

Dimensionality Reduction: Principal Component Analysis (PCA), Implementation and demonstration. **Artificial Neural Networks:** Introduction, Neural network representation, appropriate problems for neural network learning, Multilayer networks and the back propagation algorithm.

Introduction

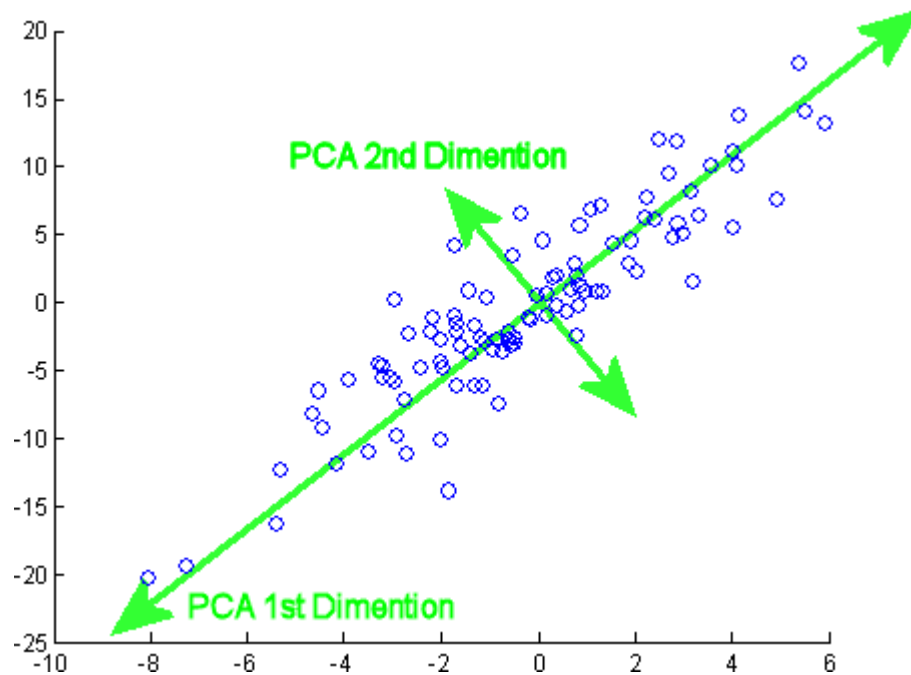
- ⇒ In machine learning problems there often involves tens of thousands of features for each training instance. This can be a problem as it makes our training extremely slow and prone to *overfitting* (refer to *overfitting* section). This problem is commonly referred to as **the curse of dimensionality**.
- ⇒ Because of the issues associated with **the curse of dimensionality**, it is necessary to reduce the number of features/dimensions considerably to help increase our model's performance and enables us to arrive at an optimal solution for our machine learning model.
- ⇒ By reducing the dimensions of our training set, we can increase the speed of our training, and reduce our dataset down to two or three dimensions, making it easier to perform data visualizations (clustering, patterns).
- ⇒ The various methods in performing Dimensionality Reduction (Projection & Manifold Learning) and a Dimensional Reduction algorithm known as Principal Component Analysis (PCA).
- ⇒ As a general rule, if you are given a data set which has more than 3 dimensions, it is often the case that dimensionality reduction algorithms will be beneficial in enhancing the performance of your model.

Principal Component Analysis (PCA)

- ⇒ PCA is fundamentally a simple dimensionality reduction technique that transforms the columns of a dataset into a new set feature. It does this by finding a new set of directions (like X and Y axes) that explain the maximum variability in the data.
 - ⇒ This new system coordinate axes is called Principal Components (PCs). The projections of the original data on the new set of coordinate axes (PCs) serves as the new transformed dataset.
- But why even compute the PCs?
- ⇒ Because, the information contained in a dataset column (variable) is the amount of variance it contains. It will effectively explain the vast information present in the original data into a fewer number of columns.
 - ⇒ The objective of Principle Component Analysis is simple, identify a hyperplane that lies closest to the data points, and project the data onto it.

Preservation our Variance

- ⇒ When mapping our original data set in a lower-dimensional hyperplane, we first must determine the right hyperplane. In Principal Component Analysis, the hyperplane, in most cases is determined by the axis through our data set, which preserves the maximum amount of variance.



Why use PCA?

⇒ Practically PCA is used for two reasons:

1. **Dimensionality Reduction:** The information distributed across a large number of columns is transformed into principal components (PC) such that the first few PCs can explain a sizeable chunk of the total information (variance). These PCs can be used as explanatory variables in Machine Learning models.
2. **Visualize Classes:** Visualizing the separation of classes (or clusters) is hard for data with more than 3 dimensions (features). With the first two PCs itself, it's usually possible to see a clear separation.

Is PCA a feature selection technique?

⇒ It is not a feature selection technique. Rather, it is a feature combination technique. Because each PC is a weighted additive combination of all the columns in the original dataset. More on this when you implement it in the next section.

⇒ However, the PCs are formed in such a way that the first Principal Component (PC1) explains more variance in original data compared to PC2. Likewise, PC2 explains more than PC3, and so on.

Principal Components

⇒ There are various techniques we can use to find our principle components. However, the most common method in finding our principle components is known as **Singular Value Decomposition (SVD)**. It turns out that if you perform a basic matrix factorisation technique known as Singular Value Decomposition (**SVD**), where you decompose the training set matrix

X, into the dot product of three matrices, our third matrix V^* actually contains all the principal components we are looking for.

$$M = U \cdot \Sigma \cdot V^*$$

Where

M = $m \times n$ matrix whose entries come from a field, where either field consists of real numbers or complex numbers.

U = $m \times m$ unitary matrix. (left singular vector)

Σ = $m \times n$ diagonal matrix with non-negative real numbers.

V = $n \times n$ unitary matrix. (Right singular vector)

V^* = conjugate transpose of the $n \times n$ unitary matrix.

Building PCA with Scikit-learn

⇒ Using scikit-learn package, the implementation of PCA is quite straight forward. The module named `sklearn.decomposition` provides the PCA object which can simply fit and transform the data into Principal components.

- Load packages
- Import data
- Build the principle components

Artificial Neural Networks: Introduction

Introduction to Artificial Neural Networks

- **What is an Artificial Neural Network ?**

- It is a computational system inspired by the

- Structure

- Processing Method

- Learning Ability

- of a biological brain

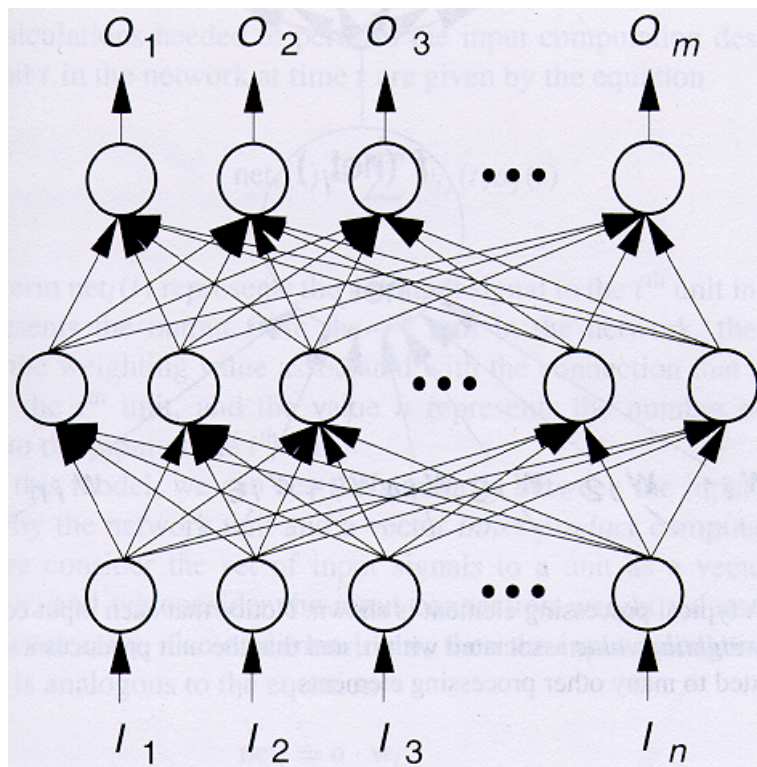
- Characteristics of Artificial Neural Networks

- A large number of very simple processing neuron-like processing elements

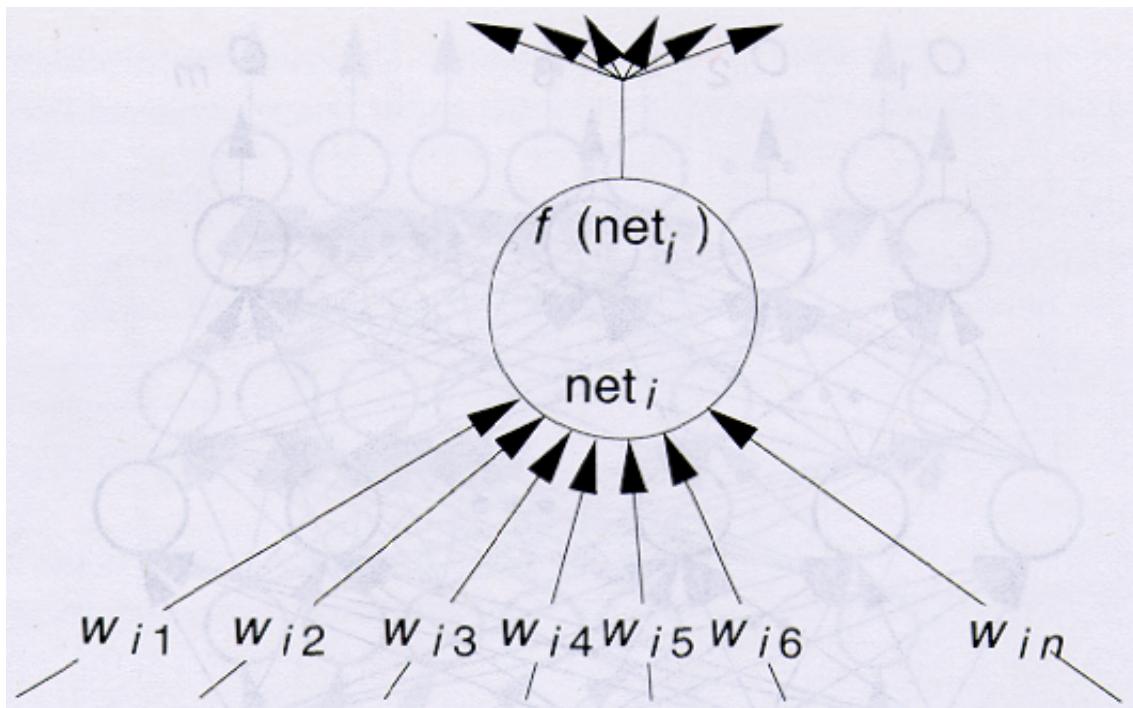
- A large number of weighted connections between the elements

- Distributed representation of knowledge over the connections

- Knowledge is acquired by network through a learning process



- **Why Artificial Neural Networks ?**
 - Massive Parallelism
 - Distributed representation
 - Learning ability
 - Generalization ability
 - Fault tolerance
- **Elements of Artificial Neural Networks**
 - Processing Units
 - Topology
 - Learning Algorithm
- **Processing Units**

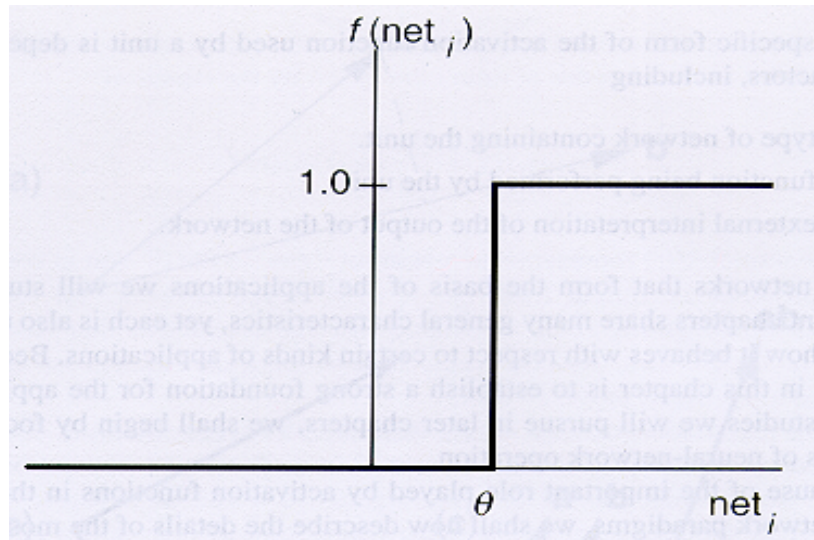


- 3 -

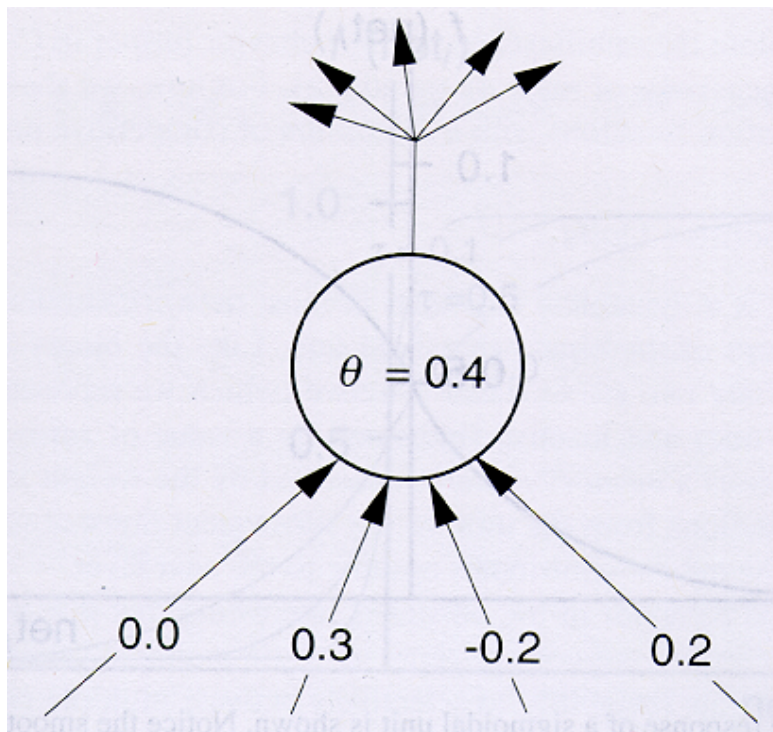
$$\text{Node input: } net_i = \sum_j w_{ij} I_j$$

$$\text{Node Output: } O_i = f(net_i)$$

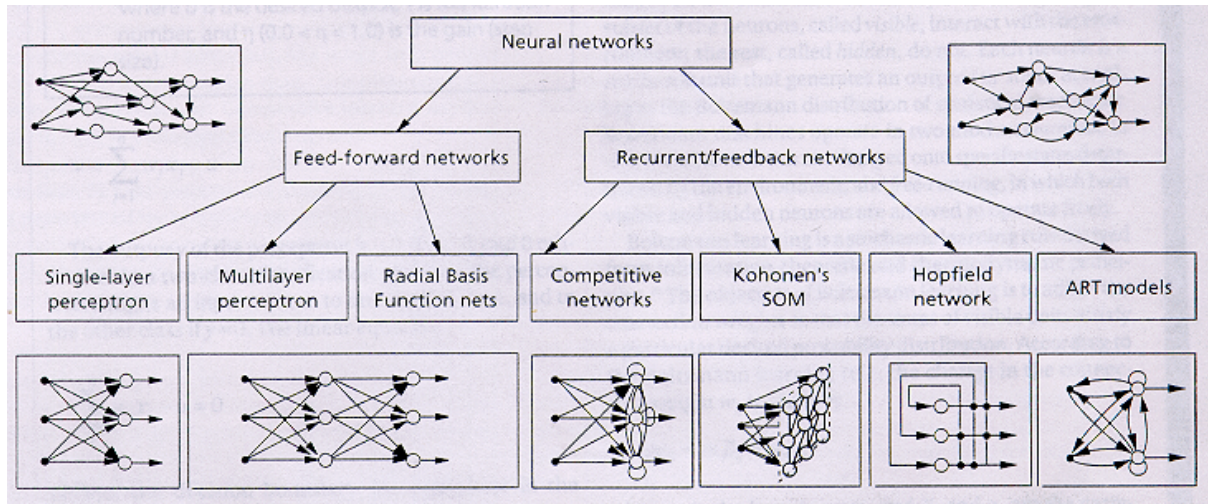
- **Activation Function**



- An example



- **Topology**



- **Learning**

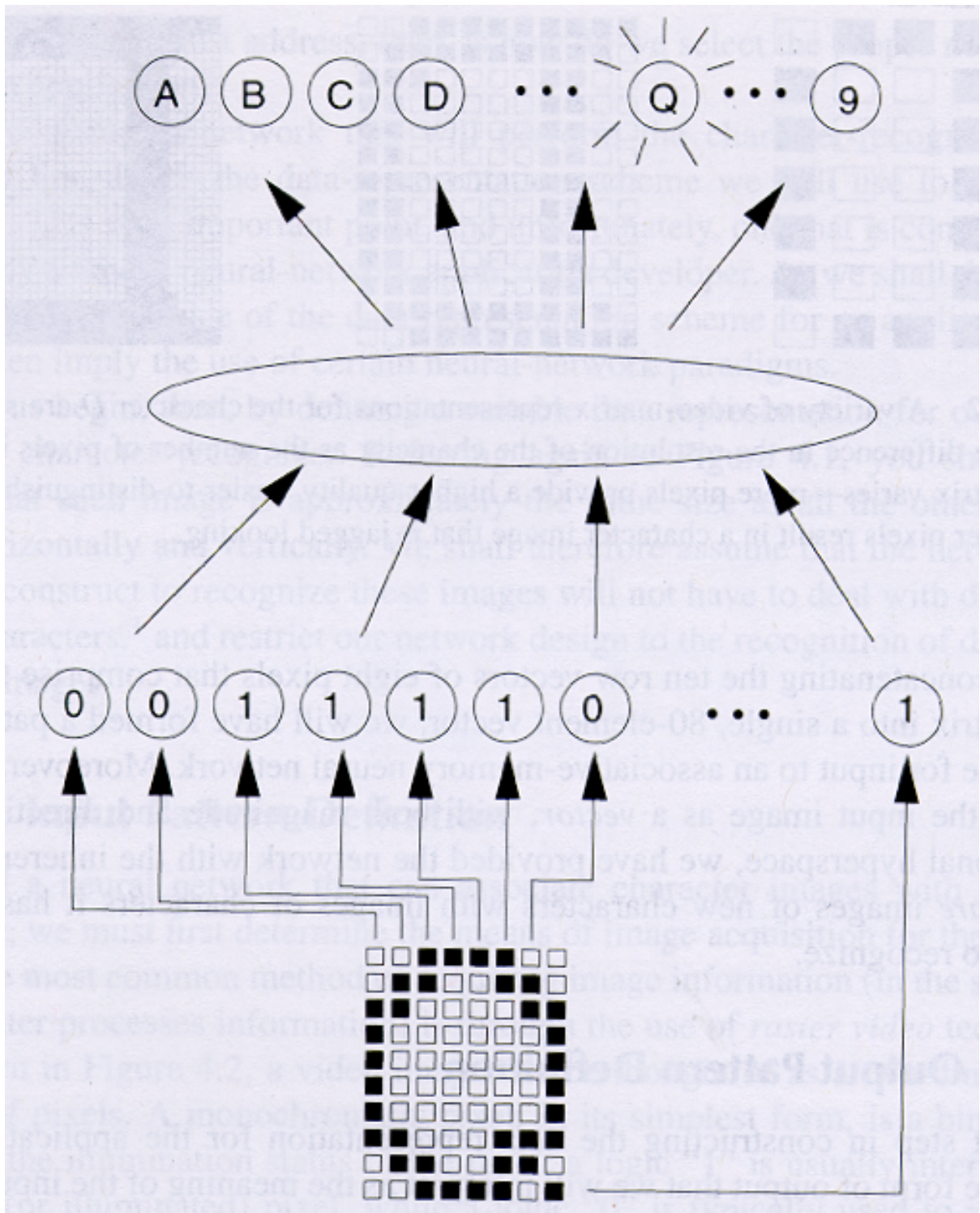
- Learn the connection weights from a set of training examples
- Different network architectures required different learning algorithms

Supervised Learning

The network is provided with a correct answer (output) for every input pattern

Weights are determined to allow the network to produce answers as close as possible to the known correct answers

The *back-propagation* algorithm belongs into this category



Unsupervised Learning

Does not require a correct answer associated with each input pattern in the training set

Explores the underlying structure in the data, or correlations between patterns in the data, and organizes patterns into categories from these correlations

The *Kohonen algorithm* belongs into this category

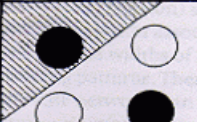
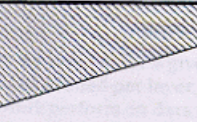
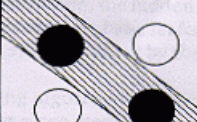
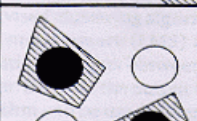
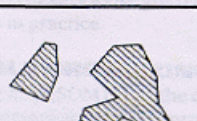
Hybrid Learning

Combines supervised and unsupervised learning

Part of the weights are determined through supervised learning and the others are obtained through unsupervised learning

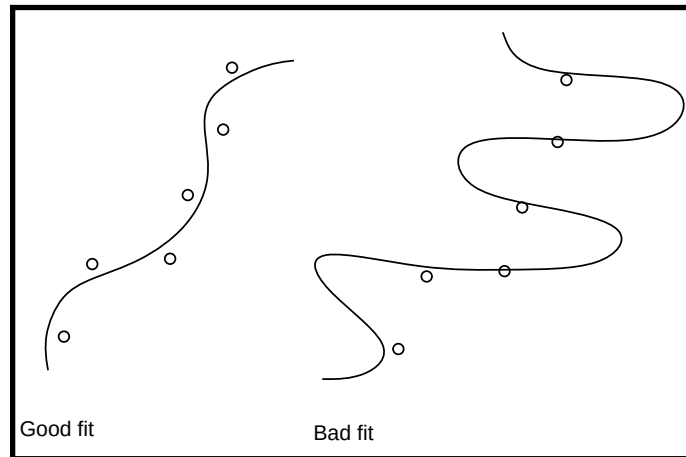
- **Computational Properties**

A single hidden layer feed-forward network with arbitrary sigmoid hidden layer activation functions can approximate arbitrarily well an arbitrary mapping from one finite dimensional space to another

Structure	Description of decision regions	Exclusive-OR problem	Classes with meshed regions	General region shapes
 Single layer	Half plane bounded by hyperplane			
 Two layer	Arbitrary (complexity limited by number of hidden units)			
 Three layer	Arbitrary (complexity limited by number of hidden units)			

- **Practical Issues**

- Generalization vs Memorization



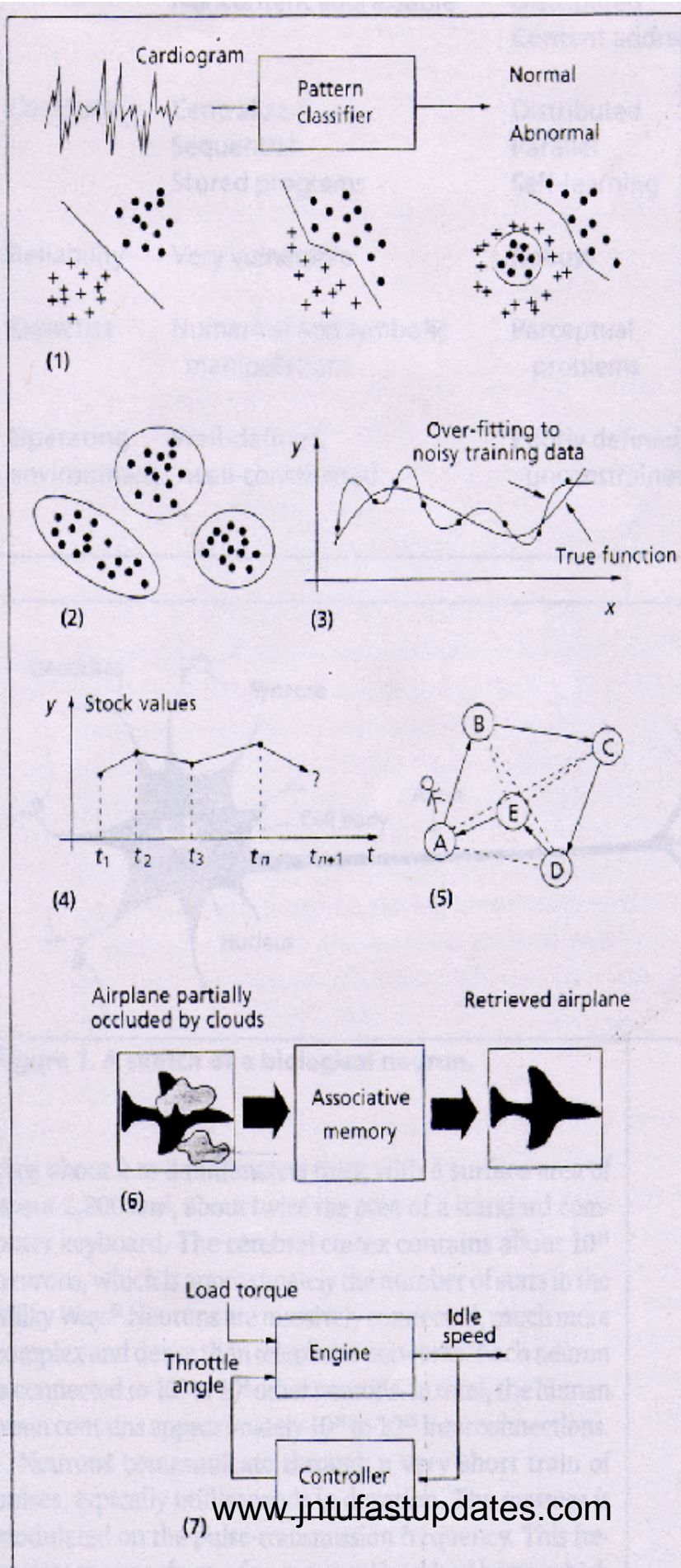
How to choose the network size (free parameters)

How many training examples

When to stop training

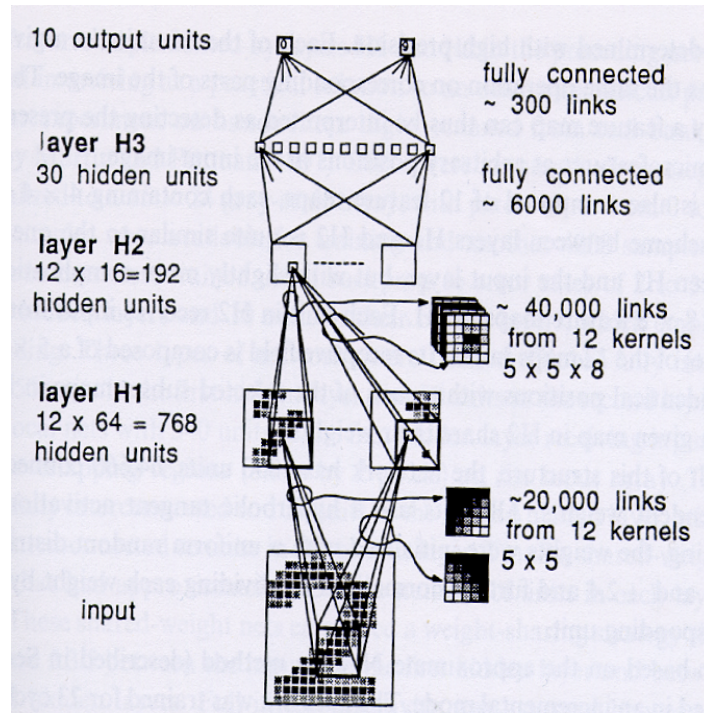
- **Applications**

- Pattern Classification
- Clustering/Categorization
- Function approximation
- Prediction/Forecasting
- Optimization
- Content-addressable Memory
- Control



- **Two Successful Applications**

- Zipcode Recognition



- Text to voice translation (NeTtalk)

