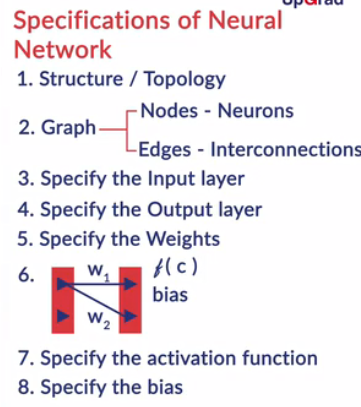
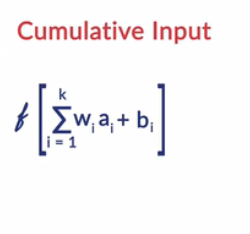
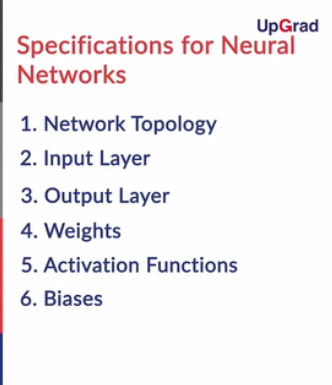


Neural Networks, or Artificial Neural Networks, are a collection of a large number of simple devices called artificial neurons. The most important takeaway is that the brain learns by **training the neurons** to behave in a certain way when given an input, such as a cat. The behaviour is basically how the **inhibitors and amplifiers**of the neurons adjust themselves.

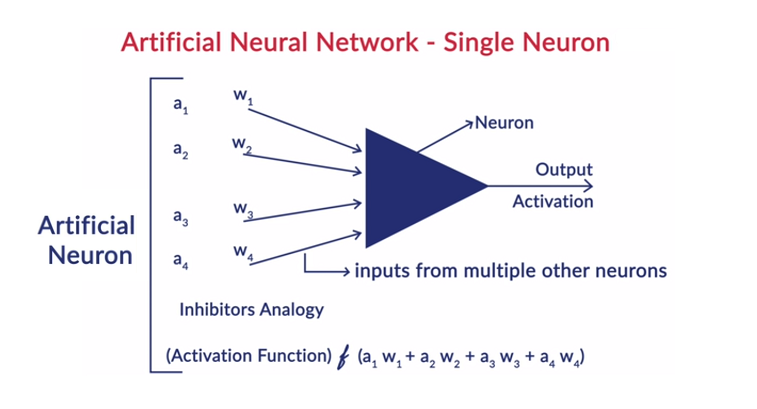
Neural networks are a collection of artificial neurons arranged in a particular topology or structure. In this segment, you will understand how an artificial neuron works i.e. how it converts inputs into outputs. You will also understand the topology or the structure of neural networks.





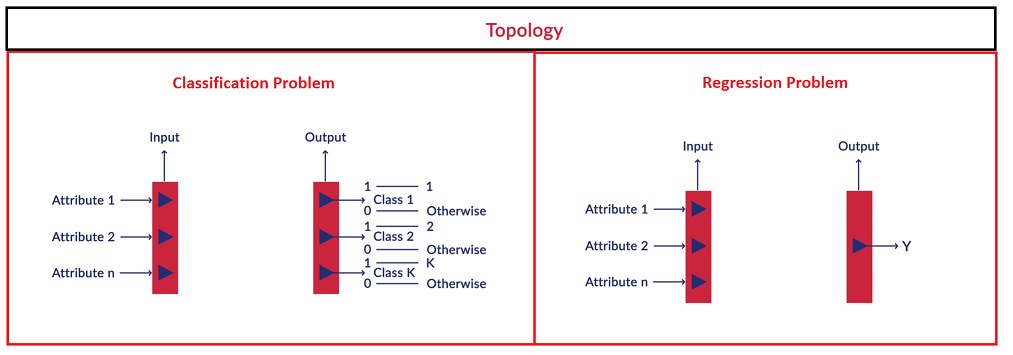


You learnt that an artificial neuron can take multiple inputs (which are outputs of other neurons). On the cumulative weighted sum of the inputs, the activation function is applied to get the output from that neuron. This output can then be fed as input to multiple other neurons. The **weights and the biases** act as **inhibitors or amplifiers** for the inputs.



**Fig:1- Artificial Neuron**

Neurons in a neural network are arranged in layers. The first and the last layer are called the input and output layers. Input layers have as many neurons as the number of attributes in the data set and output layer has as many neurons as the number of classes of the target variable (for a classification problem). For a regression problem, the number of neurons in the output layer would be 1.

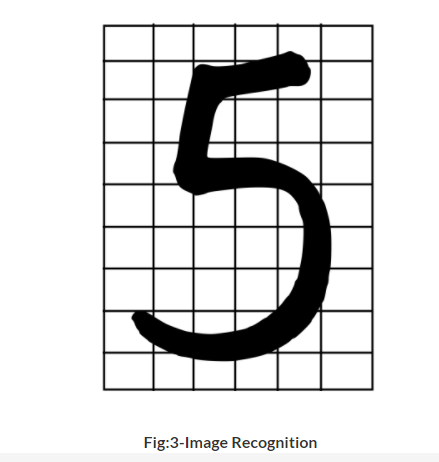


**Fig:2- Topology**

Before you can start working with neural networks you need to specify a few things:

1. **Structure** of the network
2. **Activation** Function
3. The number of neurons in the **input and output layers**

How do you decide the number of neurons in the input layer? For image recognition the inputs to the network might be the raw pixel data from a scanned, handwritten image of a digit. And we'd like the network to learn weights and biases so that the output from the network correctly classifies the digit.



**Fig:3-Image Recognition**

When you are given the image to identify the digit on it (as shown in the image above), how would you get the digit right? Let's first understand the meaning of a pixel. It is the short form for a **picture element**. When we see graphic images on digital devices, the display divides the screen into a large number of pixels arranged in rows and columns (called an array of pixel elements). Each pixel has its own address on this grid and is represented by dots or squares. Pixels build up a sample of an original image and are the smallest component of a digital image. The more pixels used to represent an image, the closer it will resemble the original.

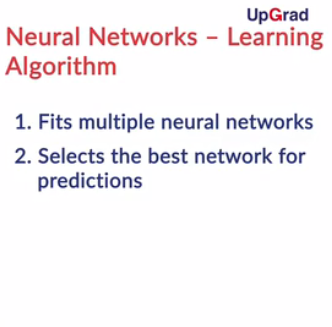
In a neural network, **each pixel**is considered as **one observation**, for example, if you have a 28 X28 pixels grid, the input layer should contain 784 neurons which will be passed to the next layer.

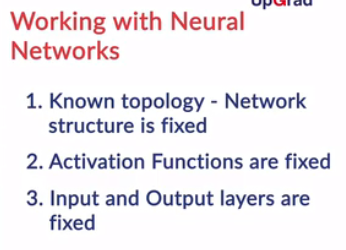
The topology or structure of a neural network defines the overall, zoomed out view of the network. There’s an input layer which takes in the input data and the output layer which generates the network’s output.

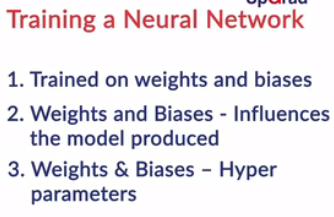
Apart from the input and output layers, a neural network becomes complete when we specify the activation function and all the weights and biases. A network thus contains:

* Input and output layers (number of neurons)
* The weights of all connections
* The bias of each neuron
* The activation function of each neuron

Each neuron i**n the input layer** takes in**one attribute as the input** and each neuron in the output layer generates **an output** of the network. For example, a binary classification task (which generates probabilities of two classes as outputs) may have 2 neurons in the output layer, each representing the probability of a data point belonging to a class.







The neural network learning algorithm, when it’s being trained on data, fits various models to the training data and selects the best model for prediction. The learning algorithm is trained with a fixed network structure, activation functions and input and output layers. It is **trained on the weights and the biases.**

This implies that the best model is the **optimal set of the weights and the biases.**The structure, activation functions and input and output layers remain the same, and thus these are the hyperparameters.

The process of finding the best combination of weights and biases is computationally very expensive. Therefore, we have to make some simplifications and make a few assumptions.

Since neural networks can have a very complex structure, and the tuning of weights and biases can be computationally very expensive, there are certain assumptions which are made to make the computation easy. . Let’s have Prof. Raghavan talk more about those assumptions.

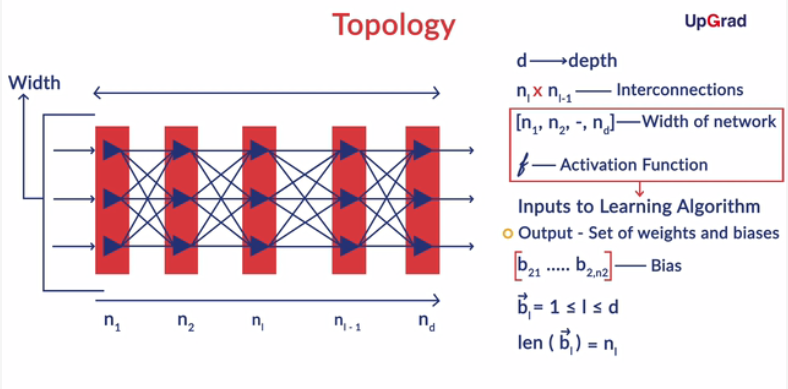
The assumptions for the most common neural network architecture are as follows:

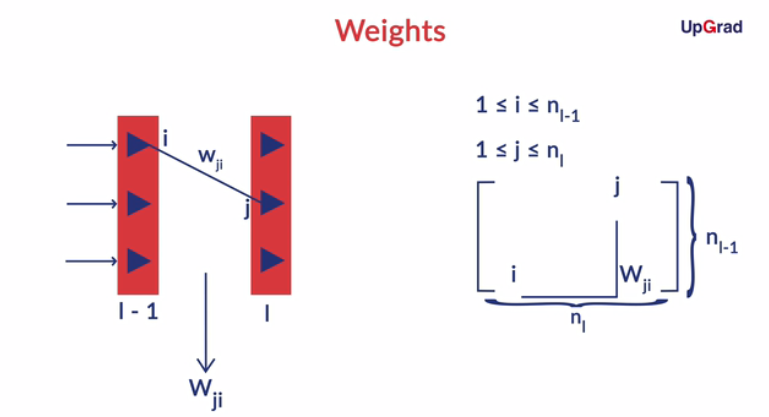
1. The neurons are **arranged in layers**and the layers are arranged **sequentially.**
2. The neurons **within a layer do not interact** with each other.
3. All the inputs enter the network through the input layer and all the outputs go out of the network through the output layer.
4. Neurons in consecutive layers are **densely connected**. This means that all the neurons in layer l_{i}are connected to all the neurons in layer l_{i+1}  and layer l_{i-1}.
5. For **every interconnection** in the neural network there is a **weight**associated with it and for **every neuron**there is a **bias** associated with it.
6. All the layers between the input and the output layers are known as the hidden layers.
7. There can only be an interconnection between neurons of adjacent/consecutive layers.
8. All neurons use same activation function

All of these assumptions have exceptions and neural networks can be made which violate these assumptions, but still perform very well on the data.

**Specifying the Hyperparameters**

To build a neural network, the hyperparameters such as the number of neurons in the input and the output layers, activation functions, the number of layers etc. are to be specified. In this segment, you will learn about hyperparamters.





To specify a network completely means to specify all of these elements:

1. The number of neurons in each layer and the number of layers
2. The activation function
3. A list of biases for each layer
4. A weight matrix for each pair of layers.

The first two elements are used by the learning algorithm to generate the neural network, i.e. they are specified by the network designer beforehand as hyperparameters. The learning algorithm then **tunes the weights and biases** to produce the best model.

Remember (the simplifying assumption) that the entire network has the same activation function.  Activation function is fixed by the network creator and then the learning algorithm uses this activation function to generate output from neurons, eventually tuning the weights and the biases.

In the next segment, you will learn more about activation functions and the most commonly used ones .

We know that a completely defined neural network contains all the following:

* The number of layers and neurons
* The weights of all connections between neurons
* The bias of each neuron
* The activation function

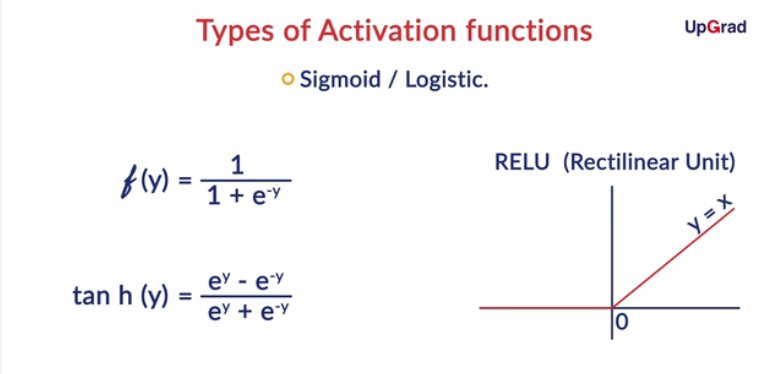
We specify the number of layers, number of neurons in the layers and the activation function. The network is then trained on the weights W and biases b, i.e. the weights and biases are found by optimising some cost function. An analogy with logistic regression is that we specify the number of inputs, outputs and the sigmoid function and train the model to find the optimal values of the coefficients.

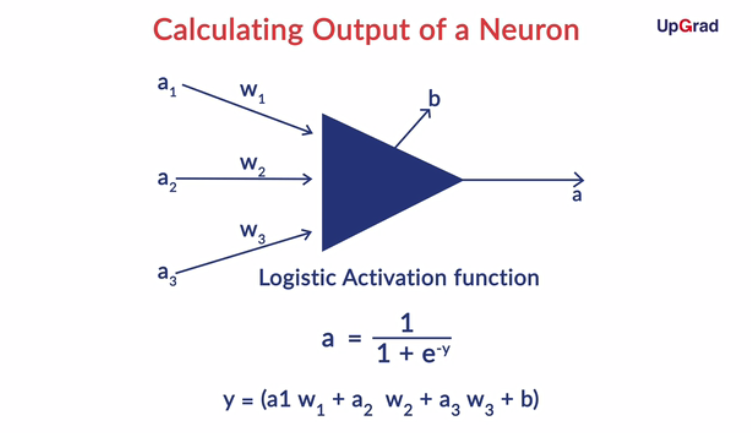
We know that a completely defined neural network contains all the following:

* The number of layers and neurons
* The weights of all connections between neurons
* The bias of each neuron
* The activation function

We specify the number of layers, number of neurons in the layers and the activation function. The network is then trained on the weights W and biases b, i.e. the weights and biases are found by optimising some cost function. An analogy with logistic regression is that we specify the number of inputs, outputs and the sigmoid function and train the model to find the optimal values of the coefficients.

In the previous segments, you learnt about the topology of neural networks, the underlying simplifying assumptions and the specifications related to the hyper-parameters. In this segment, you will understand how the output is calculated from a neuron using an activation function and the types and properties of an activation function.





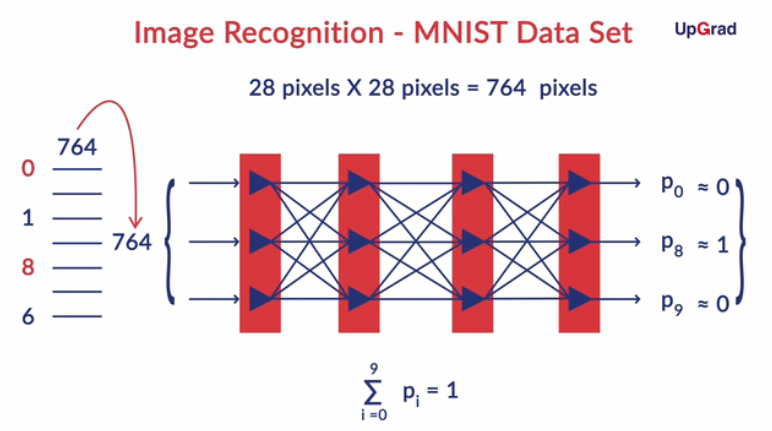
To strike the correct architecture, you need to find the best combination of the number of layers in the network and the number of neurons in each layer. Also, you must choose an activation function.

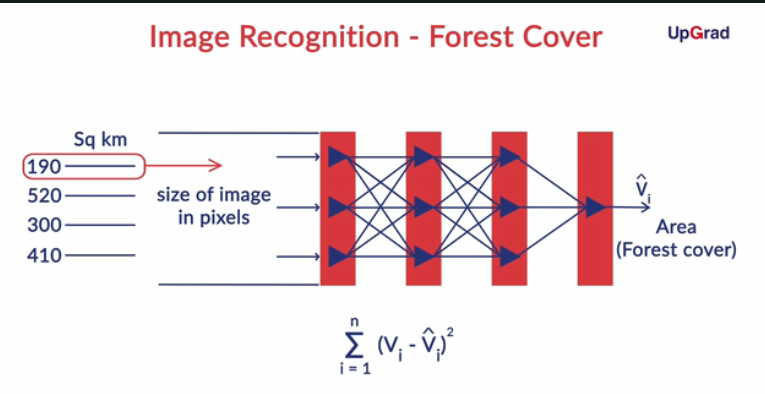
Essentially, activation function could be any function but it should have some properties. They are:

* Activation functions should be smooth i.e. they should have no abrupt changes when plotted.
* They should also make the inputs and outputs non-linear with respect to each other to some extent. This should be ensured because non-linearity helps in making neural networks more compact.

The most popular activation functions used for neural networks are:

* Logistic function - f(y) = \frac{1}{1 + e^{-y}}
* Hyperbolic tangent function - tan h(y) =\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}
* Rectilinear Unit - y = x \forall x \geq 0 \& 0 otherwise.
* The output of a neuron is basically the activation function applied to the cumulative input to that neuron. If the cumulative input to the neuron is  y = a_{1}w_{1} +a_{2}w_{2} + a_{3}w_{3} + b then using the logistic activation function the output out of that neuron will be a = \frac{1}{1 + e^{-a_{1}w_{1} -a_{2}w_{2} -a_{3}w_{3} -b}} .  Neural networks are also known as universal approximators, because with non-linear activations they can approximate any measurable function to any desired degree of accuracy.
* Interestingly, note that **each hidden layer**in a network acts as a **mini logistic regression model**if you use the sigmoid activation function. If you treat the cumulative input of the previous layer as a feature vector, the **current layer applies a sigmoid function** on it and converts it to the output the output vector.





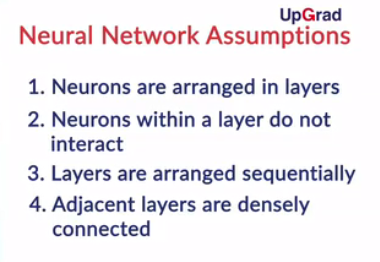
 There are various problems you face while trying to recognise handwritten text using an algorithm such as:

* Noise in the image
* Orientation of the text
* Non-uniformity in the spacing of text
* Non-uniformity in handwritings.

MNIST data set takes care of a few of problems listed above because in the images it has, the digits are written in a box and they do not touch the box. Now the only problem the network needs to take care of the non-uniformity in handwritings.  Since the images in the MNIST data set are 28 X 28 pixels, the input layer has 764 neurons (each neuron takes 1 pixel as input) and the output layer has 10 neurons each giving the probability of the input image belonging to any of the 10 classes.  The image is classified to the class represented by the neuron with the highest probability.

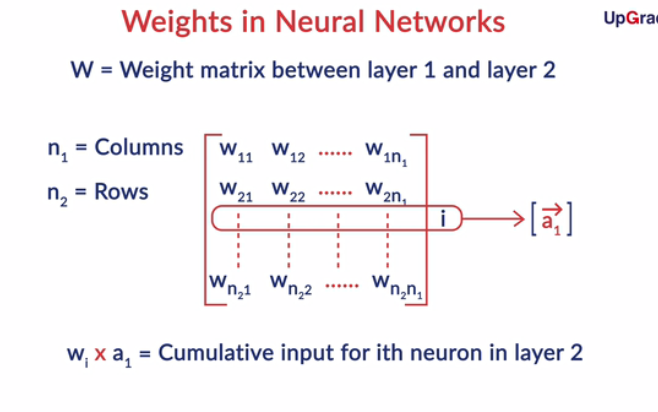
# Flow of Information in Neural Networks - Between 2 Layers

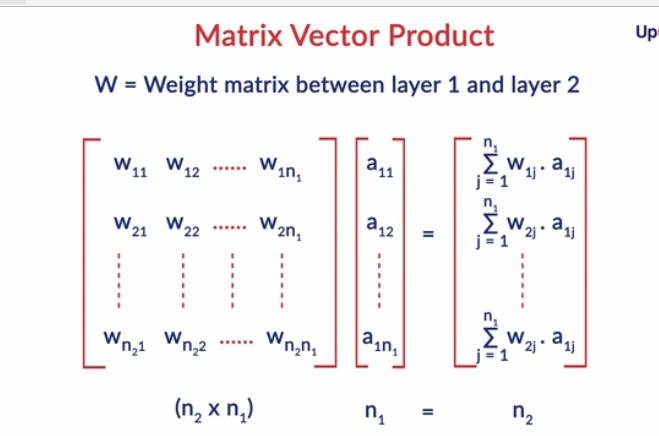
In the previous session. you learnt about the structure, topology, hyperparameters and the simplifying assumptions of neural networks. In this segment, you will understand through an example of two layers how the information transefers from one layer to the adjacent one in a neural network.  The flow of information from one layer to the next one, i.e. from left to right is called **feedforward.**

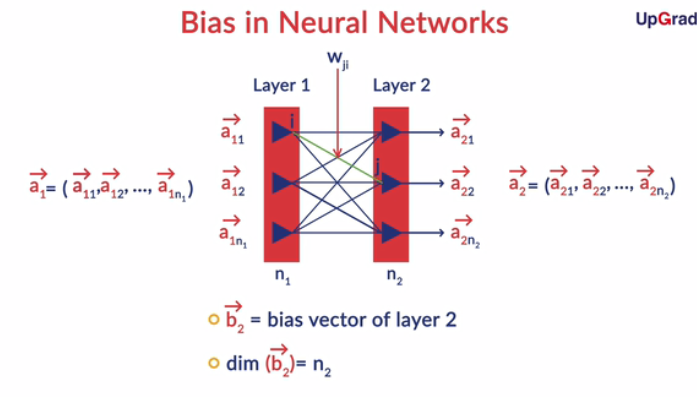


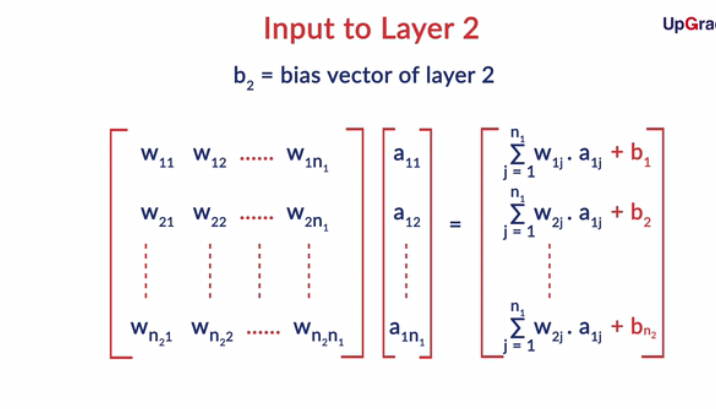
The output of a layer is also called its **activation**(since the activation function produces the output). If layer 1 has three neurons, the output / activation of  layer 1 is a: **Vector of length 3**

Each neuron has exactly one output (a number). Thus, the output of layer 1 is a vector of 3 individual outputs

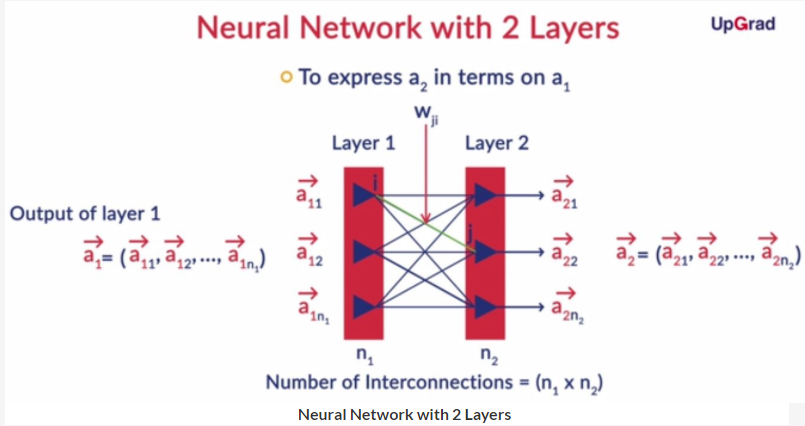








In this segment, you learnt how information flows from one layer to the other in neural networks using a simple example of 2 layers. The number of neurons in layer 1 and layer 2 are assumed to be n_{1} and n_{2} respectively. Therefore the output coming out of the first layer is the vector  a_{1} =(a_{11}, a_{12}, ..... ,a_{1n_{1}})  and the output from the second layer is the vector a_{2} = (a_{21}, a_{22}, a_{23}......a_{2n_{2}}). Also, there would be weights attached to each of the interconnection and for each pair of layers, there is a weight matrix associated which is of the order n_{2}Xn_{1}  .



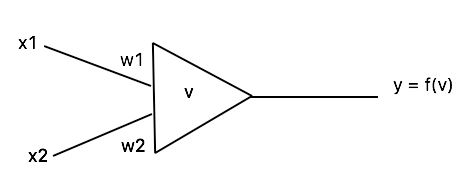
**Neural Network with 2 Layers**

Now, to compute the output of the i^{th} neuron in the second layer:

* Multiply the i^{th}  row of the weight matrix with the output of layer 1 to get the weighted sum of inputs.
* Convert the weighted sum to cumulative sum by adding the bias term associated with the i^{th} neuron to the weighted sum.
* Apply the activation function to the cumulative input to get the output of the i^{th} neuron in the second layer.

# Comprehension - Calculating Output of a Neuron

The following figure shows a single neuron which takes two logical inputs x_{1} and x_{2} each of which can take the values 0 or 1 only. The weights corresponding to them are w_{1} and w_{2} and the bias of the neuron is 0. The weighted average of the inputs is called v, which is then converted by the activation function f to the output y.



**Single Neuron**

This simple ‘network’ can act as a logical gate. For example, the AND gate returns 1 as the output only when both inputs are 1, else it returns 0.

|  |  |
| --- | --- |
| Table 1: The AND Gate | |
| Input (x_{1}, x_{2}) | Output(y) |
| (0, 0) | 0 |
| (0, 1) | 0 |
| (1, 0) | 0 |
| (1, 1) | 1 |

For an AND gate, the function f and the weights are defined as:

f(v) = 1 if v\geq 2

             0 otherwise

The weight matrix for an AND gate is w = \begin{bmatrix} 1 &1 \end{bmatrix}, which is a 1 x 2 matrix. The input vector x = \begin{bmatrix} x1 & x2 \end{bmatrix}^{T}, which is a 2 x 1 matrix.  The value of v is thus ​​​​​​wx.

Consider that the inputs x_{1} and x_{2} are two attributes and the output y is a logical value - 0 or 1. The task of the network is to implement a logical gate.

# Information Flow in Neural Networks

In the previous segment, you learnt how information is passed between two layers using an example of 2 layers. You will now understand how it is passed in a complete neural network.

The process of computing output from the input is called **feedforward**or **inference.**To summarize the feedforward process, we use the following to compute the output of a network (obviously assuming that we have the values of weights and biases which are calculated using **backpropagation**which you'll study later):

* a_{0} is the input to the network
* The network has h hidden layers and the h+1^{th} layer is the output layer.
* The weight matrix for layer l is of the order n_{l}Xn_{l-1}
* The bias vector for layer l has n_{l} elements.

The output from the network will be calculated in a sequential manner. Given the input a_{0} to the network, you multiply the input vector with the matrix w_{1} and add the bias vector b_{1} to get the cumulative input z_{1}. Activation function is then applied to z_{1} to give \sigma(z_{1}) which is the output of layer 1 (or the input to layer 2).

The input to layer 2 is now a_{1}. You multiply a_{1} with w_{2}, add the bias vector b_{2}. This becomes z_{2} and upon applying activation to this you get the output of layer 2. You can continue this process till you get the output of the last layer.

# Time Series Differencing - II

So far, you’ve briefly seen how differencing can be used as an alternative method to make time series forecasts. However, an important assumption that arises during time series differencing is that differencing the time series a few times will result in it becoming stationary. Why would that happen? What if it doesn’t happen? What do you do then?

o that’s a broad intuition behind why differencing works. In a way, you can say that differencing is like differentiating. So a few levels of differencing will give you a de-trended (weakly) stationary series.

However, because of the same reason, you can say that this method of differencing will only work efficiently for series that can be approximated to low degree polynomials, such as ax2+bx+c,ax3+bx2+cx+dax2+bx+c,ax3+bx2+cx+d, etc. A sinusoidal series, for example, will not effectively become constant after two or three levels of  differencing, and hence, cannot be modelled effectively using differencing.

So now you know two methods to forecast a time series:

**ARIMA** - Difference the series. If the differenced series is stationary, model it as an ARMA process.

**Classical Decomposition** - Remove the trend and seasonality of the time series by modelling them. If the de-trended and de-seasonalised series is stationary, model it as an ARMA process.

Let’s now listen to Prof. Raghavan as he goes through the pros and cons of each of these methodologies.

Let's revisit the pros and cons of the two methods discussed above:

* **Classical Decomposition**: The obvious downside of this methodology is that you need to manually 'guess' a good trend and seasonal pattern in the data. This is often not such a big issue, though. On visualisation, most series you encounter in practice reveal such trends rather quickly.
* **Differencing**: This method can easily be automated, and that removes the onus of guessing an appropriate trend/seasonal pattern from the analyst. An important drawback of the differencing approach is that though the differencing eventually does remove non-stationarity and produce a good fit, it is hard to visualise the trend and seasonality implicit in the data. An explicit visualisation of such trends and seasonality can be critical for several applications, which is missed out in this approach.