Architecture of neural networks

The perceptron, backpropagation and its variants

Single-Layer and Multi-Layer Perceptrons

The contents to be discussed include:

- ★ Neural network Architectures
- ★ The Single-Layer/Multi-Perceptron
- ★ Perceptron Learning Algorithm
- ★ The Back-Propagation Algorithm
- ★ Issues with the Back-Propagation Algorithm and Applications

Neural network Architectures

Neural networks can have various structures, but some common ones include:

- **Feedforward neural networks**
- **Recurrent neural networks (RNNs)**
- **Convolutional Neural Networks (CNNs)**

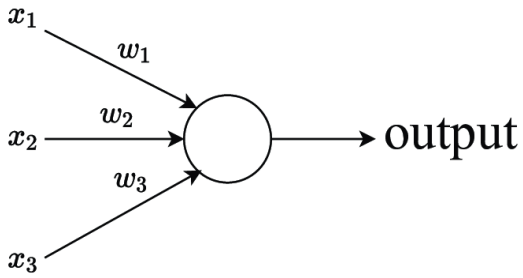
Neural network Architectures…

Feedforward neural networks:

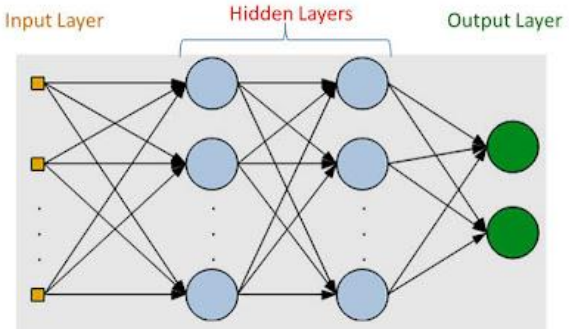
- It allow signals to travel one way only; from input to output.
- Tend to be straight forward networks that associate inputs with outputs.
- Activation flows from input layer to output layer.

There are **two types of feedforward neural network structures**:

➔ Single-layer perceptron



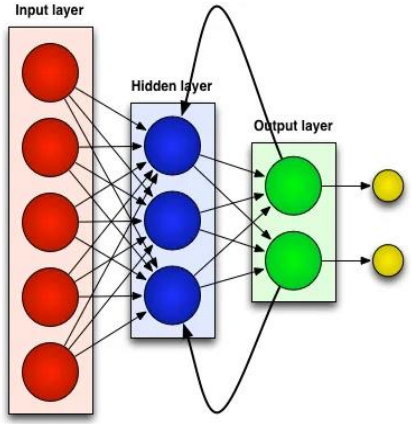
➔ Multi-layer perceptron



Neural network Architectures …

Recurrent neural networks (feedback):

- ➔ A recurrent neural network distinguishes itself from a feedforward neural network in that it has at least one feedback loop.
- ➔ A recurrent network may consist of a single layer of neurons with each neuron feeding its output signal back to the inputs of all the other neurons.

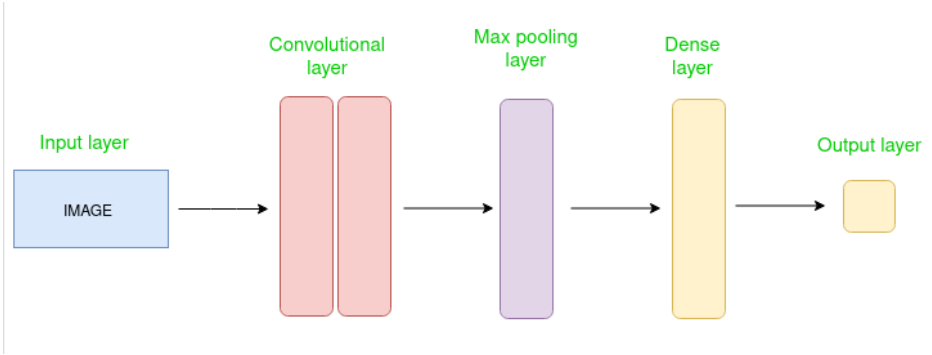


Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point

Neural network Architectures …

Convolutional Neural Networks (CNNs):

- ➔ CNNs are specifically designed for processing grid-like data, such as images.
- ➔ They use convolutional layers to automatically and adaptively learn patterns from the input data.
- ➔ CNNs excel at image recognition, object detection, and feature extraction.
- ➔ Popular architectures include **AlexNet**, **VGG**, **ResNet**, and **Inception**.



The perceptron, backpropagation and its variants

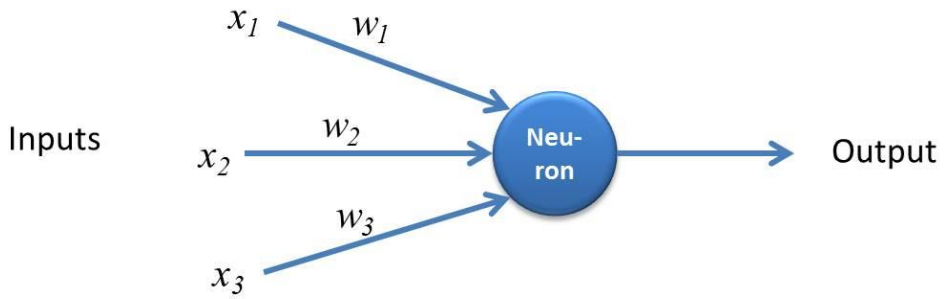
The perceptron:

- The perceptron is the simplest model of a neuron that illustrates how a neural network works.
- The perceptron is a machine learning algorithm developed in 1957 by Frank Rosenblatt and first implemented in IBM 704.
- A perceptron is a type of artificial neural network that is used for supervised learning tasks.
- The **Perceptron** is a linear machine learning algorithm for binary classification tasks.
- The **perceptron** is made up of inputs x1, x2, …, xn their corresponding weights w1, w2, …, wn. A function known as activation function takes these inputs, multiplies them with their corresponding weights and produces an output y.

The perceptron…

How the perceptron Works?

- The perceptron is a network that takes a number of inputs, carries out some processing on those inputs and produces an output as can be shown in Figure.

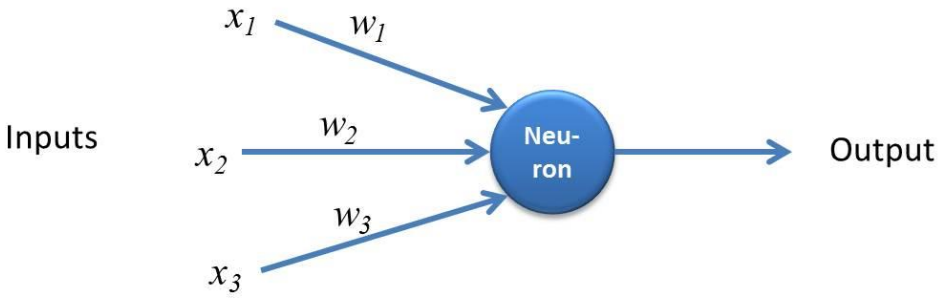


The perceptron…

How the perceptron Works?

- In the figure, the perceptron has three inputs *x1*, *x2* and *x3* and one output.
- The importance of this inputs is determined by the corresponding weights *w1*, *w2* and *w3* assigned to this inputs.

- Output is **0** if the sum is below certain threshold or **1** if the output is above certain threshold.
- This threshold could be a real number and a parameter of the neuron. Since the output of the perceptron could be either **0** or **1**, this perceptron is an example of binary classifier.



The equivalent formula

$$output = \begin{cases} 0 \ if \ \sum_j w_j x_j < threshold \\ 1 \ if \ \sum_j w_j x_j > threshold \end{cases}$$

Output = $w_1x_1 + w_2x_2 + w_3x_3 + w_nx_n$

Basic Components of perceptron

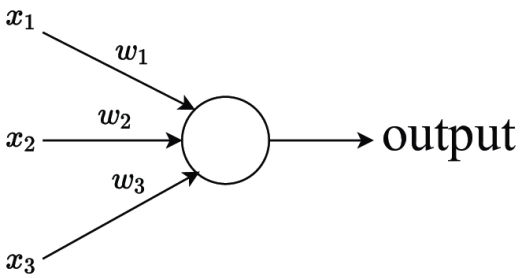
- **Input Layer:** The input layer consists of one or more input neurons, which receive input signals from the external world or from other layers of the neural network.
- **Weights:** Each input neuron is associated with a weight, which represents the strength of the connection between the input neuron and the output neuron.
- **Bias:** A bias term is added to the input layer to provide the perceptron with additional flexibility in modeling complex patterns in the input data.
- **Step function/Activation Function:** The activation function determines the output of the perceptron based on the weighted sum of the inputs and the bias term. Common activation functions used in perceptrons include the step function and sigmoid function.
- **Output**: The output of the perceptron is a single binary value, either 0 or 1, which indicates the class or category to which the input data belongs.
- **Training Algorithm:** The perceptron is typically trained using a supervised learning algorithm such as **the perceptron learning algorithm or backpropagation**.
- Overall, the **perceptron** is a simple yet powerful algorithm that can be used to perform binary classification tasks and has paved the way for more complex neural networks used in deep learning today.

Types of perceptron

- **Single Layer Perceptron model**: One of the easiest ANN(Artificial Neural Networks) types consists of a feed-forward network and includes a threshold transfer inside the model. The main objective of the single-layer perceptron model is to analyze the **linearly separable objects with binary outcomes**. A Single-layer perceptron can learn only linearly separable patterns.
- **Multi-Layered Perceptron model**: It is mainly similar to a single-layer perceptron model but has more hidden layers.
- **Forward Stage:** From the input layer in the on stage, activation functions begin and terminate on the output layer.
- **Backward Stage:** In the backward stage, weight and bias values are modified per the model's requirement. The backstage removed the error between the actual output and demands originating backward on the output layer.
- A multilayer perceptron model has a greater processing power and can process linear and non-linear patterns. Further, it also implements logic gates such as AND, OR, and XOR.

Single Layer Perceptron

- The single layer perceptron provides only one trainable weight layer.
- Connections with trainable weights go from the input layer to an output neuron, which returns the information 1 trainable layer whether the pattern entered at the input neurons was recognized or not.
- Thus, a single layer perceptron (abbreviated SLP) has only one level of trainable weights.



Perceptron Learning Algorithm

Basic Algorithm

1. While error ≠ 0
2. For each row
3. Calculate output
4. Calculate error(correct-prediction)
5. If error > 0
6. For each weight
7. Update the weights

Perceptron Learning Algorithm

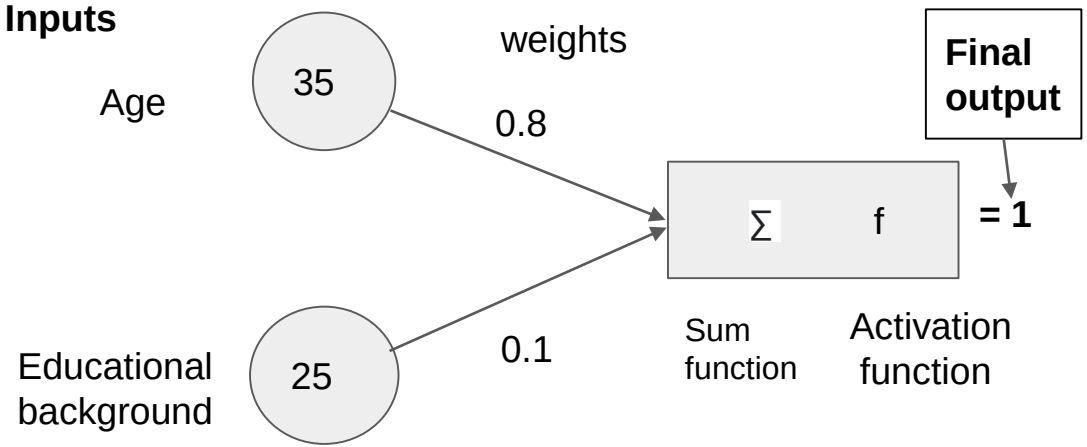
Code fragment:

```
[ ]  def train():
       total_error = 1
       while (total_error != 0):
         total_error = 0
         for i in range(len(outputs)):
           prediction = calculate_output(inputs[i])
           error = abs(outputs[i] - prediction)
           total_error += error
           if error > 0:
             for j in range(len(weights)):
               weights[j] = weights[j] + (learning_rate * inputs[i][j] * error)
               print('Weight updated: ' + str(weights[j]))
         print('Total error: ' + str(total_error))
```

Single Layer Perceptron Implementation

Problem: A person salary prediction based on education background and age using **AND operator**.

$$\text{Sum} = \sum_{i=1}^{n} xi * wi$$



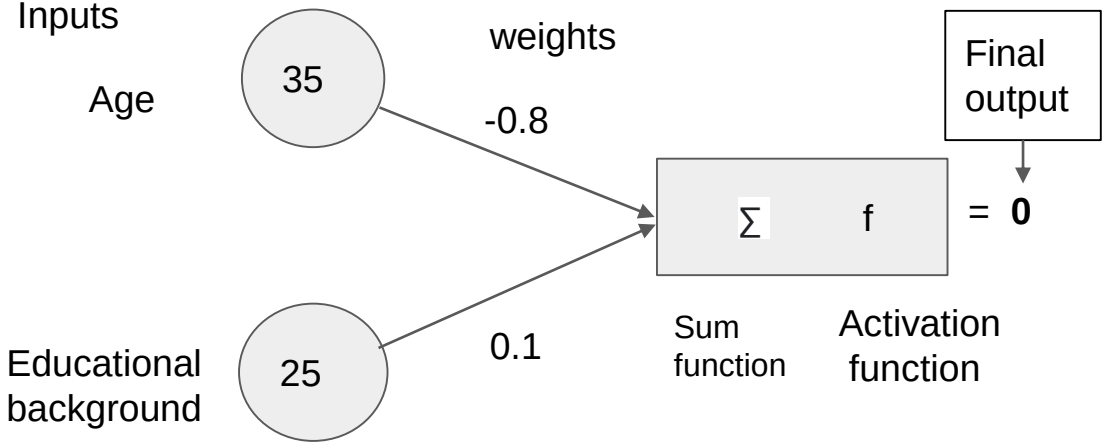
Sum = (35*0.8) + (25*0.1)
Sum = 28 + 2.5
Sum = 30.5

- Sum greater or equal to 1 is 1
- Otherwise 0

Single Layer Perceptron

Weight update: Weights can be consistently updated for better neural network learning

$$\text{Sum} = \sum_{i=1}^{n} xi * wi$$



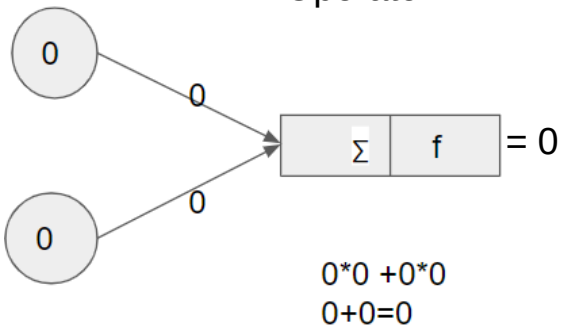
- Sum greater or equal to 1 = 1
- Otherwise 0

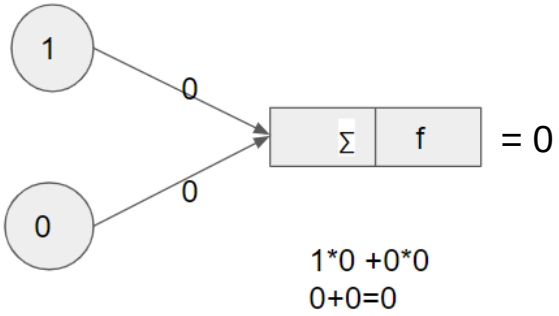
"AND" Operator

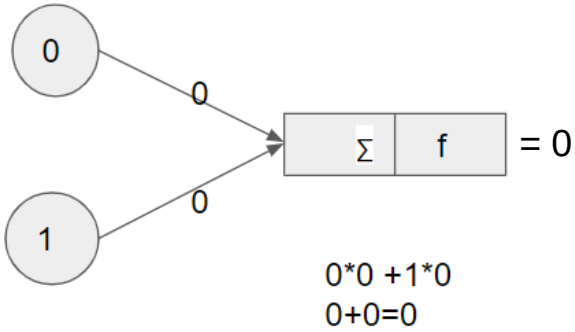
Let assume **X1** is an **age** and **X2** is also **educational background**

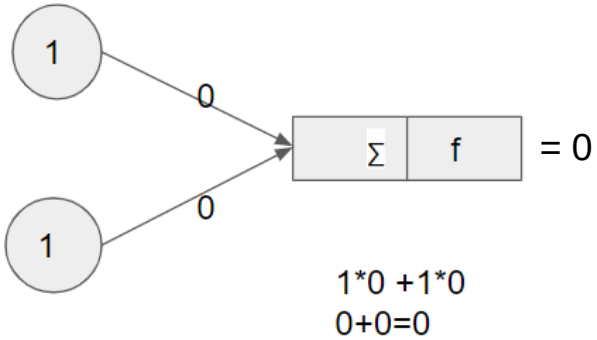
x1	x2	Class
0	0	0
0	1	0
1	0	0
1	1	1

AND Operator









x1	x2	Class
0	0	0
0	1	0
1	0	0
1	1	1

Error = correct-prediction

class	prediction	error
0	0	0
0	0	0
0	0	0
1	0	1

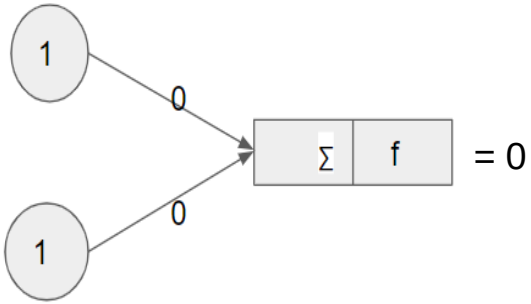
75% is correctly classified

NB: We don't consider whether a number is negative or positive. Instead calculate the absolute difference of a number

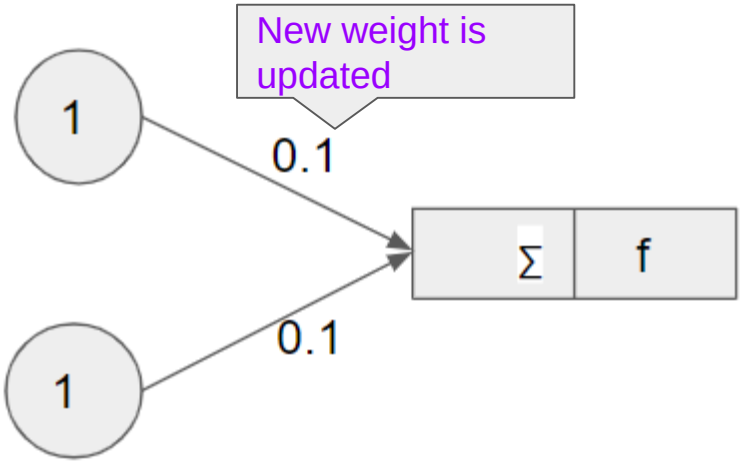
weight(n+1) = weight(n) + (learning_rate * input * error)

Error = correct-prediction

class	prediction	error
0	0	0
0	0	0
0	0	0
1	0	1

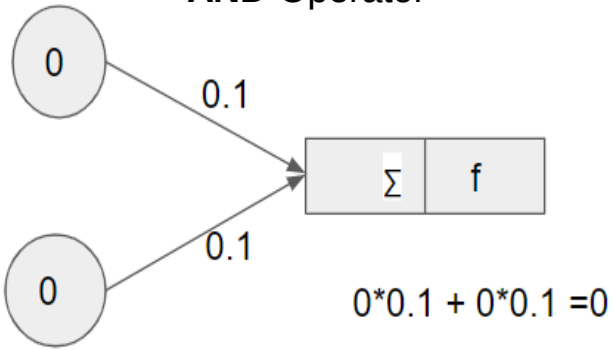


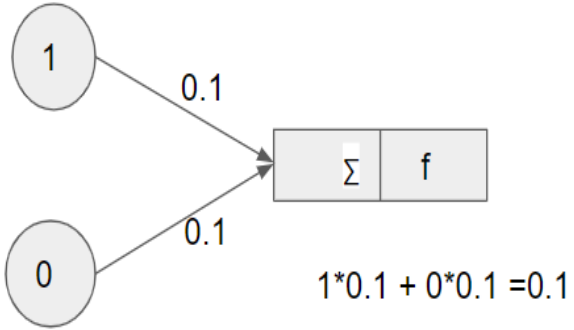
X1:
weight(n+1) = weight(n) + (learning_rate * input * error)
weight(n+1) = 0 + (0.1 * 1 * 1)
weight(n+1) = 0.1

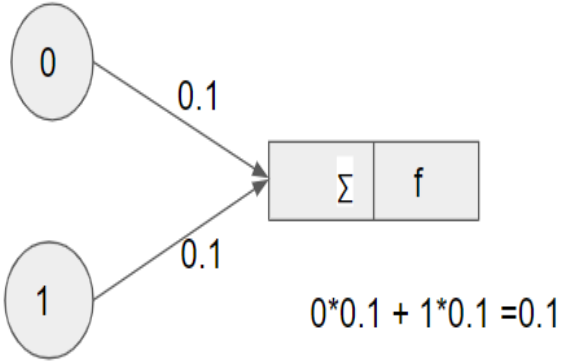


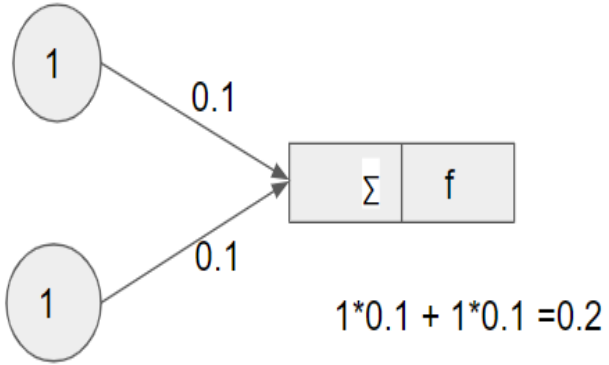
X2:
weight(n+1) = weight(n) + (learning_rate * input * error)
weight(n+1) = 0 + (0.1 * 1 * 1)
weight(n+1) = 0.1

AND Operator







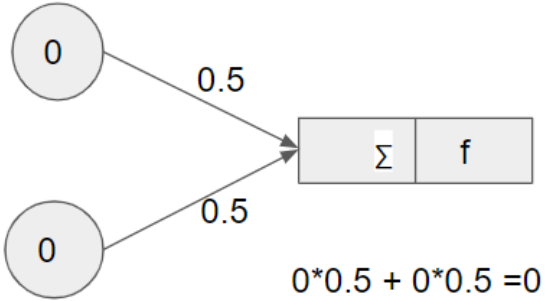


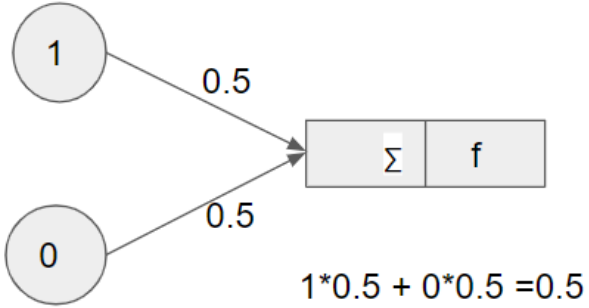
x1	x2	Class
0	0	0
0	1	0
1	0	0
1	1	1

Error = correct-prediction

class	prediction	error
0	0	0
0	0	0
0	0	0
1	0	1

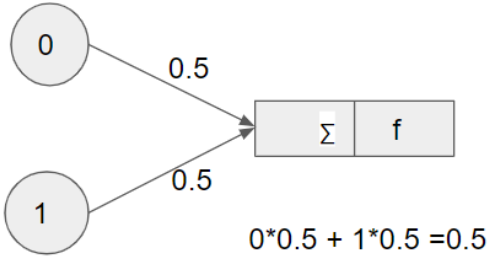
75%

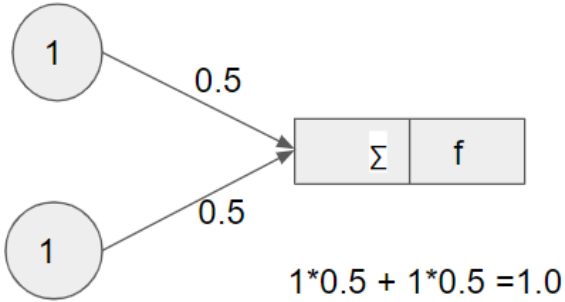




x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	1

Error = correct-prediction





class	prediction	error
0	0	0
0	0	0
0	0	0
1	1	0

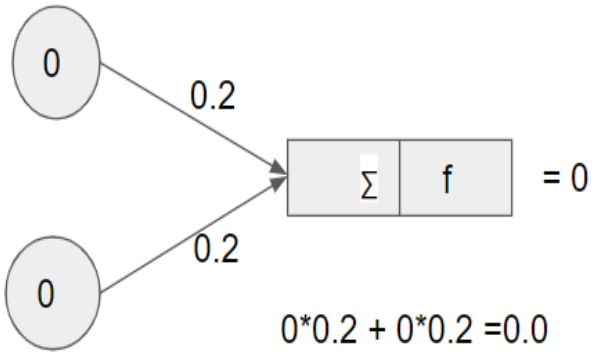
100%

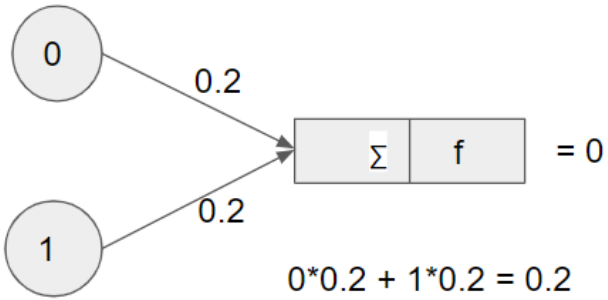
"OR" Operator

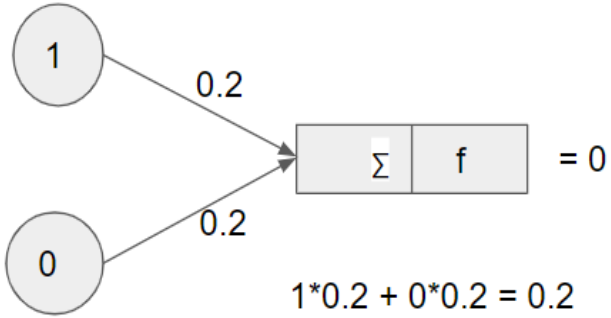
Let assume **X1** is an **age** and **X2** is also **educational background**

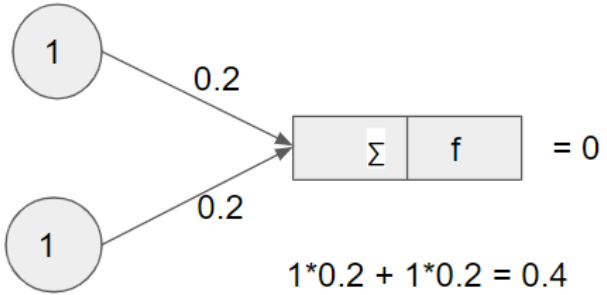
x1	x2	Class
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

OR Operator









x1	x2	Class
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Error = correct-prediction

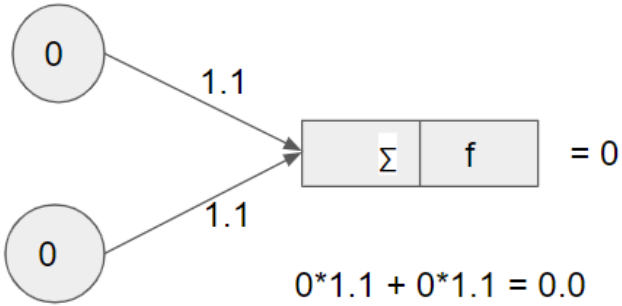
class	prediction	error
0	0	0
1	0	1
1	0	1
1	0	1

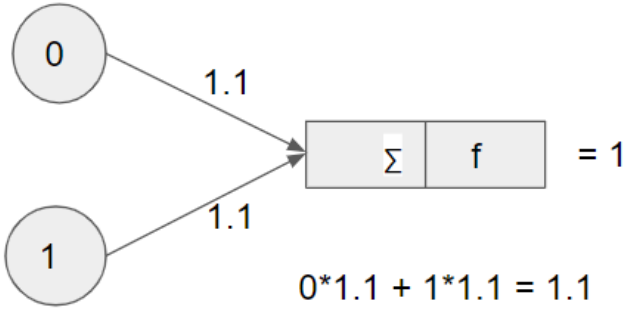
25% is correctly classified

NB: We don't consider whether a number is negative or positive. Instead calculate the absolute difference of a number

weight(n+1) = weight(n) + (learning_rate * input * error)

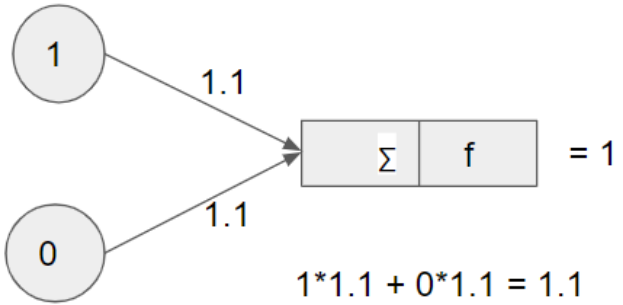
OR Operator

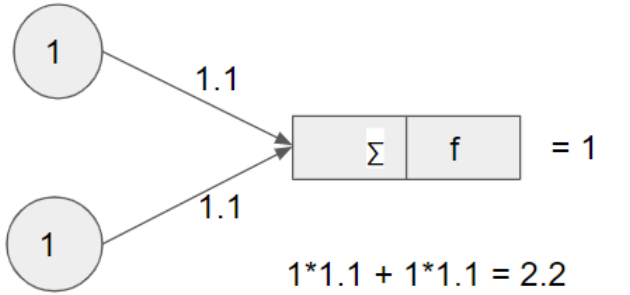




x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	1

Error = correct-prediction





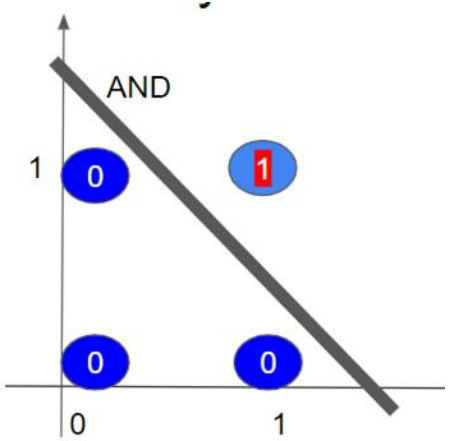
100% is correctly classified

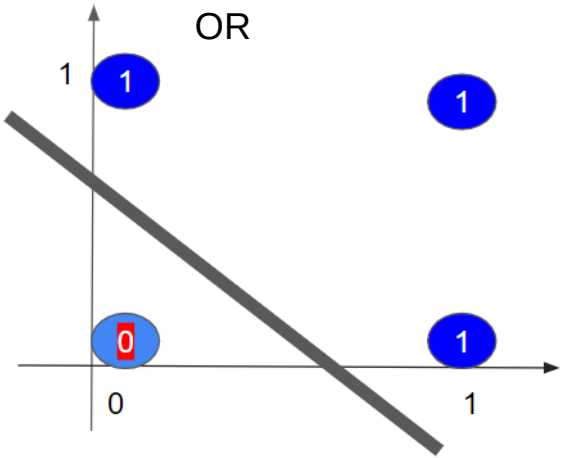
class	prediction	error
0	0	0
1	1	0
1	1	0
1	1	0

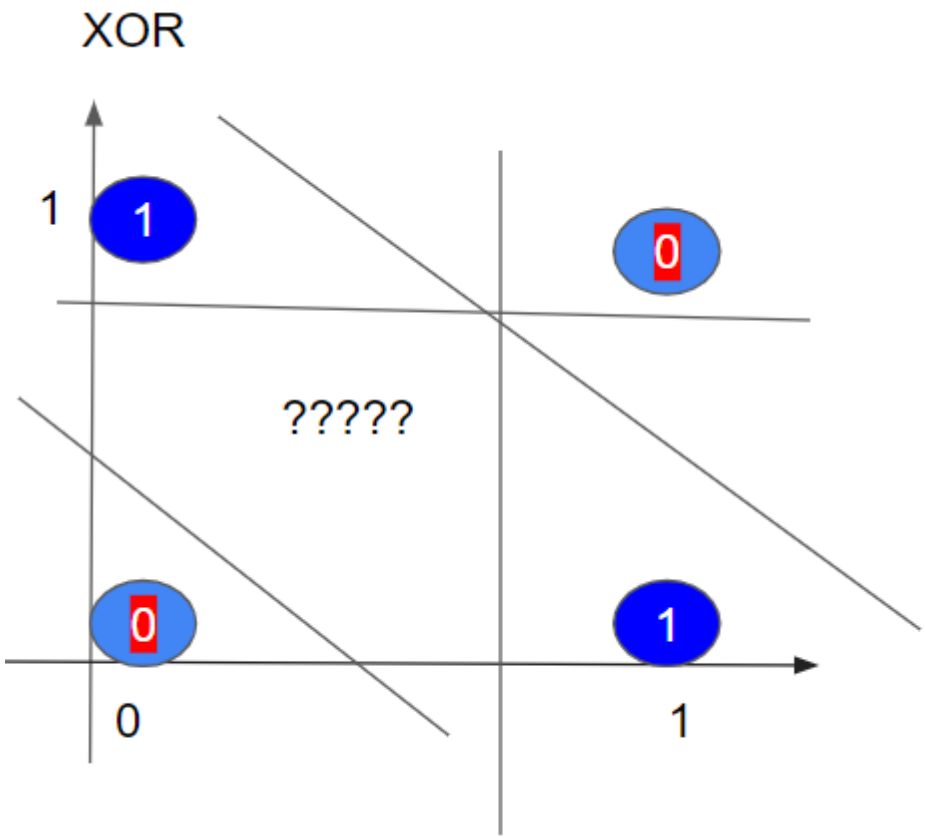
NB: We don't consider whether a number is negative or positive. Instead calculate the absolute difference of a number

weight(n+1) = weight(n) + (learning_rate * input * error)

Summary





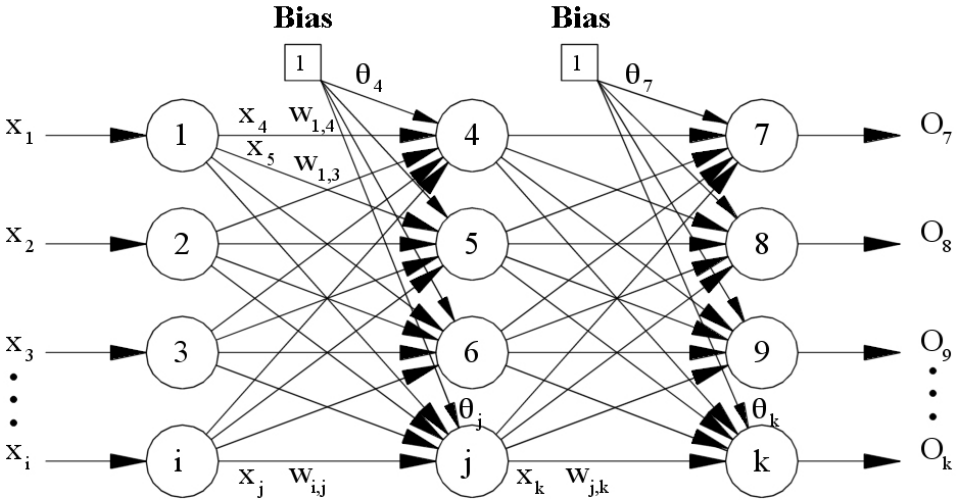


Multilayer Perceptron - Backpropagation

- The Backpropagation neural network is a multilayered, feedforward neural network and is by far the most extensively used.
- It is also considered one of the simplest and most general methods used for **supervised training** of multilayered neural networks.
- Backpropagation works by approximating the non-linear relationship between the input and the output by adjusting the weight values internally.
- Generally, the **Backpropagation network has two stages**, training and testing. During the **training phase**, the network is "shown" sample inputs and the **correct classifications(Class).**

Backpropagation…

➔ The following figure shows the topology of the Backpropagation neural network that includes **input layer**, **hidden layer** and an **output layer**. It should be noted that Backpropagation neural networks can have more than **one hidden layer.**



Backpropagation…

- The operations of the Backpropagation neural networks can be divided into two **steps**: **feedforward** and **Backpropagation.**
- **In the feedforward step,** an input pattern is applied to the input layer and its effect propagates, **layer by layer**, through the network until an **output is produced**.
- The network's **actual output value** is then compared to the **expected output**, and an error signal is computed for **each of the output nodes**.
- Since all the hidden nodes have, to some degree, contributed to the errors evident in the output layer, the output error signals are transmitted backwards from the **output layer** to **each node in the hidden layer** that immediately contributed to the output layer.
- This process is then **repeated**, layer by layer, until each node in the network has received an error signal that describes its relative contribution to the overall error.

Backpropagation…

- Once the error signal for each node has been determined, the errors are then used by the nodes to update the values for each connection weights until the network converges to a state that allows all the training patterns to be encoded.
- **The Backpropagation algorithm** looks for the minimum value of the error function in weight space using a technique called the delta rule or gradient descent.
- The weights that minimize the error function is then considered to be a **solution to the learning problem.**

Backpropagation…Algorithm

Input Layer

When a specified training pattern is fed to the input layer, the weighted sum of the input to the **jth node** in the hidden layer is given by:

$$Net_j = \sum w_{i,j} x_j + \theta_j \quad \text{..... Equation (1)}$$

Equation (1) is used to calculate the aggregate input to the neuron. The term is the weighted value from a bias node that always has an output value of 1. The bias node is considered a **"pseudo input"** to each neuron in the hidden layer and the output layer, and is used to overcome the problems associated with situations where the values of an input pattern are **zero**.

If any input pattern has zero values, the neural network could not be trained without a bias node.

Backpropagation…Algorithm

Input Layer

To decide whether a neuron should fire, the **"Net"** term, also known as the **action potential**, is passed onto an appropriate **activation function**.

The resulting value from the activation function determines the **neuron's output**, and becomes the **input value** for the neurons in the **next layer connected to it**..

Since one of the requirements for the Backpropagation algorithm is that the activation function is differentiable, a typical activation function used is the **Sigmoid equation.**

$$f(Net) = \frac{1}{1 + e^{-Net_i}}$$

Backpropagation…

Since one of the requirements for the Backpropagation algorithm is that the **activation function** is differentiable, a typical activation function used is the **Sigmoid equation:**

$$O_j = x_k = \frac{1}{1 + e^{-Net_j}} \qquad \text{..... Equation (2)}$$

It should be noted that many other types of functions can, and are, used:- hyperbolic tan being another popular choice.

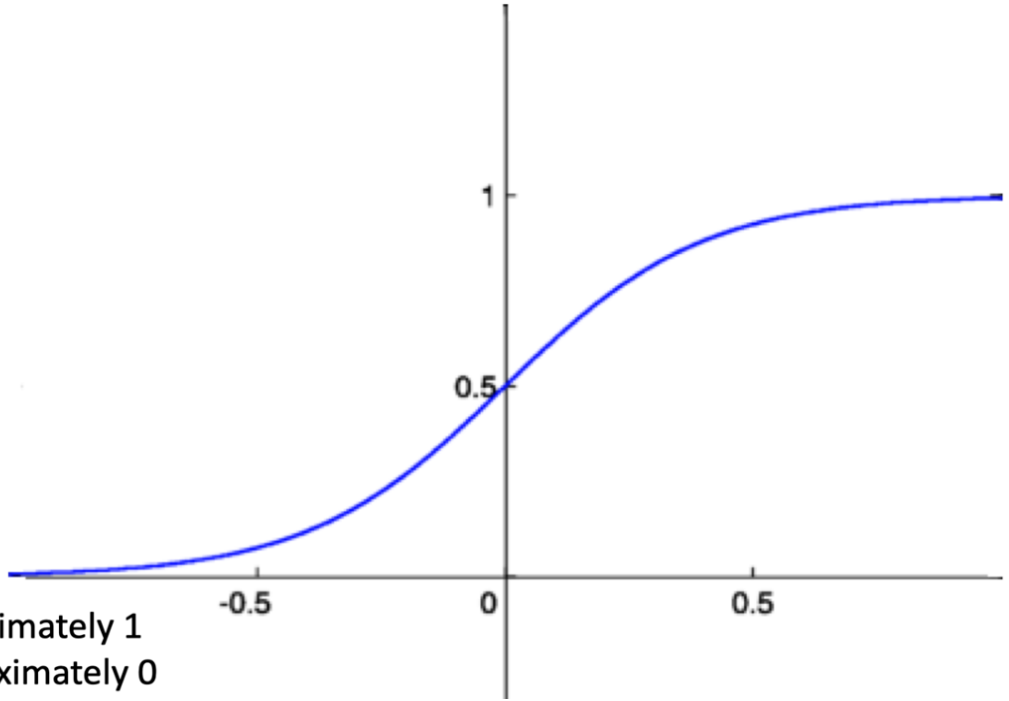
NB: e *is tht Euler's number equals to 2.71828..., where the digits go on forever in a series that never ends or repeats.*

Equations (1) and (2) are used to determine the output value for node k in the output layer.

Backpropagation…

SIGMOID FUNCTION

$$y = \frac{1}{1 + e^{-x}}$$



If X is high, the value is approximately 1
If X is small, the value is approximately 0

Backpropagation…

Error Calculations and Weight Adjustments - Backpropagation

Output Layer

If the actual activation value of the output node, **k**, is $\mathbf{O_k}$, and the expected target output for node **k** is $\mathbf{t_k}$, the difference between the actual output and the expected output is given by:

$$\Delta_k = t_k - O_k \qquad \text{..... Equation (3)}$$

The **error signal for node k** in the output layer can be calculated as:

$$\delta_k = \Delta_k O_k (1 - O_k) \qquad \text{..... Equation (4)}$$

where the $O_k(1-O_k)$ term is the **derivative of the Sigmoid function.**

Backpropagation…

Error Calculations and Weight Adjustments - Backpropagation

With the **delta rule**, the change in the weight connecting **input node j** and **output node k** is proportional to the error at node **k** multiplied by the **activation of node j**.

The formulas used to modify the weight, $w_{j,k}$, between the output node, k, and the node, j is:

$$\Delta w_{j,k} = l_r \, \delta_k \, x_k \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{ Equation (5)}$$

$$w_{j,k} = w_{j,k} + \Delta w_{j,k} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{ Equation (6)}$$

Where $\Delta w_{j,k}$ is the change in the weight between nodes **j** and k, lr is the learning rate, X_k the input variable at node K. The learning rate is a relatively small constant that indicates the relative change in weights.

Backpropagation…

Error Calculations and Weight Adjustments - Backpropagation

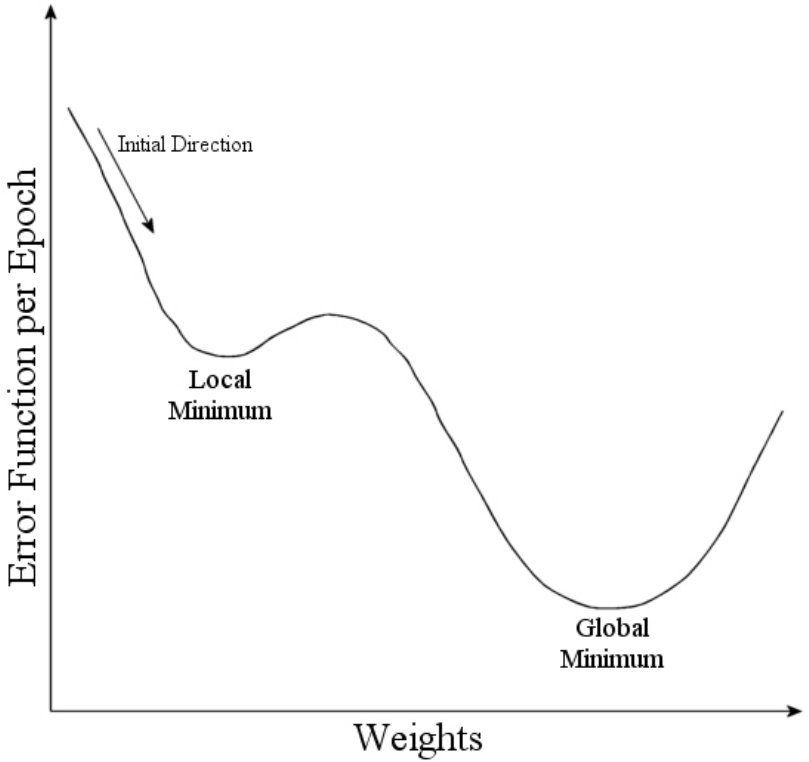
It should also be noted that, in equation (5), the $\mathbf{x_k}$ variable is the input value to the node **k**, and is the same value as the output from node **j**.

To improve the process of updating the weights, a modification to equation (5) is made:

$$\Delta w_{j,k}^{n} = l_r \delta_k x_k + \Delta w_{j,k}^{(n-1)} \mu \quad \text{..... Equation (7)}$$

Here the weight update during the **n^{th} iteration** is determined by including a momentum term (μ), which is multiplied to the (n-1)th iteration of the $\Delta w_{j,k}$ The introduction of the momentum term is used to accelerate the learning process by "encouraging" the weight changes to continue in the same direction with larger steps. Furthermore, the momentum term prevents the learning process from settling in a local minimum. by "overstepping" the small "hill". Typically, the momentum term has a value between 0 and 1.

Backpropagation…



Global and Local Minima of Error Function

Backpropagation…

Hidden Layer

The error signal for node j in the hidden layer can be calculated as:

$$\delta_k = (t_k - O_k) O_k \sum(w_{j,k} \delta_k) \quad \ldots\ldots\ldots\ldots\ldots \text{Equation (8)}$$

where the Sum term adds the weighted error signal for all nodes, **k**, in the output layer.

As before, the formula to adjust the weight, **wi,j**, between the input node, **i**, and the node, **j** is:

$$\Delta w_{i,j}^{n} = l_r \delta_j x_j + \Delta w_{i,j}^{(n-1)} \mu \quad \ldots\ldots\ldots\ldots\ldots \text{Equation (9)}$$

$$w_{i,j} = w_{i,j} + \Delta w_{i,j} \quad \ldots\ldots\ldots\ldots\ldots \text{Equation (10)}$$

Backpropagation…

The Global Error

Finally, Backpropagation is derived by assuming that it is desirable to minimize the error on the output nodes over all the patterns presented to the neural network. The following equation is used to calculate the **error function**, E, for all patterns:

$$E = \tfrac{1}{2} \sum \left(\sum \left(t_k - O_k \right)^2 \right) \qquad \ldots\ldots\ldots\ldots\ldots \text{ Equation (11)}$$

Ideally, the error function should have a value of **zero** when the neural network has been correctly trained. This, however, is numerically unrealistic.

Backpropagation… Pseudo code

```
Assign all network inputs and output

Initialize all weights with small random numbers, typically between -1 and
1

repeat

   for every pattern in the training set

      Present the pattern to the network

//       Propagated the input forward through the network:
         for each layer in the network
            for every node in the layer
               1. Calculate the weight sum of the inputs to the node
               2. Add the threshold to the sum
               3. Calculate the activation for the node
            end
         end

//       Propagate the errors backward through the network
          for every node in the output layer
            calculate the error signal
         end

         for all hidden layers
            for every node in the layer
               1. Calculate the node's signal error
               2. Update each node's weight in the network
            end
         end

//       Calculate Global Error
          Calculate the Error Function

   end

while ((maximum  number of iterations < than specified) AND
       (Error Function is > than specified))
```

THE END