P.O. Box 342-01000 Thika
Email: info@mku.ac.ke
Web: www.mku.ac.ke

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: BIT 4106

COURSE TITLE : FUZZY PATTERNS RECOGNITION AND NEURAL NETWORKS

Instructional Manual for BBIT – Distance Learning

Prepared by Mr. Paul Mutinda

E-mail: *mutindaz@yahoo.com*

Contents

PART I

1.1 Introduction to Fuzzy Logic

Fuzzy logic studies reasoning systems in which the notions of truth and falsehood are considered in a graded fashion, in contrast with classical mathematics where only absolutely true statements are considered

1:1:1 Fuzzy logic in broad sense

Fuzzy logic in broad sense serves mainly as apparatus for fuzzy control, analysis of vagueness in natural language and several other application domains.

It is one of the techniques of soft-computing, i.e. computational methods tolerant to sub optimality and impreciseness (vagueness) and giving quick, simple and sufficiently good solutions.

1:1:2 Fuzzy logic in the narrow sense

Fuzzy logic in the narrow sense is symbolic logic with a comparative notion of truth developed fully in the spirit of classical logic (syntax, semantics, axiomatization, truth-preserving deduction, completeness, etc.; both propositional and predicate logic).

It is a branch of many-valued logic based on the paradigm of inference under vagueness.

1.2 Fuzzy sets and crisp sets

In classical mathematics one deals with collections of objects called (crisp) sets.

Sometimes it is convenient to fix some universe U in which every set is assumed to be included.

It is also useful to think of a set A as a function from U which takes value 1 on objects which belong to A and 0 on all the rest. Such functions is called the characteristic function of A, XA:

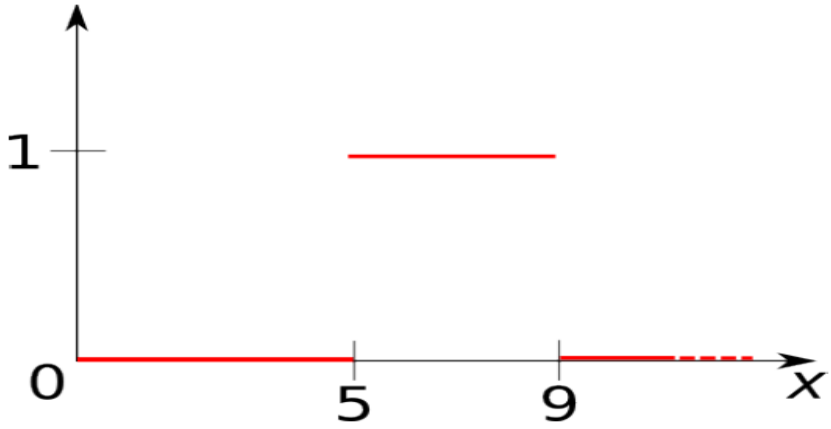
$$\chi_A(x) =_{def} \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

So there exists a objective correspondence between characteristic functions and sets.

1.2.1 Crisp sets

Example

Let X be the set of all real numbers between 0 and 10 and let A = [5; 9] be the subset of X of real numbers between 5 and 9. This results in the following figure:

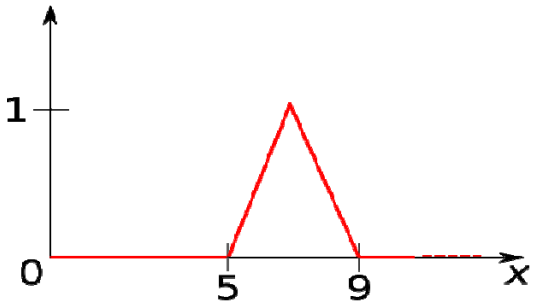


1.2.2 Fuzzy sets

Fuzzy sets generalize this definition, allowing elements to belong to a given set with a certain degree.
Instead of considering characteristic functions with value in f0; 1g we consider now functions valued in [0; 1].
A fuzzy subset F of a set X is a function _F (x) assigning to every element x of X the degree of membership of x to F:

$$x \in X \mapsto \mu_F(x) \in [0,1].$$

Let, as above, X be the set of real numbers between 1 and 10. A description of the fuzzy set of real numbers **close** to 7 could be given by the following figure:



1:3 Operations between sets

In classical set theory there are some basic operations defined over sets. Let X be a set and $P(X)$ be the set of all subsets of X or, equivalently, the set of all functions between X and $\{0,1\}$. The operation of **union**, **intersection** and **complement** are defined in the following ways:

$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$ i.e. $\chi_{A \cup B}(x) = \max\{\chi_A(x), \chi_B(x)\}$

$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$ i.e. $\chi_{A \cap B}(x) = \min\{\chi_A(x), \chi_B(x)\}$

$A = \{x \mid x \in A\}$ i.e. $\chi_A(x) = 1 -$

$\chi_A(x)$

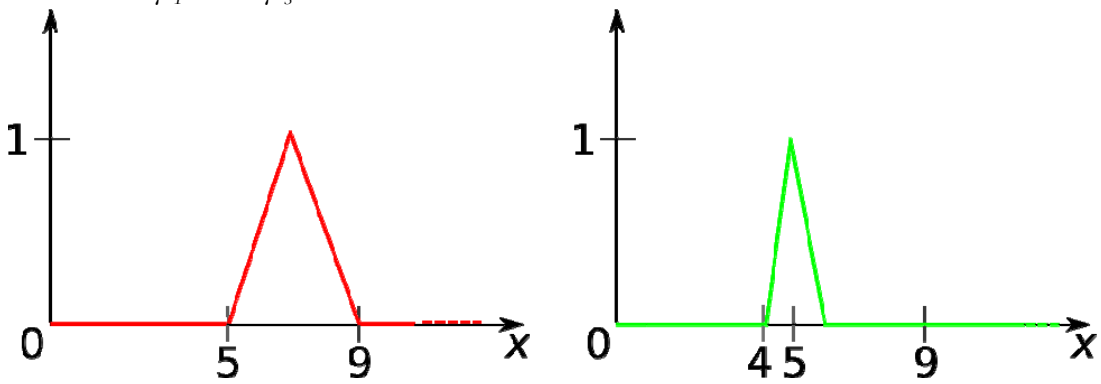
1:3:1 Operations between fuzzy sets: union

The law

$$\chi_{A \cup B}(x) = \max\{\chi_A(x), \chi_B(x)\}.$$

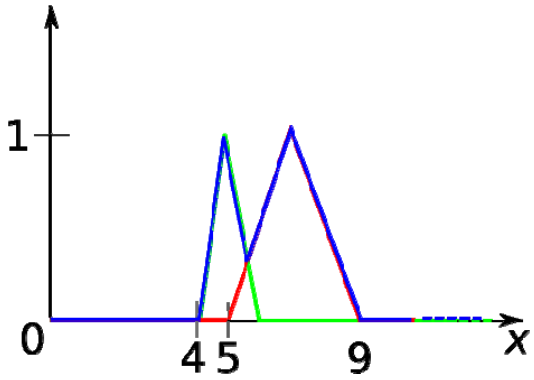
gives us an obvious way to generalise union to fuzzy sets.

Let F and S be fuzzy subsets of X given by membership functions μ_F and μ_S:



We set

$$\mu_{F \cup S}(x) = \max\{\mu_F(x), \mu_S(x)\}$$



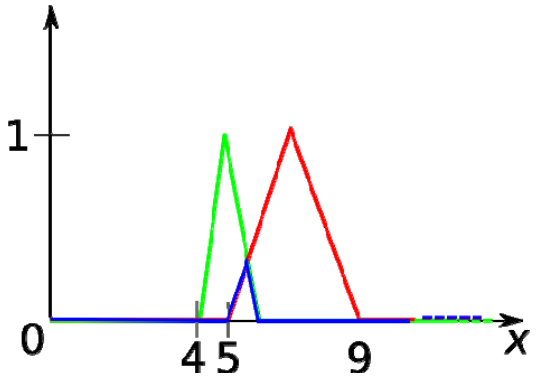
1:3:2 Operations between fuzzy sets: intersection

Analogously for intersection:

$$\chi_{A \cap B}(x) = \min\{\chi_A(x), \chi_B(x)\}.$$

We set

$$\mu_{F\cap S}(x) = \min\{\mu_F(x), \mu_S(x)\}$$



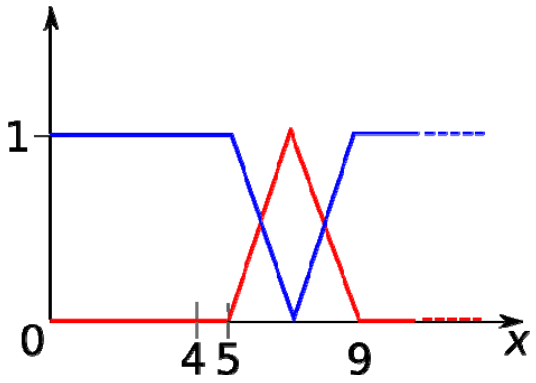
1:3:3 Operations between fuzzy sets: complement

Finally the complement for characteristic functions is defined by,

$$\chi_{A'}(x) = 1 - \chi_A(x).$$

We set

$$\mu_{F'}(x) = 1 - \mu_F(x).$$



1:3:4 Operations between fuzzy sets 2

Let's go back for a while to operations between sets and focus on intersection. We defined operations between sets inspired by the operations on characteristic functions. Since characteristic functions take values over $\{0, 1\}$ we had to choose an **extension** to the full set $[0, 1]$.

It should be noted, though, that also the product would do the job, since on $\{0, 1\}$ they coincide:

$$\chi_{A\cap B}(x) = \min\{\chi_A(x), \chi_B(x)\} = \chi_A(x) \cdot \chi_B(x).$$

So our choice for the interpretation of the intersection between fuzzy sets was a little illegitimate.

Further we have

$$\chi_{A\cap B}(x) = \min\{\chi_A(x), \chi_B(x)\} = \max\{0, \chi_A(x) + \chi_B(x) - 1\}$$

It turns out that there is an infinity of functions which have the same values as the minimum on the set $\{0, 1\}$. This leads to isolate some basic property that the our functions must enjoy in order to be good candidate to interpret the intersection between fuzzy sets.

1.4 t-norms

In order to single out these properties we look again back at the crisp case: It is quite reasonable for instance to require the fuzzy intersection to be **commutative**, i.e.

$$\mu_F(x) \cap \mu_S(x) = \mu_S(x) \cap \mu_F(x),$$

or **associative**:

$$\mu_F(x) \cap [\mu_S(x) \cap \mu_T(x)] = [\mu_F(x) \cap \mu_S(x)] \cap \mu_T(x).$$

Finally it is natural to ask that if we take a set μ_F *bigger* than μ_S than the intersection $\mu_F \cap \mu_T$ should be bigger or equal than $\mu_S \cap \mu_T$:

If for all $x \in X \mu_F(x) \geq \mu_S(x)$ then $\mu_F(x) \cap \mu_T(x) \geq \mu_S(x) \cap \mu_T(x)$

Summing up the few basic requirements that we make on a function $\odot$ that candidates to interpret intersection are:To extend the $\{0, 1\}$ case, i.e. for all $x \in [0, 1]$.

$$1 \odot x = x \text{ and } 0 \odot x = 0$$

Commutativity, i.e., for all $x, y, z \in [0, 1]$,

$$x \odot y = y \odot x$$

Associativity, i.e., for all $x, y, z \in [0, 1]$,

$$(x \ast y) \ast z = x \ast (y \ast z),$$

To be non-decreasing, i.e., for all $x_1, x_2, y \in [0, 1]$,

$$x_1 \leq x_2 \text{ implies } x_1 \ast y \leq x_2 \ast y.$$

Objects with such properties are already known in mathematics and are called **t-norms**.

Example

(i) Lukasiewicz t-norm: $x \quad y = \max(0, x + y - 1)$.

(ii) Product t-norm: $x \cdot y$ usual product between real numbers.

(iii) Gödel t-norm: $x \wedge y = \min(x, y)$.

if $(x, y) \in [0, 1[^2$

otherwise.

(v) The family of Frank t-norms is given by:

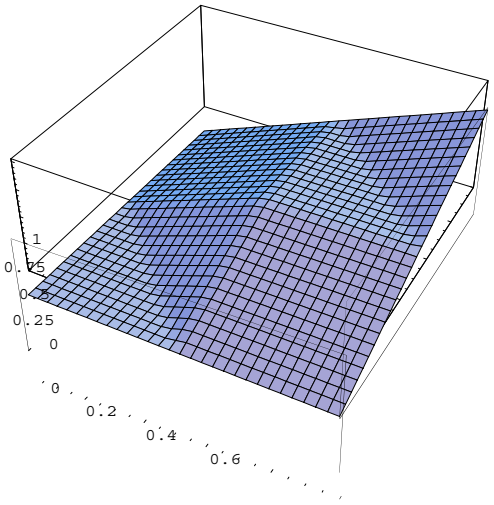
if $\lambda = 0$

if $\lambda = 1$

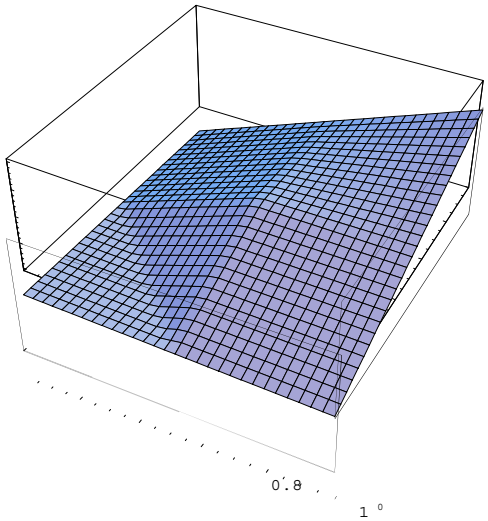
if $\lambda = \infty$

otherwise

Examples



Two copies of Lukasiewicz



Lukasiewicz plus Product

Summary

We have seen that it is possible to generalise the classic crisp sets to objects which naturally admits a notion of **graded membership**. Also the **fundamental operations** between sets can be generalised to act on those new objects....but there is not **just one** of such generalisations.

A few natural requirements drove us to isolate the concept of **t-norm** as a good candidate for intersection. There is a plenty of t-norms to choose from, but all of them can be reduced to a combination of **three** basic t-norms. **next aim**: we have fuzzy properties and we can combine them, let us try to reason about them.

Part II

Mathematical logic

2:1 What is a logic?

In mathematics a **logic** is a formal system which describes somen set of rules for building new objects form existing ones.

Example

Given the two words *ab* and *bc* is it possible to build new ones by substituting any *b* with *ac* or by substituting any *c* with *a*. So the words *aac, aaa, acc, aca,..* are **deducible** from the two given ones. The rules of chess allow to build new configurations of the pieces on the board starting from the initial one. The positions that we occupy in the space are governed by the law of physics.

2:2 Propositional logic

Propositional logic studies the way new sentences are derived from a set of given sentences (usually called **axioms**).

Example

If there is no fuel the car does not start.

There is no fuel in this car.

This car will not start.

If you own a boat you can travel in the see.

If you can travel in the see you can reach Elba island.

If you own a boat you can reach Elba island.

2:3 SYNTAX

Proportional Logic

Definition

The *objects* in propositional logic are sentences, built from a alphabet.

The language of propositional logic is given by:

A set V of **propositional variables** (the alphabet):

$\{X_1, \ldots, X_n, \ldots\}$

Connectives: $\vee, \wedge, \neg, \rightarrow$ (conjunction, disjunction, negation and implication).

Parenthesis (and).

2:3:1 Sentences of propositional calculus

Definition

Sentences (or **formulas**) of propositional logic are defined in the following way.

i) Every variable is a formula.

ii) If P and Q are formulas then $(P \vee Q)$, $(P \wedge Q)$, $(\neg P)$, $(P \rightarrow Q)$ are formulas.

iii) All formulas are constructed only using i) and ii).

Parenthesis are used in order to avoid confusion. They can be omitted whenever there is no risk of misunderstandings.

2:3:2 The axioms of propositional logic

The axioms of propositional logic are :

1. $(A \rightarrow (B \rightarrow A))$
2. $((A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)))$
3. $((\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A))$

plus **modus ponens**: if $A \rightarrow B$ is true and A is true, then B is true.

A **deduction** is a sequence of instances of the above axioms and use of the rule modus ponens. The other connectives are

defined as

$$A \vee B =_{def} \neg A \rightarrow B$$

$$A \wedge B =_{def} \neg(\neg A \vee \neg B) = \neg(\neg\neg A \rightarrow \neg B)$$

2:3:3 A deduction in propositional logic

Example

An instance of 1 gives

$$\neg X_1 \rightarrow (X_2 \rightarrow \neg X_1)$$

and an instance of 2 gives

$$\neg X_1 \rightarrow (X_2 \rightarrow \neg X_1) \rightarrow ((\neg X_1 \rightarrow X_2) \rightarrow (\neg X_1 \rightarrow \neg X_1)),$$

the use of modus ponens leads

$$(\neg X_1 \rightarrow X_2) \rightarrow (\neg X_1 \rightarrow \neg X_1)$$

which, by definition, can be written as

$$(\neg X_1 \rightarrow X_2) \rightarrow (X_1 \vee \neg X_1).$$

2:4 The semantics of a calculus

Just as happens in mathematics, where one makes calculations with numbers and those numbers *represent*, e.g. physical quantities, or amount of money, or points in a space, one can associate to a logic one (or several) **interpretation**, called the **semantics** of the logic.

Evaluations

Definition

An **evaluation** of propositional variables is a function

$$v : V \to \{0, 1\}$$

mapping every variable in either the value 0 (False) or 1 (True). In order to extend evaluations to formulas we need to interpret connectives as operations over $\{0, 1\}$. In this way we establish a **homomorphism** between the algebra of formulas (with the operation given by connectives) and the Boolean algebra on $\{0, 1\}$:

$$v : Form \to \{0, 1\}$$

2:4:1 The semantics of connectives

The evaluation v can be extended to a function $\mathbf{v}$ total on $Form$ by using induction:

Variables: $\mathbf{v}(X_1) = v(X_1), \ldots, \mathbf{v}(X_n) = v(X_n)$.

$\mathbf{v}(P \wedge Q) = 1$ if both $\mathbf{v}(P) = 1$ and $\mathbf{v}(Q) = 1$.

$\mathbf{v}(P \wedge Q) = 0$ otherwise.

$\mathbf{v}(P \vee Q) = 1$ if either $\mathbf{v}(P) = 1$ or $\mathbf{v}(Q) = 1$.

$\mathbf{v}(P \vee Q) = 0$ otherwise.

$\mathbf{v}(P \to Q) = 0$ if $\mathbf{v}(P) = 1$ and $\mathbf{v}(Q) = 0$.

$\mathbf{v}(P \to Q) = 1$ otherwise.

$\mathbf{v}(\neg P) = 1$ if $\mathbf{v}(P) = 0$, and vice-versa.

A formula is a **tautology** if it only takes values 1. Tautologies are always true, for every valuation of variables.

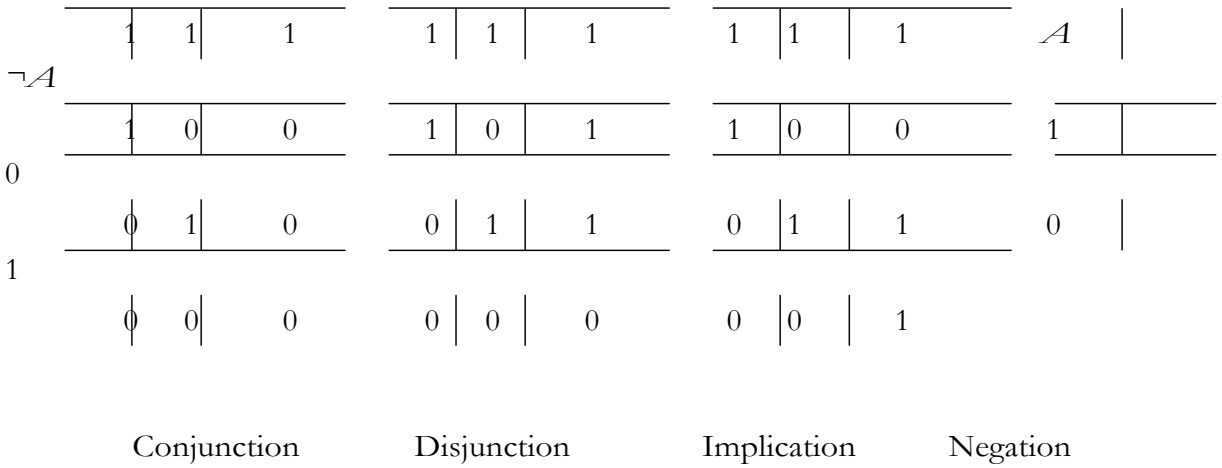
2:4:2 Truth tables

The above rules can be summarized by the following tables:

A	B	$A \wedge B$

A	B	$A \vee B$

A	B	$A \mapsto B$



Using the tables for basic connectives we can write tables for any formula:

Example

Let us consider the formula $X \to (Y \vee \neg X)$:

X	Y	$\neg X$	$Y \vee \neg X$	$X \to (Y \vee \neg X)$
1	1	0	1	1
1	0	0	0	0
0	1	1	1	1
0	0	1	1	1

2:5 Multiple truth values

2:5:1 Adding a third truth values

It is easy now to figure out how to extend the previous logical apparatus with a third truth value, say $1/2$. We keep the same syntactical structure of formulas: we just change the semantics. Evaluations are now functions from the set of variables into $\{0, 1/2, 1\}$.

Accordingly to the definitions of truth tables for connectives we have different three-valued logics.

–

2:5:2 Kleene's logic

Kleene strong three valued logic is defined as

A and B	0	1/2	1
0	0	0	0
1/2	0	1/2	1/2
1	0	1/2	1

Conjunction

A or B	0	1/2	1
0	0	1/2	1
1/2	1/2	1/2	1
1	1	1	1

Disjunction

A implies B	0	1/2	1
0	1	1	1
1/2	1/2	1/2	1
1	0	1/2	1

Implication

A	not A
1	0
1/2	1/2
0	1

Negation

2:5:3 Lukasiewicz three valued logic

Lukasiewicz three valued logic is given by the following stipulation:

A B	0	1/2	1
0	0	0	0
1/2	0	0	1/2
1	0	1/2	1

Conjunction

A ? B	0	1/2	1
0	0	1/2	1
1/2	1/2	1	1
1	1	1	1

Disjunction

$A \rightarrow B$	0	1/2	1
0	1	1	1
1/2	1/2	1	1
1	0	1/2	1

Implication

A	$\neg A$
1	0
1/2	1/2
0	1

Negation

We can also consider more than three values, and also infinitely many values, for example interpreting formulas in the real interval $[0, 1]$.

Here come back the t-norm functions defined earlier. Indeed one can think t-norms as possible semantics for the connective "conjunction".

To rescue an implication from the t-norm, one can ask for desirable properties which relate the two connectives; a very important one is

$$(A \wedge B) \to C \qquad A \to (B \to C).$$

□

2:7 The Other Connectives

2:7:1 Residuum

Proposition

Let $*$ be a continuous t-norm. Then, for every $x, y, z \in [0, 1]$, there is a unique operation satisfying the property:

$$(x * z) \leq y \quad \text{if and ony if} \quad z \leq (x \Rightarrow y)$$

and it is defined by

$$x \Rightarrow y = \max\{z \mid x * z \leq y\}$$

The operation $\Rightarrow$ is called the **residuum** of the t-norm $*$.

Example

The following are residua of the three main continuous t-norms:

	T-norm	Residuum
L	$x \otimes_L y = \max(x + y - 1, 0)$	$x \rightarrow_L y = \min(1, 1 - x + y)$
P	$x \otimes_P y = x \cdot y$	
G	$x \otimes_G y = \min(x, y)$	

2:7:2 Negation

Once we have implication we can also define negation.

Indeed in classical logic a formula that implies a false formula is false itself. Hence

$$\neg A = A \rightarrow 0.$$

In case of Lukasiewicz t-norm, we have

$$\neg x = x \rightarrow 0 = \min(1, 1 - x + 0) = 1 - x$$

For Gödel and Product logic

$$\neg x =$$

2:7:3 The complete picture

Completing the table

	T-norm $x \odot y$	Residuum $x \Rightarrow y$	Negation $\neg x$
L	$\max(x + y - 1, 0)$	$\min(1, 1 - x + y)$	$1 - x$
P	$x \cdot y$	1 if $x \leq y$ y/x otherwise	1 if $x = 0$ 0 ow
G	$\min(x, y)$	1 if $x \leq y$ y otherwise	1 if $x = 0$ 0 ow

So each of these logics is specified only by the t-norm.

–

2:7:4 Implication

We have seen that logical systems can be approached in two different ways:

Specifying the **syntax**, that means fixing axioms and deduction rules

Specifying the **semantics**, that means fixing the interpretation of formulas.

In the first approach the connective of implication plays a very important role, since it is the main ingredient of the basic deduction rule of **Modus ponens**:

If A *and* $A \to B$ *are theorems, then* B *is a theorem.*

Implication

On the other hand, the implication can be defined as an operation between sets by

$$A \to B = \neg A \cup B.$$

This means that if A and B are subsets of X, then $A \to B = X$ if and only if $A \subseteq B$ that is equivalent to say that $\chi_A(x) \leq \chi_B(x)$.

Going to the fuzzy level, implication takes care of **order**

between membership values.

Later, we shall come back to implication in fuzzy logic.

Summary

We have seen that it is possible to formalise **inside** mathematics what a logical system is. Logical systems can be presented **syntactically** by specifying axioms and rules or **semantically** by giving devising the truth tables of the connectives. Just as happens in classical logic, where the concept of intersection corresponds to the connective hand, we have seen that t-norms can be used as **generalised truth tables** for conjunction.
Clearly one can build any logical system whatsoever, but in order to obtain good **deductive properties** it is important to relate in some way the connectives **next aim**: we wish now to push these methods to infinite values and show that syntax and semantic can be reunified back.

PART III

3:1 Introduction to Pattern Recognition

Pattern recognition is the discipline of building machines to perform perceptual tasks which we humans are particularly good at. e.g. recognize faces, voice, identify species of flowers, spot an approaching storm.

There are practical needs to find more efficient ways of doing things. e.g. read hand-written symbols, diagnose diseases, identify incoming missiles from radar or sonar signals. The machines may perform these tasks faster, more accurately and cheaply.

The goal of pattern recognition research is to clarify complicated mechanisms of decision making processes and automatic these function using computers.

3:2 Pattern classification

Although we humans can perform some of the perceptual tasks with ease, there is not sufficient understanding to duplicate the performance with a computer.

Because the complex nature of the problems, many pattern recognition research has been concerned with more moderate problems of

pattern classification — the assignment of a physical object or event to one of several pre- specified categories.

Example: A lumber mill producing assorted hardwoods wants to automate the process of sorting finished lumber according to the species of trees.

Optical sensing is used to distinguish birch lumber from ash lumber. A camera takes a pictures of the lumber and passes to on to a feature extractor.

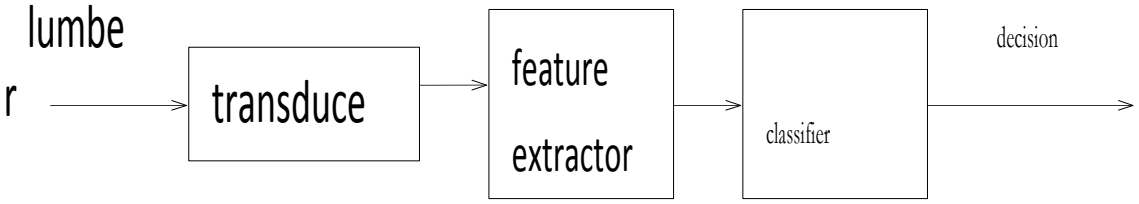


Figure: A pattern classification system

The feature extractor reduces the data by measuring certain "properties" that distinguish pictures of birch lumber from pictures of ash lumber. These features are then

passed to a classifier that evaluates the evidence presented and makes a final decision about the lumber type.

Suppose that somebody at the lumber mill tells us that birth is often lighter colored than ash. Then brightness becomes an obvious feature. We might attempt to classify the lumber merely by seeing whether or not the average brightness x exceeds some critical value.

One characteristic of human pattern recognition is that it involves a teacher. Similarly a machine pattern recognition system needs to be trained. A common mode of learning is to be given a collection of labeled examples, known as training data set. From the training data set, structure information is distilled and used for classifying new inputs.

In this case, we would obtain samples of different types of wood, make brightness measurements, and inspect the results. Suppose that we obtain the following histogram based on these data samples.

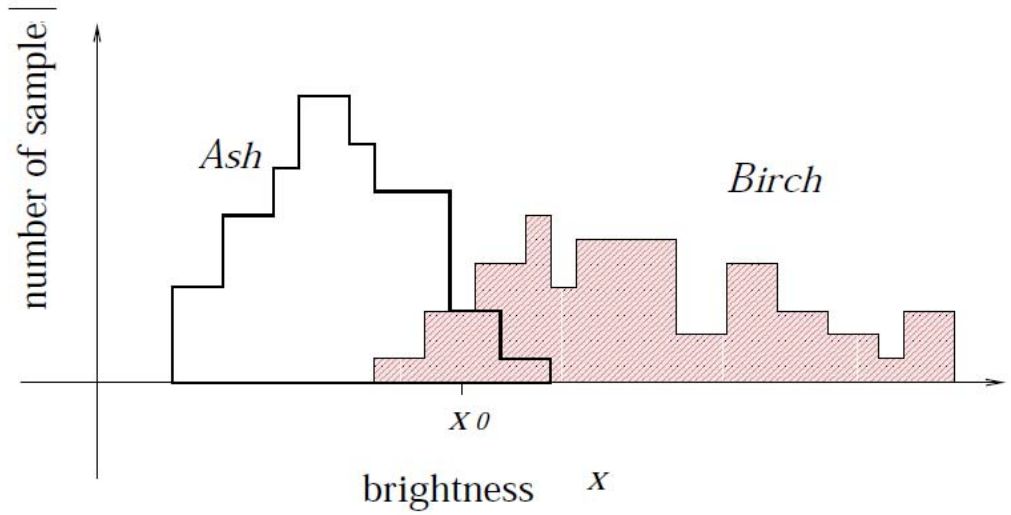


Figure: Histogram for the brightness feature.

The histogram bears out the statement that birch is usually lighter than ash, but

it is clear that this single criterion is not infallible. No matter how we choose X_0, we cannot reliably separate birch from ash by brightness alone.

The second feature is based on the observation that ash typically has a more prominent grain pattern than birch. It is reasonable to assume that we can obtain a measure of this feature from the magnitude and frequency of occurrence of light-to-dark transitions in the picture.

The feature extractor has thus reduced each picture to a point or a feature vector $\mathbf{X}$ in a two dimensional space, where,

$$\mathbf{X} = [X_1, X_2]^T$$

where X_1 denotes the brightness, X_2 denotes the grain prominence.

Our problem now is to partition the feature space into two regions for birth and ash. Suppose that we measure the feature vectors for our training data samples and obtain the following scatter diagram.

This plot suggests the rule for classifying the data: Classify the lumber as ash if its feature vector falls above the line AB, and as birch otherwise.

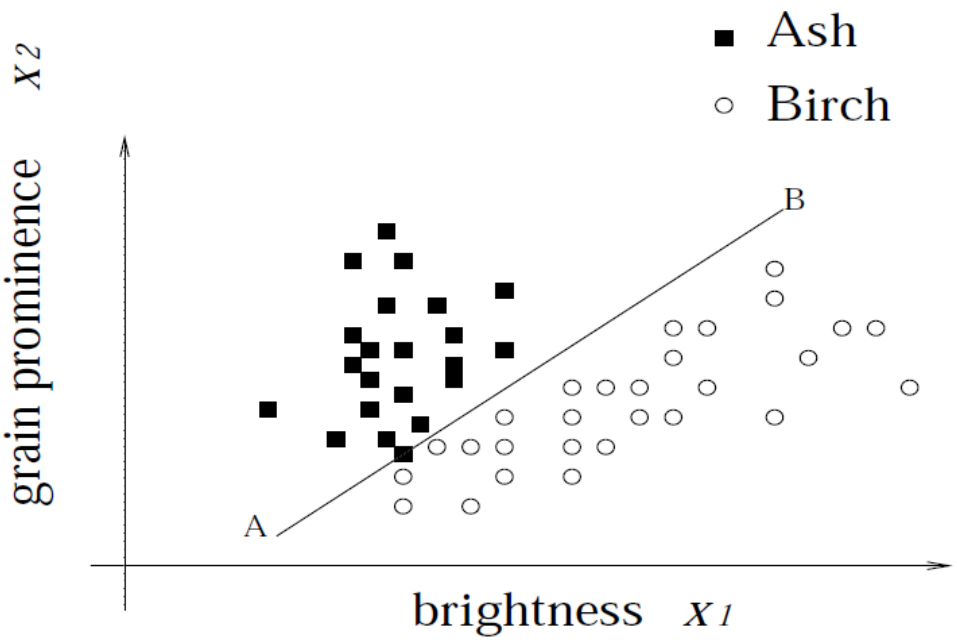


Figure: Scatter diagram for the feature vector.

In order to make sure that this rule performs well, we obtain more samples and adjust the position of line AB in order to minimize the probability of error. This suggests that pattern recognition problems has a statistical nature.

We also see from above example that pattern recognition involves transducer, feature extractor and classifier.

In this course we will focus on *introducing different types of classifiers*.

3:2:1 Concepts in pattern recognition

3:2:1:1 Nonparametric: Nonparametric techniques do not rely on a set of parameters/weights.

3:2:1:2 Parametric: These models are parameterized, with its parameters/weights to be determined through some parameter optimization algorithm, which are then determined by fitting the model to the training data set.

3:2:1:3 Supervised: The training samples are given as some input/output pairs. The output is the desired response for the input. The parameters/weights are adjusted so as to minimize the errors between the response of the networks and the desired response.

3:2:1:4 Unsupervised: Suppose that we are given data samples without being told which

classes they belong to. There are schemes that are aimed to discover significant patterns in the input data without a teacher (labelled data samples).

The nearest neighbor classifiers

The nearest neighbor rule

A set of n pairs $(\mathbf{x}_1, t_1),\ldots,(\mathbf{x}_n, t_n)$ is given, where $\mathbf{x}_i$ takes real values and t_i takes values in the set $\{1,\ldots, M\}$. Each $\mathbf{x}_i$ is the outcome

of the set of measurements made upon the ith individual. Each t_i is the index of the category to which the ith sample belongs.

For brevity we say:

$\mathbf{x}_i$ belongs to category t_i

A set of measurements is made upon a new individual as $\mathbf{x}$, and we wish to assign $\mathbf{x}$ a label in $\{1,\ldots, M\}$. Let $\mathbf{x}_k$ be the sample nearest to $\mathbf{x}$, then the nearest neighbor rule is to assign $\mathbf{x}$ the label associated to $\mathbf{x}_k$.

$$\min\{d(\mathbf{x},\mathbf{x}_i)\} = d(\mathbf{x},\mathbf{x}_k), i = 1,\ldots,n$$

A commonly used distance measure is the sum of squares.

Suppose $\mathbf{x} = [x_1, x_2]^T$ and

$\mathbf{x}_k = d(\mathbf{x},\mathbf{x}_k) = (x_1 - x_{k1})^2 + (x_2 - x_{k2})^2$

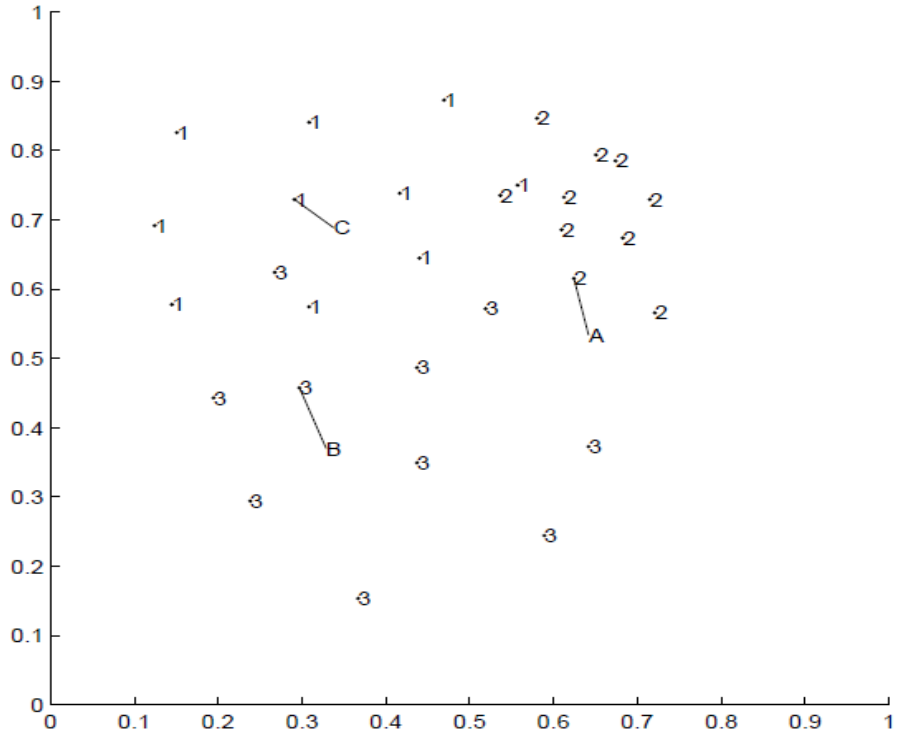


Table: There are three classes each having 10 known samples. Three new samples A,B and C are presented unlabelled. The algorithm can output the class label, for each new sample, as the label of its nearest neighbor. The results are generated by *nn.m*.

Example 1: In order to select the best candidates, an over-subscribed secondary school sets an entrance exam on two subjects of English and Mathematics. Suppose that we know the marks and the classification results of 5 applicants as in the Table below. If an applicant has been accepted, this is denoted as class 1, otherwise class 2. Use the nearest neighbor rule to determine if Andy should be accepted if his marks of English and Mathematics are 70 and 70 respectively.

Candidate No.	English	Math	Class		
1		80		85	1
2		70		60	2
3		50		70	2
4		90		70	1
5		85		75	1

Solution:

1. Calculate the distance between Andy's marks and those of 5 applicants.

$$d_1 = (70 - 80)^2 + (70 - 85)^2 = 225$$

$$d_2 = (70 - 70)^2 + (70 - 60)^2 = 100$$

$$d_3 = (70 - 50)^2 + (70 - 70)^2 = 400$$

$$d_4 = (70 - 90)^2 + (70 - 70)^2 = 400$$

$$d_5 = (70 - 85)^2 + (70 - 75)^2 = 150$$

2. Find out the minimum value amongst

$\{d_1, d_2, d_3, d_4, d_5\}$, which is $d_2 = 100$.

3. Look for the value of the Class for the No.2 applicant, which is 2. Hence the applicant is determined as not acceptable by the algorithm.

The k nearest neighbor rule (k-nn)

An obvious extension of the nearest neighbor rule is the k nearest neighbor rule. This rule classifies the new sample $\mathbf{x}$ by assigning it the label most frequently represented among the k nearest samples.

We will restrict our discussion on the case of two classes.

A decision is made by examining the labels on the k nearest neighbors and taking a vote (k is odd to avoid ties).

Using the same example, we can determine if Andy should be accepted with k nearest neighbor rule, with $k = 3$.

1.Calculate the distance between Andy's marks and those of 5 applicants.

$d_1 = 125$, $d_2 = 100$, $d_3 = 400$, $d_4 = 400$ and $d_5 = 150$.

2. Find out the 3 smallest values amongst

$\{d_1, d_2, d_3, d_4, d_5\}$, which is d_1, d_2, d_5.

3. Look for the values of the Class labels for No.1, No. 2 and No.3

applicants, which are {1,2,1}.

4. There are more ones in the set of {1,2,1}, so the applicant is determined as acceptable by the 3 – *nn* algorithm.

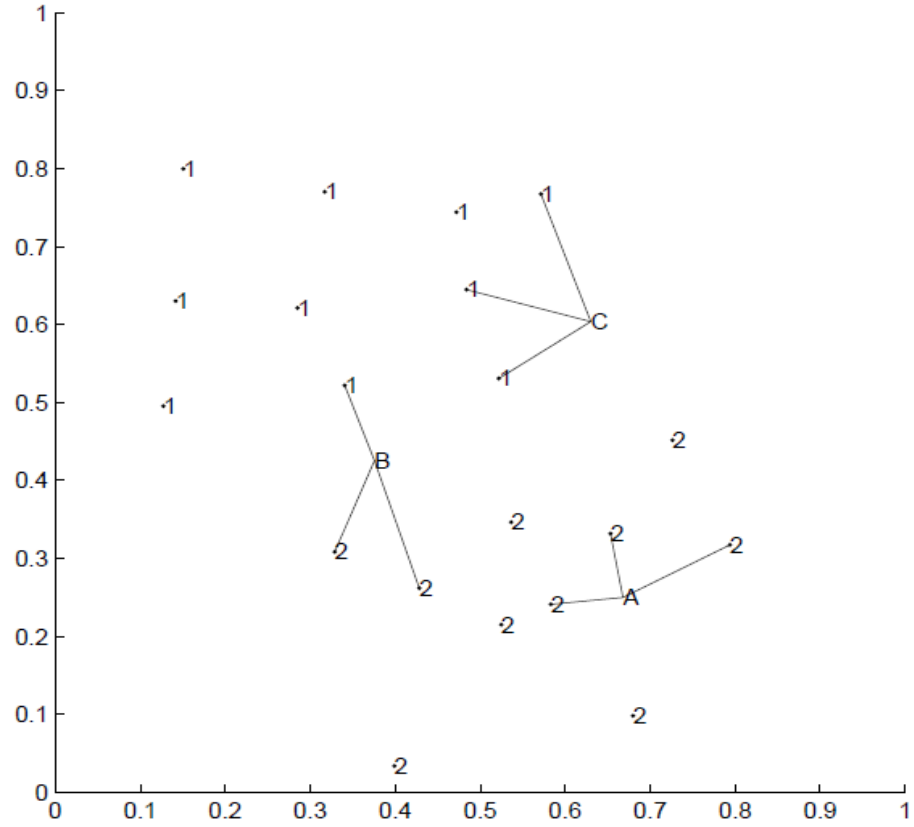
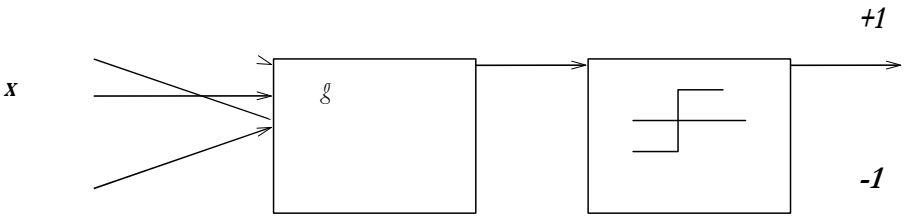


Table: There are two classes each having 10 two known samples. Three new samples A,B and C are presented unlabelled. The algorithm can output the class label, for each new sample, as the label of the most represented 3 nearest neighbors. The results are generated by *knn.m*.

Linear discriminant analysis

Linear discriminant function

There are many different ways to represent a two class pattern classifier. One way is in terms of a discriminant function $g(\mathbf{x})$.



For a new sample $\mathbf{x}$ and a given discriminant function, we can decide on $\mathbf{x}$ belongs to Class 1 if $g(\mathbf{x}) > 0$, otherwise it's Class 2.

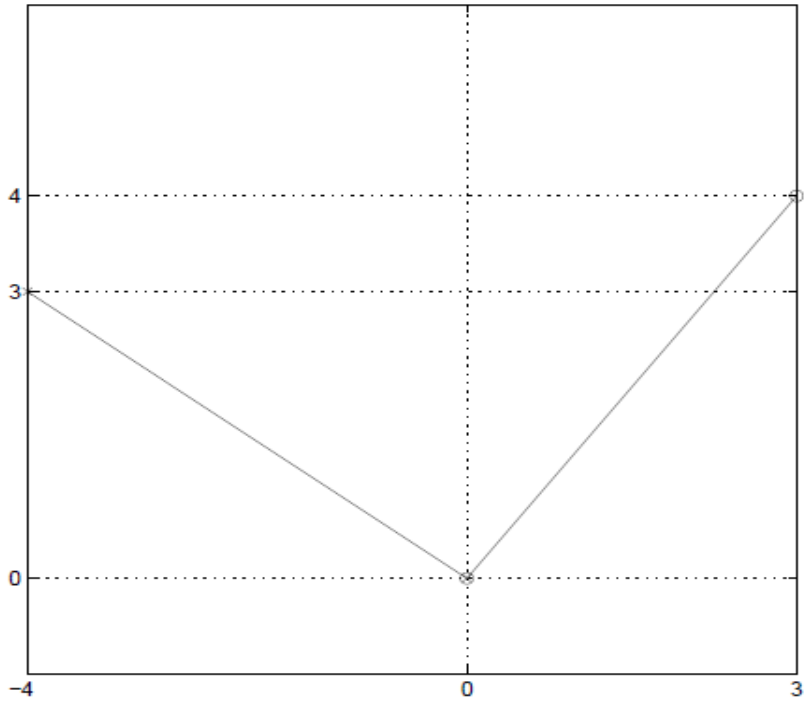
A discriminant function that is a linear combination of the components of $\mathbf{x}$ can be written as

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$$

where $\mathbf{w}$ is called the weight vector and w_0 the threshold weight.

The equation $g(\mathbf{x}) = 0$ defines the decision surface that separates data samples assigned to Class 1 from data samples assigned to Class 2. This is a hyperplane when $g(\mathbf{x})$ is linear.

Two vectors $\mathbf{a}$ and $\mathbf{b}$ are normal to each other if $\mathbf{a}^T\mathbf{b} = 0$. In Figure below we see $[3, 4]$ and $[-4, 3]$ are normal to each other, in algebraic terms, $[3, 4][-4, 3]^T = 3 \times (-4) + 4 \times 3 = 0$



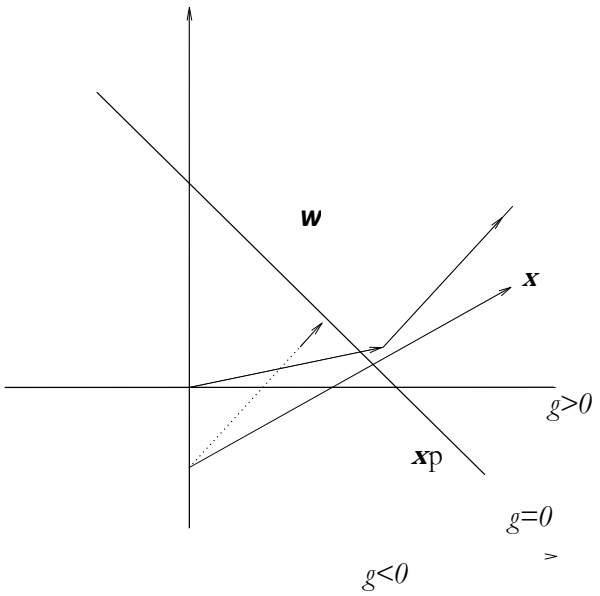
If two points $\mathbf{x}_1$, $\mathbf{x}_2$ are both on the decision surface, then

$$g(\mathbf{x}_1) = g(\mathbf{x}_2) = 0$$

$$\mathbf{w}^T\mathbf{x}_1 + w_0 = \mathbf{w}^T\mathbf{x}_2 + w_0 = 0$$

$$\mathbf{w}^T(\mathbf{x}_1 - \mathbf{x}_2) = 0$$

This means that w is normal to any vector lying in the hyperplane ((x1 − x2) is a vector lying on the the decision surface as it starts from x2, ends at x1).



$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$$

where $\mathbf{x}_p$ is the projection of $\mathbf{x}$ on the hyperplane. r is the distance from $\mathbf{x}$ to the hyperplane.

$$\begin{aligned} g(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + w_0 \\ &= \mathbf{w}^T[\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}] + w_0 \\ &= \mathbf{w}^T\mathbf{x}_p + r\frac{\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|} + w_0 \\ &= \underbrace{\mathbf{w}^T\mathbf{x}_p + w_0}_{0} + r\|\mathbf{w}\| \\ &= r\|\mathbf{w}\| \end{aligned}$$

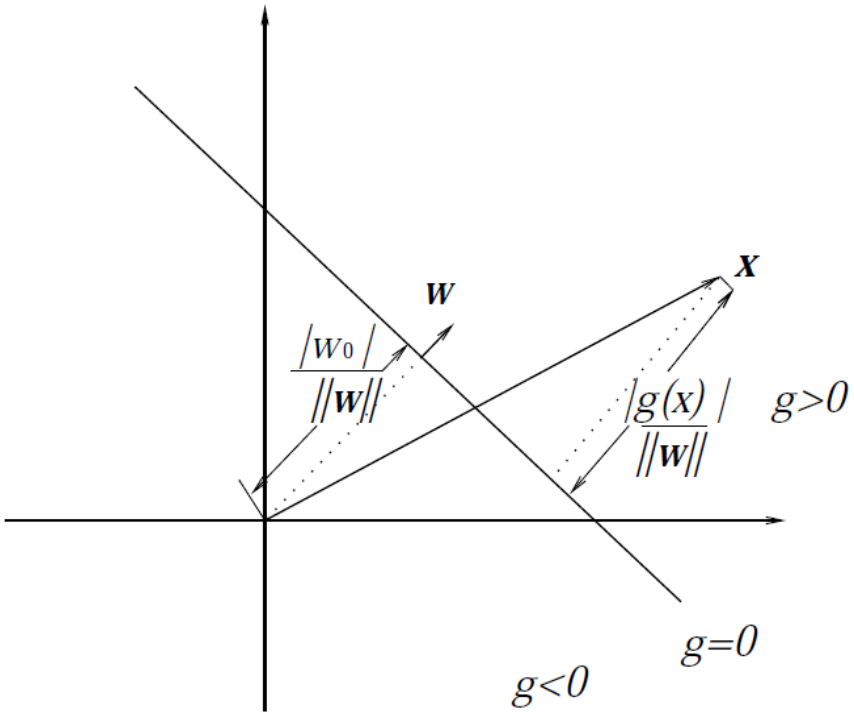
Hence the distance of any data point to the hyperplane is given by

$$r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}$$

In particular, when $\mathbf{x} = [0, 0]^T$.

$$r = \frac{w_0}{\|\mathbf{w}\|}$$

A linear discriminant function divides the feature space by a hyperplane, of which the orientation is determined by the normal vector $\mathbf{w}$, and the location is determined by w_0. If $w_0 = 0$, then the hyperplane passes origin. If $w_0 > 0$, the origin is on the positive side of the hyperplane.



Example 1: In order to select the best candidates, an over-subscribed secondary school sets an entrance exam on two subjects of English and Mathematics. The marks of 5 applicants as listed in the Table below and the decision for acceptance is passing an average mark of 75.

(i) Show that the decision rule is equivalent of the method of linear discriminant function.

(ii) Plot the decision hyperplane, indicating the half planes of both Accept and Reject, and location of the 5 applicants.

Candidate No.	English	Math	Decision
1	80	85	Accept
2	70	60	Reject
3	50	70	Reject
4	90	70	Accept
5	85	75	Accept

Solution: (i) Denote marks of English and Mathas x_1 and x_2, respectively. The decision rule is if $\frac{x_1+x_2}{2} > 75$, accept, otherwise reject.

This is equivalent to using a linear discriminant function

$$g(\mathbf{x}) = x_1 + x_2 - 150$$

with decision rule: if $g(\mathbf{x}) > 0$, accept, otherwise reject.

(ii) To plot $g(\mathbf{x}) = 0$, the easiest way is to set $x_1 = 0$, find the value of x_2 so that $g(\mathbf{x}) = 0$.

i.e. $0 = 0 + x_2 - 150$, so $x_2 = 150$.

$[0, 150]^T$ is on the hyperplane.

Likewise we can also set $x_2 = 0$, find the value of x_1 so that $g(\mathbf{x}) = 0$. i.e. $0 = x_1 + 0 - 150$,

so $x_1 = 150$.

$[150, 0]^T$ is on the hyperplane.

Plot a straight line linking $[0, 150]^T$ and $[150, 0]^T$.

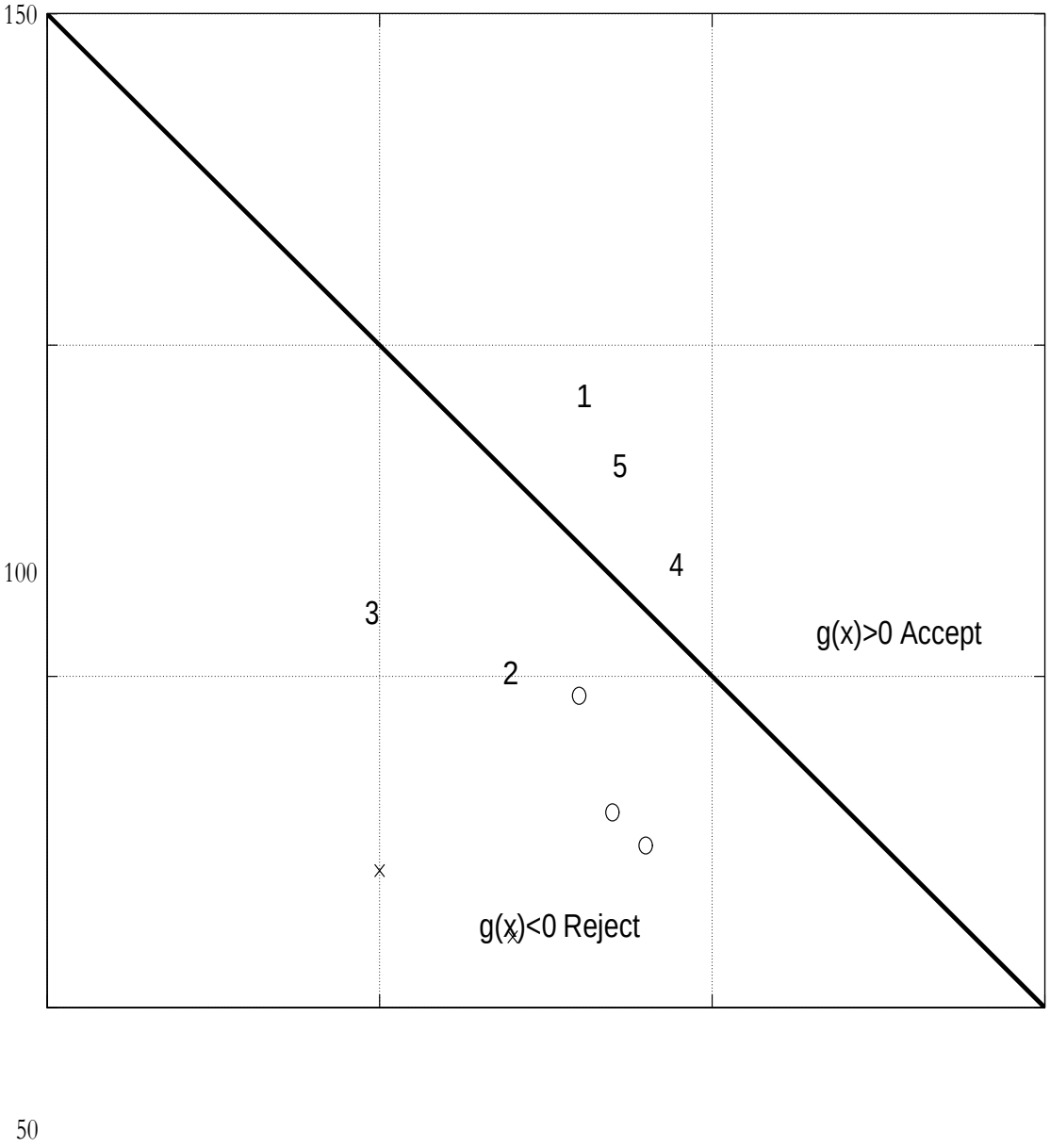

150
100
50
1
5
4
3
2
g(x)>0 Accept
g(x)<0 Reject

Figure: The solution to the example 1 (ii).

There are many ways of determining the linear discriminant function $g(\mathbf{x})$ given a set of training data samples. One way is to set the labelled data samples some target values. e.g. +1 for one class and -1 for another class, then

the weights of the linear discriminant function are adjusted.

Using the same example, a set of linear equations can be constructed based on values in the previous Table.

$$\begin{cases} 80w_1 + 85w_2 + w_0 = 1 \\ 70w_1 + 60w_2 + w_0 = -1 \\ 50w_1 + 75w_2 + w_0 = -1 \\ 90w_1 + 70w_2 + w_0 = 1 \\ 85w_1 + 75w_2 + w_0 = 1 \end{cases}$$

There are 5 equations to solve 3 unknown parameters. There is no exact solution. Instead, the weights are determined by minimizing the overall errors between both sides.

The solution to this problem is often based on the least squares estimate, given by

$$\begin{bmatrix} w_1 \\ w_2 \\ w_0 \end{bmatrix} =$$

$$\{\begin{pmatrix} 80 & 70 & 50 & 90 & 85 \\ 85 & 60 & 75 & 70 & 75 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 80 & 85 & 1 \\ 70 & 60 & 1 \\ 50 & 75 & 1 \\ 90 & 70 & 1 \\ 85 & 75 & 1 \end{pmatrix}\}^{-1}$$

$$\times \begin{pmatrix} 80 & 70 & 50 & 90 & 85 \\ 85 & 60 & 75 & 70 & 75 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

$$= [0.0571, 0.0580, -8.3176]^T$$

So $g(\mathbf{x}) = 0.0571x_1 + 0.0580x_2 - 8.3176$. Note that this is the same hyperplane defined by $g(\mathbf{x}) = x_1 + 1.0106x_2 - 145.6684$, close to the hyperplane used in Example 1.

PART IV
Artificial Neural Networks

4:1 Introduction

In this section of the course we are going to consider neural networks. More correctly, we should call them Artificial Neural Networks (ANN) as we not building neural networks from animal tissue. Rather, we are simulating, on a computer, what we understand about neural networks in the brain.

We start this section of the course by looking at a brief history of the work done in the field of neural networks. Next we look at how a real brain operates (or as much as we know about how a real brain operates). This will provide us with a model we can use in implementing a neural network.

Following this we will look at how we can solve simple algebraic problems using a neural network. In doing so we will discover the limitations of such a model.

4:2 Research History

McCulloch & Pitts (McCulloch, 1943) are generally recognised as being the designers of the first neural network. They recognised that combining many simple processing units together could lead to an overall increase in computational power. Many of the ideas they suggested are still in use today. For example, the idea that a neuron has a threshold level and once that level is reached the neuron fires is still the fundamental way in which artificial neural networks operate.

The McCulloch and Pitts network had a fixed set of weights and it was Hebb (Hebb, 1949) who developed the first learning rule. His premise was that if two neurons were active at the same time then the strength between them should be increased.

In the fifties and throughout the sixties many researchers worked on the ***perceptron*** (Block, 1962, Minsky & Papert, 1988 (originally published in 1969) and Rosenblatt, 1958, 1959 and 1962). This neural network model can be proved to converge to the correct weights, if there are weights that will solve the problem. The learning algorithm (i.e. weight adjustment) used in the perceptron is more powerful than the learning rules used by Hebb.

The perceptron caused great excitement at the time as it was thought it was the path to producing programs that could think. But in 1969 (Minsky & Papert, 1969) it was shown that the perceptron had severe limitations which meant that it could not learn certain types of functions (i.e. those which are not linearly separable).

Due to Minsky and Papert's proof that the perceptron could not learn certain type of (important) functions, research into neural networks went into decline throughout the 1970's.

It was not until the mid 1980's that two people (Parker, 1985) and (LeCun, 1986) independently discovered a learning algorithm for multi-layer networks called backpropogation that could solve problems that were not linearly separable. In fact, the process had been discovered in (Werbos, 1974) and was similar to another algorithm presented by (Bryson & Ho, 1969) but it took until the mid 1980's to make the link to neural networks.

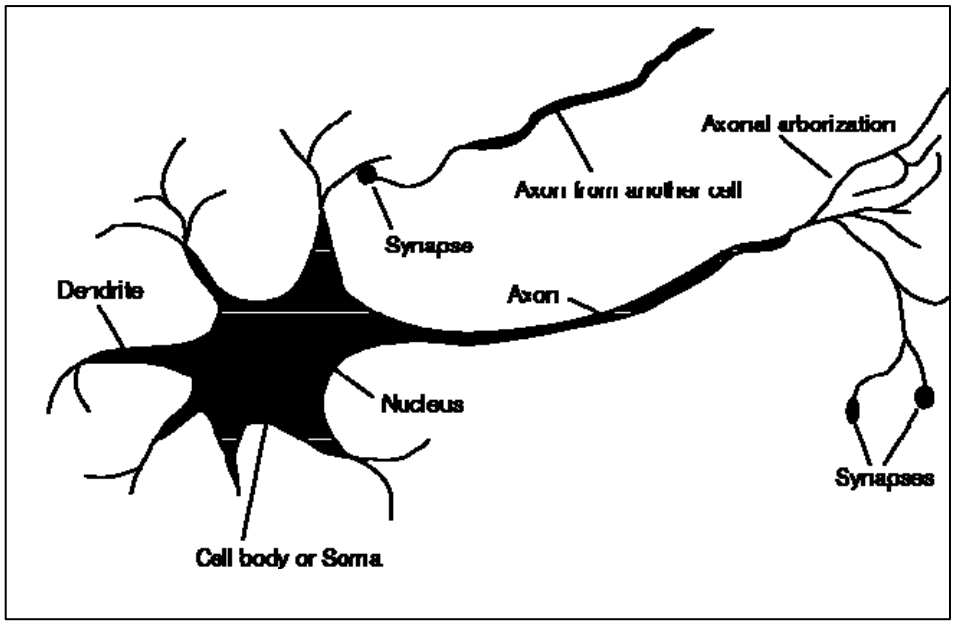
4:3 The Brain

We still do not know exactly how the brain works. For example, we are born with about 100 billion neurons in our brain. Many die as we progress through life, and are not replaced, yet we continue to learn.

Although we do not know exactly how the brain works, we do know certain things about it. We know it is resilient to a certain amount of damage (in addition to the continual loss we suffer as we get older). There have been reports of objects being passed (if passed is the right word) all the way through the brain with only slight impairment to the persons mental capability. We also know what certain parts of the brain do. We know, for example, that much information processing goes on in the cerebral cortex, which is the outer layer of the brain.

From a computational point of view we also know that the fundamental processing unit of the brain is a ***neuron***. A neuron consists of a cell body, or ***soma***, that contains a nucleus. Each neuron has a number of ***dendrites*** which receive connections from other neurons. Neurons also have an ***axon*** which goes out from the neuron and eventually splits into a number of strands to make a connection to other neurons. The point at which neurons join other neurons is called a ***synapse***. A neuron may connect to as many as 100,000 other neurons.

A simplified view of a neuron is shown in the diagram below.



Signals move from neuron to neuron via electrochemical reactions. The synapses release a chemical transmitter which enters the dendrite. This raises or lowers the electrical potential of the cell body. The soma sums the inputs it receives and once a threshold level is reached an electrical impulse is sent down the axon (often known as firing). These impulses eventually reach synapses and the cycle continues.

Synapses which raise the potential within a cell body are called ***excitatory***. Synapses which lower the potential are called ***inhibitory***. It has been found that synapses exhibit ***plasticity***. This means that long-term changes in the strengths of the connections can be formed depending on the firing patterns of other neurons. This is thought to be the basis for learning in our brains.

4:5 The First Artificial Neuron

Much of this section is taken from (Fausett, 1994).

As mentioned in the research history McCulloch and Pitts (1943) produced the first neural network, which was based on their artificial neuron. Although this work was developed in the early forties, many of the principles can still be seen in the neural networks of today.

We can make the following statements about a McCulloch-Pitts network

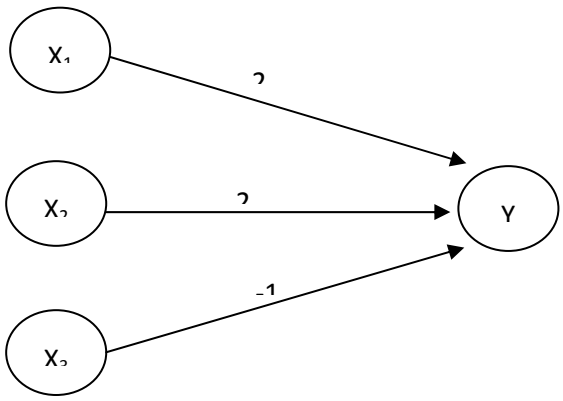
- The activation of a neuron is binary. That is, the neuron either fires (activation of one) or does not fire (activation of zero).
  For the network shown below the activation function for unit *Y* is

f(y_in) = 1, if *y_in* >= θ else 0

where *y_in* is the total input signal received

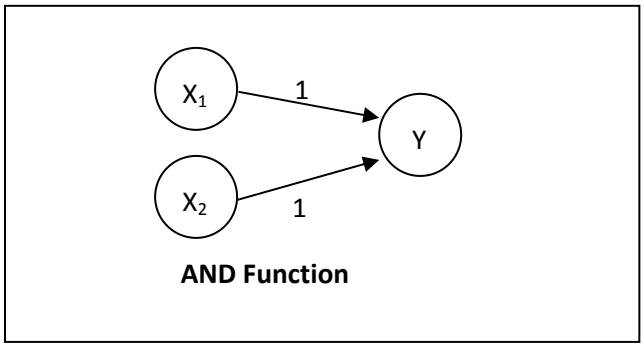
θ is the threshold for *Y*.

- Neurons is a McCulloch-Pitts network are connected by directed, weighted paths.
- If the weight on a path is positive the path is excitatory, otherwise it is inhibitory.
- All excitatory connections into a particular neuron have the same weight, although different weighted connections can be input to different neurons.
- Each neuron has a fixed threshold. If the net input into the neuron is greater than the threshold, the neuron fires.
- The threshold is set such that any non-zero inhibitory input will prevent the neuron from firing.
- It takes one time step for a signal to pass over one connection.

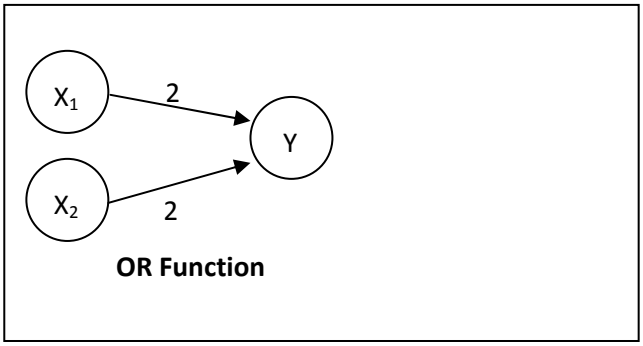


A sample McCulloch-Pitts network is shown above and some of the statements can be observed. In particular, note that the threshold for Y was equal 4 as this is the only value that allows it to fire, taking into account that a neuron cannot fire if it receives a nonzero inhibitory input.

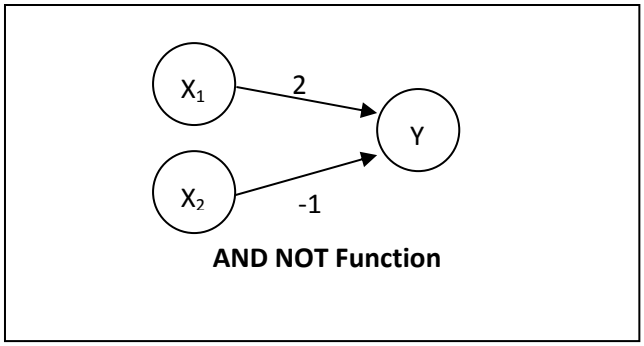
Using the McCulloch-Pitts model we can model logic functions. Below we show and describe the architecture for four logic functions (the truth tables for each function is also shown)



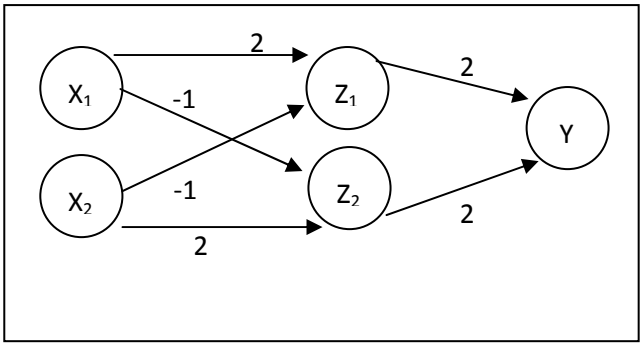
AND Function



OR Function



AND NOT Function



AND		
X_1	X_2	Y
1	1	1
1	0	0
0	1	0

OR		
X_1	X_2	Y
1	1	1
1	0	1
0	1	1

AND NOT		
X_1	X_2	Y
1	1	0
1	0	1
0	1	0

XOR		
X_1	X_2	Y
1	1	0
1	0	1
0	1	1

0	0	0

0	0	0

0	0	0

0	0	0

AND Function

As both inputs (X_1 and X_2) are connected to the same neuron the connections must be the same, in this case 1. To model the AND function the threshold on Y is set to 2.

OR Function

This is almost identical to the AND function except the connections are set to 2 and the threshold on Y is also set to 2.

AND NOT Function

Although the truth table for the AND NOT function is shown above it deserves just a small explanation as it is not often seen in the textbooks. The function is not symmetric in that an input of 1,0 is treated differently to an input of 0,1. As you can see from the truth table the only time true (value of one) is returned is when the first input is true and the second input is false. Again, the threshold on Y is set to 2 and if you apply each of the inputs to the AND NOT network you will find that we have modeled X_1 AND NOT X_2.

XOR Function

XOR can be modeled using AND NOT and OR;

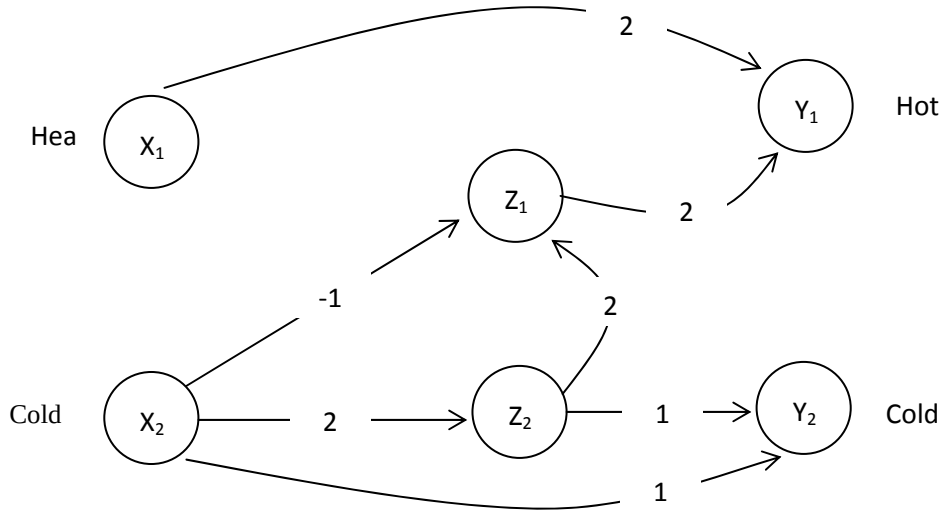
X_1 XOR X_2 = (X_1 AND NOT X_2) OR (X_2 AND NOT X_1)

(To prove it draw the truth table)
This explains the network shown above. The first layer performs the two AND NOT's and the second layer performs the OR. Both Z neurons and the Y neuron have a threshold of 2.

As a final example of a McCulloch-Pitts network we will consider how to model the phenomenon that if you touch something very cold you initially perceive heat. Only after you have left your hand on the cold source for a while do you perceive cold. This example (from Fausett, 1994) is an elaboration of one originally presented by (McCulloch and Pitts, 1943). To model this we will assume that time is discrete. If cold is applied for one time step then heat will

be perceived. If a cold stimulus is applied for two time steps then cold will be perceived. If heat is applied then we should perceive heat.

Take a look at this figure. Each neuron has a threshold of 2. This, as we shall see, allows us to model this phenomenon.



First though, remember that time is discrete, so it takes time for the stimulus (applied at X_1 and X_2) to make its way to Y_1 and Y_2 where we perceive either heat or cold.
Therefore, at t(0), we apply a stimulus to X_1 and X_2.
At t(1) we can update Z_1, Z_2 and Y_1.
At t(2) we can perceive a stimulus at Y_2.
At t(2+n) the network is fully functional.

Before we see if the network performs as we hope let's consider what we are trying to do.

Input to the system will be (1,0) or (0,1), which represents hot and cold respectively.
We want the system to perceive cold if a cold stimulus is applied for two time steps. That is

$$Y_2(t) = X_2(t-2) \; AND \; X_2(t-1) \qquad (1)$$

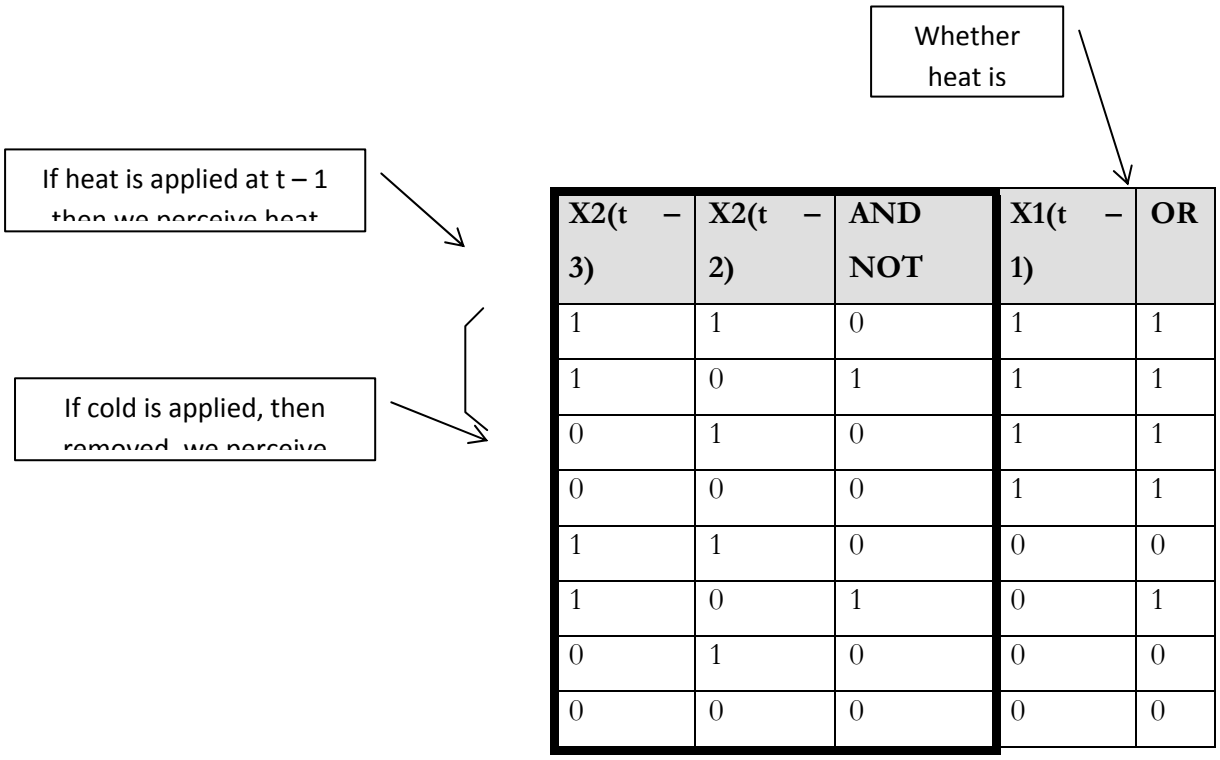
This truth table (i.e. the truth table for AND) shows that this model is correct

$X_2(t-2)$	$X_2(t-1)$	$Y_2(t)$
1	1	1
1	0	0
0	1	0
0	0	0

We want the system to perceive heat if either a hot stimulus is applied or a cold stimulus is applied (for one time step) and then removed. We can express this as follows

$$Y_1(t) = [\, X_1(t-1) \,] \text{ OR } [\, X_2(t-3) \text{ AND NOT } X_2(t-2) \,] \quad (2)$$

This truth table shows that it gives us the required result



X2(t – 3)	X2(t – 2)	AND NOT	X1(t – 1)	OR
1	1	0	1	1
1	0	1	1	1
0	1	0	1	1
0	0	0	1	1
1	1	0	0	0
1	0	1	0	1
0	1	0	0	0
0	0	0	0	0

So, if we are convinced that we have the correct logical statements to represent the hot/cold problem, we now need to convince ourselves that the network above represents the logical statements.

The figure of the network shows that

$$Y_1(t) = X_1(t-1) \text{ OR } Z_1(t-1) \qquad (3)$$

(compare this to the OR network we developed earlier).

Now consider the Z_1 neuron. This is how it is formed

$$Z_1(t-1) = Z_2(t-2) \text{ AND NOT } X_2(t-2) \qquad (4)$$

(again, compare this to the AND NOT network above).

Now Z_2, this is simply

$$Z_2(t-2) = X_2(t-3) \qquad (5)$$

If we take formula (3), and substitute in formula (4) and (5) we end up with

$$Y_1(t) = [\, X_1(t-1) \,] \text{ OR } [\, X_2(t-3) \text{ AND NOT } X_2(t-2) \,] \qquad (6)$$

which is the same as formula (2), showing that our network (Y_1 anyway) works correctly (as we have proved formula 2 works using a full analysis by using the truth table).

We can perform a similar analysis for Y_2, and show that Y_2 in the network acts in the way we developed above (formula (1)). You should do this to convince yourself that this is correct.

If you still don't believe it, there is a spreadsheet available from the course web site that implements this network so that you can see that it works as we expect.

4:6 Modelling a Neuron

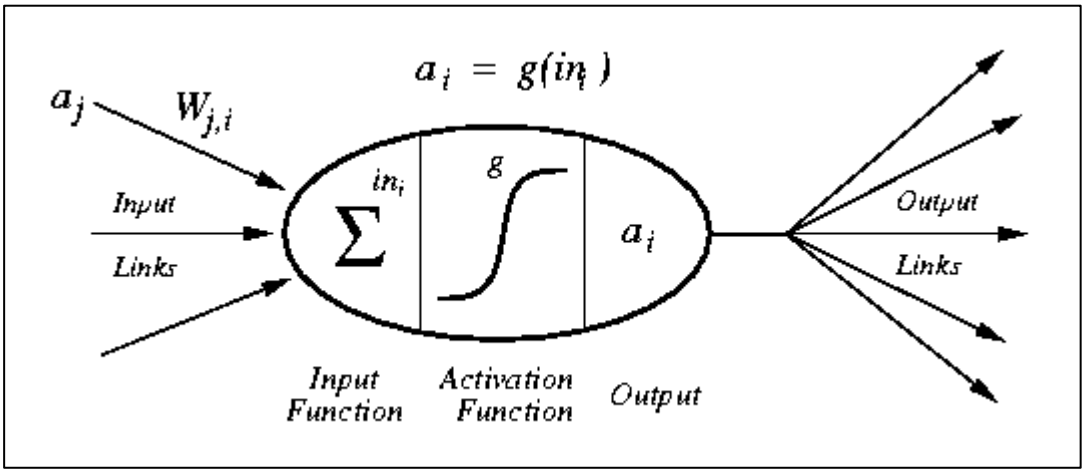
To model the brain we need to model a neuron. Each neuron performs a simple computation. It receives signals from its input links and it uses these values to compute the activation level (or output) for the neuron. This value is passed to other neurons via its output links.

The input value received of a neuron is calculated by summing the weighted input values from its input links. That is

$$in_i = \sum_j W_{j,i} a_j$$

An activation function takes the neuron input value and produces a value which becomes the output value of the neuron. This value is passed to other neurons in the network.

This is summarised in this diagram and the notes below.



a_j : Activation value of unit j

$w_{j,i}$: Weight on the link from unit j to unit i

in_i : Weighted sum of inputs to unit i

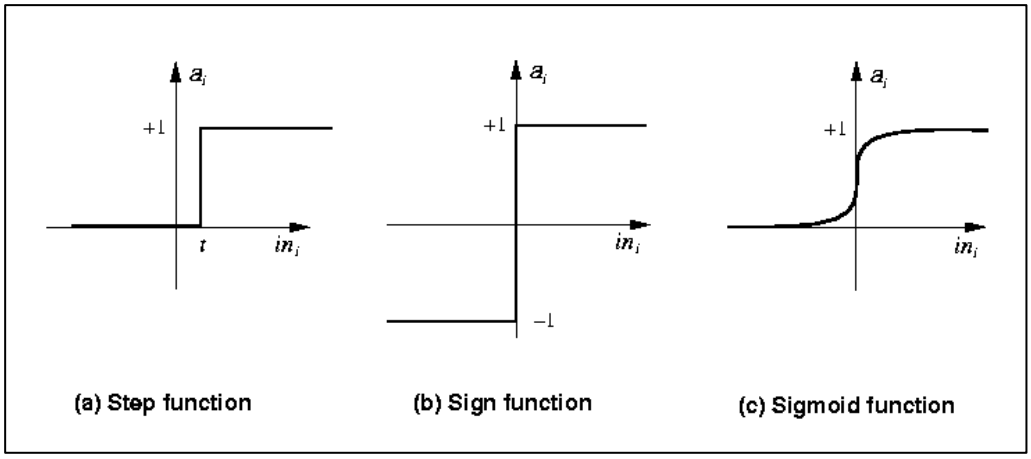
a_i : Activation value of unit i (also known as the output value)

g : Activation function

Or, in English.

A neuron is connected to other neurons via its input and output links. Each incoming neuron has an activation value and each connection has a weight associated with it.
The neuron sums the incoming weighted values and this value is input to an activation function. The output of the activation function is the output from the neuron.

Some common activation functions are shown below.



These functions can be defined as follows.

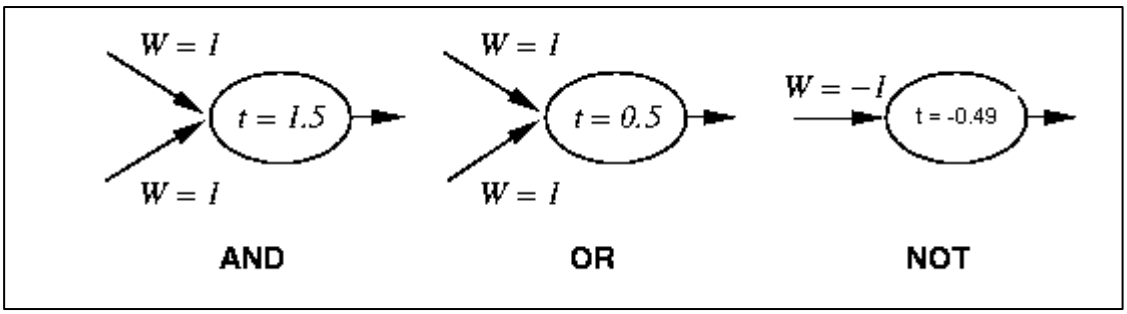
$Step_t(x)$ = 1 if x >= t, else 0
Sign(x) = +1 if x >= 0, else –1
Sigmoid(x) = $1/(1+e^{-x})$

On occasions an identify function is also used (i.e. where the input to the neuron becomes the output). This function is normally used in the input layer where the inputs to the neural network are passed into the network unchanged.

4:7 Some Simple Networks

We can use what we have learnt above to demonstrate a simple neural network which acts as a logic gate.
The diagram below is modelling the following truth tables

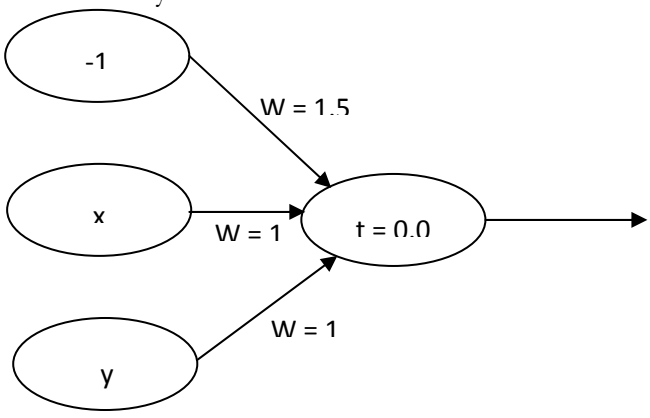
	AND				OR				NOT	
Input 1	0	0	1	1	0	0	1	1	0	1
Input 2	0	1	0	1	0	1	0	1		
Output	0	0	0	1	0	1	1	1	1	0



In these networks we are using the step activation function.
You should convince yourself that these networks produce the correct output for all the allowed inputs.

You will notice that each neuron has a different threshold. From a computational viewpoint it would be easier if all the neurons had the same threshold value and the actual threshold was somehow modelled by the weights.

In fact, it is possible to do exactly this. Consider this network



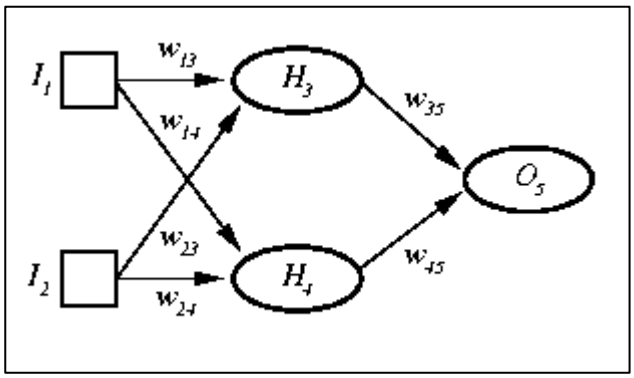
It is the same as the AND network in the diagram above except that there is an extra input neuron whose activation is always set to –1. The threshold of the neuron is represented by the weight that links this extra neuron to the output neuron.
This means the threshold of the neuron can be set to zero.

You might like to work through this network using the four possible combinations of x and y and convince yourself that the network operates correctly.

This advantage of this method is that we can always set the threshold for every neuron to zero and from a computational point of view, when "training" the network, we only have to update weights and not both thresholds and weights.

4:8 Types of Network

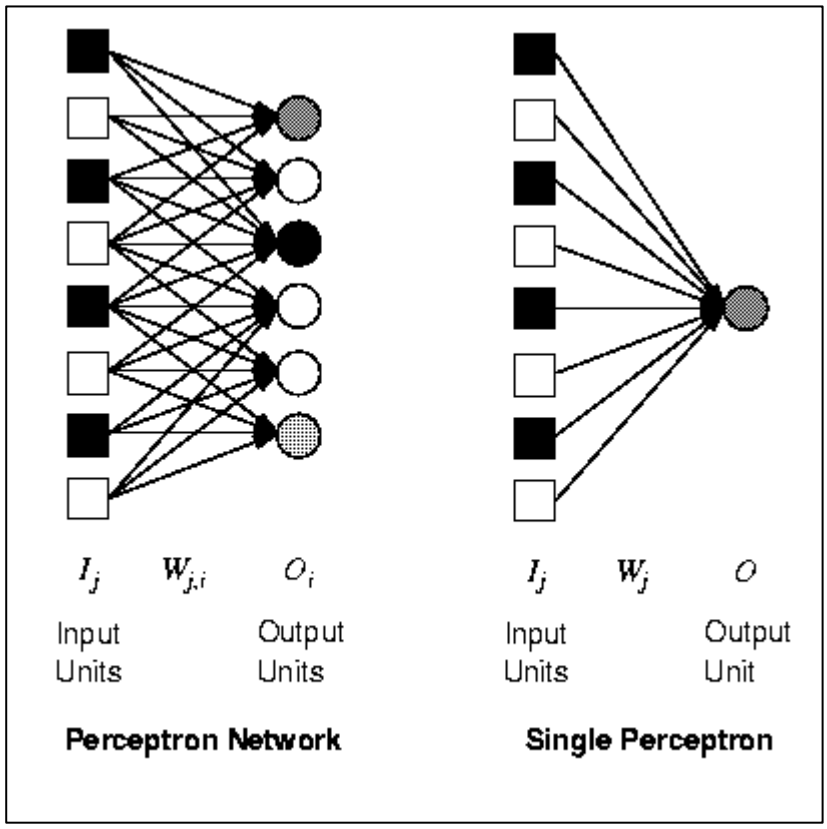
The simple networks we have considered above only have input neurons and output neurons. It is considered a one layer network (the input neurons are not normally considered to form a layer as they are just a means of getting data into the network).
Also, in the networks we have considered, the data only travels in one direction (from the input neurons to the output neurons). In this respect it is known as a feed-forward network.
Therefore, we have been looking at one-layer, feed-forward networks.
There are many other types of network. An example of a two-layer, feed-forward network is shown below.



The Perceptron

The name perceptron is now used as a synonym for single-layer, feed-forward networks. They were first studied in the 1950's and although other network architectures were known about the perceptron was the only network that was known to be capable of learning and thus most of the research at that time concentrated on perceptrons.

The diagram below shows example of perceptrons.



You will see, in the left hand network that a single weight only affects one of the outputs. This means we can make our study of perceptrons easier by only considering networks with a single output (i.e. similar to the network shown on the right hand side of the diagram).

As we only have one output we can make our notation a little simpler. Therefore the output neuron is denoted by O and the weight from input neuron j is denoted by W_j.
Therefore, the activation function becomes

$$O = Step_0 \sum_j WjIj$$

(Note, we are assuming the use of additional weight to act as a threshold so that we can use a $step_0$ function, rather than $step_t$).

What can perceptrons represent?

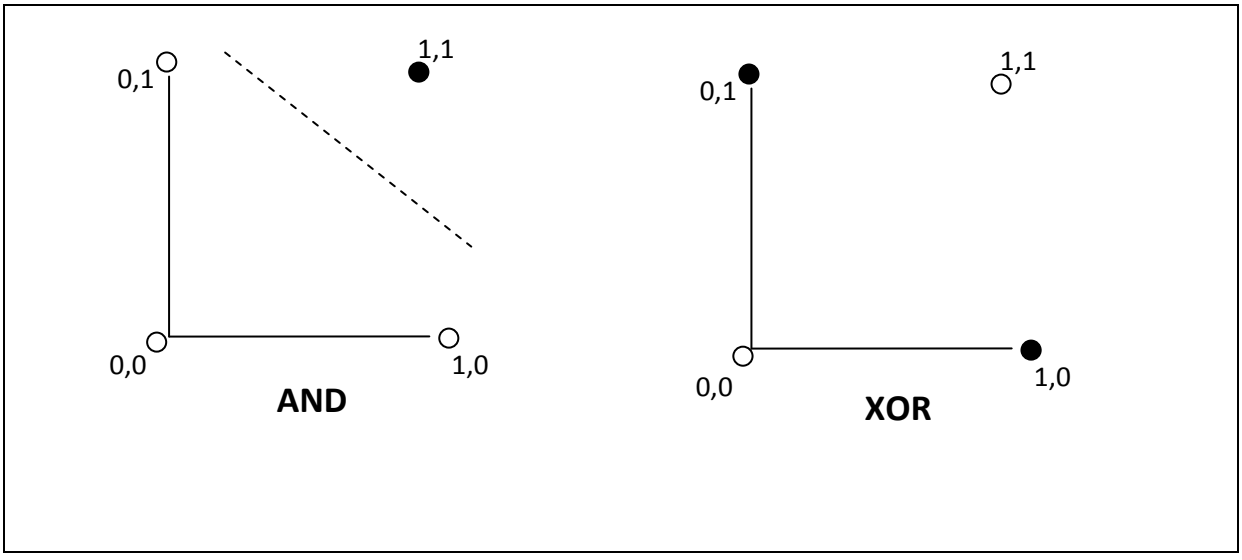
We have already seen that perceptrons can represent the AND, OR and NOT logic functions. But does it follow that a perceptron (a single-layer, feed-forward network) can represent any boolean function?

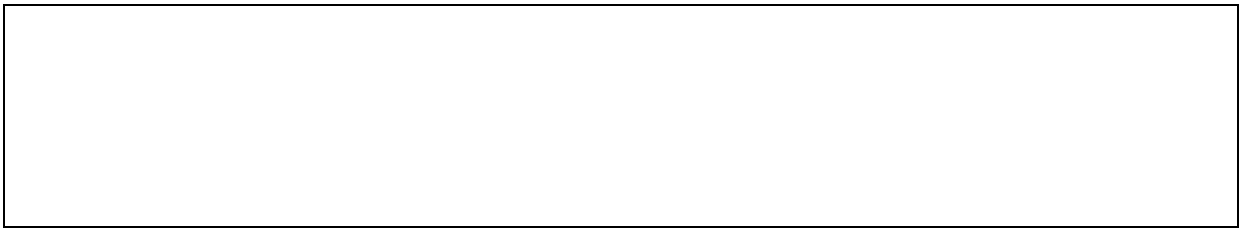
Unfortunately, this is not the case.

To see why, consider these two truth tables

	AND				XOR			
Input 1	0	0	1	1	0	0	1	1
Input 2	0	1	0	1	0	1	0	1
Output	0	0	0	1	0	1	1	0

We can represent these two truth tables graphically. Like this



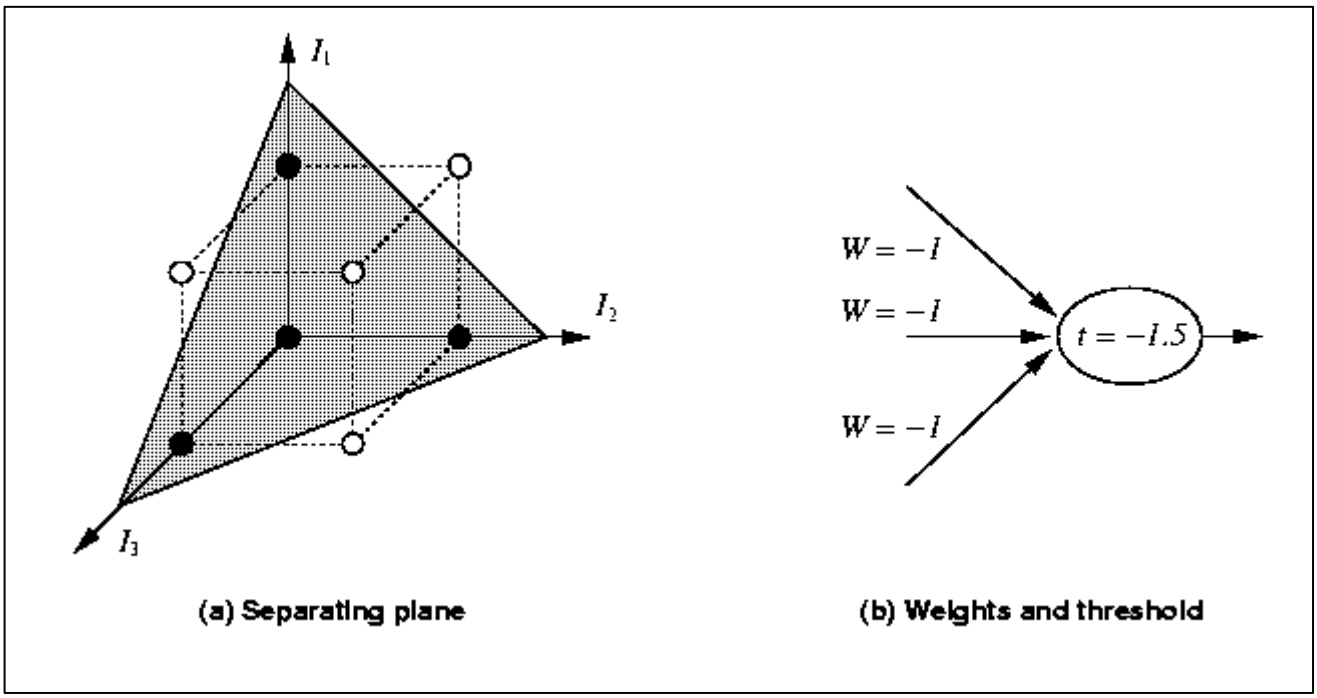


(where a filled circle represents an output of one and a hollow circle represents an output of zero).

If you look at the AND graph you will see that we can divide the one's from the zero's with a line. This is not possible with XOR. Functions such as AND are called ***linearly separable***.

It was the proof by Minsky & Papert in 1969 that perceptrons could only learn linearly separable functions that led to decline in neural network research until the mid 1980's when it was proved that other network architectures could learn these type of functions.

Although, only being able to learn linearly separable functions is a major disadvantage of the perceptron it is still worth studying as it is relatively simple and can help provide a framework for other architectures.
It should however, be realised that perceptrons are not only limited to two inputs (e.g. the AND function). We can have n inputs which gives us an n-dimension problem.
When n=3 we can still visualise the linear separability of the problem (see diagram below).



But once *n* is greater than three we find it difficult to visualise the problem.

4:9 Learning Linearly Separable Functions

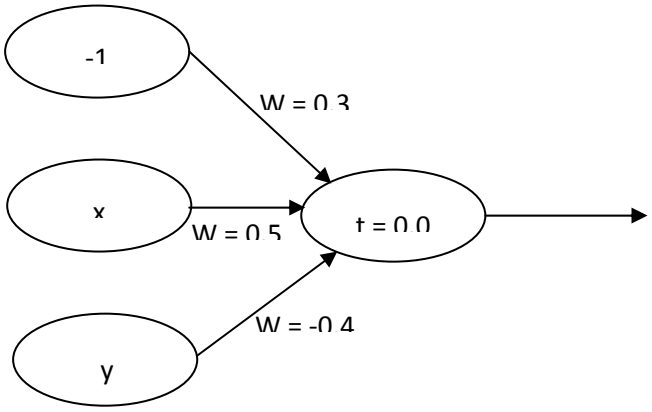
With a function such as AND (with only two inputs) we can easily decide what weights to use to give us the required output from the neuron. But with more complex functions (i.e. those with more than two inputs it may not be so easy to decide on the correct weights).
Therefore, we would like our neural network to "learn" so that it can come up with its own set of weights.

We will consider this aspect of neural networks for a simple function (i.e. one with two inputs). We could obviously scale up the problem to accommodate more complex problems, providing the problems are linearly separable.

Consider this truth table (AND) and the neuron that we ***hope*** will represent it.

AN

	D			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	0	0	1

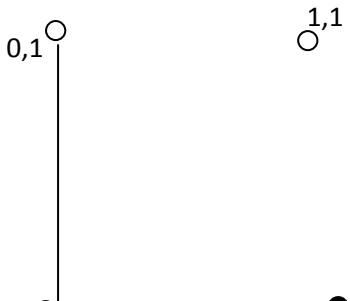


In fact, all we have done is set the weights to random values between –0.5 and 0.5

By applying the activation function for each of the four possible inputs to this neuron it actually gives us the following truth table

	???			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	0	1	0

Which can be graphically represented as follows



The network, obviously, does not represent the AND function so we need to adjust the weights so that it "learns" the function correctly.

The algorithm to do this follows, but first some terminology.

Epoch : An epoch is the presentation of the entire training set to the neural network. In the case of the AND function an epoch consists of four sets of inputs being presented to the network (i.e. [0,0], [0,1], [1,0], [1,1]).

Training Value, T : When we are training a network we not only present it with the input but also with a value that we require the network to produce. For example, if we present the network with [1,1] for the AND function the training value will be 1.

Error, Err : The error value is the amount by which the value output by the network differs from the training value. For example, if we required the network to output 0 and it output a 1, then Err = -1.

Output from Neuron, O : The output value from the neuron

I_j : Inputs being presented to the neuron

W_j : Weight from input neuron (I_j) to the output neuron

LR : The learning rate. This dictates how quickly the network converges. It is set by a matter of experimentation. It is typically 0.1.

The Perceptron Training Algorithm

While epoch produces an error

Present network with next inputs from epoch

Err = T – O

If Err <> 0 then

Note : If the error is positive we need to increase O. If the error is negative we need to decrease O. Each input contributes WjIj to the total input so if Ij is positive, an increase in Wj will increase O. If Ij is negative an increase in Wj will decrease O). This can be achieved with the following

$W_j = W_j + LR * I_j * Err$

Note : This is often called the delta learning rule.

End If

End While

Perceptron Learning – An Example

Let's take a look at an example. The initial weight values are 0.3, 0.5, and -0.4 (taken from the above example) and we are trying to learn the AND function.

If we present the network with the first training pair ([0,0]), from the first epoch, nothing will happen to the weights (due to multiplying by zero).
The next training pair ([0,1]) will result in the network producing zero (by virtue of the $step_0$ function). As zero is the required output there is no error so training continues.
The next training pair ([1,0]) produces an output of one. The required output is 0. Therefore the error is –1. This means we have to adjust the weights.

This is done as follows (assuming LR = 0.1)

$W_0 = 0.3 + 0.1 * -1 * -1 = 0.4$
$W_1 = 0.5 + 0.1 * 1 * -1 = 0.4$
$W_2 = -0.4 + 0.1 * 0 * -1 = -.04$

Therefore, the new weights are 0.4, 0.4, -0.4.

Finally we apply the input [1,1] to the network. This also produces an error and the new weight values will be 0.3, 0.5 and –0.3.

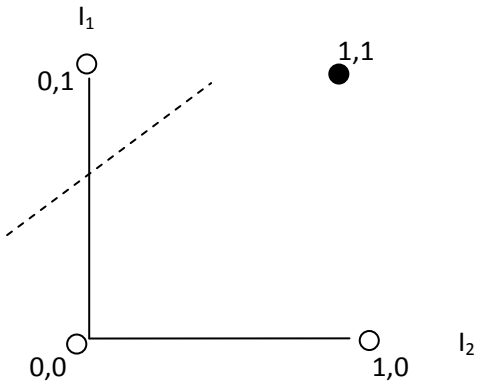
As this presentation of the epoch produced an error (two in fact) we need to continue the training and present the network with another epoch.

Training continues until an epoch is presented that does not produce an error.

If we consider the state of the network at the and of the first epoch (weights = 0.3, 0.5, -0.3) we know the weights are wrong as the epoch produced an error (in fact, the weights *may* be correct

at the end of the epoch but we need to present another epoch to show this). We can also produce a graph that shows the current state of the network.

We are trying to achieve the AND function which can be represented as follows



The current "linear separability" line can be drawn by using the weights to draw a line on the graph. Two points on the I_1 and I_2 axis can be found as follows

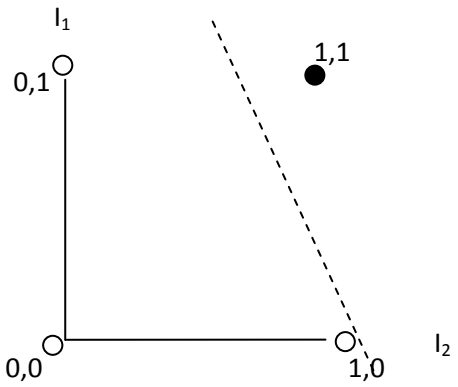
I_1 point = W_0/W_1
I_2 point = W_0/W_2

Note : As discussed above, W_0 actually represents the threshold. Therefore, we are dividing the threshold by the weights.

That is, $I_1 = 0.6$ and $I_2 = -1$

This line is shown (*roughly* in the right position) on the above graph. It is clearly not in the correct place as the line does not divide the one's and the zero's correctly.

If we continue with the training until we get no errors from an entire epoch the weights would be 0.4, 0.4, 0.1. If we plot these weights on the graph we get (again the line is *roughly* in the correct position).



And we can see that the linearly separability lies is in a valid position and this can be confirmed by checking that the neural network is producing the correct outputs. You might like to do this for weights of 0.4, 0.4 and 0.1.

Unless I plan to set some coursework that involves you building a neural network, a spreadsheet available via the course web site will show you how we can implement a perceptron using a spreadsheet. This will allow you to experiment with it (for example, showing that an XOR type problem can never be learnt).

5:0 References

- Aleksander, I., Morton, H. 1995. *An Introduction to Neural Computing (2nd ed)*. Chapman and Hall
- Anderson, J.A., Rosenfeld, E. (eds). 1988. *Neurocomputing: Foundations of Research*. Cambridge, MA: MIT Press
- Block, H.D. 1962. *The Perceptron: A Model for Brain Functioning, I*. Reviews of Modern Physics, Vol 34, pp 123-135. Reprinted in Anderson and Rosenfeld,1988, pp 138-150
- Bryson, A.E., Ho, Y-C. 1969. *Applied Optimal Control*. New York: Blaisdell
- Callan, R. (1999) *The Essence of Neural Networks*, Prentice Hall, ISBN 0-13-908732-X
- Davalo, E., Naim, P. (1991). *Neural Networks*, The Macmillan Press Ltd, ISBN 0-333-54996-1
- Fausett, L. 1994. *Fundamentals of Neural Networks : Architectures, Algorithms and Applications*. Prentice-Hall, ISBN 0-13-103805-2
- Hebb, D.O. 1949. *The Organization of Behavior*. New York: John Wiley & Sons. Introduction and Chapter 4 reprinted in Anderson & Rosenfeld, 1988, pp 45-56
- Le Cun, Y. 1986. *Learning Processes in an Asymmetric Theshold Network*. Disordered Systems and Biological Organization (Bienenstock, E., Fogelman-Smith, F., Weisbuch, G. (eds)), NATO ASI Series, F20, Berlin: Springer-Verlag
- McCulloch, W.S., Pitts, W. 1943. *A Logical Calculus of the Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics*, Vol 5, pp 115-133. Reprinted in Anderson & Rosenfeld, 1988, pp 18-28.

- Minsky, M.L., Papert, S.A. 1988. *Perceptrons, Expanded Edition*. Cambridge, MA: MIT Press. Original Edition, 1969.
- Parker, D. 1985. *Learning Logic*. Technical Report TR-87, Cambridge, MA: Center for Computational Research in Economics and Management Science, MIT
- Rosenblatt, F. 1958. *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*. Psychological Review, Vol 65, pp 386-408. Reprinted in Anderson and Rosenfeld, 1988, pp 92-114
- Rosenblatt, F. 195. *Two Theorems of Statistical Separability in the Perceptron*. Mechanization of Thought Processes of a Symposium held at the National Physical Laboratory, November, 1958. London: HM Stationery Office, pp 421-456
- Rosenblatt, F. 1962. *Principles of Neurodynamics*. New York: Spartan
- Russell, S., Norvig, P. 1995. *Artificial Intelligence A Modern Approach*. Prentice-Hall. ISBN 0-13-103805-2
- Werbos, P. 1974. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D Thesis, Cambridge, MA: Harvard U. Committee on Applied Mathematics.