Supervised Learning

Artificial Neural Networks

Dr. Neeta Nain

Department of Computer Science and Engineering
Malviya National Institute of Technology, Jaipur

October 2018

Outline

- Machine Learning
- Neural Network
- Activation Function
- Model of *ANN*
- Perceptron and Learning
- Modeling *AND*, *OR* and *XOR* gate
- Example
- Implementation Parameters
- Back Propagation

Machine Learning

- Machine learning is an adaptive learning mechanism that simulates the behaviour of human neuro cells and enables computer to learn and classify/recognize.
- The machine is fed with machine understandable pattern in the form of a vector consisting of numerical measures of the significant features of concerned entity.
- Each recognition system is applicable to a certain specific domain of patterns. For example, a character recognizer may not work fine for identification of flowers.
- The most popular approach to machine learning is artificial neural networks.

Neuron

Neuron Cell

neuron.png

The neuron is the basic working unit of the brain, a specialized cell designed to transmit information to other nerve cells, muscle, or gland cells. Most neurons have a cell body, an axon, and dendrites. The activity of a neuron is an all-or-none process. A certain no of synapse (> 1) must be excited within a specific period to activate a neuron.

The Basic Neuron

Simple mathematical model presented by McCulloch and Pitts (1943). It consists of :-

- n input nodes I_1, I_2 ,.., I_n and one output node O.
- An n-vector X applied at input nodes, in one-to-one mapping.
- Weight vector w of size n applied to net connectors, s.t. w_i is assigned between I_i and O. Inputs with Positive weights are called **excitatory** and with negative weights are called **inhibitory**.
- The structure of the interconnection network does not change with time
- Summation Unit computes weighted sum of inputs

$$y = w_1x_1 + w_2x_2 + .. + w_nx_n = \sum_{i=1}^{n} w_ix_i$$

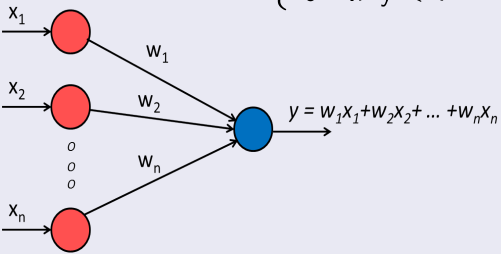
The Basic Neuron

Output Function

- Y is thresholded against threshold value T stalled at O using function F as follows:

$$O = F(y) = \begin{cases} 1 & \text{If } y \geq T \\ 0 & \text{If } y < T \end{cases}$$

$$O = F(y - T) = \begin{cases} 1 & \text{If } y \geq T \\ 0 & \text{If } y < T \end{cases}$$



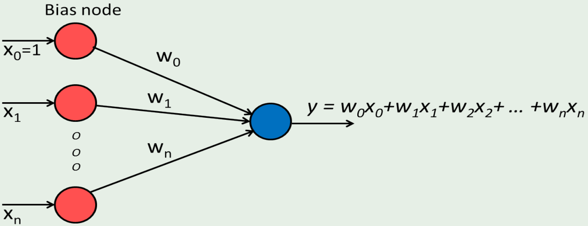
The Basic Neuron

Output with bias input

- Due to the importance of threshold an augmented network is created by adding extra input (bias input) and weight (bias weight) with $x_0 = 1$ and $w_0 = -T$. Thus,

$$y' = w_0x_0 + w_1x_1 + w_2x_2 + .. + w_nx_n = \sum_{i=0}^{n} w_ix_i$$

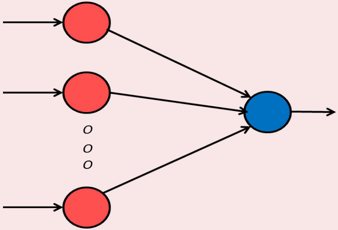
$$O' = F(y') = \begin{cases} 1 & \text{If } y' \geq 0 \\ 0 & \text{If } y' < 0 \end{cases}$$

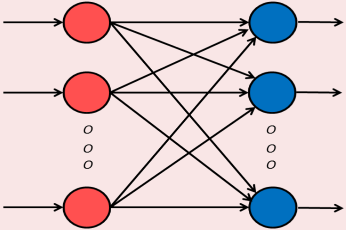


Types of Neural Network

Single output and Multiple output Neural Network

- A neuron is a NW with a single node. A number of neurons with separate output nodes but using the same inputs is a general single layer neural NW. Fig (a) Single layer single output node and Fig (b) is a single layer multiple output nodes neural NW.

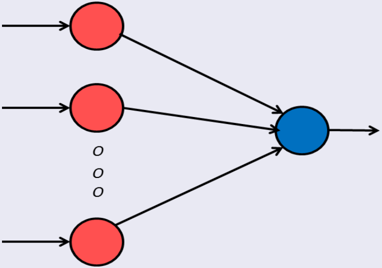




Types of Neural Network

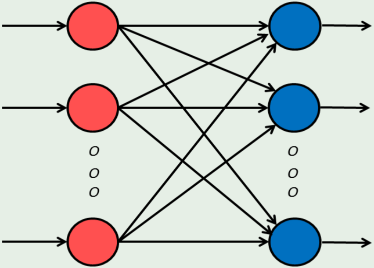
Single output

Single layer single output node neural network



Multiple output

Multiple output nodes have different weight vectors. Weight between input x_i and output node o_j is given by w_{ij}. Output vector is defined as $O = [o_1 \ \ o_2 \ \ .. \ \ o_m]^T$.



Activation Function

- Also known as a Transfer function, Squash function.
- Decides the amount and nature of activation needed to stimulate the output/next neuron.
- There are three types of activation functions.

1. Step function

Also known as **Heaviside function**.

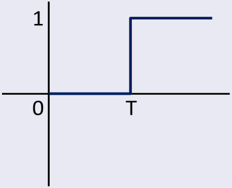
The weighted sum is compared with a fixed threshold value **T**.

The activation function is expressed as:

$$O = F(y) \qquad \text{OR} \qquad O = F(y - T)$$

The step function is represented as:

$$F(y) = \begin{cases} 1 & \text{If } y \geq T \\ 0 & \text{If } y < T \end{cases}$$

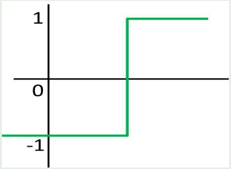


Activation Function

2. Signum function:

Also known as a **Quantizer function**. It is defined as:

$$F(y) = \begin{cases} +1 & \text{If } y \geq T \\ -1 & \text{If } y < T \end{cases}$$

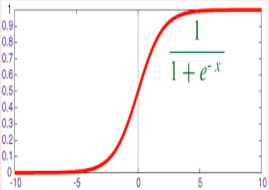


3. Sigmoid function:

It is a continuous function that varies asymptotically between 0 and 1, defined as:

$$F(y) = \frac{1}{(1 + e^{-\alpha y})}$$

where, α is the slope parameter.



Models of ANN

1. Feedforward Network

The input and output vectors are $x = [x_1 \;\; x_2 \;\; .. \;\; x_n]$ and $O = [O_1 \;\; O_2 \;\; .. \;\; O_m]^T$.

weight between x_i input and O_j output is given by w_{ij}.
Activation value for j^{th} neuron is computed as:

$$y_j = \sum_{i=1}^{n} w_{ij} x_i \qquad \text{for } j = 1, 2, ..., m$$

Output from the j^{th} neuron is expressed as:

$$O_j = F(\sum_{i=1}^{n} w_{ij} x_i) = F(w_j^T x)$$

where, $w_j = [w_{1j} \;\; w_{2j} \;\; .. \;\; w_{nj}]^T$ contains weights leading to the j^{th} output node.

Models of ANN

Output of the complete network is expressed as: $O = F(W_x)$

$$W = \begin{bmatrix} w_{11} & w_{21} & \dots & w_{n1} \\ w_{12} & w_{22} & \dots & w_{n2} \\ \dots & \dots & \dots & \dots \\ w_{1m} & w_{2m} & \dots & w_{mn} \end{bmatrix} \qquad F = \begin{bmatrix} f(.) & 0 & 0 & 0 \\ 0 & f(.) & 0 & 0 \\ .. & .. & .. & .. \\ 0 & 0 & .. & f(.) \end{bmatrix}$$

where, W is the weight matrix or the connection matrix and F is the component wise activation function matrix. Individual activation values are the scalar products of input vectors with the corresponding weight vectors.

Perceptron and Learning

Perceptron

- Developed by Rosenblatt (1953), is a trainable classifier.
- Acts as a 2-class classifier previously trained with sufficient training patterns.
- An advantage of perceptron is that nodes in a layer can operate simultaneously enabling high degree of parallelism, realized in a true hardware implementation.

Minimum Squared Error (MSE) Criterion

- Error is the difference between the values of actual output of perceptron and desired output.
- During training, we establish a fixed training set of weight values by satisfying the MSE criterion.

Sequential MSE Algorithm

1. To find the values of the weights $w_1, w_2 ... w_M$ that minimise the error $E = \frac{1}{2}\sum_{p=1}^{N}(D_p - d_p)^2, D_p = w_0 + w_1 x_{p1} + \cdots + w_M x_{pM}$
2. d_p is the desired output for sample p, and $x_{p1}, \cdots, x_{pM}$ are the feature values of that sample.
3. the weights are optimized all at once for the entire dataset by setting all the partial derivatives of $\frac{\delta E}{\delta w_i} = 0$, and solving the resulting set of linear equations simultaneously for the weights.
4. weights are changed in directions that will decrease the error between desired (d) and actual output (D). The direction of steepest descent of a differentiable function D of M variables is the vector $(-\eta\frac{\delta D}{\delta w_1}, \cdots, \eta\frac{\delta D}{\delta w_M})$, η is a $+$ve constant of proportionality.
5. pick a starting guess $w_1, \cdots, w_M$ and move a short distance in the direction of the steepest decrease in function.

Perceptron and Learning

Algorithm:

1. Initialize $w_1, w_2 ... w_M$ with random values; choose a +ve η.
2. Input training pattern **x** along with desired output **d**.
3. Calculate the actual output $D = \sum_{i=0}^{n} w_i x_i$
4. Compute error, E, of classification and take partial differentiation of E with respect to w_i

   $E = \frac{1}{2}(D - d)^2 \qquad \frac{\delta E}{\delta w_i} = \frac{\delta E}{\delta D}\frac{\delta D}{\delta w_i}$

   $\frac{\delta E}{\delta D} = (D - d) \qquad \frac{\delta D}{\delta w_i} = x_i \qquad \frac{\delta E}{\delta w_i} = (D - d)x_i$
5. Update weight values in order to achieve MSE criterion

$$w_i = w_i - \eta(D - d)x_i$$

   where, η is a learning rate factor whose values are kept between 0 and 1.

Steepest Descent Minimization Procedure

- The weights are changed in directions that will decrease the error between the desired output of the network d and its actual output D
- Pick a starting guess $w_1, \cdots, w_M$ and choose a positive learning rate $0 < \eta < 1$
- Compute the partial derivatives $\dfrac{\delta D}{\delta w_i}$, for $i = 1, \cdots, M$.

  Replace w_i by $w_i - \eta \dfrac{\delta D}{\delta w_i}$ for $i = 1, \cdots, M$.
- Repeat step 2 until $w_1, \cdots, w_M$ cease to change significantly.

Function Minimization by Steepest Descent

- If partial derivatives are not available analytically, they may be estimated by finding the increase in E produced by increasing w_i a very small amount, i..e, $\frac{\delta E}{\delta w_i} \approx \frac{\Delta E}{\Delta w_i}$
- Minimize the function $F(w_1, w_2) = w_1^4 + w_2^4 + 3w_1^2 w_2^2 - 2w_1 - 3w_2 + 4$, which has the partial derivatives
- $\frac{\delta F(w_1, w_2)}{\delta w_1} = 4w_1^3 + 6w_1 w_2^2 - 2$ and
- $\frac{\delta F(w_1, w_2)}{\delta w_2} = 4w_2^3 + 6w_1^2 w_2 - 3$, $\eta = 0.1$

Function Minimization by Steepest Descent

Iteration	$F(w_1, w_2)$	$\frac{\delta F(w_1, w_2)}{\delta w_1}$	$\frac{\delta F(w_1, w_2)}{\delta w_2}$	w_1	w_2
1	4.0	-2.0	-3.0	0.0	0.0
2	2.7	-1.86	-2.82	0.2	0.3
3	1.77	-0.9855	-1.691	0.386	0.582
4	1.548	0.09	-0.2458	.4845	0.7511
5	1.543	0.144	-0.080	0.475	0.776
10	1.539	0.0221	-0.0161	0.438	0.802
30	1.539	0.0	0.0	0.43	0.807

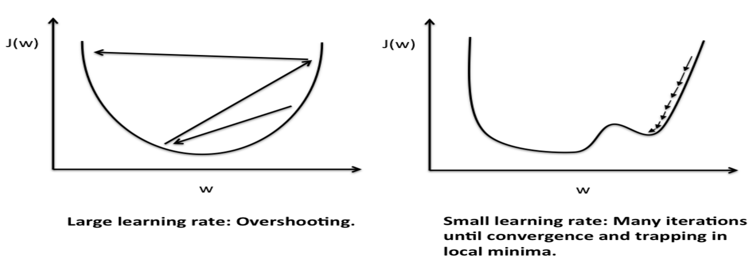
After 30 iterations, w_1 and w_2 cease to change significantly, so a local minimum has reached.

Difficulties with Steepest Descent

- choosing a good starting guess and choosing an appropriate η
- If a starting guess is too far from the values that produce the absolute global minimum is chosen, it may converge to a local minimum
- If the η is chosen too small, progress toward the minimum will be so slow that a huge no of iterations will be required
- If η is too large, it may repeatedly overshoot the minimum and progress will be slow, the procedure may even cycle between two or more local minima and fail to converge
- Decreasing η as the iteration number increases can be useful

Gradient Descent

Effect of Learning Rate Parameter

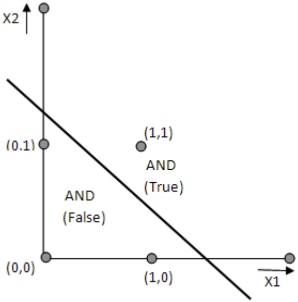


Modeling AND gate

- The simple perceptron can be used to model this as an *AND* gate.
- It is done by applying known input output combinations ensuring training and adjusting weight vectors of the perceptron.
- The output is *TRUE* if the input combination satisfies logical *AND*, otherwise *FALSE*.
- Let us consider 2 inputs x_1 (Registered Candidate) and x_2 (Attendance) resulting in 4 input patterns as shown in Table.
- The input to the perceptron is $w_1 x_1 + w_2 x_2$ with weights w_1 and w_2.

Modeling AND gate

Training Pattern	Attributes		Class	Desired Output
	x_1	x_2		
1	1	1	TRUE	1
2	0	1	FALSE	0
3	1	0	FALSE	0
4	0	0	FALSE	0



Modeling AND gate

- During learning phase, we start with random values of w_1 and w_2 and update weights until it become stable.
- The output O is defined as
- The threshold $T = 0.75$ and learning rate $\eta = 0.2$

$$O = \begin{cases} 0 & \text{If } w_1 x_1 + w_2 x_2 < 0.75 \\ 1 & \text{Otherwise} \end{cases}$$

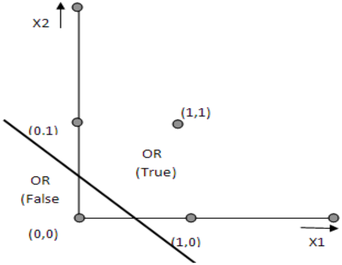
$$w_{i+1} = w_i - \eta(O - d)x_i$$

Iterations	x1	x2	Weights		$\sum_i w_i x_i$	o	d	Δw_1	Δw_2
1	1	1	0.2	0.3	0.5	0	1	0.2	0.2
	0	1	0.4	0.5	0.5	0	0	0.0	0.0
	1	0	0.4	0.5	0.4	0	0	0.0	0.0
	0	0	0.4	0.5	0.0	0	0	0.0	0.0
2	1	1	0.4	0.5	0.9	1	1	0.0	0.0
	0	1	0.4	0.5	0.5	0	0	0.0	0.0
	1	0	0.4	0.5	0.4	0	0	0.0	0.0
	0	0	0.4	0.5	0.0	0	0	0.0	0.0

Modeling OR gate

- The simple perceptron can be used to model an OR gate.
- Let us consider 2 inputs x_1 and x_2 resulting in 4 input patterns as shown in Table.
- The input to the perceptron is $w_1x_1 + w_2x_2$ with weights w_1 and w_2.
- Objective is to compute the values of w_1 and w_2 so that the truth table of OR gate is satisfied by the input combination.

Training Pattern	Attributes		Class	Desired Output
	x_1	x_2		
1	1	1	TRUE	1
2	0	1	TRUE	1
3	1	0	TRUE	1
4	0	0	FALSE	0



Modeling OR gate

- The output O is defined as
- The threshold $T = 0.6$ and learning rate $\eta = 0.2$

$$O = \begin{cases} 0 & \text{If } w_1 x_1 + w_2 x_2 < 0.60 \\ 1 & \text{Otherwise} \end{cases}$$

Iterations	x1	x2	Weights		$\sum_i w_i x_i$	o	d	Δw_1	Δw_2
1	1	1	0.2	0.3	0.5	0	1	0.2	0.2
	0	1	0.4	0.5	0.5	0	1	0.0	0.2
	1	0	0.4	0.7	0.4	0	1	0.2	0.0
	0	0	0.6	0.7	0.0	0	0	0.0	0.0
2	1	1	0.6	0.7	1.3	1	1	0.0	0.0
	0	1	0.6	0.7	0.7	1	1	0.0	0.0
	1	0	0.6	0.7	0.6	1	1	0.0	0.0
	0	0	0.6	0.7	0.0	0	0	0.0	0.0

Example

- Consider a pattern classification problem as given in Table and classify the unknown pattern $p = (0.5, 0.7)$. Consider $\eta = 0.1$, $w_1 = 0.2$, $w_2 = 0.3$ and $T = 0.6$

Training Pattern	Attributes		Class
	x_1	x_2	
1	0.1	0.2	0
2	0.2	0.3	0
3	0.6	0.6	1
4	0.8	0.5	1

- The variation of weight parameters during learning is presented in Table.
- It is observed that the network is completely trained after 3^{rd} iteration and final weights turn out to be

Example

- Consider a pattern classification problem as given in Table and classify the unknown pattern $p = (0.5, 0.7)$. Consider $\eta = 0.1$, $w_1 = 0.2$, $w_2 = 0.3$ and $T = 0.6$

Training Pattern	Attributes		Class
	x_1	x_2	
1	0.1	0.2	0
2	0.2	0.3	0
3	0.6	0.6	1
4	0.8	0.5	1

- The variation of weight parameters during learning is presented in Table.
- It is observed that the network is completely trained after 3^{rd} iteration and final weights turn out to be $w_1 = 0.48$, $w_2 = 0.52$.
- For, pattern $p = (0.5, 0.7)$ the response is $y = w_1x_1 + w_2x_2 = 0.48 \times 0.5 + 0.52 \times 0.7 = 0.604$

Example

- Consider a pattern classification problem as given in Table and classify the unknown pattern $p = (0.5, 0.7)$. Consider $\eta = 0.1$, $w_1 = 0.2$, $w_2 = 0.3$ and $T = 0.6$

Training Pattern	Attributes		Class
	x_1	x_2	
1	0.1	0.2	0
2	0.2	0.3	0
3	0.6	0.6	1
4	0.8	0.5	1

- The variation of weight parameters during learning is presented in Table.
- It is observed that the network is completely trained after 3^{rd} iteration and final weights turn out to be $w_1 = 0.48$, $w_2 = 0.52$.
- For, pattern $p = (0.5, 0.7)$ the response is $y = w_1x_1 + w_2x_2 = 0.48 \times 0.5 + 0.52 \times 0.7 = 0.604$
- Therefore, p belongs to category 1.

Example

Iterations	x1	x2	Weights		$\sum_i w_i x_i$	o	d	Δw_1	Δw_2
1	0.1	0.2	0.2	0.3	0.08	0	0	0.0	0.0
	0.2	0.3	0.2	0.3	0.13	0	0	0.0	0.0
	0.6	0.6	0.2	0.3	0.30	0	1	0.06	0.06
	0.8	0.5	0.26	0.36	0.388	0	1	0.08	0.05
2	0.1	0.2	0.34	0.41	0.116	0	0	0.0	0.0
	0.2	0.3	0.34	0.41	0.191	0	0	0.0	0.0
	0.6	0.6	0.34	0.41	0.450	0	1	0.06	0.06
	0.8	0.5	0.40	0.47	0.555	0	1	0.08	0.05
3	0.1	0.2	0.48	0.52	0.152	0	0	0.0	0.0
	0.2	0.3	0.48	0.52	0.252	0	0	0.0	0.0
	0.6	0.6	0.48	0.52	0.60	1	1	0.0	0.0
	0.8	0.5	0.48	0.52	0.644	1	1	0.0	0.0

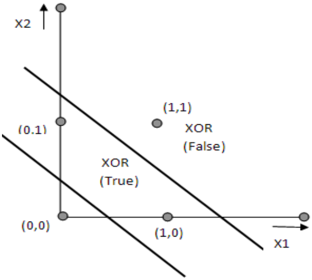
MSE for a two layer net with multiple outputs

- We have for each of the N samples the feature values $x_0, x_1, \cdots, x_M$ and desired outputs $d_1, \cdots, d_{M_1}$. w_{ij} is the weight on input i for output node j
- $E = \frac{1}{2}\sum_{j=1}^{M_1}(D_j - d_j)^2$, since $D_j = \sum_{i=0}^{M_1} w_{ij}x_i, \quad \frac{\delta E}{\delta w_{ij}} = (D_j - d_j)x_i$
- Pick starting weights and learning rate η
  1. Present samples $1, \cdots, N$ repeatedly to the classifier, cycling back to sample 1 after N.
  2. For each sample, compute $D_j = w_{0j} + w_{1j}x_1 + \cdots + w_{Mj}x_M$, for nodes $j = 1, \cdots, M_1$
  3. Replace w_{ij} by $wij - \eta(D_j - d_j)x_i$ for all i
- Repeat step 1 - 3 until weights cease to change significantly.

Problems in Modeling XOR gate

- XOR gate cannot be modeled by a simple perceptron as it is nonlinearly separable.
- Figure shows that XOR inputs cannot be categorized using a single straight line.
- The input-output combination and corresponding classes is shown in Table.

Training Pattern	Attributes		Class	Desired Output
	X_1	X_2		
1	1	1	FALSE	0
2	0	1	TRUE	1
3	1	0	TRUE	1
4	0	0	FALSE	0



Problems in Modeling XOR gate

Proof:

Input to the perceptron is $w_1 x_1 + w_2 x_2$ with weights w_1 and w_2.
Output is

$$O = \begin{cases} 0 & \text{If } w_1 x_1 + w_2 x_2 < T \\ 1 & \text{Otherwise} \end{cases}$$

Applying actual inputs in above equation:

$$w_1 \text{ x } 1 + w_2 \text{ x } 1 < T \qquad w_1 \text{ x } 0 + w_2 \text{ x } 1 \geq T$$
$$w_1 \text{ x } 0 + w_2 \text{ x } 0 < T \qquad w_1 \text{ x } 1 + w_2 \text{ x } 0 \geq T$$

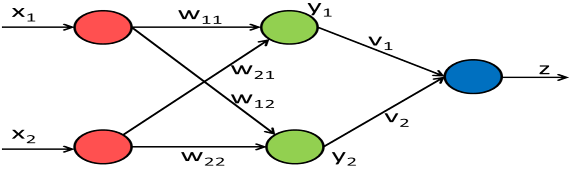
After simplification:

$w_1 + w_2 < T$ and $w_1, w_2 \geq T$

which are contradictory to each other. Hence, *XOR* cannot be implemented using a perceptron.

Problems in Modeling XOR gate

- XOR outputs are separable by a pair of hyperplanes.
- Multiple perceptrons can be used to model XOR gate by using 2 layers of neurodes between input and output.



- Inputs x_1 and x_2 are applied at the input nodes, their weighted sum is computed and thresholded.
- Let $w_{1,1} = 1, w_{1,2} = -1, w_{2,1} = -1$ and $w_{2,2} = 1$. The output of 1^{st} layer is

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{OR} \quad \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Modeling XOR gate

- For second layer inputs are y_1 and y_2, z denotes activation function, v_1 and v_2 weights, and o is the thresholded output.
- The input-output relation for the 2 layers is written as:

$$O_j = \begin{cases} 0 & \text{If } w_{1j}x_1 + w_{2j}x_2 < 0.1 \\ 1 & \text{Otherwise} \end{cases}$$

$$O = \begin{cases} 0 & \text{If } v_1 y_1 + v_2 y_2 < 0.1 \\ 1 & \text{Otherwise} \end{cases}$$

x_1	x_2	w_{11}/w_{22}	w_{11}/w_{22}	y_1	y_2	o_1	o_2	v_1	v_2	z	o
1	1	1	-1	0	0	0	0	1	1	0	0
0	1	1	-1	-1	1	0	1	1	1	1	1
1	0	1	-1	1	-1	1	0	1	1	1	1
0	0	1	-1	0	0	0	0	1	1	0	0

Limitations of a Perceptron

Limitations of a Perceptron

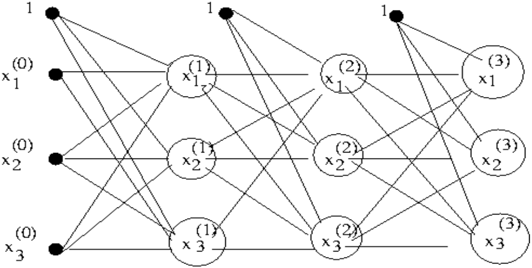
- A simple perceptron is very limited in its capacity.
- Classify a set of patterns into two linearly separable classes only.

Nets with Hidden layers (MLP)

- After Rosenblatt developed the perceptron classifier (1953), research in 1960s, it became apparent that the perceptron was too limited to use on any but the simplest learning situations
- Minsky and Pappert (1969) stated - Multilayer nets were so general that they were equivalent to a universal computer and hence too complex for general analysis
- Research continued to develop many types of mutilevel adaptive networks. Except for the input nodes, each node is a neuron (or processing element) with a nonlinear activation function.
- MLP utilizes a supervised learning technique called backpropagation for training the network. MLP is a modification of the standard linear perceptron and can distinguish data that are not linearly separable.
- The two main activation functions used in current applications are both sigmoids, S-shaped curve, so that each weight has some effect on the final output. e.g., , $R(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$ and $R(s) = \frac{1}{1+e^{-s}}$, in which the former function is a hyperbolic tangent which ranges from -1 to 1, and the latter, the logistic function, is similar in shape but ranges from 0 to 1.
- MLPs are universal function approximators as showed by Cybenko's theorem, so they can be used to create mathematical models by regression analysis.

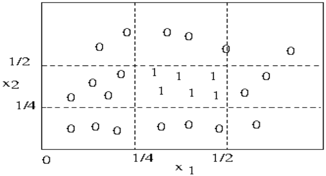
MultiLayer ANN

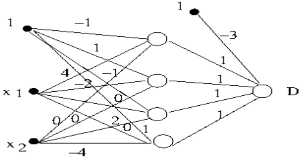
- A multilayer net has $K+1$ layers, denoted as $0, 1, \cdots, K$.
- Input layer has index $k = 0$, called retina. Nodes in layer K are called the output nodes, and those in layers $1, \cdots, K-1$ are called the hidden nodes
- The output of i^{th} node in layer k is denoted $x_i^{(k)}$. This is the value obtained after computing the weighted sum of the inputs and applying the threshold or other function.
- A two-layer net can classify samples into two classes which are separated by a hyperplane. If the problem is to classify into two decision regions where class 1 is convex and class 0 is the complement of class 1, three layers are required



Example

- A point is inside the square when $\frac{1}{4} < x_1 < \frac{1}{2}$, AND, $\frac{1}{4} < x_2 < \frac{1}{2}$





- These inequalities can be written as

$$
\begin{array}{ll}
0 < 4x_1 - 1 + 0x_2 & 0 < -2x_1 + 1 + 0x_2 \\
0 < 2x_2 - 1 + 0x_1 & 0 < -4x_2 + 1 + 0x_2
\end{array}
$$

- The hidden nodes (top to bottom), compute the right-hand side of these inequalities, which is then combined with a logical AND (with the right-hand layer of weights)
- $D = -3 + 1.1 + 1.1 + 1.1 + 1.1 = 1$ (weighted sum (OR)). If any of the hidden nodes are 0, the thresholded sum is negative, so the output is 0.

Nets with hidden layers

- Layer 1 determine whether the sample lies in each of the half-planes, making up the convex region
- Layer 2 performs a logical AND to see if the pattern is in all of those half planes simultaneaously
- A 3 layer AND-OR network, can always perfectly separate two classes if they can be distinguished by a set of binary features, no matter how many are there (discrete values can be encoded as binary features)
- Each node in layer-2 specializes in detecting one of the possible input combinations (as many as) that requires an output $= 1$
- The single node in the third layer computes the logical OR of the inputs from the second layer (if any one output in layer-2 is 1)

Back Propagation

- The back-propagation algorithm for training the weights in a multilayer net uses the steepest descent minimization procedure and the sigmoid threshold function.
- The back propagation algorithm consists of two main steps
  1. a feed-forward step in which the outputs of the nodes are computed starting at layer 1 and working forward to the output layer.
  2. a back propagation step where the weights are updated in an attempt to get better agreement between the observed output and the desired output.
- The feed-forward step begins from first layer and works forward to last layer.
- The back-propagation step begins from last layer and works backward to first layer.

Back Propagation Algorithm

- Training a multilayer threshold network is complicated as a small change in one of the weights will usually not effect the outputs of the network at all, so steepest descent is not feasible.
- The outputs from a node will be affected only if one of the weights changes enough to cause its weighted sum to change its sign
- Even if an output changes at one layer, the outputs at the next layer may not necessarily change, so the final outputs are very resistant to most small weight changes.
- We might consider eliminating the threshold elements and simply compute weighted sums at each node (then there would be no need to use a multilayer net, since any net without thresholds is $\approx$ to a two layer net without thresholds)

Back Propagation

- A compromise between the two extremes of using a discontinuous threshold at each node and completely linear nodes is to use a smooth S-shaped function that gradually changes from 0 when the weighted sum is much less that the theshold to 1 when s is much greater than the threshold.
- Sigmoid popularized by Rummelhart $R(s) = \frac{1}{(1+e^{-s})}$. It varies from 0 when $s \to -\infty$, to 1 when $s \to +\infty$, to $1/2$ when $s = 0$
- The property that $\frac{\delta R}{\delta s} = R(1 - R)$, is used in back-propagation
- Use of sigmoid threshold R at each node in a net causes the outputs of the net to be differentiable functions of the weights, so the steepest descent can be used to find a locally optimal set of weights
- The outputs of such a net can be very complicated nonlinear functions of the inputs, so the weights to which the net converges may be dependent on the starting guess and the step size.

Back Propagation for Training Weights in a Multilayer Net

- Uses the steepest descent minimization procedure and the sigmoid threshold function.
- Layers are $k = 0, \cdots, K$, with $k = 0$ as input and $k = K$ as output layer.
- The output of node j in layer k is denoted as $x_j^{(k)}$ for $j = 1, \cdots, M_k$, where M_k is the number of nodes in layer k (not counting the node with a bias weight)
- The input layer j passes its input x_j along as its output: $x_j^{(0)} = x_j$ for $j = 1, \cdots, M_0$
- In every layer except the output, the bias node outputs 1, so $x_0^{(k)} = 1$ for $k = 0, \cdots, K - 1$
- The outputs are $x_j^{(K)}$ for $j = 1, \cdots, M_K$(output layer doesnot have a node of index 0).
- The weight of the connection from node i in layer $k - 1$ to node j in layer k is denoted as $w_{ij}^{(k)}$
- Feed-forward step (outputs of the nodes are computed) begins at layer 1 and works forward to layer K. The back-propagation step (weights are updated in an attempt to get better agreement between the observed outputs $x_1^{(K)}, \cdots, x_{M_K}^{(K)}$ and the desired outputs $(d_1, \cdots, d_{M_K})$ begins at layer K and works backward to layer 1.

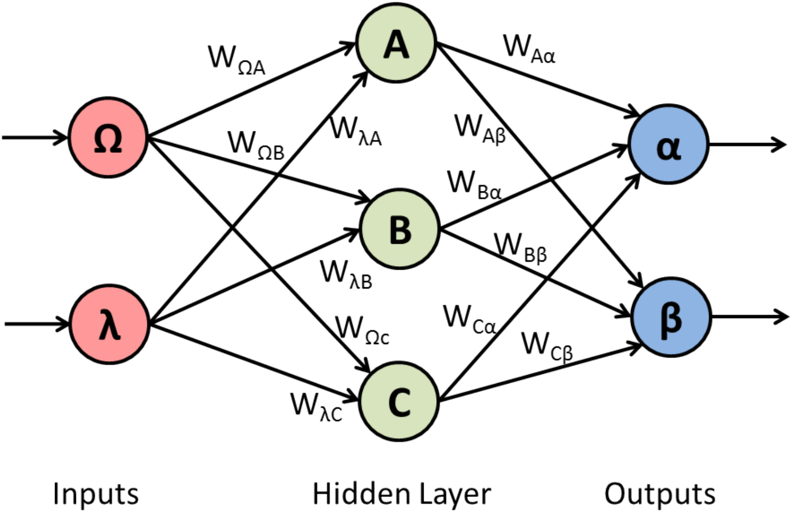
Error Function Minimization

- $E = \frac{1}{2}\sum_{j=1}^{M_K}(x_j^{(K)} - d_j)^2$, d_j is the desired output of the j^{th} node in the last layer.
  The partial derivatives of E are first computed with respect to the weights in the last layer, then in the next-to-last layer, and so on.
- Since the partial derivatives of E in layer k involve the partial derivatives of E in layer $k+1$, when the partial derivatives of E in layer k are computed, all of the terms involved in the expression will already have been computed
- $\frac{\delta E}{\delta w_{ij}^{(K)}} = (x_j^{(K)} - d_j)\frac{\delta x_j^{(K)}}{\delta w_{ij}^{(K)}} = (x_j^{(K)} - d_j)x_j^{(K)}(1 - x_j^{(K)})x_i^{(K-1)}$. We have used
  $\frac{\delta R(s)}{\delta s} = R(s)(1 - R(s))$
- $\frac{\delta x_j^{(K)}}{\delta w_{ij}^{(K)}} = \frac{\delta}{\delta w_{ij}^{(K)}} R(\sum_{i=0}^{M_{K-1}} w_{ij}^{(K)} x_i^{(K-1)}) = x_j^{(K)}(1 - x_j^{(K)})x_i^{(K-1)}$
- Therefore $w_{ij}^{(K)}$ should be replaced by $w_{ij}^{(K)} - \eta\frac{\delta E}{\delta w_{ij}^{(K)}} = w_{ij}^{(K)} - \eta(x_j^{(K)} - d_j)x_j^{(K)}(1 - x_j^{(K)})x_i^{(K-1)}$,
  which becomes $w_{ij}^{(K)} - \eta\delta_j^{(K)}x_i^{(K-1)}$
- If we let $\delta_j^{(K)} = (x_j^{(K)} - d_j)x_j^{(K)}(1 - x_j^{(K)}) = x_j^{(K)}(1 - x_j^{(K)})(x_j^{(K)} - d_j)$

Back Propagation Algorithm

- Initialize the weights $w_{ij}^{(k)}$ to small ramdom values, and η a postive constant
- Repeatedly set $x_1^{(0)}, \cdots, x_{M_0}^{(0)}$ equal to the features of samples $1, \cdots, N$, cycling back to sample 1 after N
- Feed-forward step: for $k = 0, \cdots, K-1$, compute $x_j^{(k+1)} = R(\sum_{1=0}^{M_k} w_{ij}^{(k+1)} x_i^{(k)})$ for nodes $j = 1, \cdots, M_{k+1}$, use the sigmoid function $R(s) = \frac{1}{(1+e^{-s})}$
- Back-propagation step: For the nodes in the output layer, $j = 1, \cdots, M_k$, compute $\delta_j^{(K)} = x_j^{(K)}(1 - x_j^{(K)})(x_j^{(K)} - d_j)$.
  For layers $k = K-1, \cdots, 1$ compute $\delta_i^{(k)} = x_i^{(k)}(1 - x_i^{(k)}) \sum_{j=1}^{M_{k+1}} \delta_j^{(k+1)} w_{ij}^{(k+1)}$ for $i = 1, \cdots, M_k$
- Update weights: $w_{ij}^{(k)} = w_{ij}^{(k)} - \eta \delta_j^{(k)} x_i^{(k-1)}$ for all i, j, k.
- Repeat steps 2 to 5 until the weights $w_{ij}^{(k)}$ cease to change significantly

Back Propagation: Example



Back Propagation

1. Calculate errors of output neurons
$\delta_\alpha = out_\alpha(1 - out_\alpha)(Target_\alpha - out_\alpha)$
$\delta_\beta = out_\beta(1 - out_\beta)(Target_\beta - out_\beta)$
2. Change output layer weights

$$W^+_{A\alpha} = W_{A\alpha} + \eta\delta_\alpha out_A \qquad W^+_{A\beta} = W_{A\beta} + \eta\delta_\beta out_A$$
$$W^+_{B\alpha} = W_{B\alpha} + \eta\delta_\alpha out_B \qquad W^+_{B\beta} = W_{B\beta} + \eta\delta_\beta out_B$$
$$W^+_{C\alpha} = W_{C\alpha} + \eta\delta_\alpha out_C \qquad W^+_{C\beta} = W_{C\beta} + \eta\delta_\beta out_C$$

3. Calculate (back-propagate) hidden layer errors
$\delta_A = out_A(1 - out_A)(\delta_\alpha W_{A\alpha} + \delta_\beta W_{A\beta})$
$\delta_B = out_B(1 - out_B)(\delta_\alpha W_{B\alpha} + \delta_\beta W_{B\beta})$
$\delta_C = out_C(1 - out_C)(\delta_\alpha W_{C\alpha} + \delta_\beta W_{C\beta})$
4. Change hidden layer weights

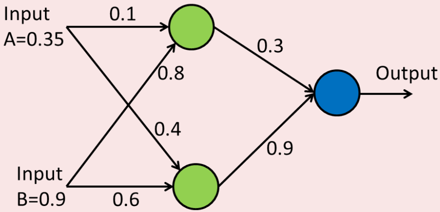
$$W^+_{\lambda A} = W_{\lambda A} + \eta\delta_A in_\lambda \qquad W^+_{\lambda A} = W_{\lambda A} + \eta\delta_A in_\lambda$$
$$W^+_{\lambda B} = W_{\lambda B} + \eta\delta_B in_\lambda \qquad W^+_{\lambda B} = W_{\lambda B} + \eta\delta_B in_\lambda$$
$$W^+_{\lambda C} = W_{\lambda C} + \eta\delta_C in_\lambda \qquad W^+_{\lambda C} = W_{\lambda C} + \eta\delta_C in_\lambda$$

5. The constant $0 < \eta < 1$ (learning rate) is put in to speed up or slow down the learning if required.

Back Propagation

Example

Consider the simple network below:



Assume that the neurons have a Sigmoid activation function and
(i) Perform a forward pass on the network.
(ii) Perform a backpropagation once (target $= 0.5$, learning rate $\eta = 0.1$).
(iii) Perform a further forward pass and comment on the result.

Back Propagation

Answer

Input to top neuron $= (0.35x0.1) + (0.9x0.8) = 0.755$. Out $= 0.68$.
Input to bottom neuron $= (0.9x0.6) + (0.35x0.4) = 0.68$. Out $= 0.6637$.
Input to final neuron $= (0.3x0.68) + (0.9x0.6637) = 0.80133$. Out $= 0.69$.

Output error
$\delta = (t - o)(1 - o)o = (0.5 - 0.69)(1 - 0.69)0.69 = -0.0406$.

Errors for hidden layers:
$\delta_1 = \delta \times w_1 = -0.0406 \times 0.3(1 - 0.68)0.68 = -2.65 \times 10^{-3}$
$\delta_2 = \delta \times w_2 = -0.0406 \times 0.9(1 - 0.67)0.67 = -8.078 \times 10^{-3}$

New weights for hidden layer
$w_{11}^{+} = w_{11} + \eta(\delta_1 \times input) = 0.1 + 0.1(-2.65 \times 10^{-3} \times 0.35) = 0.099$
$w_{21}^{+} = w_{21} + \eta(\delta_1 \times input) = 0.8 + 0.1(-2.65 \times 10^{-3} \times 0.9) = 0.799$
$w_{12}^{+} = w_{12} + \eta(\delta_2 \times input) = 0.4 + 0.1(-8.078 \times 10^{-3} \times 0.35) = 0.399$
$w_{22}^{+} = w_{22} + \eta(\delta_2 \times input) = 0.6 + 0.1(-8.078 \times 10^{-3} \times 0.9) = 0.599$

Back Propagation

New weights for output layer:

$w_{13}^{+} = w_{13} + \eta(\delta_3 \times input_1) = 0.3 + 0.1(-0.0406) \times 0.68 = 0.297$

$w_{23}^{+} = w_{23} + \eta(\delta_3 \times input_2) = 0.9 + 0.1(-0.0406) \times 0.66 = 0.897$

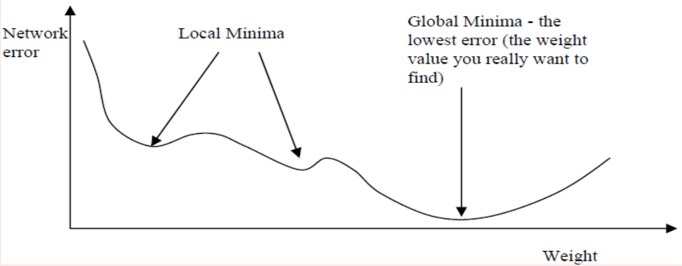
(iii)

Old error was -0.0406. New error is -0.18205. Therefore error has reduced.

Back Propagation

Problem with Back Propagation

- Backpropagtion has some problems associated with it. The best known is called Local Minima.
- This occurs because the algorithm always changes the weights in such a way as to cause the error to fall. But the error might briefly have to rise as part of a more general fall.
- The algorithm will *stuck* (because it cant go uphill) and the error will not decrease further.



Back Propagation

Solutions of Problem with Back Propagation

- A very simple solution is to reset the weights to different random numbers and try training again (this can also solve several other problems).
- Another solution is to add *momentum* to the weight change. This means that the weight change for this iteration depends not just on the current error, but also on previous changes. For example:
  $W^+ = W + \text{Current change} +$
  $(\text{Change on previous iteration} * \textit{constant})$ where the constant < 1.

Thank You