Artificial Neural Networks

INTRODUCTION TO NEURAL NETWORKS

Topics

1. Reducing Dimensionality
   b) Principal Component Analysis (PCA)
2. Introduction to Artificial Neural Networks
   a) Motivation
   b) Biological Model
   c) Neuron Model
   d) Perceptron

Overview

Principal Component Analysis (PCA) is a way to reduce data dimensionality

PCA projects high dimensional data to a lower dimension

PCA projects the data in the least square sense– it captures big (principal) variability in the data and ignores small variability

Variance and Co-Variance

In order to capture big (principal) variability in the data we need to measure the variabilities

Variance: In probability theory and statistics, variance is the expectation of the squared deviation of a random variable from its mean.

Co-Variance: is a measure of the joint variability of two random variables.

Sample Two-dimensional Data

Covariance: measures the correlation between X and Y

- Cov(X,Y)=0: X and Y are independent
- Cov(X,Y)>0: X and Y move in the same direction
- Cov(X,Y)<0: X and Y move in opposite directions

$$\mathrm{cov}(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)} = 57.14$$

X=Temperature	Y=Humidity
40	90
40	90
40	90
30	90
15	70
15	70
15	70
30	90
15	70
30	70
30	70
30	90
40	70
30	90

Covaiance Matric: More than two attributes

$$C^{nxn} = (c_{ij} \mid c_{ij} = \text{cov}(Dim_i, Dim_j))$$

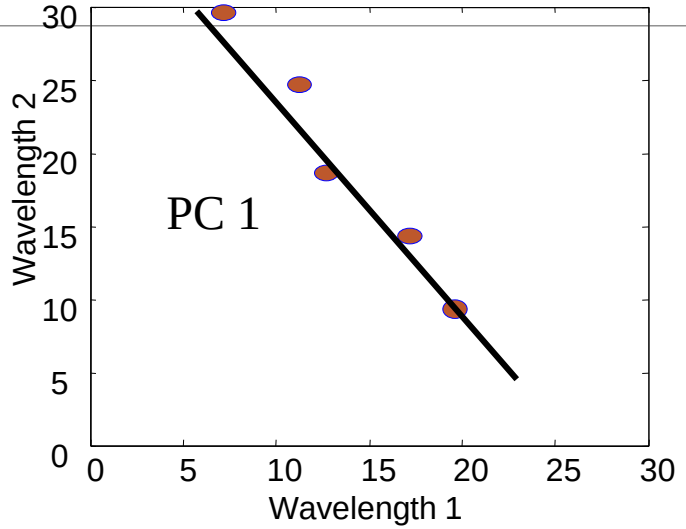
Example for three attributes (x,y,z):

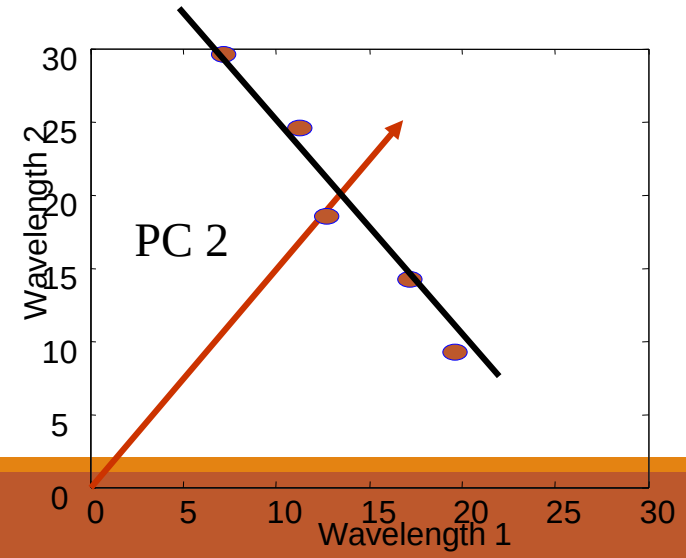
$$C = \begin{pmatrix} \text{cov}(x,x) & \text{cov}(x,y) & \text{cov}(x,z) \\ \text{cov}(y,x) & \text{cov}(y,y) & \text{cov}(y,z) \\ \text{cov}(z,x) & \text{cov}(z,y) & \text{cov}(z,z) \end{pmatrix}$$

Principal Components

First PC is direction of maximum variance from origin

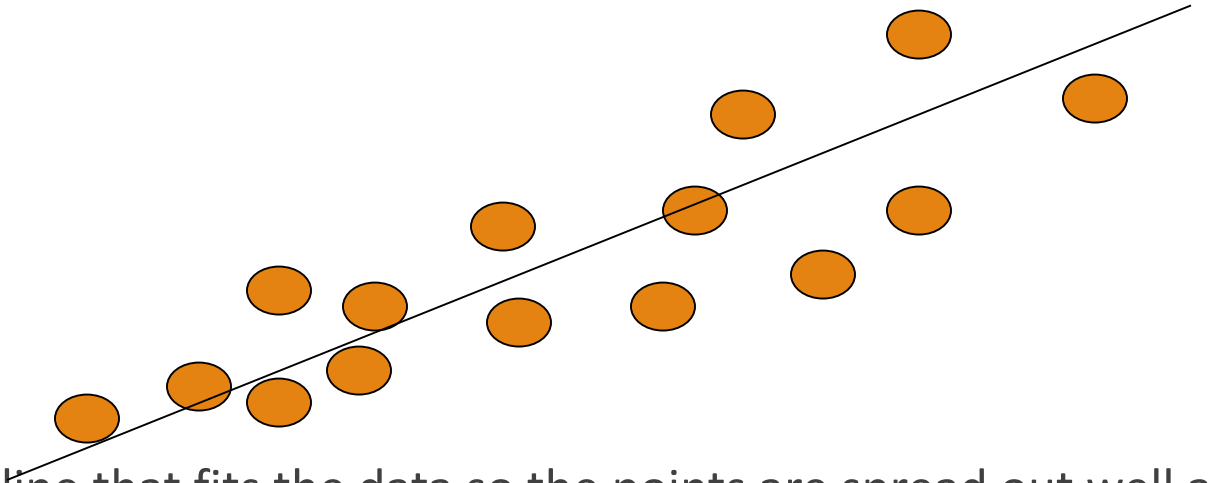
Subsequent PCs are orthogonal to 1st PC and describe maximum residual variance





Algebraic Interpretation – 1D

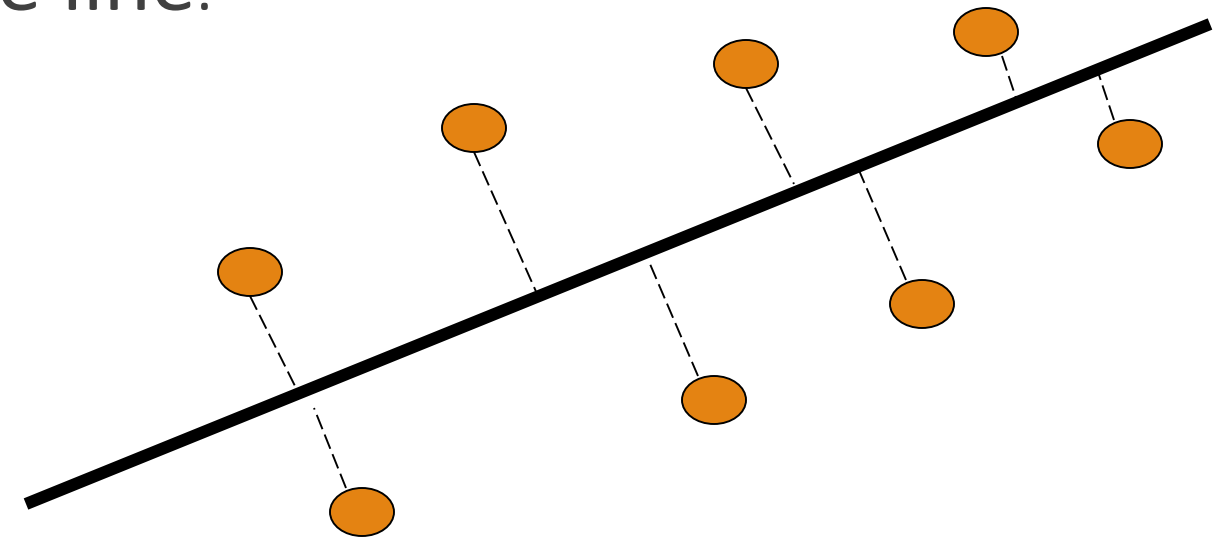
Given m points in a n dimensional space, what is the best line to represent this data?



Choose a line that fits the data so the points are spread out well along the line

Algebraic Interpretation – 1D

Formally, minimize sum of squares of distances to the line.



PCA: *General*

From k original variables: $x_1,x_2,\ldots,x_k$:

Produce k new variables: $y_1,y_2,\ldots,y_k$:

$y_1 = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1k}x_k$

$y_2 = a_{21}x_1 + a_{22}x_2 + \ldots + a_{2k}x_k$

...

$y_k = a_{k1}x_1 + a_{k2}x_2 + \ldots + a_{kk}x_k$

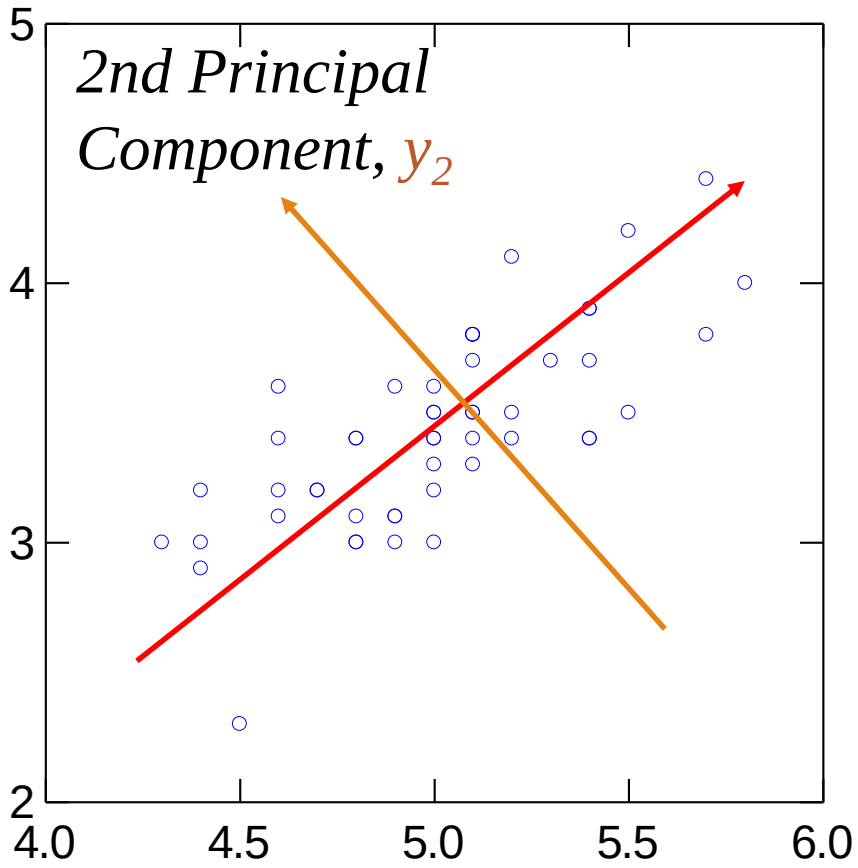
such that:

y_k's are uncorrelated (orthogonal)

y_1 explains as much as possible of original variance in data set

y_2 explains as much as possible of remaining variance

etc.


2nd Principal Component, y_2
1st Principal Component, y_1
5
4
3
2
4.0
4.5
5.0
5.5
6.0

Principal Components

Principal components are give by the eigenvectors of the covariance matrix

Main principal component is given by the eigenvector corresponding to the largest eigenvalue.

Eigenvalues & Eigenvectors

Vectors **x** having same direction as *A***x** are called *eigenvectors* of *A* (*A* is an n by n matrix).

In the equation *A***x**=λ**x**, λ is called an *eigenvalue* of *A*.

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} x \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 12 \\ 8 \end{pmatrix} = 4x \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

Eigenvalues & eigenvectors

$A\mathbf{x}=\lambda\mathbf{x} \Leftrightarrow (A-\lambda I)\mathbf{x}=0$

How to calculate **x** and λ:

- Calculate $det(A-\lambda I)$, yields a polynomial (degree n)
- Determine roots to $det(A-\lambda I)=0$, roots are eigenvalues λ
- Solve $(A-\lambda I)\ \mathbf{x}=0$ for each λ to obtain eigenvectors **x**

Principal components

1. principal component (PC1)
   - The eigenvalue with the largest absolute value will indicate that the data have the largest variance along its eigenvector, the direction along which there is greatest variation
2. principal component (PC2)
   - the direction with maximum variation left in data, orthogonal to the 1. PC

In general, only few directions manage to capture most of the variability in the data.

Steps of PCA

Let $\overline{X}$ be the mean vector (taking the mean of all rows)

Adjust the original data by the mean

$X' = X - \overline{X}$

Compute the covariance matrix C of adjusted X

Find the eigenvectors and eigenvalues of C.

PCA Example

DATA:

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

ZERO MEAN DATA:

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

PCA Example

Calculate the covariance matrix

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

since the non-diagonal elements in this covariance matrix are positive, we should expect that both the x and y variable increase together.

PCA Example

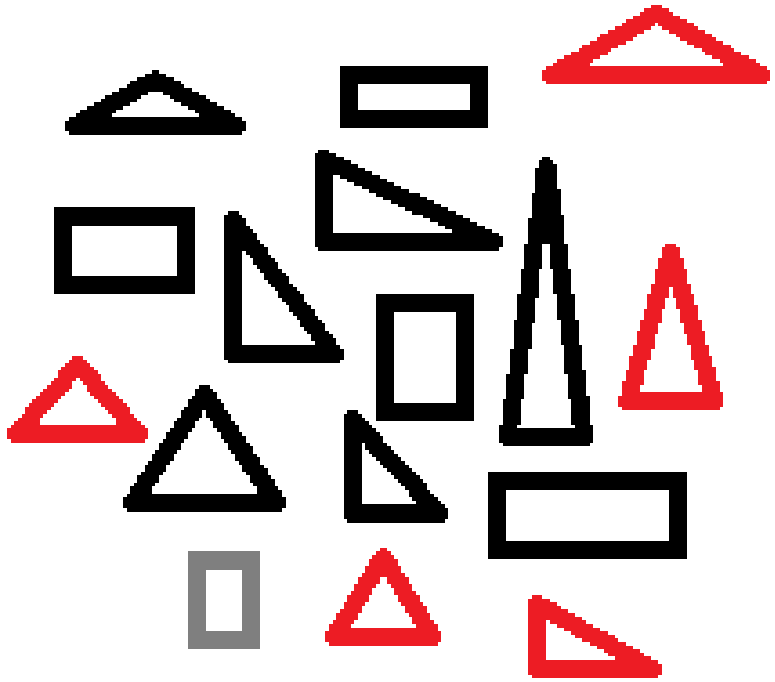
Calculate the eigenvectors and eigenvalues of the covariance matrix

$$\text{eigenvalues} = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$\text{eigenvectors} = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

Introduction to Neural Networks

Classification Problem



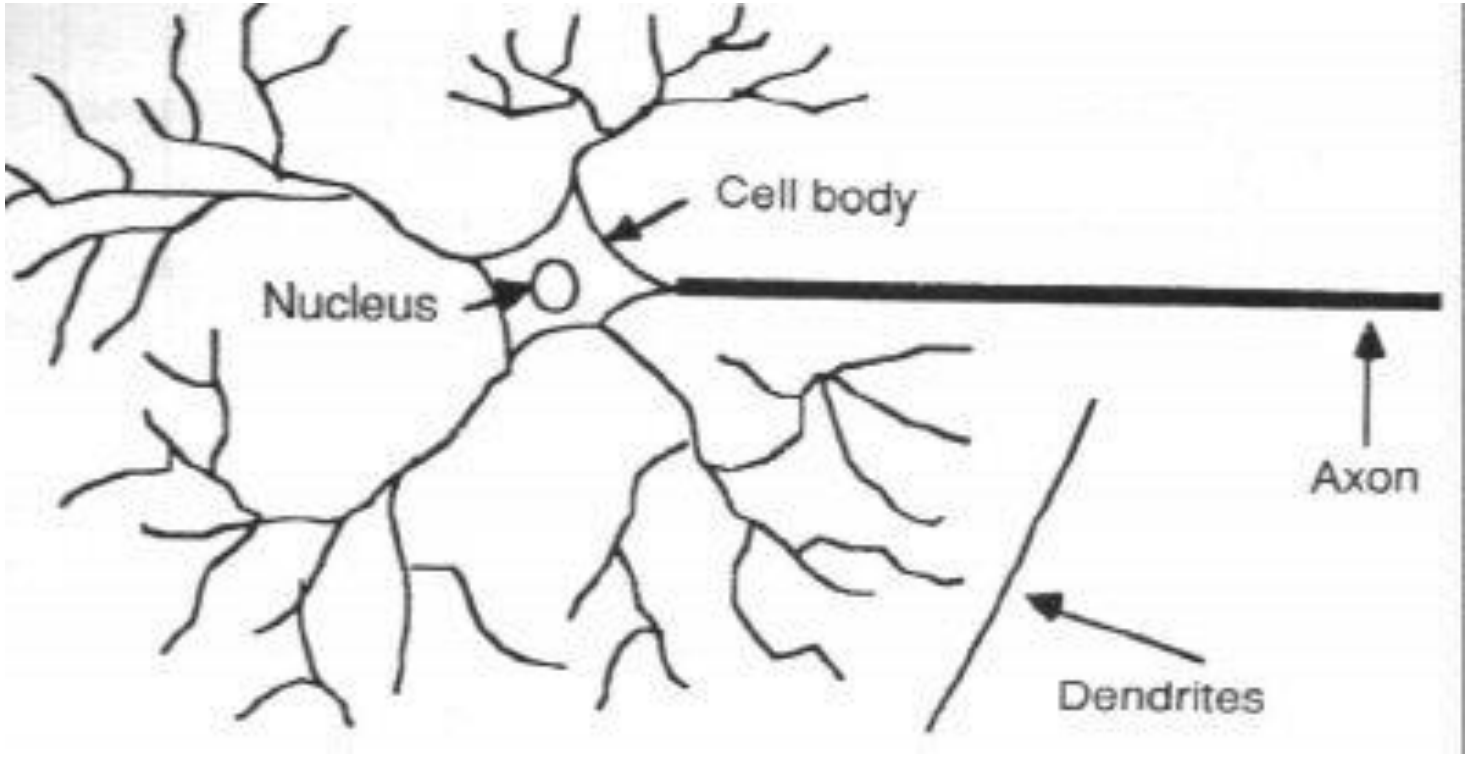
?

Motivation

Is it possible to develop a classifier similar to the biological nervous systems,

1. Which does not require explicit feature selection
2. Learns from data
3. Capable of processing information in the same way that the human brain does?

The Structure of the Human Brain



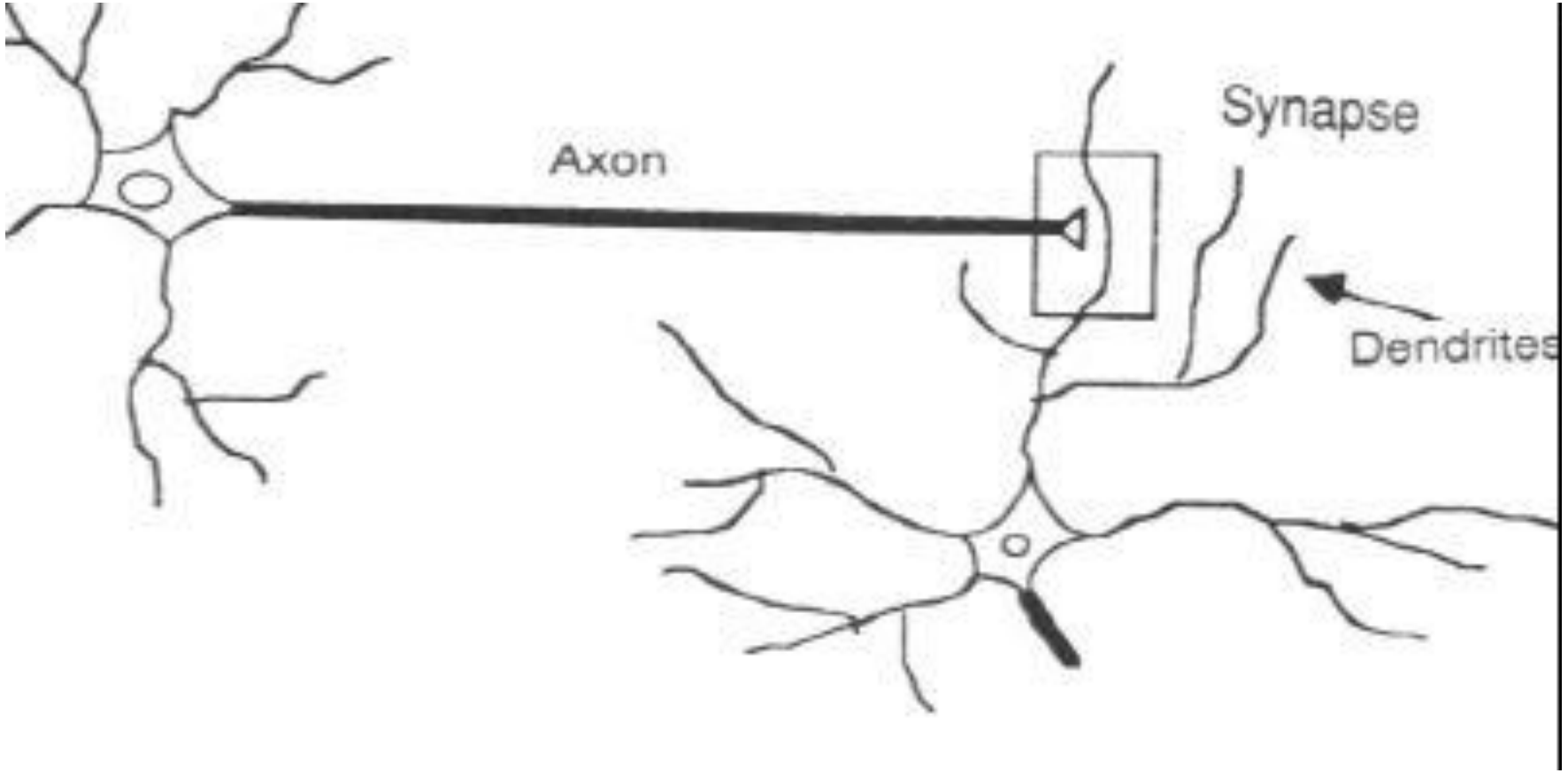
The Structure of the Human Brain

In the human brain, a typical neuron collects signals from others through a host of fine structures called *dendrites*.

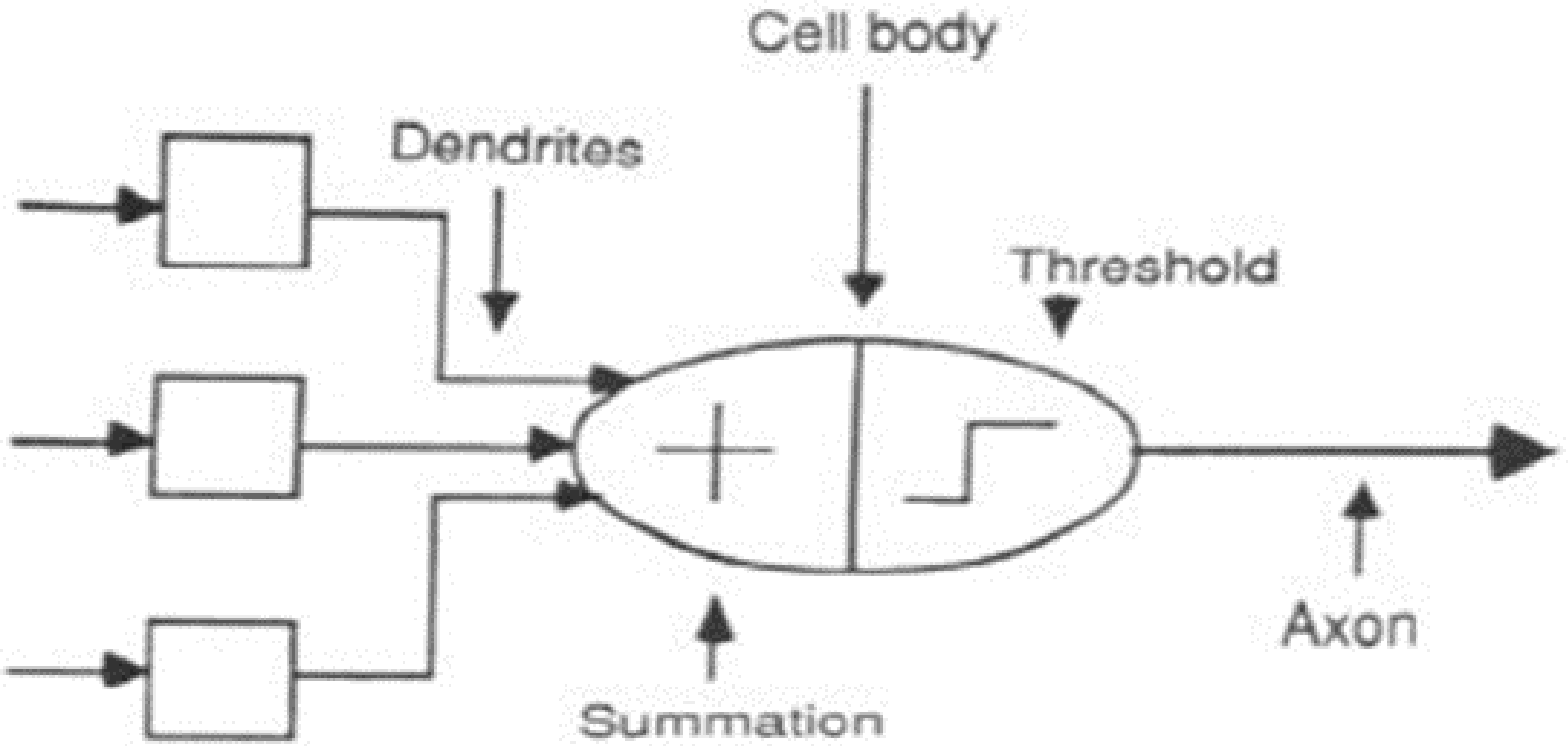
The neuron sends out spikes of electrical activity through a long, thin stand known as an *axon*, which splits into thousands of branches.

At the end of each branch, a structure called a *synapse* converts the activity from the axon into electrical effects that inhibit or excite activity in the connected neurons.

The Structure of the Human Brain



Neural Model



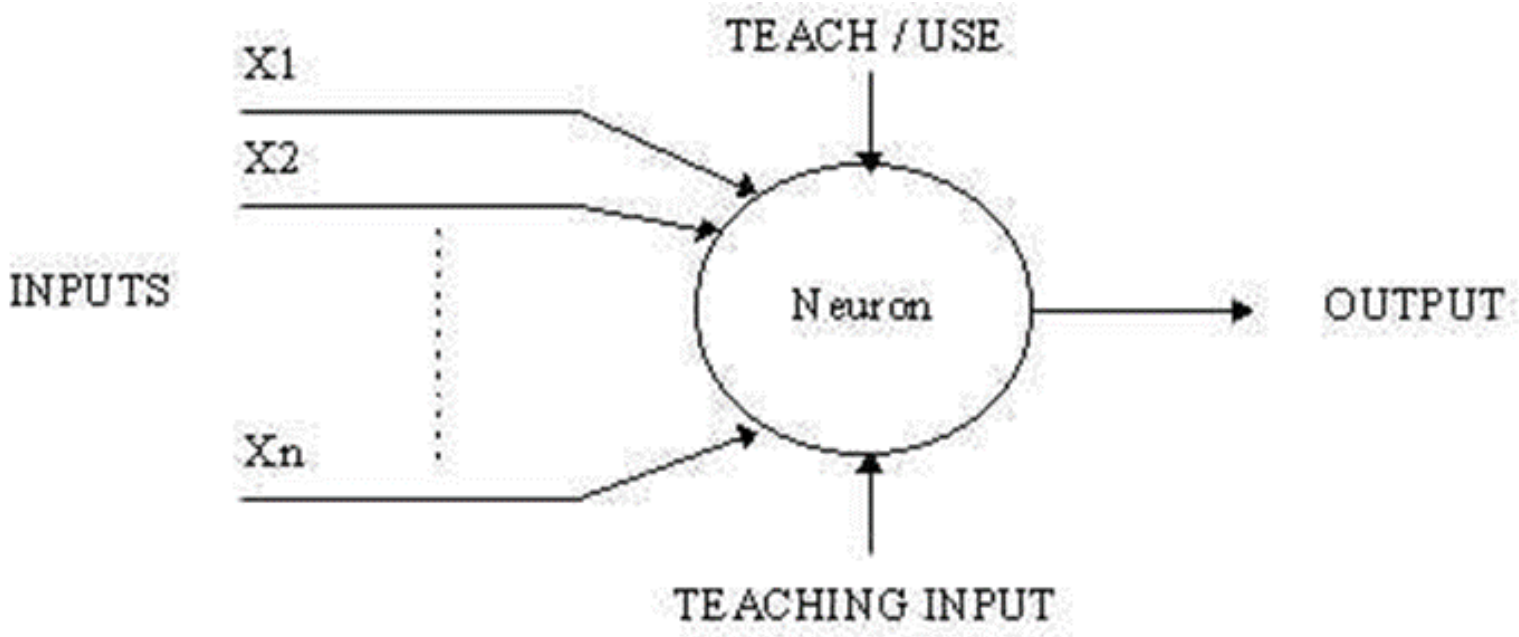
Neural Model

When a neuron receives sufficiently large excitatory input, it sends a spike of electrical activity down its axon.

Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes.

We conduct these neural networks by first trying to deduce the essential features of neurons and their interconnections.

An Artificial Neuron



An Artificial Neuron

An artificial neuron is a device with many inputs and one output.

The neuron has two modes of operation:

- the training mode and
- the using mode.

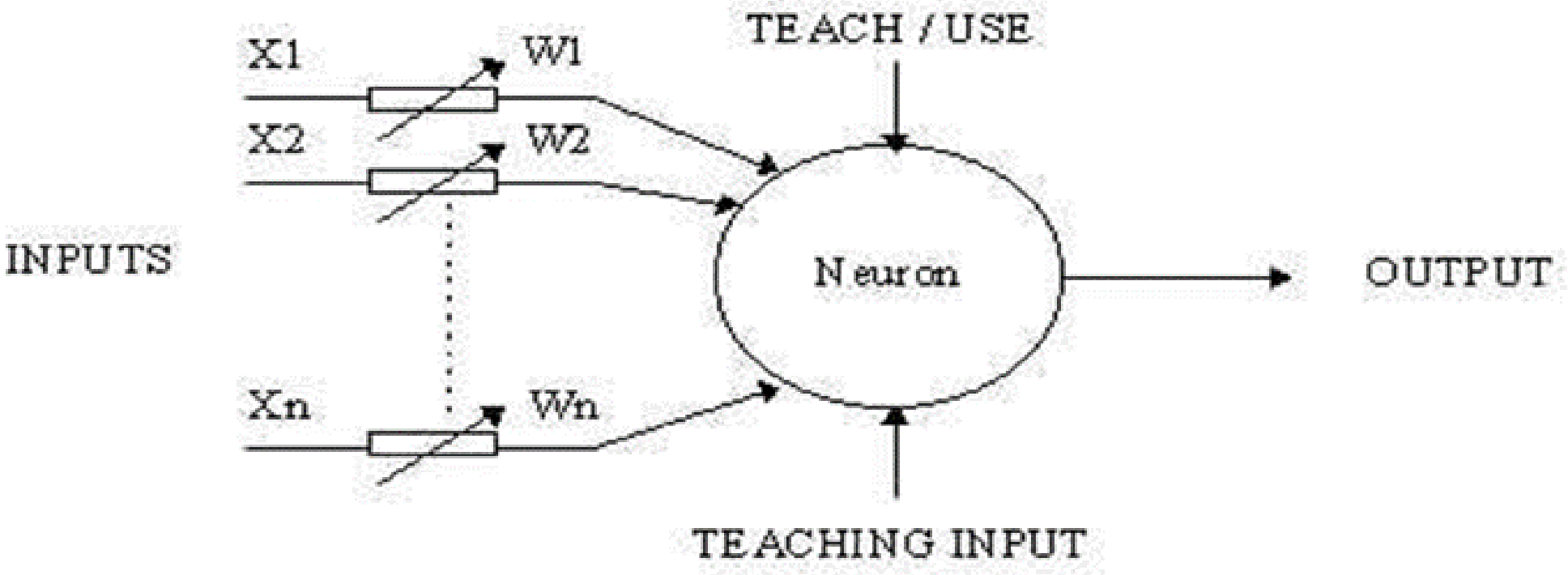
An Artificial Neuron

In the training mode, the neuron is trained to react (fire) to some particular input patterns.

In the using mode, when a taught pattern is detected at the input, its associated output is generated.

A rule named the firing rule is used to relate some output to every possible input pattern, (not only the ones on which the node was trained on previously).

More Sophisticated Neuron Model



More Sophisticated Neuron Model

A more sophisticated Neuron is know as the McCulloch and Pitts model (MCP).

The difference is that in the MCP model, the inputs are weighted and the effect that each input has at decision making, is dependent on the weight of the particular input.

The weight of the input is a number which is multiplied with the input to give the weighted input.

More Sophisticated Neuron Model

The MCP neuron has the ability to adapt to a particular situation by changing its weights and/or threshold.

Various algorithms exist that cause the neuron to 'adapt'; the most used ones are the Delta rule and the back error propagation.

Feed-Forward and Feedback Neural Networks

Feed-forward networks

- Feed-forward NNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer.
- Feed-forward NNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition.

Feed-Forward and Feedback Neural Networks

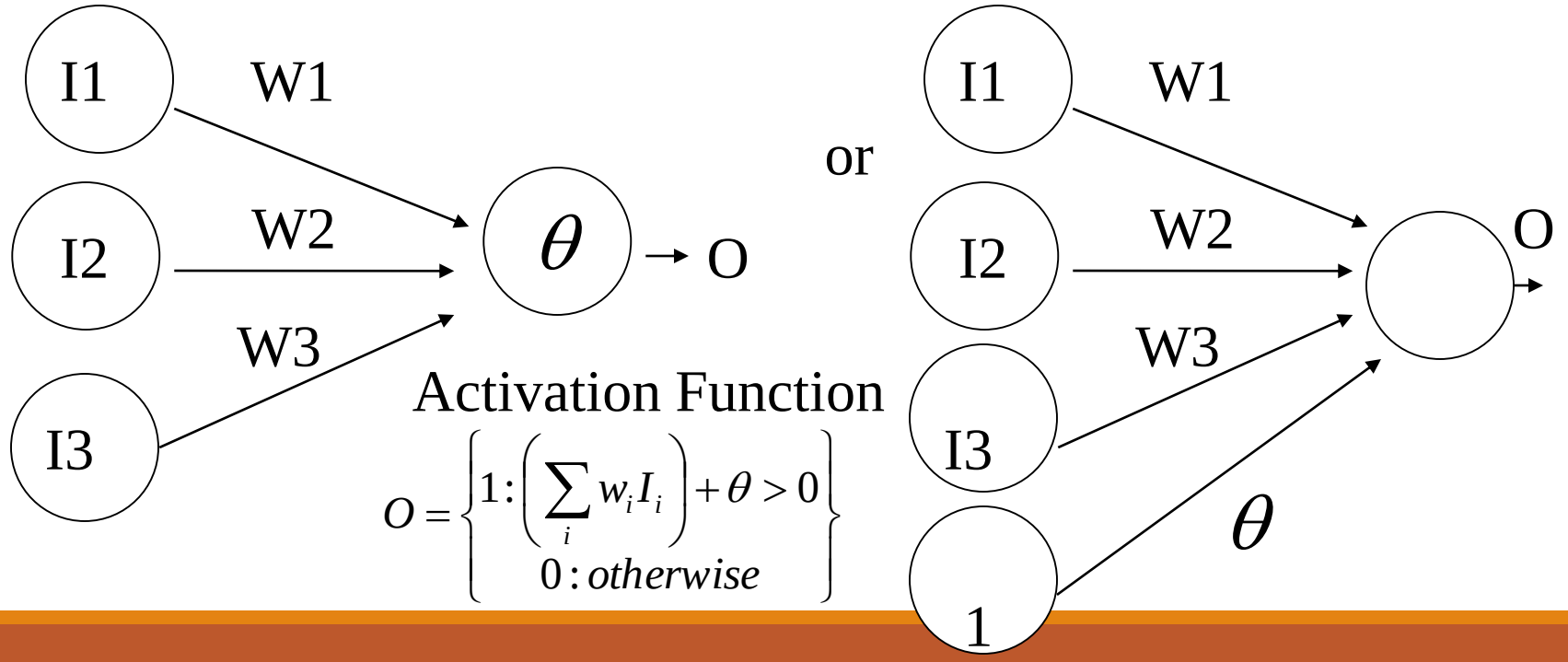
Feedback networks

- Feedback networks can have signals traveling in both directions by introducing loops in the network.
- Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point.
- They remain at the equilibrium point until the input changes and a new equilibrium needs to be found.

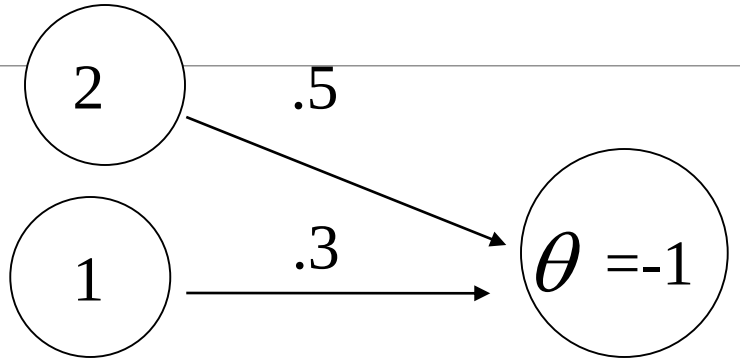
Perceptrons

Initial proposal of connectionist networks Rosenblatt, 50's and 60's

Essentially a linear discriminant composed of nodes, weights



Perceptron Example



$$2(0.5) + 1(0.3) + -1 = 0.3 \text{ , } O=1$$

Learning Procedure:

Randomly assign weights (between 0-1)

Present inputs from training data

Get output O, nudge weights to gives results toward our desired output T

Repeat; stop when no errors, or enough epochs completed

Perception Training

$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

$$\Delta w_i(t) = (T - O) I_i$$

Weights include Threshold. T=Desired, O=Actual output.

Example: T=0, O=1, W1=0.5, W2=0.3, I1=2, I2=1,Theta=-1

$$w_1(t+1) = 0.5 + (0-1)(2) = -1.5$$
$$w_2(t+1) = 0.3 + (0-1)(1) = -0.7$$
$$w_\theta(t+1) = -1 + (0-1)(1) = -2$$

If we present this input again, we'd output 0 instead

How might you use a perceptron network?

This (and other networks) are generally used to learn how to make classifications

Say you have collected some data regarding the diagnosis of patients with heart disease

- Age, Sex, Chest Pain Type, Resting BPS, Cholesterol, …, Diagnosis (<50% diameter narrowing, >50% diameter narrowing)

- `67,1,4,120,229,…, 1`
- `37,1,3,130,250,… ,0`
- `41,0,2,130,204,… ,0`

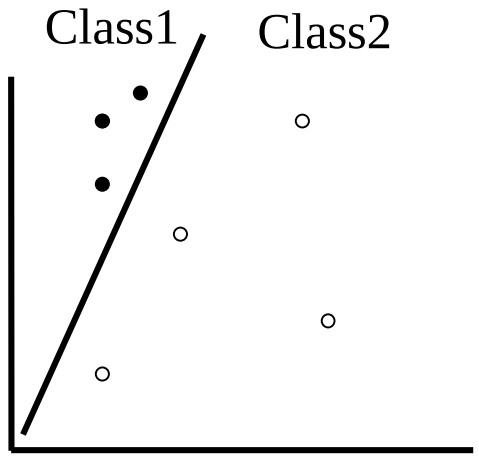
Train network to predict heart disease of new patient

Perceptrons

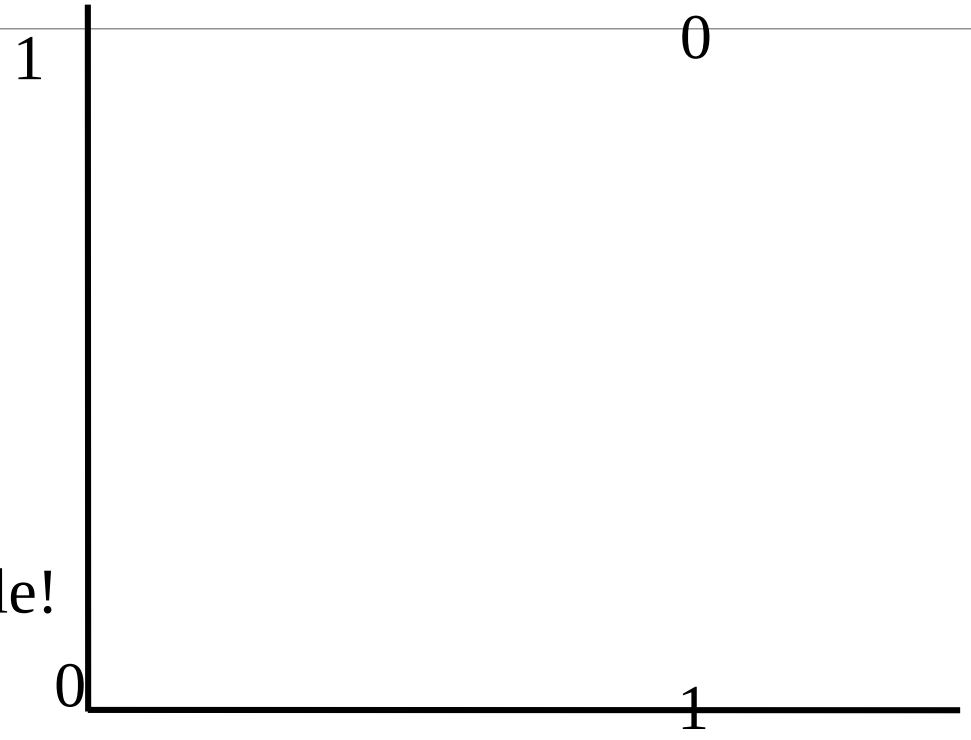
Can add learning rate to speed up the learning process; just multiply in with delta computation

Essentially a linear discriminant

Perceptron theorem: If a linear discriminant exists that can separate the classes without error, the training procedure is guaranteed to find that line or plane.



Exclusive Or (XOR) Problem

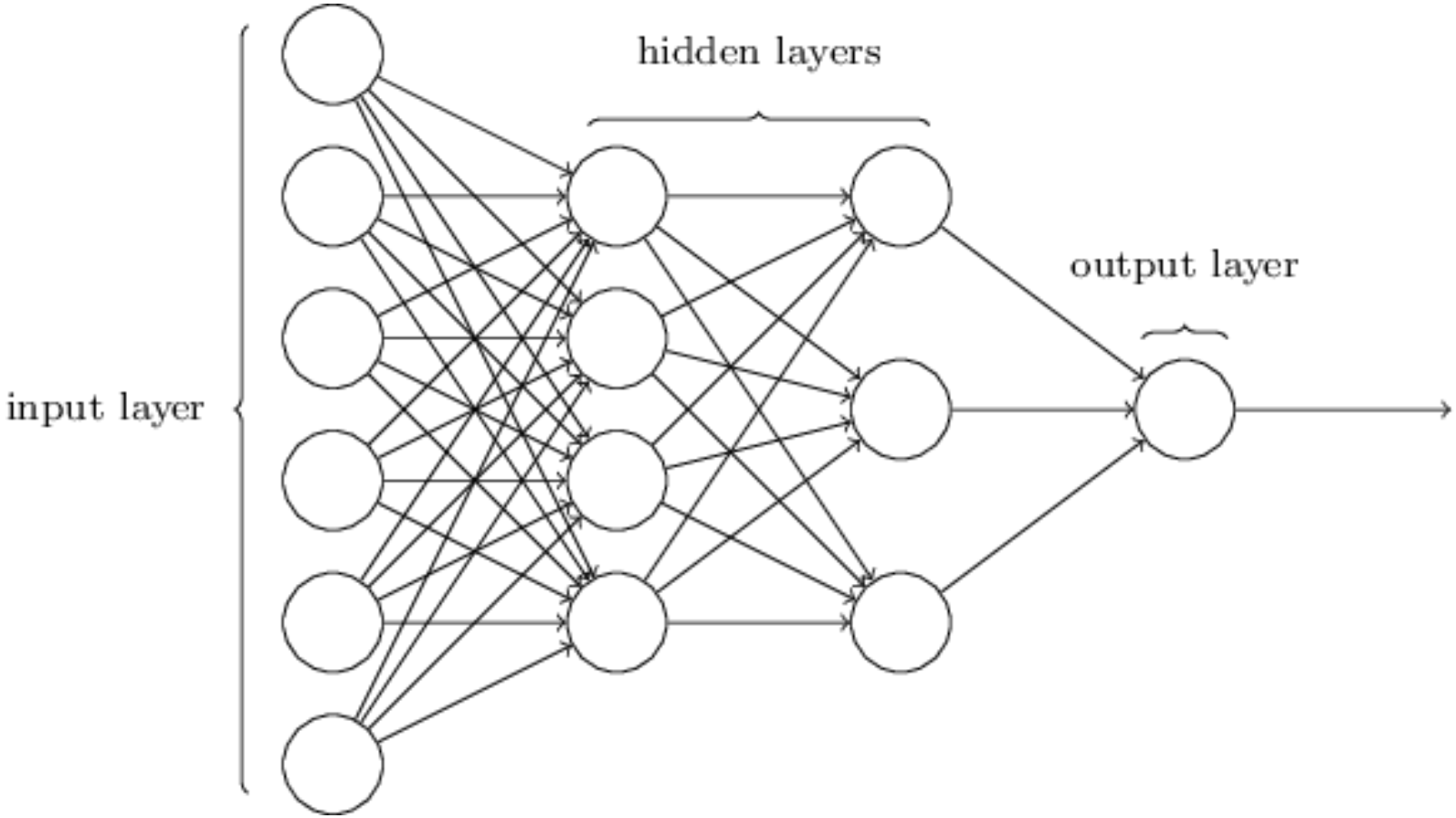


Input: 0,0 Output: 0
Input: 0,1 Output: 1
Input: 1,0 Output: 1
Input: 1,1 Output: 0

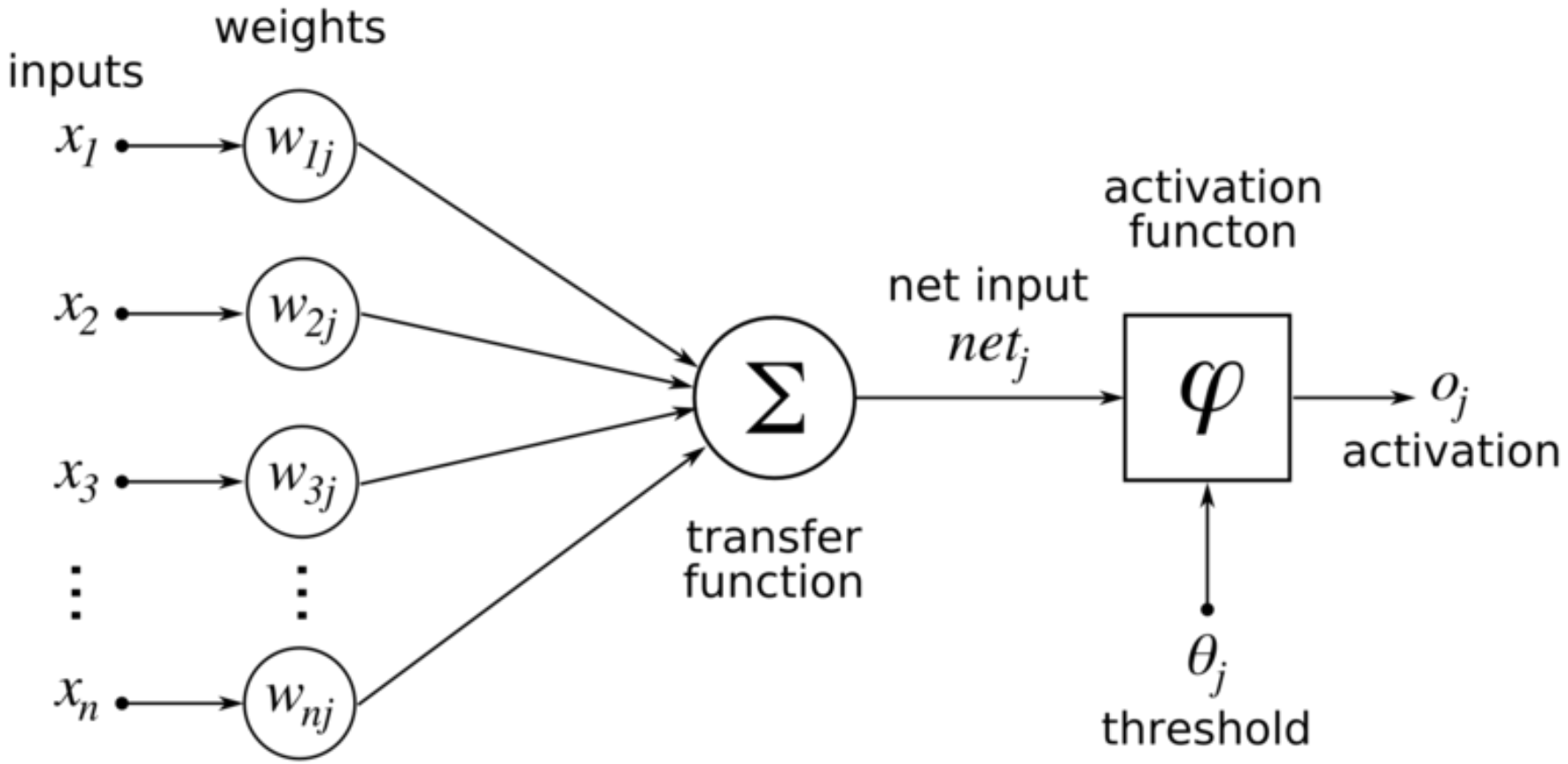
XOR Problem: Not Linearly Separable!

We could however construct multiple layers of perceptrons to get around this problem. A typical multi-layered system minimizes LMS Error,

Multilayer Perceptron



Structure of a Neuron



LMS Learning

LMS (Least Mean Square) learning systems, are more general than the previous perceptron learning rule.
In LMS. The total error measured over all training samples is minimized

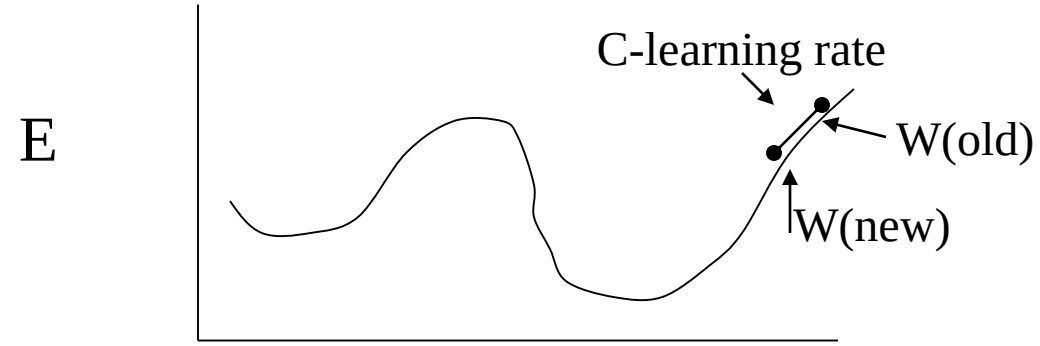
LMS Learning

$$O = \sum_i w_i I_i + \theta$$

$$LMS = \frac{1}{2}\sum_P \left(T_P - O_P\right)^2$$

E.g. if we have two patterns and
T1=1, O1=0.8, T2=0, O2=0.5 then D=(0.5)[(1-0.8)^2+(0-0.5)^2]=.145

We want to minimize the LMS:



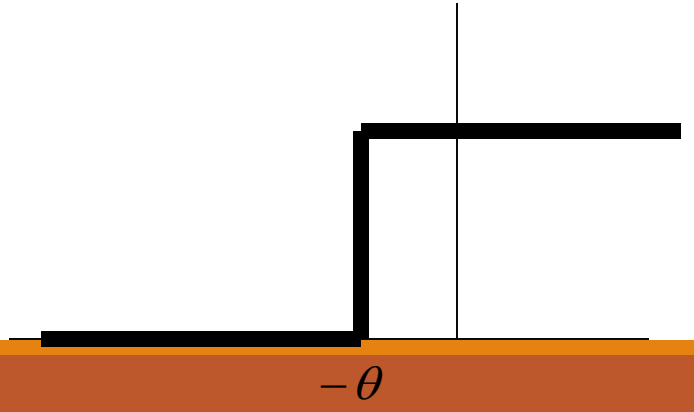
Activation Function

To apply the LMS learning rule, also known as the delta rule, we need a differentiable activation function.

$$\Delta w_k = cI_k\left(T_j - O_j\right)f'\left(ActivationFunction\right)$$

Old:

$$O = \begin{Bmatrix} 1 : \sum_i w_i I_i + \theta > 0 \\ 0 : otherwise \end{Bmatrix}$$



New:

$$O = \frac{1}{1 + e^{-\sum_i w_i I_i + \Theta}}$$