Machine Learning: Lecture 4

Artificial Neural Networks

(Based on Chapter 4 of Mitchell T.., Machine Learning, 1997)

What is an Artificial Neural Network?

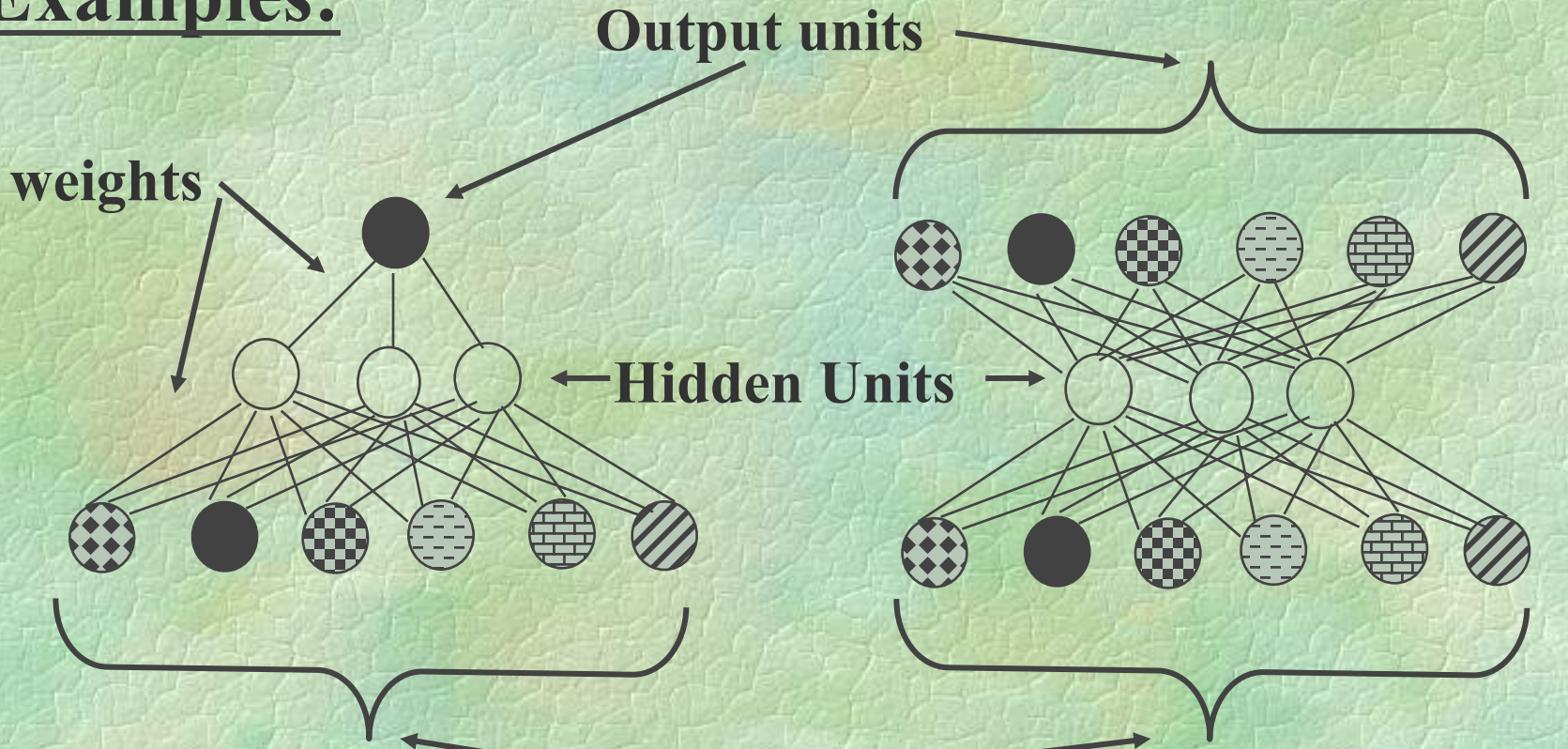
- It is a formalism for representing functions inspired from biological systems and composed of parallel computing units which each compute a simple function.
- Some useful computations taking place in ***Feedforward Multilayer*** Neural Networks are:
  - Summation
  - Multiplication
  - Threshold (e.g., $1/(1+e^{-x})$) [the sigmoidal threshold function]. Other functions are also possible

Biological Motivation

- Biological Learning Systems are built of very complex webs of interconnected neurons.
- Information-Processing abilities of biological neural systems must follow from highly parallel processes operating on representations that are distributed over many neurons
- ANNs attempt to capture this mode of computation

Multilayer Neural Network Representation

Examples:



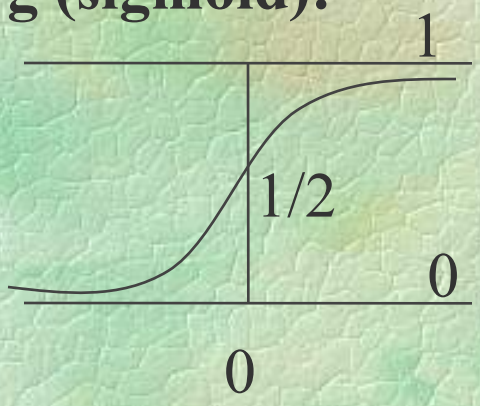
How is a function computed by a Multilayer Neural Network?

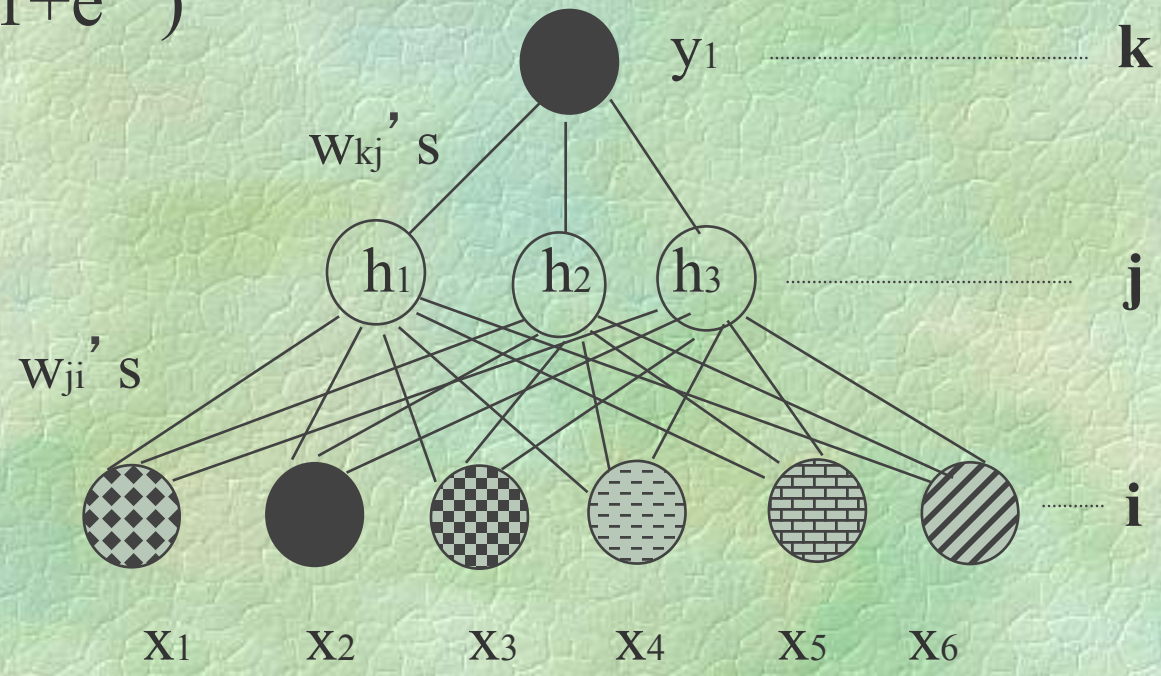
- $h_j = g(\sum_i w_{ji}.x_i)$
- $y_1 = g(\sum_j w_{kj}.h_j)$

where $g(x) = 1/(1+e^{-x})$

Typically, y_1=1 for positive example and y_1=0 for negative example

g (sigmoid):





Learning in Multilayer Neural Networks

- Learning consists of searching through the space of all possible matrices of weight values for a combination of weights that satisfies a database of positive and negative examples (multi-class as well as regression problems are possible).
- Note that a Neural Network model with a set of adjustable weights defines a restricted hypothesis space corresponding to a family of functions. The size of this hypothesis space can be increased or decreased by increasing or decreasing the number of hidden units present in the network.

Appropriate Problems for Neural Network Learning

- Instances are represented by many attribute-value pairs (e.g., the pixels of a picture. ALVINN [Mitchell, p. 84]).
- The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
- The training examples may contain errors.
- Long training times are acceptable.
- Fast evaluation of the learned target function may be required.
- The ability for humans to understand the learned target function is not important.

History of Neural Networks

- **1943: McCulloch and Pitts proposed a model of a neuron --> Perceptron (read [Mitchell, section 4.4])**
- **1960s: Widrow and Hoff explored Perceptron networks (which they called "Adelines") and the delta rule.**
- **1962: Rosenblatt proved the convergence of the perceptron training rule.**
- **1969: Minsky and Papert showed that the Perceptron cannot deal with nonlinearly-separable data sets---even those that represent simple function such as X-OR.**
- **1970-1985: Very little research on Neural Nets**
- **1986: Invention of Backpropagation [Rumelhart and McClelland, but also Parker and earlier on: Werbos] which can learn from nonlinearly-separable data sets.**
- **Since 1985: A lot of research in Neural Nets!**

Backpropagation: Purpose and Implementation

- **Purpose:** To compute the weights of a feedforward multilayer neural network adaptatively, given a set of labeled training examples.
- Method: By minimizing the following cost function (the sum of square error):
  $E = 1/2 \sum_{n=1}^{N} \sum_{k=1}^{K} [y_k^n - f_k(x^n)]^2$

where N is the total number of training examples and K, the total number of output units (useful for multiclass problems) and f_k is the function implemented by the neural net

Backpropagation: Overview

- Backpropagation works by applying the ***gradient descent*** rule to a feedforward network.
- The algorithm is composed of two parts that get repeated over and over until a pre-set maximal number of ***epochs***, ***EPmax***.
- Part I, the ***feedforward*** pass: the activation values of the hidden and then output units are computed.
- Part II, the ***backpropagation*** pass: the weights of the network are updated--starting with the hidden to output weights and followed by the input to hidden weights--with respect to the sum of squares error and through a series of weight update rules called the ***Delta Rule***.

Backpropagation: The Delta Rule I

- **<u>For the hidden to output connections</u>**
  (easy case)
- $\Delta w_{kj} = -\eta \, \partial E / \partial w_{kj}$

  $= \eta \sum_{n=1}^{N} [y_k^n - f_k(x^n)] \, g'(h_k^n) \, V_j^n$

  $= \eta \sum_{n=1}^{N} \delta_k^n \, V_j^n$

  with
  - η corresponding to the ***learning rate***
    (an extra parameter of the neural net)
  - $h_k^n = \sum_{j=0}^{M} w_{kj} \, V_j^n$
  - $V_j^n = g(\sum_{i=0}^{d} w_{ji} x_i^n)$ and
  - $\delta_k^n = g'(h_k^n)(y_k^n - f_k(x^n))$

M is the number of hidden units and d the number of input units

Backpropagation: The Delta Rule II

- **For the input to hidden connections**
  (hard case: no pre-fixed values for the hidden units)
- $\Delta w_{ji} = -\eta \, \partial E / \partial w_{ji}$

  $= -\eta \sum_{n=1}^{N} \partial E / \partial V_j^n \, \partial V_j^n / \partial w_{ji}$ (Chain Rule)

  $= \eta \sum_{k,n} [y_k^n - f_k(x^n)] \, g'(h_k^n) \, w_{kj} \, g'(h_j^n) x_i^n$

  $= \eta \, \delta_k^n w_{kj} g'(h_j^n) x_i^n$

  $= \eta \sum_{n=1}^{N} \delta_j^n x_i^n$ with

  - $h_j^n = \sum_{i=0}^{d} w_{ji} x_i^n$
  - $\delta_j^n = g'(h_j^n) \sum_{k=1}^{K} w_{kj} \, \delta_k^n$
  - and all the other quantities already defined

Backpropagation: The Algorithm

1. Initialize the weights to small random values; create a random pool of all the training patterns; set ***EP***, the number of epochs of training to 0.
2. Pick a training pattern μ from the remaining pool of patterns and propagate it forward through the network.
3. Compute the deltas, δ_k^{μ} for the output layer.
4. Compute the deltas, δ_j^{μ} for the hidden layer by propagating the error backward.
5. Update all the connections such that

$$w_{ji}^{New} = w_{ji}^{Old} + \Delta w_{ji} \quad \text{and} \quad w_{kj}^{New} = w_{kj}^{Old} + \Delta w_{kj}$$

6. If any pattern remains in the pool, then go back to Step 2. If all the training patterns in the pool have been used, then set **EP = EP+1**, and if $\mathbf{EP} < \mathbf{EP}_{Max}$, then create a random pool of patterns and go to Step 2. If $\mathbf{EP} = \mathbf{EP}_{Max}$, then stop.

Backpropagation: The Momentum

- To this point, Backpropagation has the disadvantage of being too slow if η is small and it can oscillate too widely if η is large.
- To solve this problem, we can add a ***momentum*** to give each connection some inertia, forcing it to change in the direction of the downhill "force".
- **New Delta Rule:**

$$\Delta w_{pq}(t+1) = -\eta \, \partial E/\partial w_{pq} + \alpha \, \Delta w_{pq}(t)$$

where p and q are any input and hidden, or, hidden and outpu units; t is a time step or epoch; and α is the momentum parameter which regulates the amount of inertia of the weights.