or 1, since we are looking for a false or true result. We could try with a sigmoid function and obtain a decimal number between 0 and 1, normally very close to one of those limits.

```
import numpy, random, os
lr = 1 #learning rate
bias = 1 #value of bias
weights = [random.random(),random.random(),random.random()]
#weights generated in a list (3 weights in total for 2 neurons and
the bias)
```

```
def Perceptron(input1, input2, output) :
    outputP = input1*weights[0]+input2*weights[1]+bias*weights[2]
    if outputP > 0 : #activation function (here Heaviside)
        outputP = 1
    else :
        outputP = 0
    error = output - outputP
    weights[0] += error * input1 * lr
    weights[1] += error * input2 * lr
    weights[2] += error * bias * lr
```

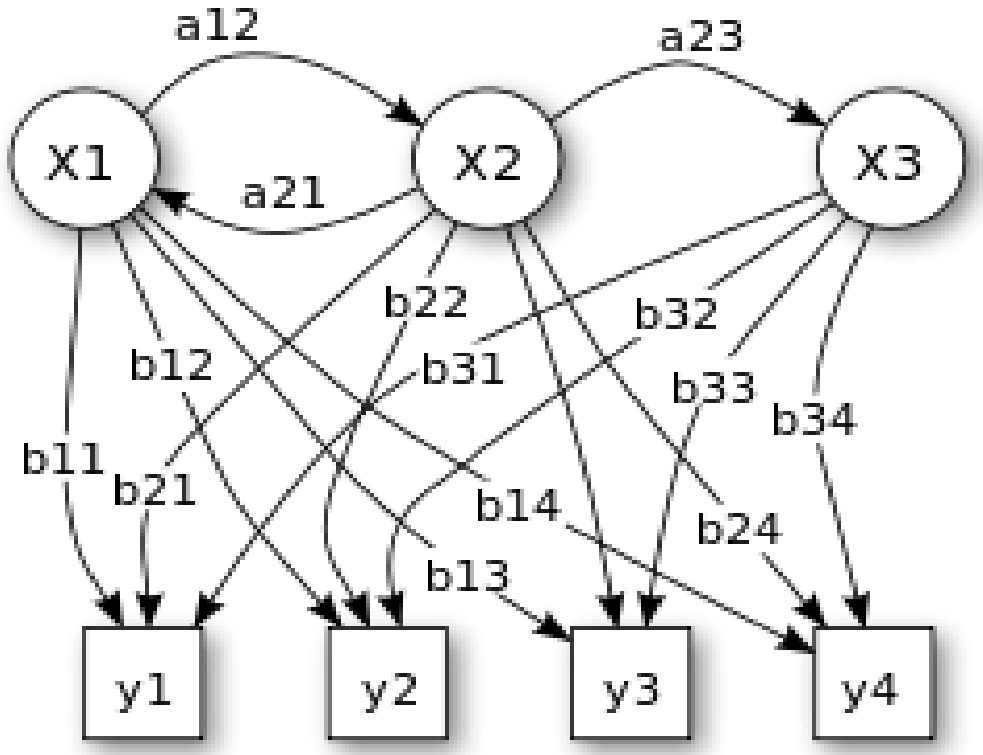
```
for i in range(50) :
    Perceptron(1,1,1) #True or true
    Perceptron(1,0,1) #True or false
    Perceptron(0,1,1) #False or true
    Perceptron(0,0,0) #False or false
```

```
x = int (input ())
y = int (input ())
outputP = x * ağırlıklar [0] + y * ağırlıklar [1] + önyargılar *
eğer çıkışP> 0 ise ağırlıklar [2] #activation işlevi
   outputP = 1
başkası :
   outputP = 0
print (x, "veya", y, "::", outputP)
```

```
outputP = 1/(1+numpy.exp(-outputP)) #sigmoid function
```

Hidden Markov Model

- The Hidden Markov Model is used as a classifier in signal processing and is used mostly in the fields of sound, handwriting, body motion recognition,word recognition, music notation monitoring, partial charge discharge, bioinformatics, gene prediction and crypto analysis.
- In the normal Markov Model, states are visible to the observer, and therefore the only parameter is the state transition probabilities.
- In the hidden Markov Model, the situation is not directly visible, but the dependent outputs are visible.
- The term Markov Analysis refers to a technique used to predict future behavior, taking into account the current behavior of the system.



- states:X
- transition probabilities state:a
- Output probabilities:b
- possible observations: Y

- Hidden Markov Models (HMMs) are a class of probabilistic graphical model that allow us to predict a sequence of unknown (hidden) variables from a set of observed variables.
- A simple example of an HMM is predicting the weather (hidden variable) based on the type of clothes that someone wears (observed).
- HMMs are probabilistic models. They allow us to compute the joint probability of a set of hidden states given a set of observed states. The hidden states are also referred to as latent states. Once we know the joint probability of a sequence of hidden states, we determine the best possible sequence.

In order to compute the joint probability of a sequence of hidden states, we need to assemble three types of information.Generally, the term "states" are used to refer to the hidden states and "observations" are used to refer to the observed states.

- Transition data — the probability of transitioning to a new state conditioned on a present state.
- Emission data — the probability of transitioning to an observed state conditioned on a hidden state.
- Initial state information — the initial probability of transitioning to a hidden state. This can also be looked at as the prior probability.

The above information can be computed directly from our training data.

For example, in the case of weather example in, our training data would consist of the hidden state and observations for a number of days. We could build our transition matrices of transitions, emissions and initial state probabilities directly from this training data.

Priors

Hot	0.6
Mild	0.3
Cold	0.1

Transitions

	Hot	Mild	Cold
Hot	0.6	0.3	0.1
Mild	0.4	0.3	0.2
Cold	0.1	0.4	0.5

Emissions

	Hot	Mild	Cold
Casual Wear	0.8	0.19	0.01
Semi Casual Wear	0.5	0.4	0.1
Winter apparel	0.01	0.2	0.79

The joint probability of the sequence, by the conditional probability chain rule and by Markov assumption, can be shown to be proportional to P(Y) below.

The probability of observing a sequence

$$Y = y(0), y(1), \ldots, y(L-1)$$

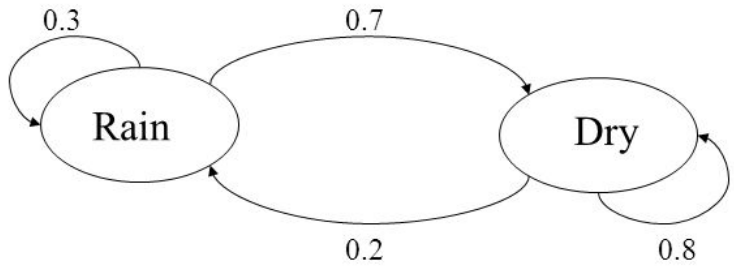
of length *L* is given by

$$P(Y) = \sum_X P(Y \mid X) P(X),$$

where the sum runs over all possible hidden-node sequences

$$X = x(0), x(1), \ldots, x(L-1).$$

Example of Markov Model

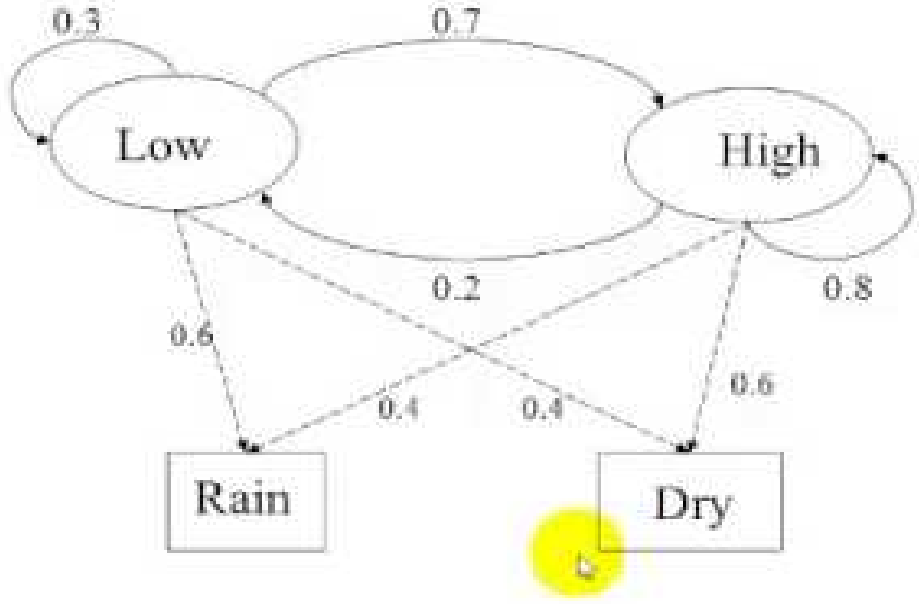


- Two states : 'Rain' and 'Dry'.
- Transition probabilities: P('Rain'|'Rain')=0.3 ,
P('Dry'|'Rain')=0.7 , P('Rain'|'Dry')=0.2, P('Dry'|'Dry')=0.8
- Initial probabilities: say P('Rain')=0.4 , P('Dry')=0.6 .

Suppose we want to calculate a probability of a sequence of states in our example, 'Dry','Dry','Rain',Rain'.

- (Dry,Dry,Rain,Rain) =
$P(Rain \mid Rain)P(Rain \mid Dry)P(Dry \mid Dry)P(Dry) = 0.3 * 0.2 * 0.8 * 0.6$

Example of Hidden Markov Model



Two states : 'Low' and 'High' **atmospheric pressure**.

Suppose we want to calculate a probability of a sequence of observations in our example, 'Dry','Rain'.Consider all possible hidden state sequences:

- P('Dry','Rain') = P(['Dry','Rain'] , ['Low','Low']) + P(['Dry','Rain'] , ['Low','High']) + P(['Dry','Rain'] , ['High','Low']) + P(['Dry','Rain'] , ['High','High'])
- P([Dry,Rain] , [Low,Low]) = P([Dry,Rain] | [Low, Low]) P([Low,Low]) =$P(Dry \mid Low)P(Rain \mid Low)$ P(Low)$P(Low \mid Low)$= 0.4*0.4*0.6*0.4*0.3

Training Algorithms

- Viterbi algorithm, it's directly applicable to the decoding problem in HMM.
- The Baum-Welch algorithm is proposed to solve the learning problem in HMM.

Fuzzy Sets and Fuzzy Logic

- The concept of fuzziness was first described in the 1960s by Lotfi Zadehfrom U.C. Berkeley in his seminal papers on fuzzy sets.
- It was extended to fuzzy logic, which is a superset of conventional Boolean logic.
- Fuzzy logic deals with a concept of partial truths, that is, truth values between "completely true" and "completely false" similar to the way fuzzy sets deal with partial memberships of an element.
- When applied to real-world problems, fuzzy logic becomes a structured, model-free estimator that approximates a function through linguistic input–output associations.

A typical fuzzy system consists of It has found many applications, some of which include fuzzy control in robotics, automation, tracking, consumer electronics, fuzzy information systems such as the Internet, information retrieval, fuzzy pattern recognition in image processing, machine vision, and in decision support problems

- Rule Base
- Membership function
- Inerface Procedure

Membership functions characterize the fuzziness in a fuzzy set – whether the elements in the set are discrete or continuous – in a graphical form for eventual use in the mathematical formalisms of fuzzy set theory.

- The most basic element of fuzzy systems is the fuzzy set. The mathematical models we have created to transfer verbal expressions to the computer.
- In a classical set, an element is either inside (1) or outside (0) of the set. In fuzzy sets, an element has any membership value between 0 and 1.

A fuzzy set is characterized by a membership function, which maps the elements of a domain, space, or an universe of discourse X to the unit interval [0, 1], written as

$$A = X \in [0, 1] \tag{2}$$

Membership functions characterize the fuzziness in a fuzzy set – whether the elements in the set are discrete or continuous – in a graphical form for eventual use in the mathematical formalisms of fuzzy set theory.

Properties of Fuzzy Sets

Having three fuzzy sets A, B and C and universal set U

- Commutative Property

$$A \cup B = B \cup A \tag{3}$$

- Associative Property

$$(A \cup B) \cup C = (B \cup C) \cup A \tag{4}$$

- Distributive Property

$$(A \cup B) \cap C = (C \cap A) \cup (C \cap B) \tag{5}$$

- Identity Property

$$A \cup U = A \tag{6}$$

$$A \cap U = U \tag{7}$$

Properties of Fuzzy Sets

Having three fuzzy sets A, B and C and universal set U

- Idempotency Property

$$A \cup A = A \tag{8}$$

$$A \cap A = A \tag{9}$$

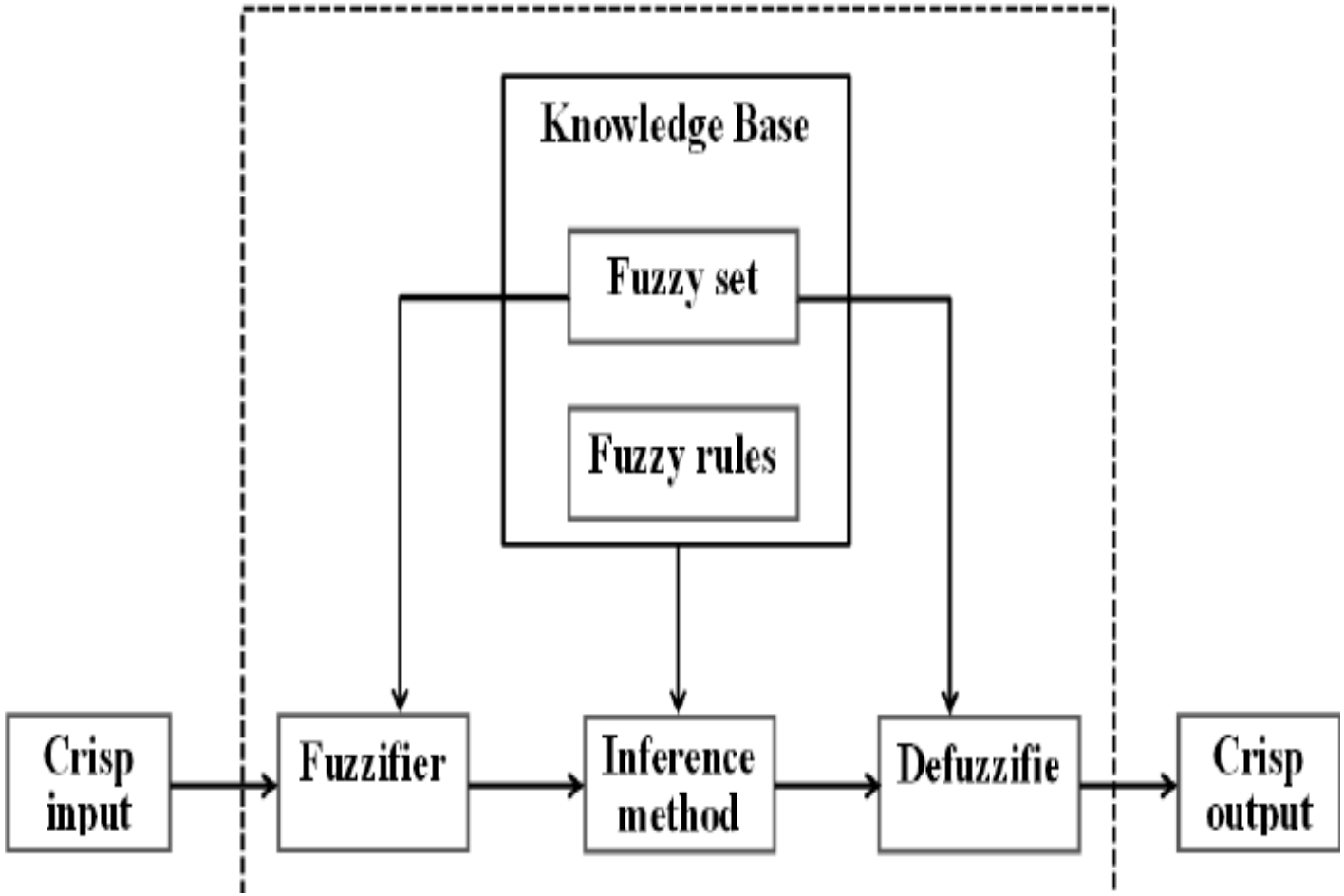
- Transitive Property

$$(A \subseteqq B \subseteqq C) \dashrightarrow (A \subseteqq C) \tag{10}$$

- Involution Property
- De Morgan's Law

Fuzzy Systems

- Fuzzy systems define the clusters and rules by associating all the inputs with all the outputs. Therefore the operation of fuzzy systems is similar to the operation of a mathematical cause-effect function.
- One of the most important concepts in fuzzy systems is fuzzy rules. Fuzzy rules contain fuzzy control rules designed to achieve the control objective. The main purpose of this stage is to express expert knowledge in cause-effect relationship. A fuzzy inference system basically consists of 4 stages:Fuzzification, Inference, Aggregation, Defuzzification

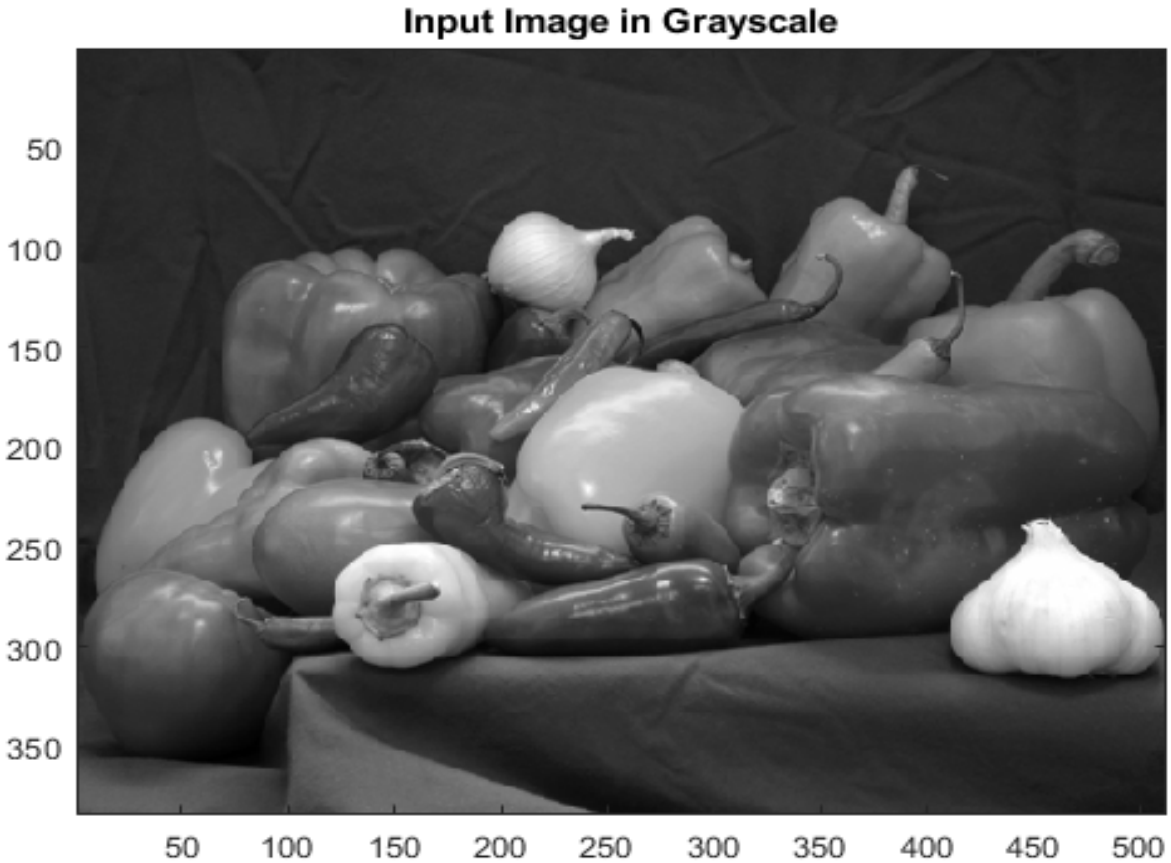

Knowledge Base
Fuzzy set
Fuzzy rules
Crisp input
Fuzzifier
Inference method
Defuzzifie
Crisp output

- Fuzzy Rule Base: Rule base that ontains a number of fuzzy rules.
- Fuzzification: A process to convert the crisp input to a linguistic variable using the membership functions stored in the fuzzy.The purpose of fuzzification is to map the inputs from a set of sensors (or features of those sensors) to values from 0 to 1 using a set of input membership functions.
- Inference Method: Using the fuzzy rules converts the fuzzy input to the fuzzy output.Inputs are applied to a set of if/then control rules.
- Defuzzification: A process to convert the fuzzy output of the inference engine to crisp using membership functions analogous to the ones used by the fuzzifier.Fuzzy outputs are combined into discrete values needed to drive the control mechanism.

To compute the output of this FIS (Fuzzy Interference System) given the inputs, one must go through six steps:

- 1. determining a set of fuzzy rules
- 2. fuzzifying the inputs using the input membership functions,
- 3. combining the fuzzified inputs according to the fuzzy rules to establish a rule strength,
- 4. finding the consequence of the rule by combining the rule strength and the output membership function (if it's a mamdani FIS),
- 5. combining the consequences to get an output distribution, and
- 6. defuzzifying the output distribution (this step applies only if a crisp output (class) is needed).

Fuzzy Logic Image Processing



Fuzzy Logic Image Processing

- This example shows how to use fuzzy logic for image processing. Specifically, this example shows how to detect edges in an image.
- An edge is a boundary between two uniform regions. You can detect an edge by comparing the intensity of neighboring pixels. However, because uniform regions are not crisply defined, small intensity differences between two neighboring pixels do not always represent an edge. Instead, the intensity difference might represent a shading effect.
- The fuzzy logic approach for image processing allows you to use membership functions to define the degree to which a pixel belongs to an edge or a uniform region.

Import RGB Image and Convert to Grayscale.Convert Irgb to grayscale so that you can work with a 2-D array instead of a 3-D array.

- Irgb = imread('peppers.png');
- Igray = rgb2gray(Irgb);
- figure
- image(Igray,'CDataMapping','scaled')
- colormap('gray')
- title('Input Image in Grayscale')

Convert Image to Double-Precision Data. The evalfis function for evaluating fuzzy inference systems supports only single-precision and double-precision data. Therefore, convert Igray to a double array using the im2double function.

- I = im2double(Igray);

Obtain Image Gradient. The fuzzy logic edge-detection algorithm for this example relies on the image gradient to locate breaks in uniform regions. Calculate the image gradient along the x-axis and y-axis. Gx and Gy are simple gradient filters. To obtain a matrix containing the x-axis gradients of I, you convolve I with Gx using the conv2 function. The gradient values are in the [-1 1] range. Similarly, to obtain the y-axis gradients of I, convolve I with Gy.

- Gx = [-1 1];
- Gy = Gx';
- Ix = conv2(I,Gx,'same');
- Iy = conv2(I,Gy,'same');

Plot the image gradients.

- figure
- image(Ix,'CDataMapping','scaled')
- colormap('gray')
- title('Ix')
- figure
- image(Iy,'CDataMapping','scaled')
- colormap('gray')
- title('Iy')

Define Fuzzy Inference System (FIS) for Edge Detection.Create a fuzzy inference system (FIS) for edge detection, edgeFIS.

- edgeFIS = mamfis('Name','edgeDetection');

Specify the image gradients, Ix and Iy, as the inputs of edgeFIS

- edgeFIS = addInput(edgeFIS,[-1 1],'Name','Ix');
- edgeFIS = addInput(edgeFIS,[-1 1],'Name','Iy');

Specify a zero-mean Gaussian membership function for each input. If the gradient value for a pixel is 0, then it belongs to the zero membership function with a degree of 1.

- sx = 0.1;
- sy = 0.1;
- edgeFIS = addMF(edgeFIS,'Ix','gaussmf',[sx 0],'Name','zero');
- edgeFIS = addMF(edgeFIS,'Iy','gaussmf',[sy 0],'Name','zero');

sx and sy specify the standard deviation for the zero membership function for the Ix and Iy inputs. To adjust the edge detector performance, you can change the values of sx and sy. Increasing the values makes the algorithm less sensitive to the edges in the image and decreases the intensity of the detected edges.Specify the intensity of the edge-detected image as an output of edgeFIS.

- edgeFIS = addOutput(edgeFIS,[0 1],'Name','Iout');

Specify the triangular membership functions, white and black, for Iout

- wa = 0.1;
- wb = 1;
- wc = 1;
- ba = 0;
- bb = 0;
- bc = 0.7;
- edgeFIS = addMF(edgeFIS,'Iout','trimf',[wa wb wc],'Name','white');
- edgeFIS = addMF(edgeFIS,'Iout','trimf',[ba bb bc],'Name','black');

Plot the membership functions of the inputs and outputs of edgeFIS.

- figure
- subplot(2,2,1)
- plotmf(edgeFIS,'input',1)
- title('Ix')
- subplot(2,2,2)
- plotmf(edgeFIS,'input',2)
- title('Iy')
- subplot(2,2,[3 4])
- plotmf(edgeFIS,'output',1)
- title('Iout')

Specify FIS Rules Add rules to make a pixel white if it belongs to a uniform region and black otherwise. A pixel is in a uniform region when the image gradient is zero in both directions. If either direction has a nonzero gradient, then the pixel is on an edge.

- r1 = "If Ix is zero and Iy is zero then Iout is white";
- r2 = "If Ix is not zero or Iy is not zero then Iout is black";
- edgeFIS = addRule(edgeFIS,[r1 r2]);
- edgeFIS.Rules

Evaluate FIS. Evaluate the output of the edge detector for each row of pixels in I using corresponding rows of Ix and Iy as inputs.

- Ieval = zeros(size(I));
- for ii = 1:size(I,1)
- Ieval(ii,:) = evalfis(edgeFIS,[(Ix(ii,:));(Iy(ii,:))]');
- end

Plot Results. Plot the original grayscale image.

- figure
- image(I,'CDataMapping','scaled')
- colormap('gray')
- title('Original Grayscale Image')

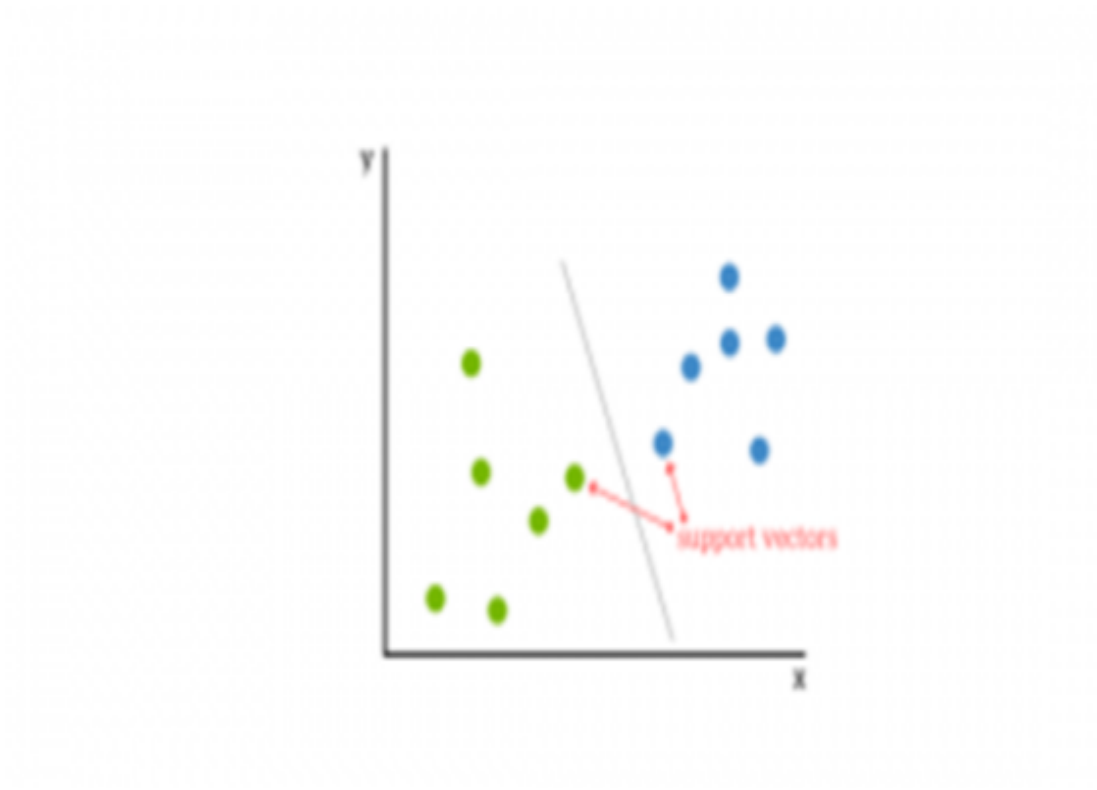
Plot the detected edges.

- figure
- image(Ieval,'CDataMapping','scaled')
- colormap('gray')
- title('Edge Detection Using Fuzzy Logic')

Support Vector Machines

- In machine learning, support-vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.SVM becomes popular because of its success in handwritten digit recognition .
- We could classify new emails into spam or non-spam, based on a large corpus of documents that have already been marked as spam or non-spam by humans. SVMs are highly applicable to such situations.

- SVMs are based on the idea of finding a hyperplane that best divides a dataset into two classes.



Support Vectors

- Support Vectors are simply the co-ordinates of individual observation. Support vectors are the data points that lie closest to the decision surface (or hyperplane) .They are the data points most difficult to classify.They have direct bearing on the optimum location of the decision surface(hyperline).
- Support vectors are the data points nearest to the hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane. Because of this, they can be considered the critical elements of a data set.

What is a hyperplane?

- Hyperplane is a line that linearly separates and classifies a set of data.
- Intuitively, the further from the hyperplane our data points lie, the more confident we are that they have been correctly classified.We therefore want our data points to be as far away from the hyperplane as possible, while still being on the correct side of it.

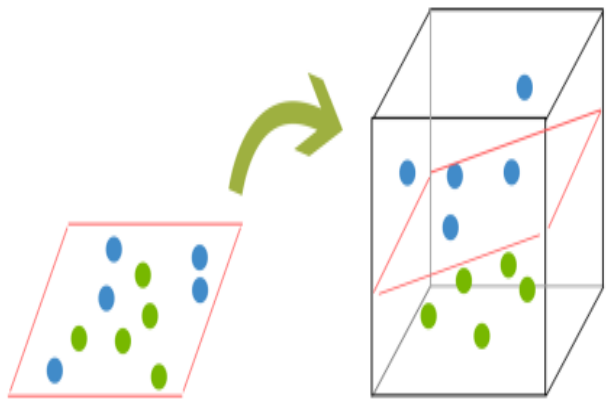
How do we find the right hyperplane?

How do we best segregate the two classes within the data?

- The distance between the hyperplane and the nearest data point from either set is known as the margin. The goal is to choose a hyperplane with the greatest possible margin between the hyperplane and any point within the training set, giving a greater chance of new data being classified correctly.

What happens when there is no clear hyperplane?

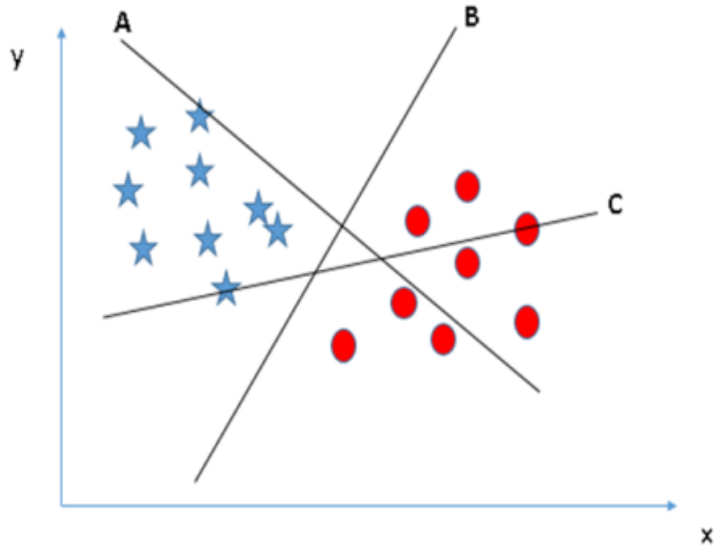
In order to classify a dataset like the one above it's necessary to move away from a 2d view of the data to a 3d view. Explaining this is easiest with another simplified example. Imagine that our two sets of colored balls above are sitting on a sheet and this sheet is lifted suddenly, launching the balls into the air. While the balls are up in the air, you use the sheet to separate them. This 'lifting' of the balls represents the mapping of data into a higher dimension. This is known as kernelling.



Because we are now in three dimensions, our hyperplane can no longer be a line. It must now be a plane as shown in the example above. The idea is that the data will continue to be mapped into higher and higher dimensions until a hyperplane can be formed to segregate it.

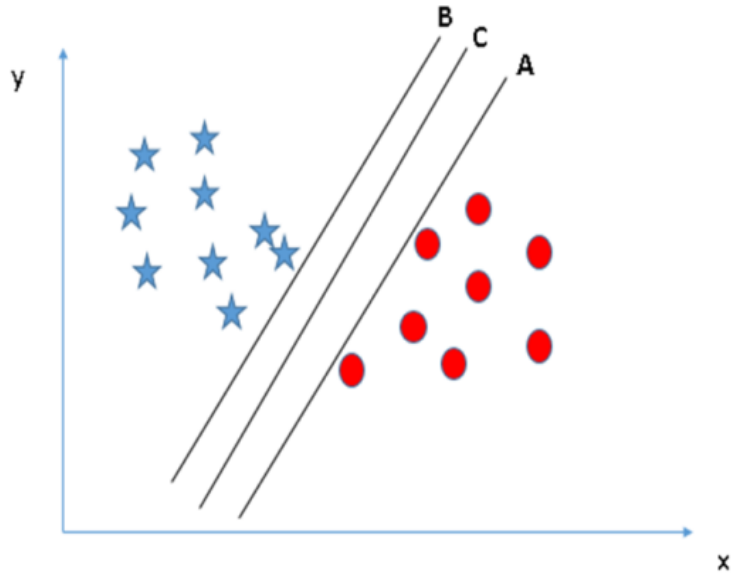
Support Vectors

- You need to remember a thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better”.



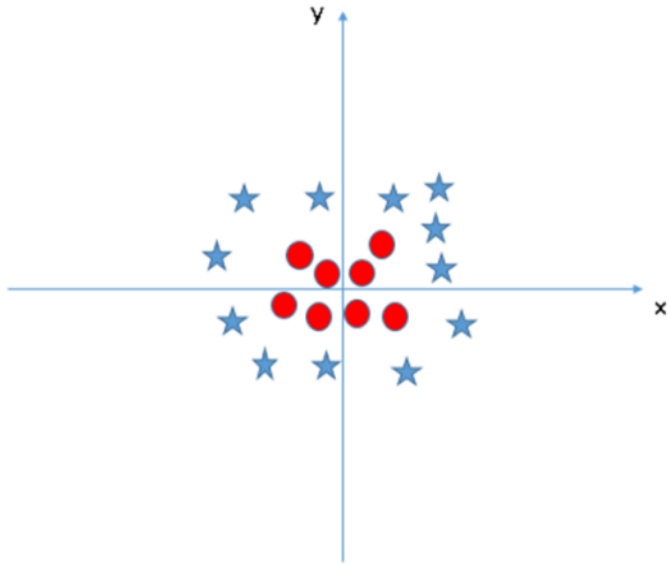
Hyperplane “B” has excellently performed

Frame Title



Maximizing the distances between nearest data point (either class) and hyperplane will help us to decide the right hyperplane. The margin for hyperplane C is high as compared to both A and B. Hence, we name the right hyperplane as C.

We are unable to segregate the two classes using a straight line, as one of star lies in the territory of other (circle) class as an outlier. One star at other end is like an outlier for star class. SVM has a feature to ignore outliers and find the hyperplane that has maximum margin. Hence, we can say, SVM is robust to outliers.

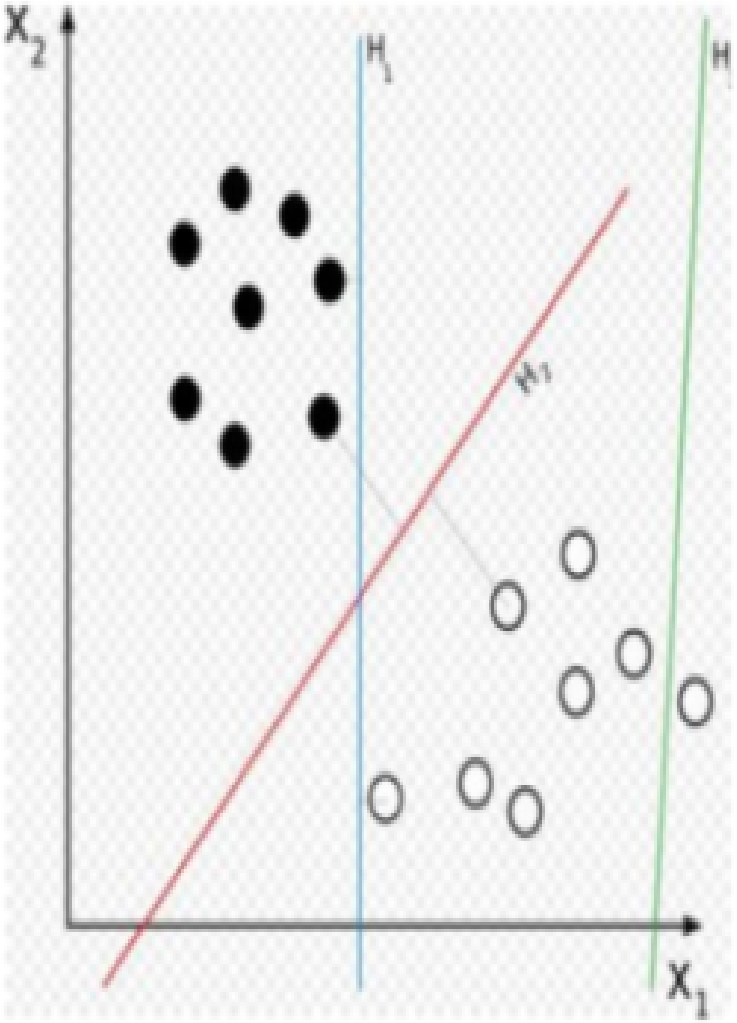


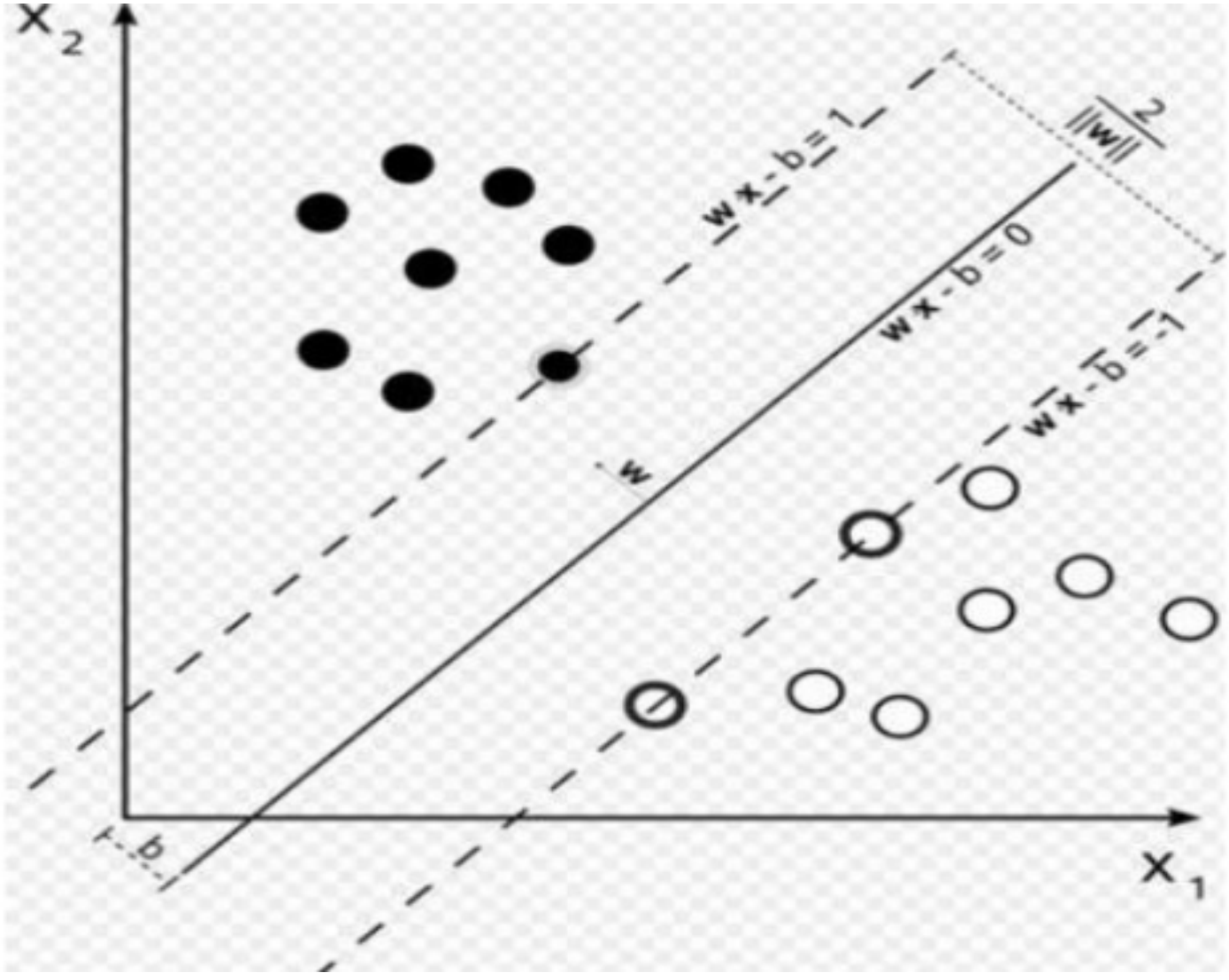
SVM can solve this problem. It solves this problem by introducing additional feature. Here, we will add a new feature $z = x^2 + y^2$

SVM Kernel Functions

- SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be different types. For example linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.
- Introduce Kernel functions for sequence data, graphs, text, images, as well as vectors. The most used type of kernel function is RBF.Because it has localized and finite response along the entire x-axis.
- The kernel functions return the inner product between two points in a suitable feature space. Thus by defining a notion of similarity, with little computational cost even in very high-dimensional spaces.

Build a simple SVM using Matlab




X2
wx-b=1
2/||w||
wx-b=0
wx-b=-1
w
-b
X1

Load the sample data

- load dataname

Create data, a two-column matrix containing sepal length and sepal width measurements for 150 irises.

- data = [meas(:,1), meas(:,2)];

From the species vector, create a new column vector, groups, to classify data into two groups: data and non-data

- groups = ismember(dataset,'data');

Randomly select training and test sets.

- (train,test)= crossvalind('holdOut',groups);
- cp = classperf(groups);

Train an SVM classifier using a linear kernel function and plot the grouped data.

- svmStruct = svmtrain(data(train,:),groups(train),'showplot',true);

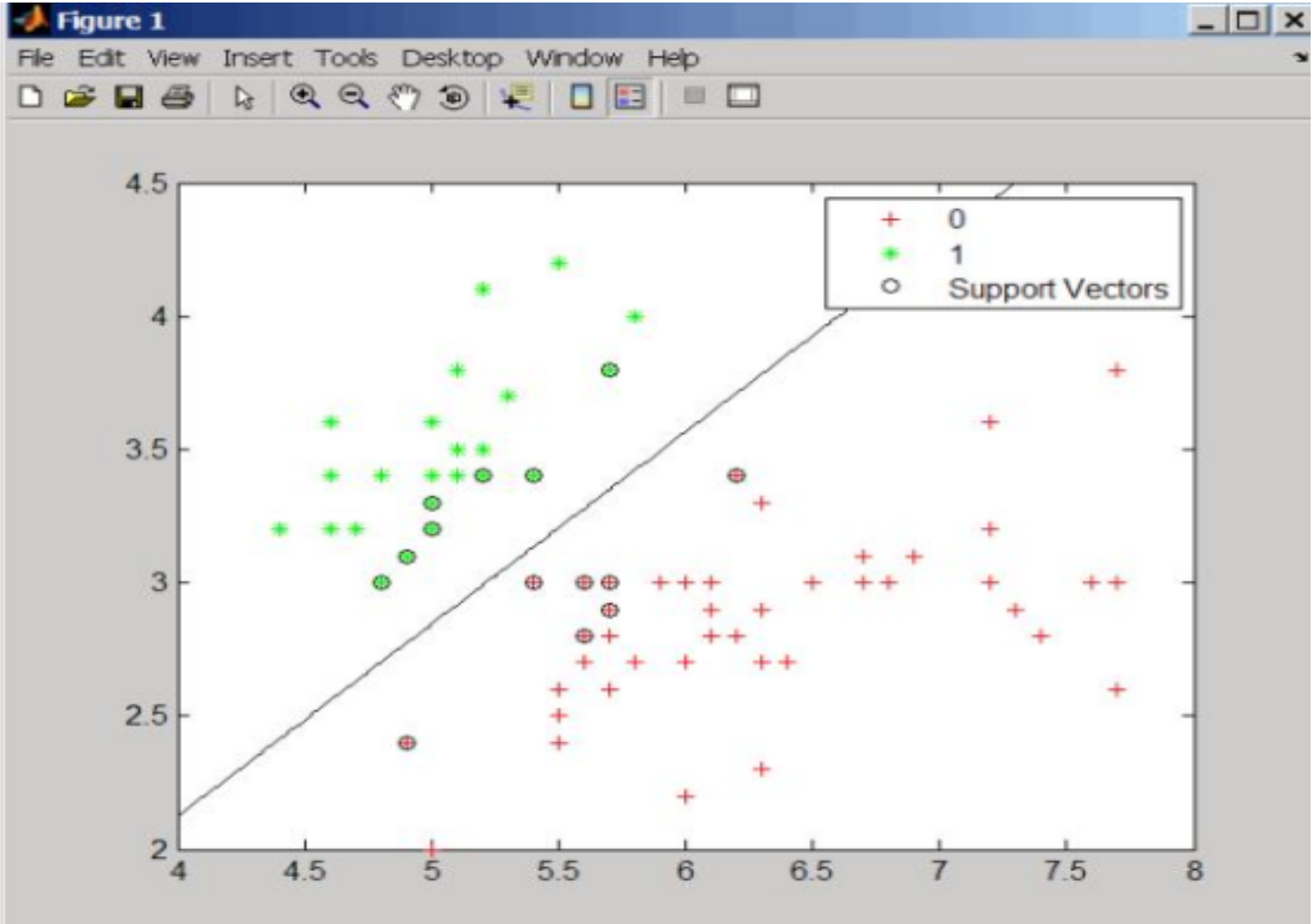


Figure 1
File Edit View Insert Tools Desktop Window Help
0
1
Support Vectors
4.5
4
3.5
3
2.5
2
4
4.5
5
5.5
6
6.5
7
7.5
8

Add a title to the plot, using the KernelFunction field from the svmStruct structure as the title.

- title(sprintf('Kernel Function: func2str(svmStruct.KernelFunction)),...
  'interpreter','none');

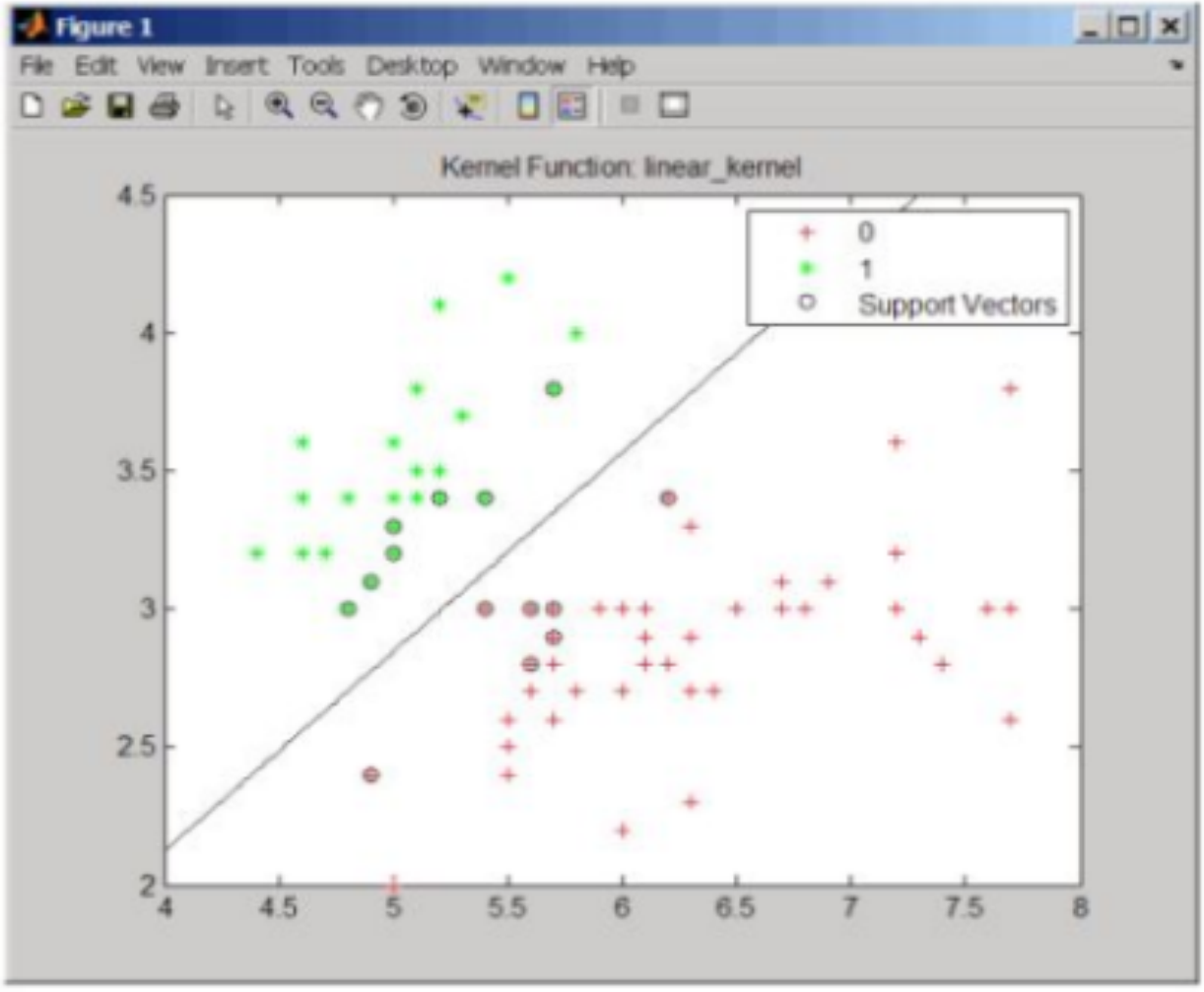


Figure 1
File Edit View Insert Tools Desktop Window Help
Kernel Function: linear_kernel
0
1
Support Vectors
4.5
4
3.5
3
2.5
2
4
4.5
5
5.5
6
6.5
7
7.5
8

Use the svmclassify function to classify the test set.

- classes = svmclassify(svmStruct,data(test,:),'showplot',true);

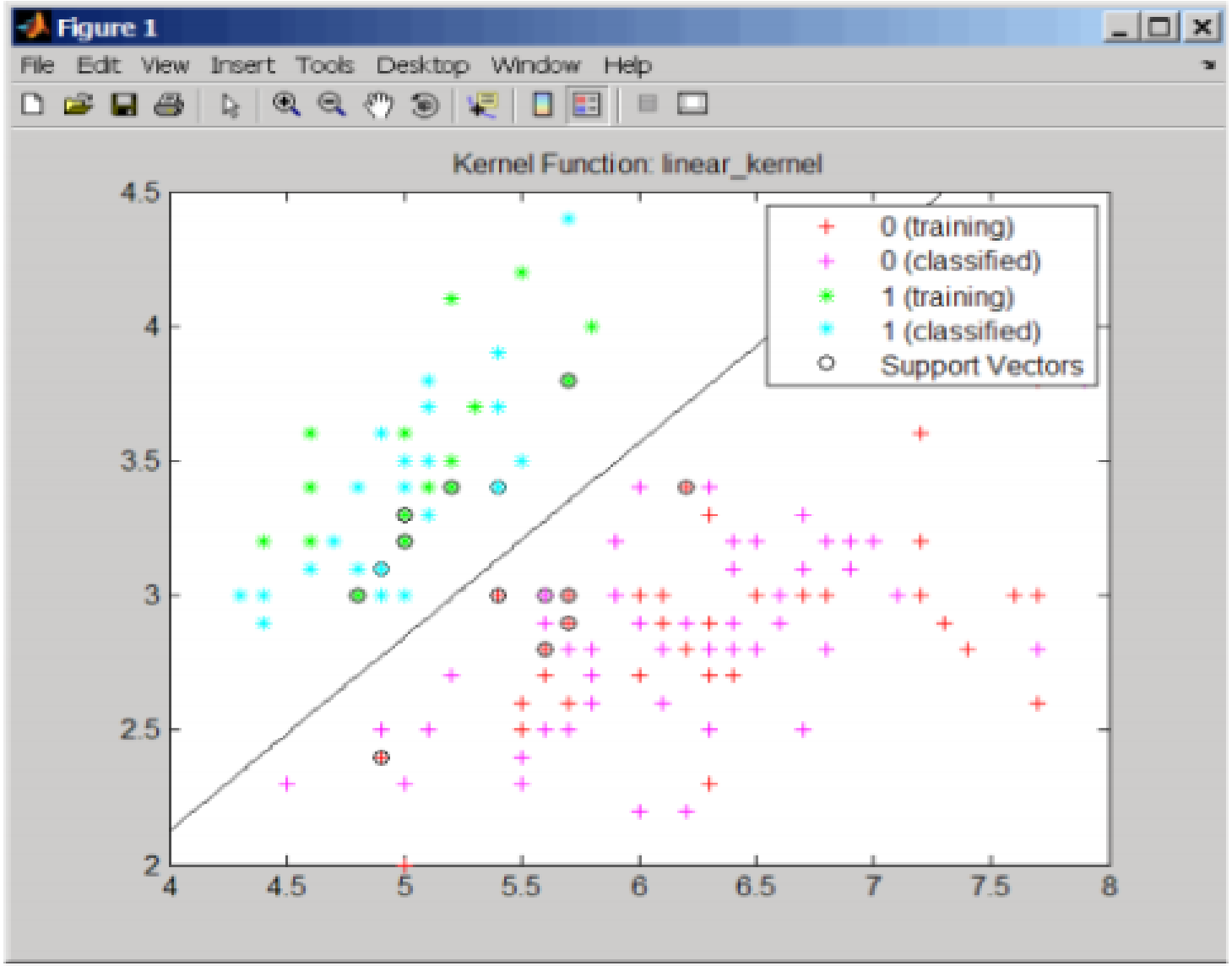


Figure 1
File Edit View Insert Tools Desktop Window Help
Kernel Function: linear_kernel
0 (training)
0 (classified)
1 (training)
1 (classified)
Support Vectors
4.5
4
3.5
3
2.5
2
4
4.5
5
5.5
6
6.5
7
7.5
8

Evaluate the performance of the classifier.

- classperf(cp,classes,test);cp.CorrectRate
- ans = 0.9867

Hybrid Systems

Fuzzy Neural Networks

- While neural networks are low-level computational structures that perform well when dealing with raw data, fuzzy logic deals with reasoning on a higher level, using linguistic information acquired from domain experts.
- However, fuzzy systems lack the ability to learn and cannot adjust themselves to a new environment. On the other hand, although neural networks can learn, they are opaque to the user.

Fuzzy Neural Networks

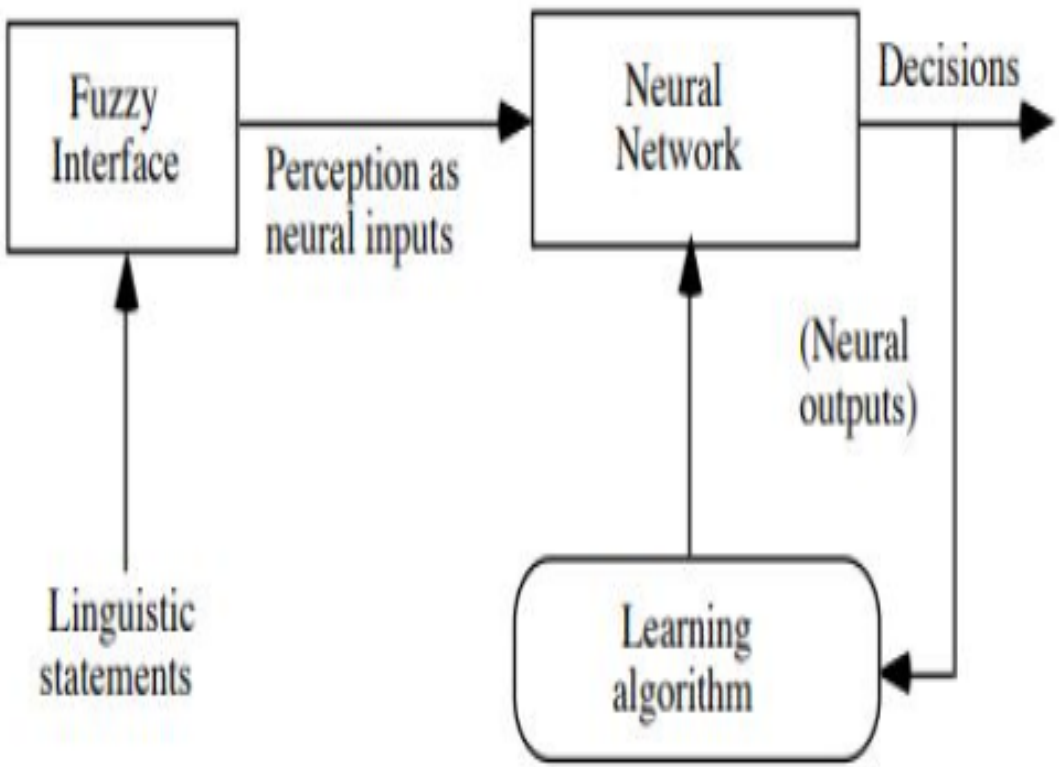
- A neuro-fuzzy system is a neural network which is functionally equivalent to a fuzzy inference model. It can be trained to develop IF-THEN fuzzy rules and determine membership functions for input and output variables of the system.
- While fuzzy logic provides an inference mechanism under cognitive uncertainty, computational neural networks offer exciting advantages, such as learning, adaptation, fault-tolerance, parallelism and generalization. To enable a system to deal with cognitive uncertainties in a manner more like humans, one may incorporate the concept of fuzzy logic into the neural networks.

The computational process envisioned for fuzzy neural systems is as follows:

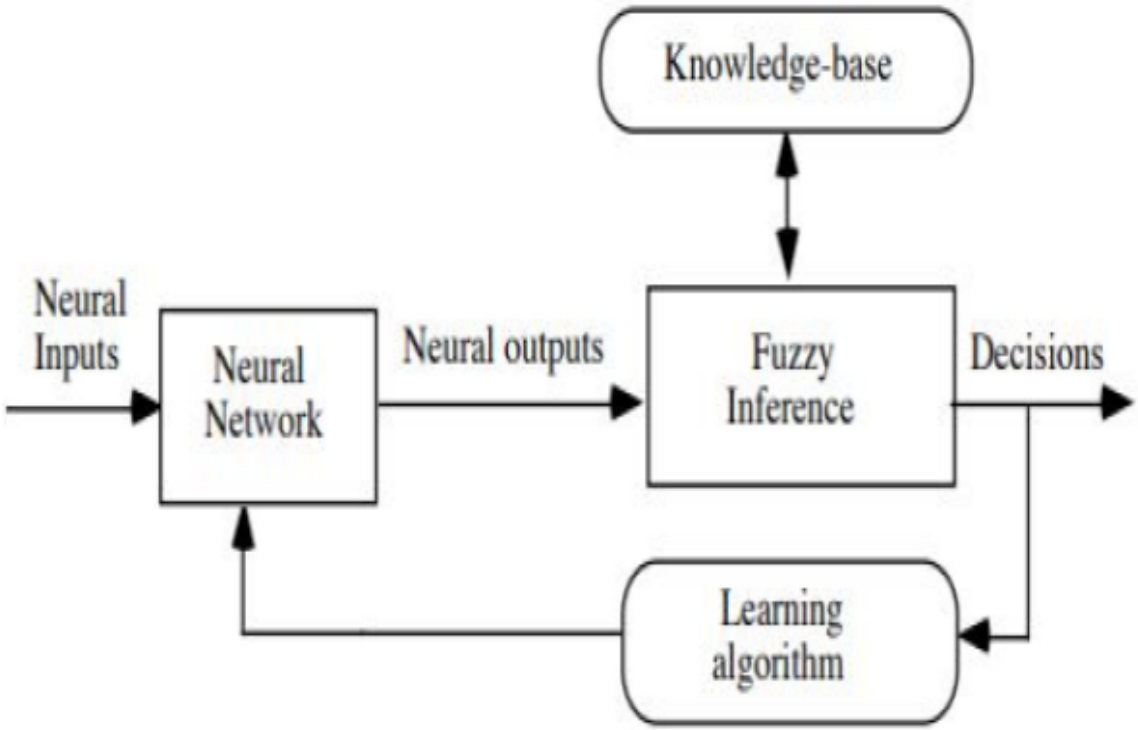
- It starts with the development of a "fuzzy neuron" based on the understanding of biological neuronal morphologies, followed by learning mechanisms. This leads to the following three steps in a fuzzy neural computational process.
- development of fuzzy neural models motivated by biological neurons,
- models of synaptic connections which incorporates fuzziness into neural network
- development of learning algorithms (that is the method of adjusting the synaptic weights)

Two possible models of fuzzy neural systems

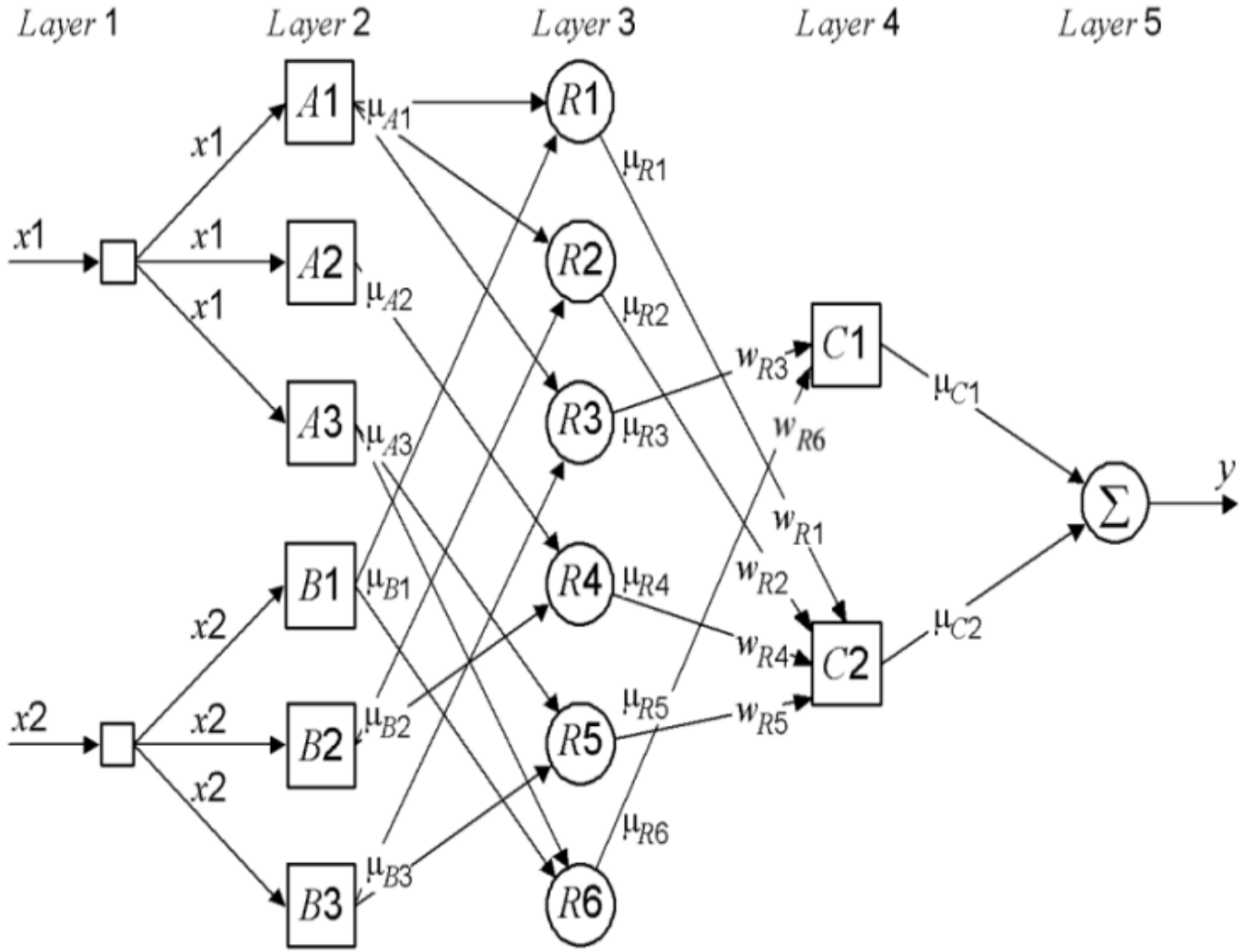
In response to linguistic statements, the fuzzy interface block provides an input vector to a multi-layer neural network. The neural network can be adapted (trained) to yield desired command outputs or decisions.



A multi-layered neural network drives the fuzzy inference mechanism.



The structure of a neuro-fuzzy system is similar to a multi-layer neural network. In general, a neuro-fuzzy system has input and output layers, and three hidden layers that represent membership functions and fuzzy rules.



- Neural networks are used to tune membership functions of fuzzy systems that are employed as decision-making systems for controlling equipment.
- Although fuzzy logic can encode expert knowledge directly using rules with linguistic labels, it usually takes a lot of time to design and tune the membership functions which quantitatively define these linquistic labels.
- Neural network learning techniques can automate this process and substantially reduce development time and cost while improving performance.
- Based upon the computational process involved in a fuzzy-neuro system, one may broadly classify the fuzzy neural structure as feedforward (static) and feedback (dynamic).

A typical fuzzy-neuro system is Berenji's ARIC (Approximate Reasoning Based Intelligent Control) architecture. It is a neural network model of a fuzy controller and learns by updating its prediction of the physical system's behavior and fine tunes a predefined control knowledge base.

The End