Classification by Backpropagation

- Backpropagation: A **neural network** learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a **weight** associated with it
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units

Neural Network as a Classifier

- Weakness

o Long training time

o Require a number of parameters typically best determined empirically, e.g., the network topology or ``structure."

o Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of ``hidden units" in the network

- Strength

o High tolerance to noisy data

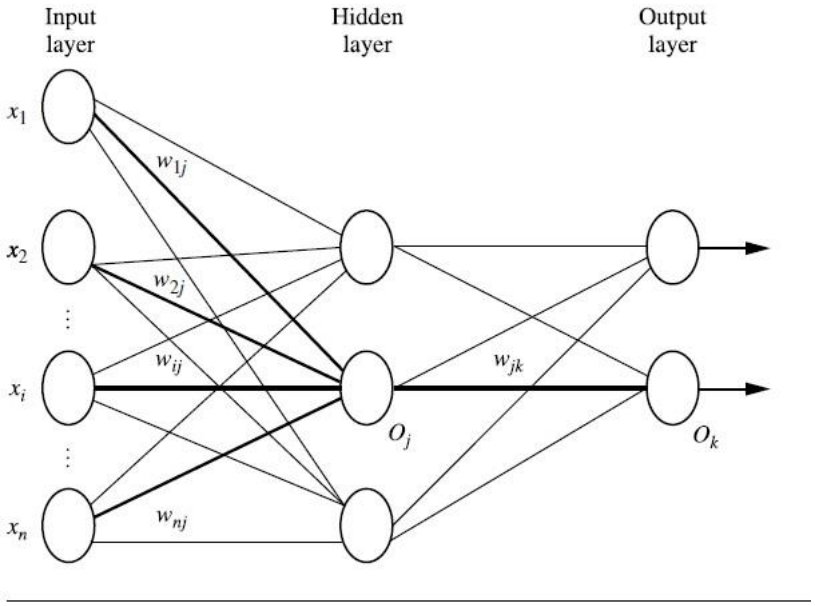
o Ability to classify untrained patterns

o Well-suited for continuous-valued inputs and outputs o Successful on a wide array of real-world data

o Algorithms are inherently parallel

o Techniques have recently been developed for the extraction of rules from trained neural networks

A Multilayer Feed-Forward Neural Network



Multilayer feed-forward neural network.

- The inputs to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the input layer
- They are then weighted and fed simultaneously to a hidden layer
- The number of hidden layers is arbitrary, although usually only one
- The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network's prediction
- The network is feed-forward in that none of the weights cycles back to an input unit or to an output unit of a previous layer
- From a statistical point of view, networks perform nonlinear regression: Given enough hidden units and enough training samples, they can closely approximate any function

Defining a network topology:

- Specifying the number of units in the input layer, the number of hidden layers (if more than one), the number of units in each hidden layer, and the number of units in the output layer.
- input values are normalized so as to fall between 0.0 and 1.0.
- if an attribute *A* has three possible or known values, namely f*a*0, *a*1, *a*2g, then we may assign three input units to represent *A*. That is, we may have, say, *I*0, *I*1, *I*2 as input units.

- Each unit is initialized to 0. If *A* D *a*0, then *I*0 is set to 1 and the rest are 0. If *A* D *a*1, then *I*1 is set to 1 and the rest are 0, and so on.

Backpropagation Algorithm:

Neural network learning for classification or numeric prediction, using the backpropagation algorithm.

Input:

- *D*, a data set consisting of the training tuples and their associated target values;
- *l*, the learning rate;
- *network*, a multilayer feed-forward network.

Output: A trained neural network.

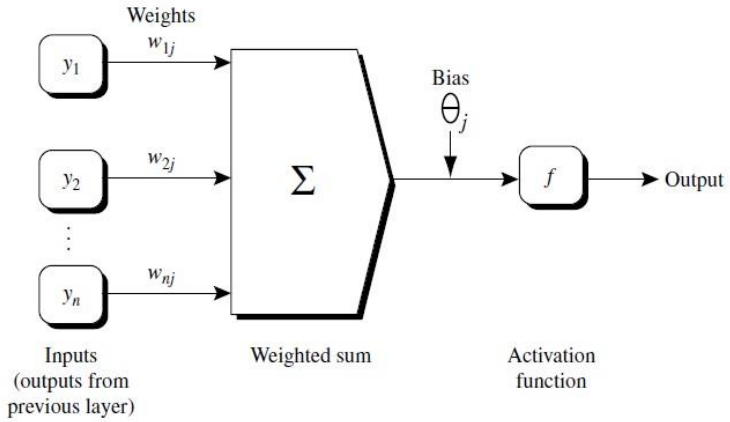
Method

Initialize all weights and biases in *network*;
while terminating condition is not satisfied {
  for each training tuple X in D {
    // Propagate the inputs forward:
    for each input layer unit j {
      $O_j = I_j$; // output of an input unit is its actual input value
    for each hidden or output layer unit j {
      $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i
      $O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j
    // Backpropagate the errors:
    for each unit j in the output layer
      $Err_j = O_j(1 - O_j)(T_j - O_j)$; // compute the error
    for each unit j in the hidden layers, from the last to the first hidden layer
      $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k
    for each weight w_{ij} in *network* {
      $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
      $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
    for each bias θ_j in *network* {
      $\Delta \theta_j = (l) Err_j$; // bias increment
      $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
  } }

Initialize the weights: The weights in the network are initialized to small random numbers (e.g., ranging from -1.0 to 1.0, or -0.5 to 0.5). Each unit has a *bias* associated with it, as explained later. The biases are similarly initialized to small random numbers. Each training tuple, ***X***, is processed by the following steps.

Propagate the inputs forward: First, the training tuple is fed to the network's input layer. The inputs pass through the input units, unchanged. That is, for an input unit, *j*,

its output, *Oj* , is equal to its input value, *Ij* . Next, the net input and output of each unit in the hidden and output layers are computed.



Hidden or output layer unit *j*: The inputs to unit *j* are outputs from the previous layer. These are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated with unit *j*. A nonlinear activation function is applied to the net input. (For ease of explanation, the inputs to unit *j* are labelled *y*1, *y*2, ….. , *yn*. If unit *j* were in the first hidden layer, then these inputs would correspond to the input tuple (*x*1, *x*2, …. , *xn*).)