Artificial Intelligence (AI)

Lab Sheet No: 5

Introduction

Constraint An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

Why Use Artificial Neural Networks?

Artificial Neural Networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained Artificial Neural Network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions.

Other advantages include:

1. *Adaptive learning:* An ability to learn how to do tasks based on the data given for training or initial experience.

2. *Self-Organization:* An ANN can create its own organization or representation of the information it receives during learning time.

3. *Real Time Operation:* ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

4. *Fault Tolerance via Redundant Information Coding:* Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

A Simple 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. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.

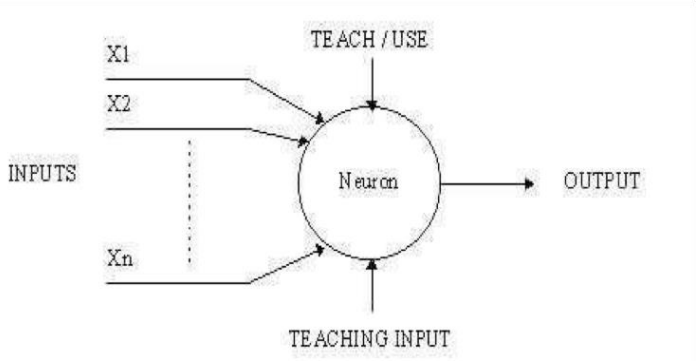


Figure 1.1: *A simple neuron*

Network layers

The commonest type of Artificial Neural Network consists of three groups, or layers, of units: a layer of "**input**" units is connected to a layer of "**hidden**" units, which is connected to a layer of "**output**" units. (See figure 1.2)

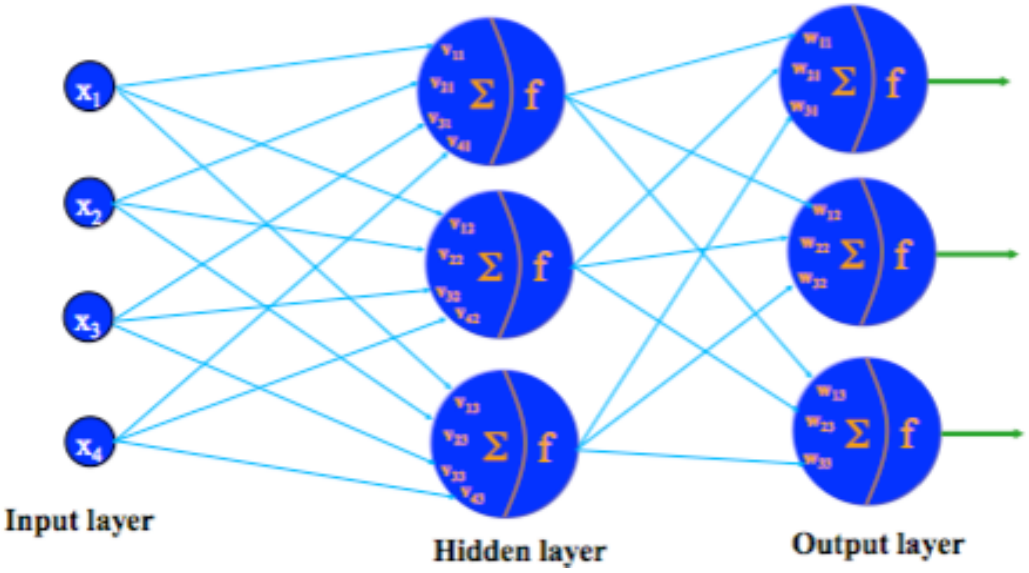


Figure 1.2: *A Simple Feed Forward Network*

The activity of the input units represents the raw information that is fed into the network. The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units.The behavior of the output units depends on the activity of the hidden units and the eights between the hidden and output units.

This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents.

We also distinguish single-layer and multi-layer architectures. The single-layer organization, in which all units are connected to one another, constitutes the most general case and is of more potential computational power than hierarchically structured multi-layer organizations. In multi-layer networks, units are often numbered by layer, instead of following a global

numbering.

Perceptrons

The most influential work on neural nets in the 60's went under the heading of 'perceptrons' a term coined by Frank Rosenblatt. The perceptron (See figure 1.3) turns out to be an MCP model (neuron with weighted inputs) with some additional, fixed, pre-processing. Units labeled A1, A2, Aj , Ap are called association units and their task is to extract specific, localized featured from the input images. Perceptrons mimic the basic idea behind the mammalian visual system. They were mainly used in pattern recognition even though their capabilities extended a lot more.

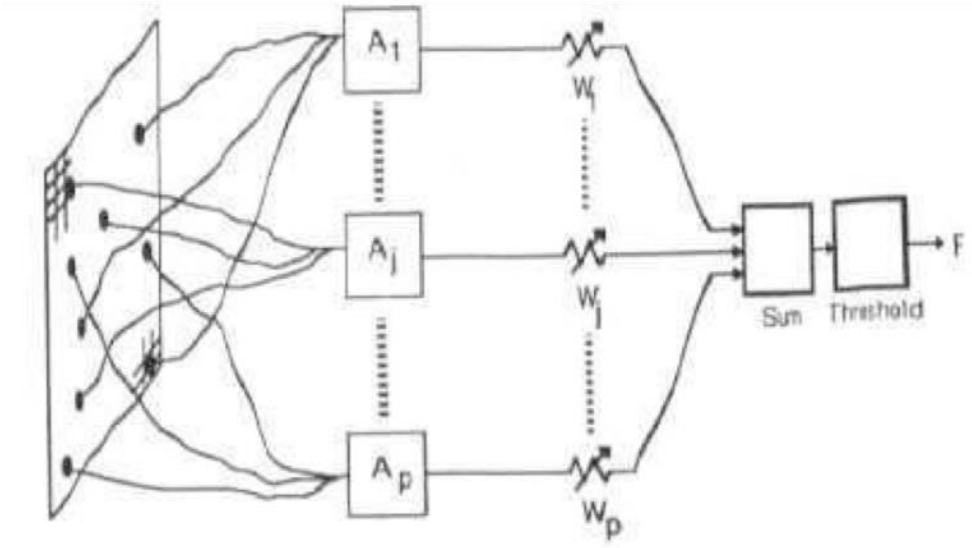


Figure 1.3: *A Perceptron*

Transfer Functions

The behavior of an ANN depends on both the weights and the input-output function (transfer function) that is specified for the units. This function typically falls into one of three categories:

- For ***linear*** (or ramp) the output activity is proportional to the total weighted output.
- For ***threshold units***, the output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value.
- For ***sigmoid units***, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units, but all three must be considered rough approximations.

To make an Artificial Neural Network that performs some specific task, we must choose how the units are connected to one another and we must set the weights on the connections appropriately. The connections determine whether it is possible for one unit to influence another. The weights specify the strength of the influence.

We can teach a network to perform a particular task by using the following procedure:

1. We present the network with training examples, which consist of a pattern of activities for the input units together with the desired pattern of activities for the output units.
2. We determine how closely the actual output of the network matches the desired output.
3. We change the weight of each connection so that the network produces a better approximation of the desired output.

Algorithm for Adaline Learning

The training is done using the Delta Rule which is also known as the least mean squares (LMS) or Widrow-Hoff rule

Step 0: initialize the weights to small random values and select a learning rate, α

Step 1: for each input vector s, with target output, t set the inputs to s

Step 2: compute the neuron inputs

$$y\_in = b + \sum x_i w_i$$

Step 3: use the delta rule to update the bias and weights

$$b(new) = b(old) + a(t - y\_in)$$
$$wi(new) = wi(old) + a(t - y\_in)xi$$

Step 4: stop if the largest weight change across all the training samples is less than a specified tolerance, otherwise cycle through the training set again

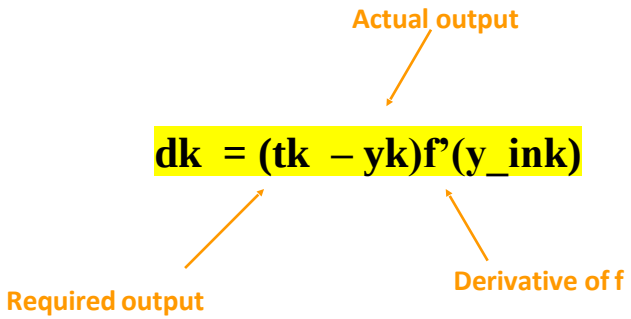
Algorithm for Backpropagation

The backpropagation algorithm provides a computational efficient method for training multi-layer networks

Step 0: Initialize the weights to small random values

Step 1: Feed the training sample through the network and determine the final output

Step 2: Compute the error for each output unit, for unit k it is:



Step 3: Propagate the delta terms (errors) back through the weights of the hidden units where the delta input for the jth hidden unit is:

$$\delta\_in_j = \Sigma \delta_\kappa w_{jk}$$

$$\delta_j = \delta\_in_j f'(z\_in_j)$$

Step 4: Calculate the weight correction term for each output unit, for unit k it is:

$$\Delta w_{jk} = \alpha \delta_\kappa z_j$$

Step 5: Calculate the weight correction term for the hidden units:

$$\Delta w_{ij} = \alpha \delta_j x_i$$

Step 6: Update the weights:

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij}$$

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$$

Step 7: Test for stopping (maximum epoch, small changes, etc)

Implementation of Logic Functions

In this practical, we will learn how basic logic functions can be implemented and trained. The primitive procedure is explained by implementing a 2-input AND gate.

X : (column for inputs)

Zi : (column for true output)

Assignment (1): Design **a** McCulloch-Pitts neural network which behaves as AND function using Adaline learning. Consider unipolar case. Perform analysis by varying NN parameters.

Assignment (2): Similarly develop a McCulloch-Pitts neural net for OR, NAND and NOR gate and draw neural nets.

Assignment (3): Perform test for bipolar model as well.

Assignment (4): Implement McCulloch-Pitts neural network model for XOR and give all the formula you used in the implementation. Draw the MLPs used for the implementation of above functions.

Assignment (5): Implement MLP model for XOR by using backpropagation algorithm

NOTE: The XOR function could not be solved by a single layer perceptron network