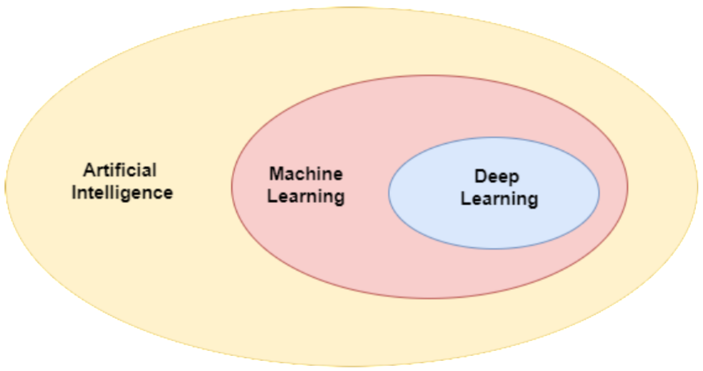
# Deep Learning With Computer Vision And Advanced NLP (DL_CV_NLP)

## Introduction to Deep Learning:

Deep Learning is a subset of Machine Learning. Deep Learning is mostly about Neural Network. The main aim of Deep Learning or Neural Network is to mimic the human Brain. Means, It makes the machine learn like human being learn. To understand what deep learning is, we first need to understand the relationship deep learning has with machine learning, neural networks, and artificial intelligence. The best way to think of this relationship is to visualize them as concentric circles:



- At the outer most ring you have artificial intelligence. One layer inside of that is machine learning. With artificial neural networks and deep learning at the center.
- Deep learning is a more approachable name for an artificial neural network. The “deep” in deep learning refers to the depth of the network. An artificial neural network can be very shallow & deep.

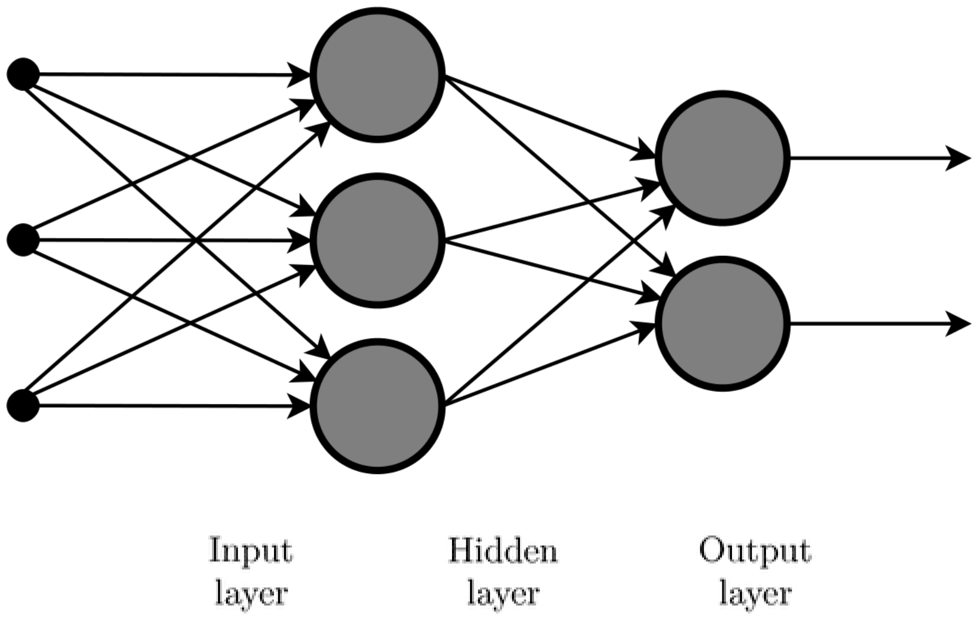
#Some requirements to learn Deep Learning:

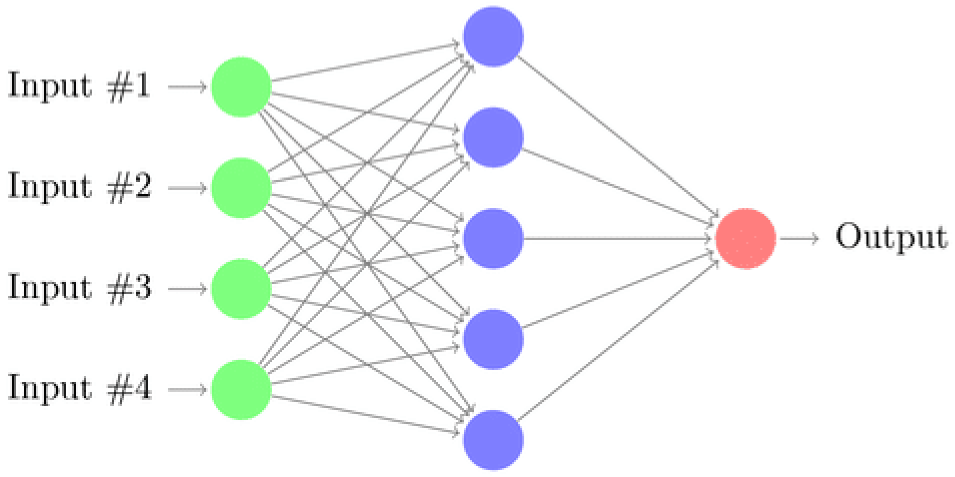
- Mathematics:
  - Functions
  - Vectors
  - Linear Algebra ->> [Metrix, Operations of Metrix]
  - Differential Calculus ->> [Gradient, Partial Derivatives, Differentiation]
  - Graphs
- Programming:
  - Python with OOPs concepts
  - Code readability
- System:
  - At least i3 or i5 processor
  - At least 8gb of RAM
  - Good internet connection
- IDE / Code Editor:
  - Google Colab
  - Paperspace
  - Pycharm
  - VS code
  - jupyter notebook

    - spyder

And very last requirement is `Dedication`

## Neural Neworks:

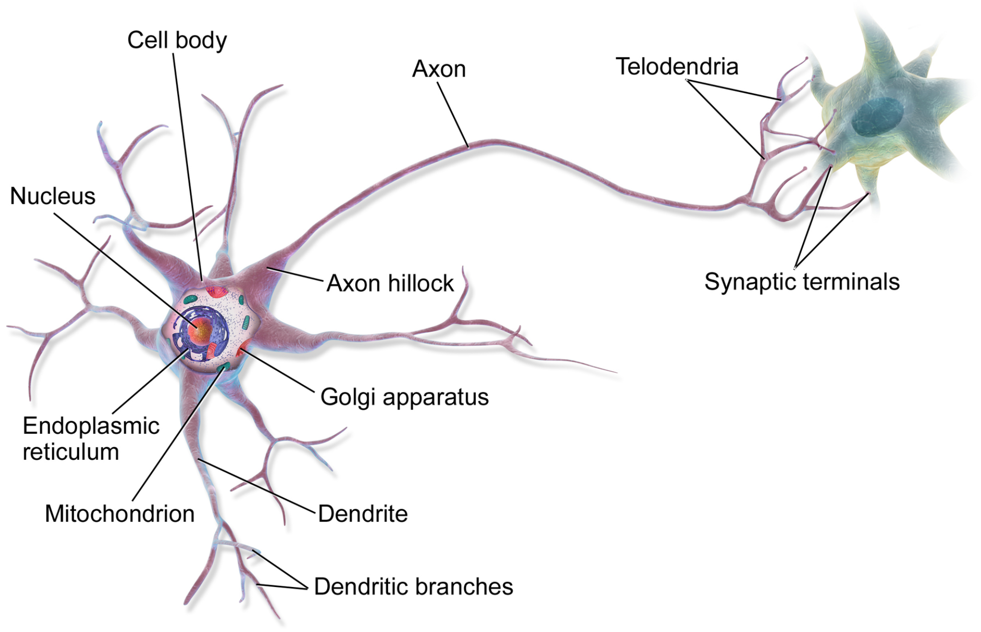




## ANN: Artificial Neural Networks:-

- Neural networks are inspired by the structure of the cerebral cortex. At the basic level is the perceptron, the mathematical representation of a biological neuron. Like in the cerebral cortex, there can be several layers of interconnected perceptrons.
- ANNs are **core of deep learning**. Hence one of the most important topic to understand.
- ANNs are **versatile, scalable and powerfull**. Thus it can tackle highly complex ML tasks like classifying images, identying object, speech recognition etc.
- ANN can be single layer or Multiple layers.
- Above the ANN has just 3 layers, The green one is Input layer , Blue one is Hidden layer and the red one is output layer.
- In each neuron except input neuron we use some kinds of activation function for activating the neuron based on some threshold. Activation function also filters out the important data.
- Here each neuron passes some information to others neuron then others neuron take this information and do some calculation and pass to other , that's how it makes the neuron networks.

## Biological Neuron:



- Biological Neuron produce short electrical impulses known as `action potentials` which travels through axons to the synapses which releases chemical signals i.e `neurotransmitters` .
- When a connected neuron recieves a sufficient amount of these neurotransmitters within a few milliseconds, it fires (or does not fires, think of a logic gate) its own action potential or elctrical impulse.
- These simple units form a strong network known as Biological Neural Network (BNN) to perform very complex computation task.

## The first Artificial Neuron:

- It was in year 1943, Artificial neuron was introduced by-
  - Neurophysiologist Warren McCulloh and
  - Mathematician Walter Pitts
- They have published their work in `McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133 (1943). https://doi.org/10.1007/BF02478259` . read full paper at link (https://www.cs.cmu.edu/~./epxing/Class/10715/reading/McCulloch.and.Pitts.pdf)
- They have shown that these simple neurons can perform small logical operation like OR, NOT, AND gate etc. These neuron only fires when they get two active inputs.

## Deep Learning classification: -

- (1) **ANN** --> Artificial Neural Network. This ANN works on all kinds of Tabular dataset (excel sheet data). Any kinds of Regression & Classification task of Machine Learning can be solved by ANN.
- (2) **CNN** --> Convolutional Neural Network. CNN works on computer vision like Images, Videos. CNN architecture works well on these kinds of Image & videos data. Some application of CNN are:-
  - (a) Image Classification --> CNN, Transfer Learning
  - (b) Object Detection --> RCNN, FAST RCNN, FASTER RCNN, SSD, YOLO, DETECTRON
  - (c) Segmentation
  - (d) Tracking

    - (e) GAN >> Generative Adversarial Network
- (3) **RNN** --> Recurrent Neural Network. RNN works whenever input data are text, time series or sequential data like Audio, Time series et cetra. Some techniques are used in RNN:-
    - RNN
    - LSTM RNN
    - Bidirectional LSTM RNN
    - Encoder, Decoder
    - Transformers
    - Bert
    - GPT1, GPT2, GPT3

**NLP - Natural Language Processing is the part of RNN.**

NLP --> Text --> Vector

- Bag of words (BOW)
- TFIDF
- word2vec
- word embedding

## Why Deep Learning?

- Computers have long had techniques for recognizing features inside of images. The results weren't always great. **Computer vision** has been a main beneficiary of deep learning. Computer vision using deep learning now rivals humans on many image recognition tasks.
- **Facebook** has had great success with **identifying faces** in photographs by using deep learning. It's not just a marginal improvement, but a game changer: "Asked whether two unfamiliar photos of faces show the same person, a human being will get it right 97.53 percent of the time. New software developed by researchers at Facebook can score 97.25 percent on the same challenge, regardless of variations in lighting or whether the person in the picture is directly facing the camera."
- **Speech recognition** is a another area that's felt deep learning's impact. Spoken languages are so vast and ambiguous. Baidu – one of the leading search engines of China – has developed a voice recognition system that is faster and more accurate than humans at producing text on a mobile phone. In both English and Mandarin.
- What is particularly fascinating, is that generalizing the two languages didn't require much additional design effort: "Historically, people viewed Chinese and English as two vastly different languages, and so there was a need to design very different features," Andrew Ng says, chief scientist at Baidu. "The learning algorithms are now so general that you can just learn."
- **Google** is now using deep learning to **manage the energy** at the company's data centers. They've cut their energy needs for cooling by 40%. That translates to about a 15% improvement in power usage efficiency for the company and hundreds of millions of dollars in savings.

`To understand more about ANN (Artificial Neural Network)` Please visit Tensorflow Playground (https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.41926&showTestData
and play with the Neural Network.

# Installation of Tensorflow, Keras, pytorch & Basic of Google Colab

## Tensorflow:

- If you are using google colab then you don't have to install tensorflow additionaly, Colab already has tensorflow pre-installed you have to just import that. To import tensorflow just write

```
import tensorflow as tf
```

- But if you are using your local system like jupyter notebook then you have to install it. To install that first of all create a virtual Environment then activate your Environment and write to your anaconda promt `pip install tensorflow` it will install latest version of tensorflow for you. To check the version of tensorflow `tf.__version__` and for keras `tf.keras.__version__`

## pytorch:

- For installing pytorch please visit their website [Pytorch (https://pytorch.org)](https://pytorch.org)

## Activate GPU on colab:

To activate GPU on colab go to the Runtime above and change the runtype type to GPU. To check your GPU just write `!nvidia-smi`

##Deep Learning Frameworks

Deep learnings is made accessible by a number of open source projects. Some of the most popular technologies include, but are not limited to, Deeplearning4j (DL4j), Theano, Torch, TensorFlow, and Caffe. The deciding factors on which one to use are the tech stack they target, and if they are low-level, academic, or application focused. Here's an overview of each:

DL4J:

- JVM-based
- Distrubted
- Integrates with Hadoop and Spark

Theano:

- Very popular in Academia
- Fairly low level
- Interfaced with via Python and Numpy

Torch:

- Lua based
- In house versions used by Facebook and Twitter
- Contains pretrained models

TensorFlow:

- Google written successor to Theano
- Interfaced with via Python and Numpy
- Highly parallel
- Can be somewhat slow for certain problem sets

Caffe:

- Not general purpose. Focuses on machine-vision problems
- Implemented in C++ and is very fast
- Not easily extensible
- Has a Python interface

> **pip install tensorflow==2.0.0**

> To run from Anaconda Prompt

> **!pip install tensorflow==2.0.0**

> To run from Jupyter Notebook

Both Tensorflow 2.0 and Keras have been released for four years (Keras was released in March 2015, and Tensorflow was released in November of the same year). The rapid development of deep learning in the past days, we also know some problems of Tensorflow1.x and Keras:

- Using Tensorflow means programming static graphs, which is difficult and inconvenient for programs that are familiar with imperative programming
- Tensorflow api is powerful and flexible, but it is more complex, confusing and difficult to use.
- Keras api is productive and easy to use, but lacks flexibility for research

#### Official docs link for DETAILED INSTALLATION STEPS (https://www.tensorflow.org/install) for Tensorflow 2

In [ ]:
```
# Verify installation
import tensorflow as tf
```

In [ ]:
```
print(f"Tensorflow Version: {tf.__version__}")
print(f"Keras Version: {tf.keras.__version__}")
```

```
Tensorflow Version: 2.5.0
Keras Version: 2.5.0
```

Tensorflow2.0 is a combination design of Tensorflow1.x and Keras. Considering user feedback and framework development over the past four years, it largely solves the above problems and will become the future machine learning platform.

> Tensorflow 2.0 is built on the following core ideas:

- The coding is more pythonic, so that users can get the results immediately like they are programming in numpy
- Retaining the characteristics of static graphs (for performance, distributed, and production deployment), this makes TensorFlow fast, scalable, and ready for production.
- Using Keras as a high-level API for deep learning, making Tensorflow easy to use and efficient
- Make the entire framework both high-level features (easy to use, efficient, and not flexible) and low-level features (powerful and scalable, not easy to use, but very flexible)

> Eager execution is by default in TensorFlow 2.0 and, it needs no special setup. The following below code can be used to find out whether a CPU or GPU is in use

## GPU/CPU Check

In [ ]:
```
tf.config.list_physical_devices('GPU')
```

Out[3]:
```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

In [ ]:
```
tf.config.list_physical_devices('CPU')
```

Out[4]:
```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]
```

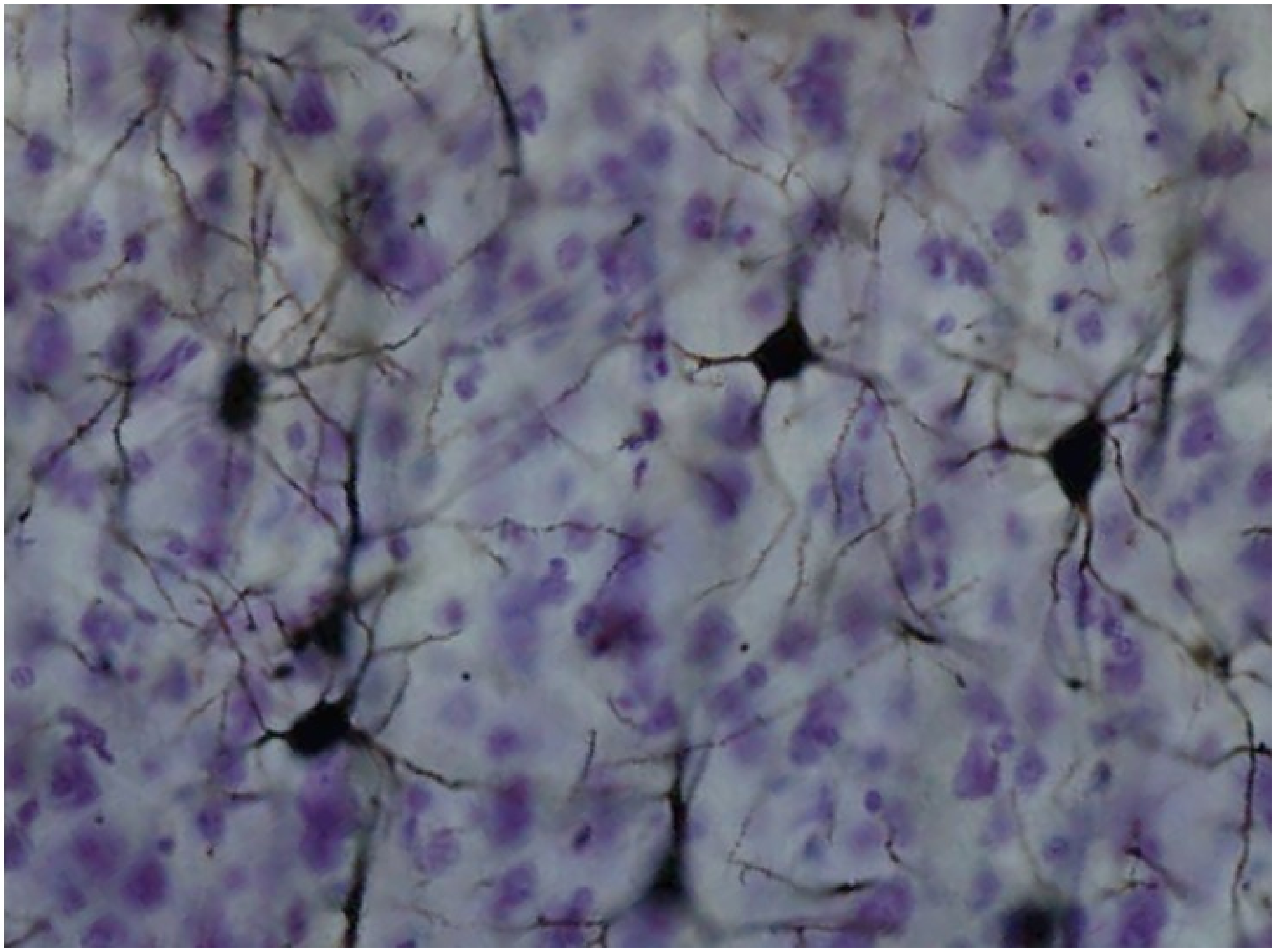
In [ ]:
```
CheckList = ["GPU", "CPU"]
for device in CheckList:
    out_ = tf.config.list_physical_devices(device)
    if len(out_) > 0:
        print(f"{device} is available")
        print("details\n",out_)
    else:
        print(f"{device} not available")
```

```
GPU is available
details
 [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
CPU is available
details
 [PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]
```

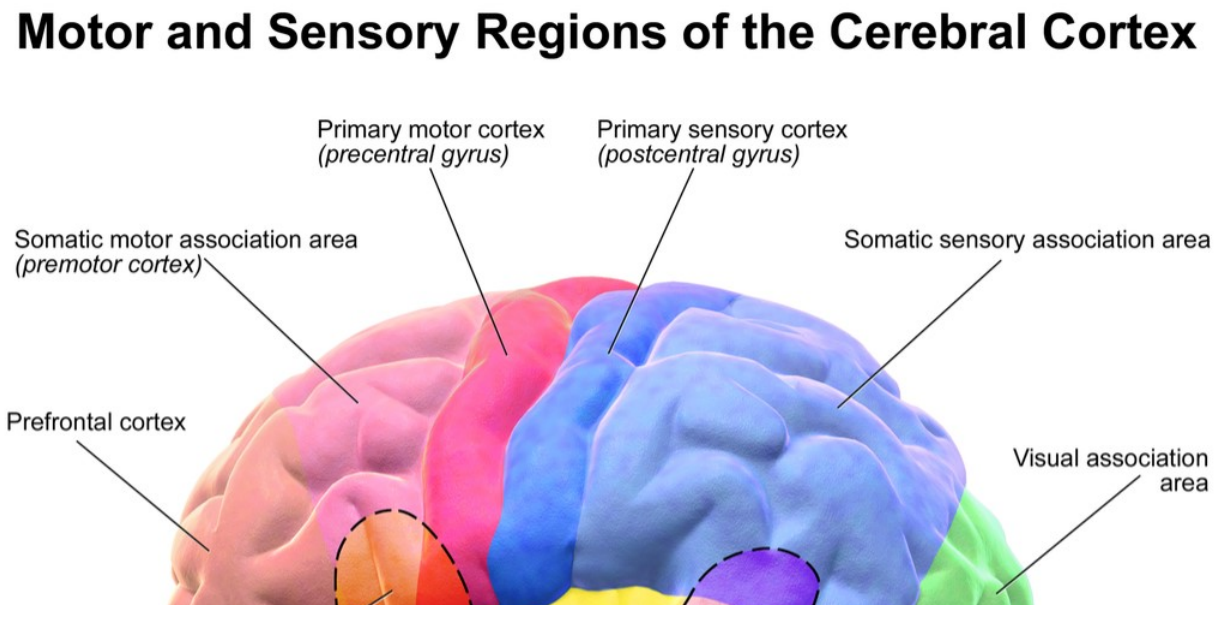
In [ ]:
```

```

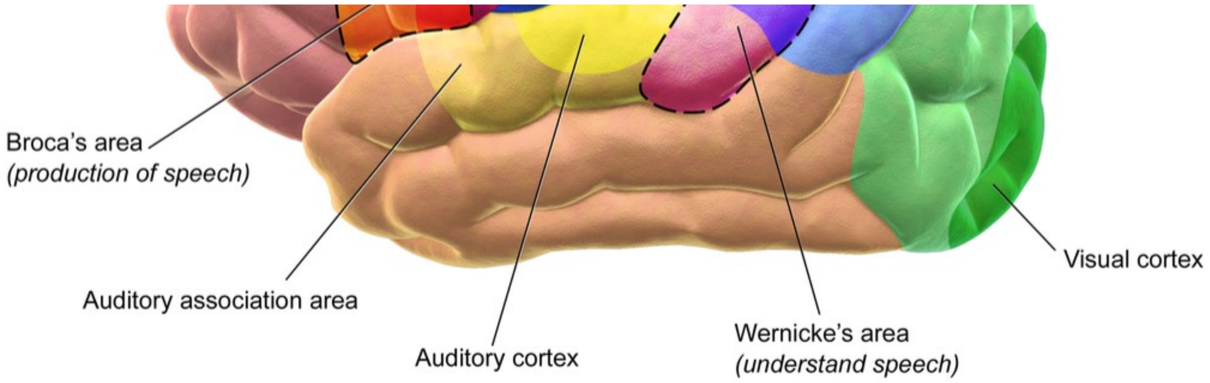
## ANN: Artificial Neural Networks

- ANNs are inspired by biological neurons found in cerebral cortex of our brain.
- The cerebral cortex (plural cortices), also known as the cerebral mantle, is the outer layer of neural tissue of the cerebrum of the brain in humans and other mammals.



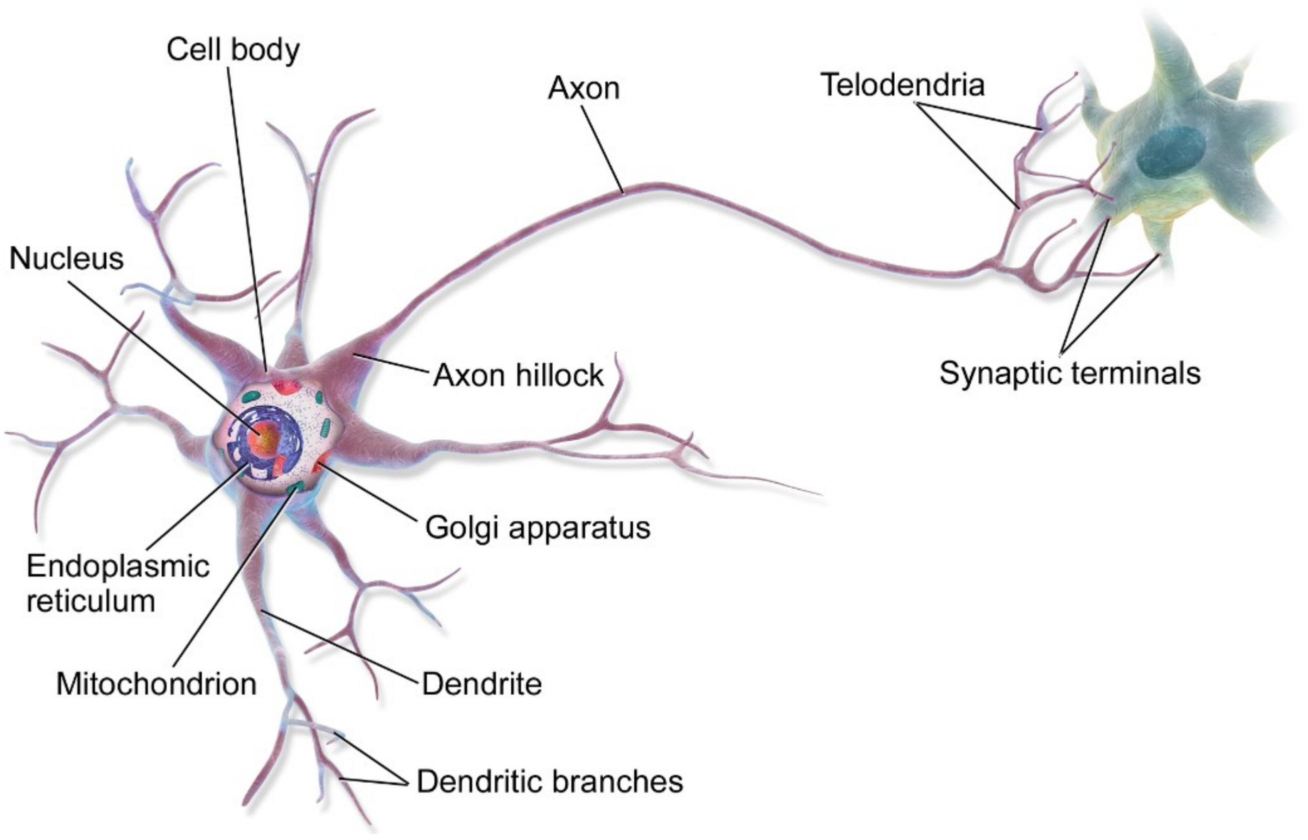
In the above diagram we can neurons of human brains, these neurons resemble the ANN.


Motor and Sensory Regions of the Cerebral Cortex
Primary motor cortex (precentral gyrus)
Primary sensory cortex (postcentral gyrus)
Somatic motor association area (premotor cortex)
Somatic sensory association area
Prefrontal cortex
Visual association area



The largest and most important part of the human brain is the cerebral cortex. Although it cannot be observed directly, various regions within the cortex are responsible for different functions, as shown in the diagram. The cortex plays a crucial role in important cognitive processes such as memory, attention, perception, thinking, language, and awareness.

## Biological Neuron

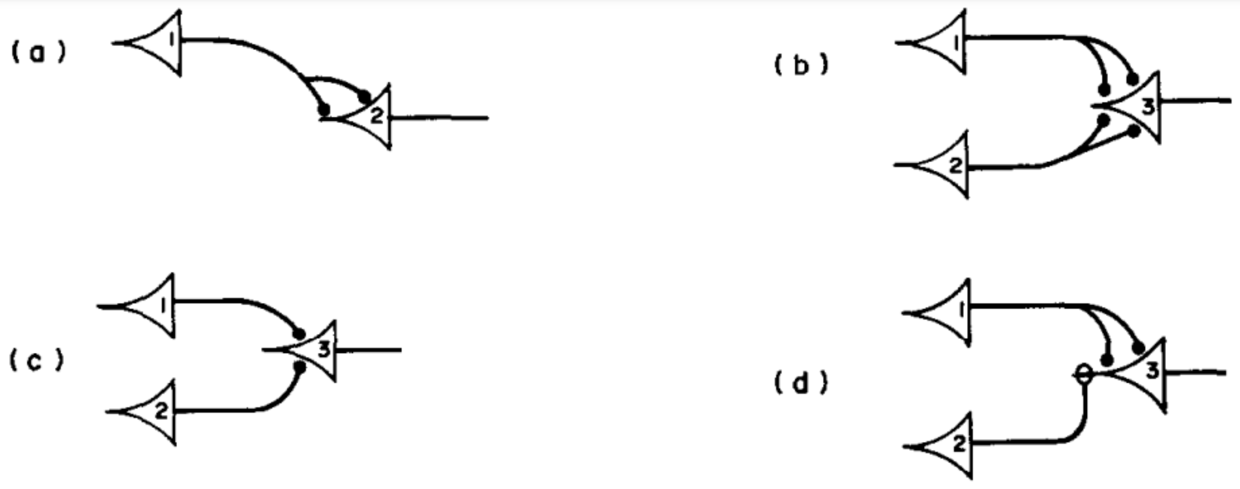


- Biological Neuron produce short electrical impulses known as action potentials which travels through axons to the synapses which releases chemical signals i.e neurotransmitters.
- When a connected neuron receives a sufficient amount of these neurotransmitters within a few milliseconds, it fires ( or does not fires, think of a NOT gate here) its own action potential or electrical impulse.
- These simple units form a strong network known as Biological Neural Network (BNN) to perform very complex computation task.
- Similar to the Biological neuron we have artificial neuron which can be used to perform complex computation task.

## The first artificial neuron

- It was in year 1943, Artificial neuron was introduced by-
  - Neurophysiologist Warren McCulloh and
  - Mathematician Walter Pitts
- They have published their work in `McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133 (1943). https://doi.org/10.1007/BF02478259` . read full paper at this link (https://www.cs.cmu.edu/~./epxing/Class/10715/reading/McCulloch.and.Pitts.pdf)
- They have shown that these simple neurons can perform small logical operation like OR, NOT, AND gate etc.
- Following figure represents these ANs which can perform (a) Buffer, (b) OR, (c) AND and (d) A-B operation

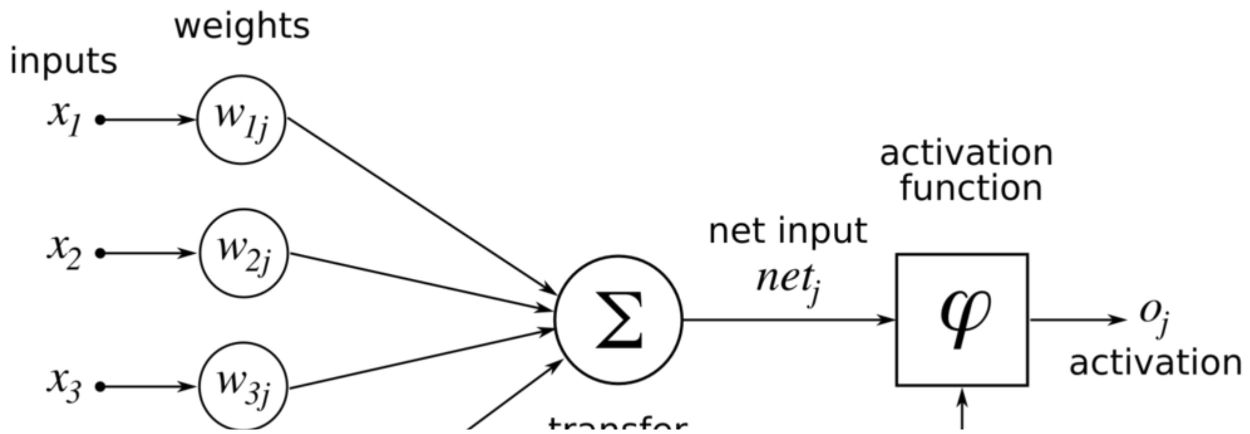
  image source (https://www.cs.cmu.edu/~./epxing/Class/10715/reading/McCulloch.and.Pitts.pdf)



- These neuron only fires when they get two active inputs.

## The Perceptron

- Its the simplest ANN architecture. It was invented by Frank Rosenblatt in 1957 and published as `Rosenblatt, Frank (1958), The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386–408. doi:10.1037/h0042519`
- It has different architecture then the first neuron that we have seen above. Its known as threshold logic unit(TLU) or linear threshold unit (LTU).
- Here inputs are not just binary.
- Lets see the architecture shown below -

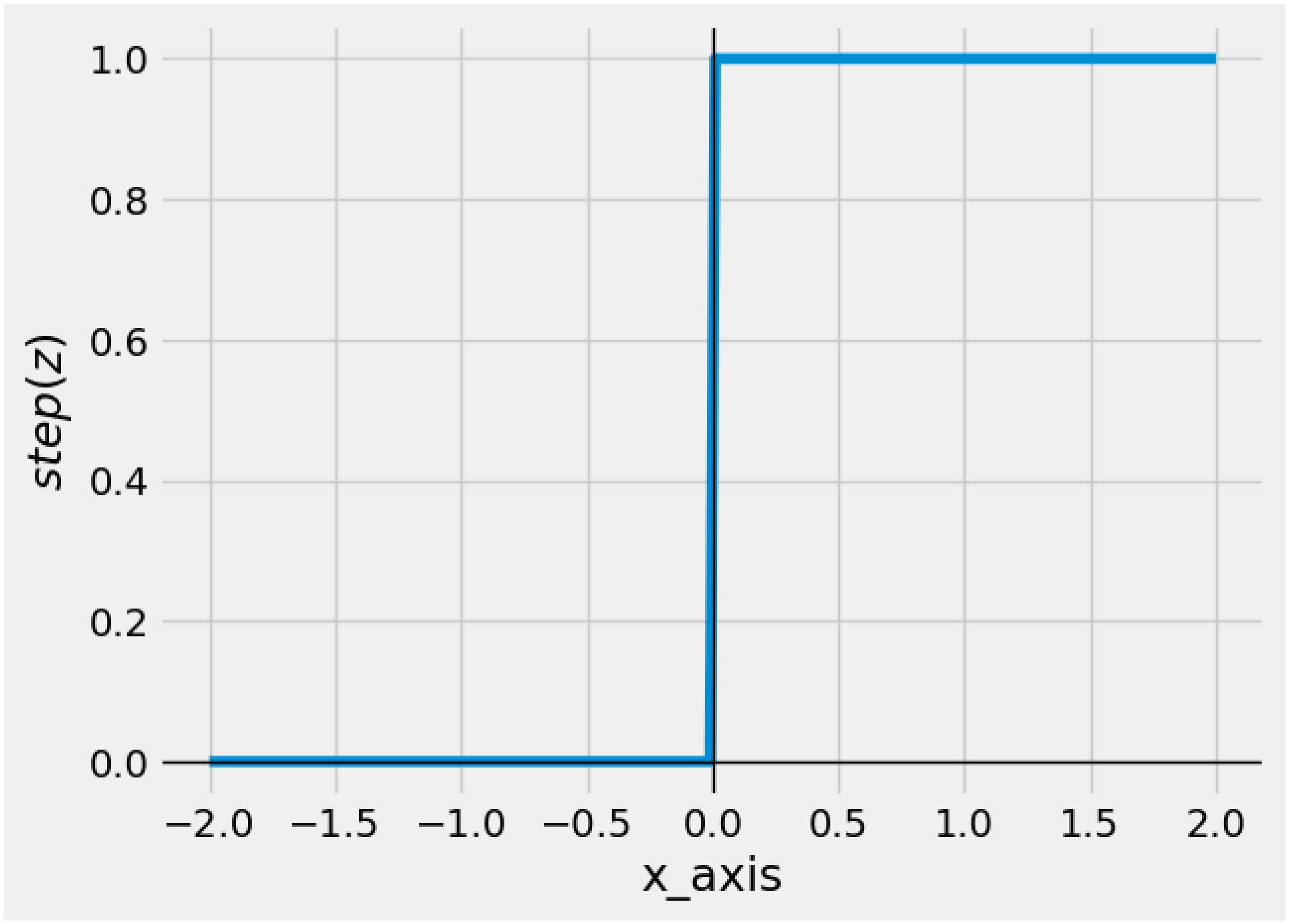


- Common activation functions used for Perceptrons are (with threshold at 0)-

$$step(z)\ or\ heaviside(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$$

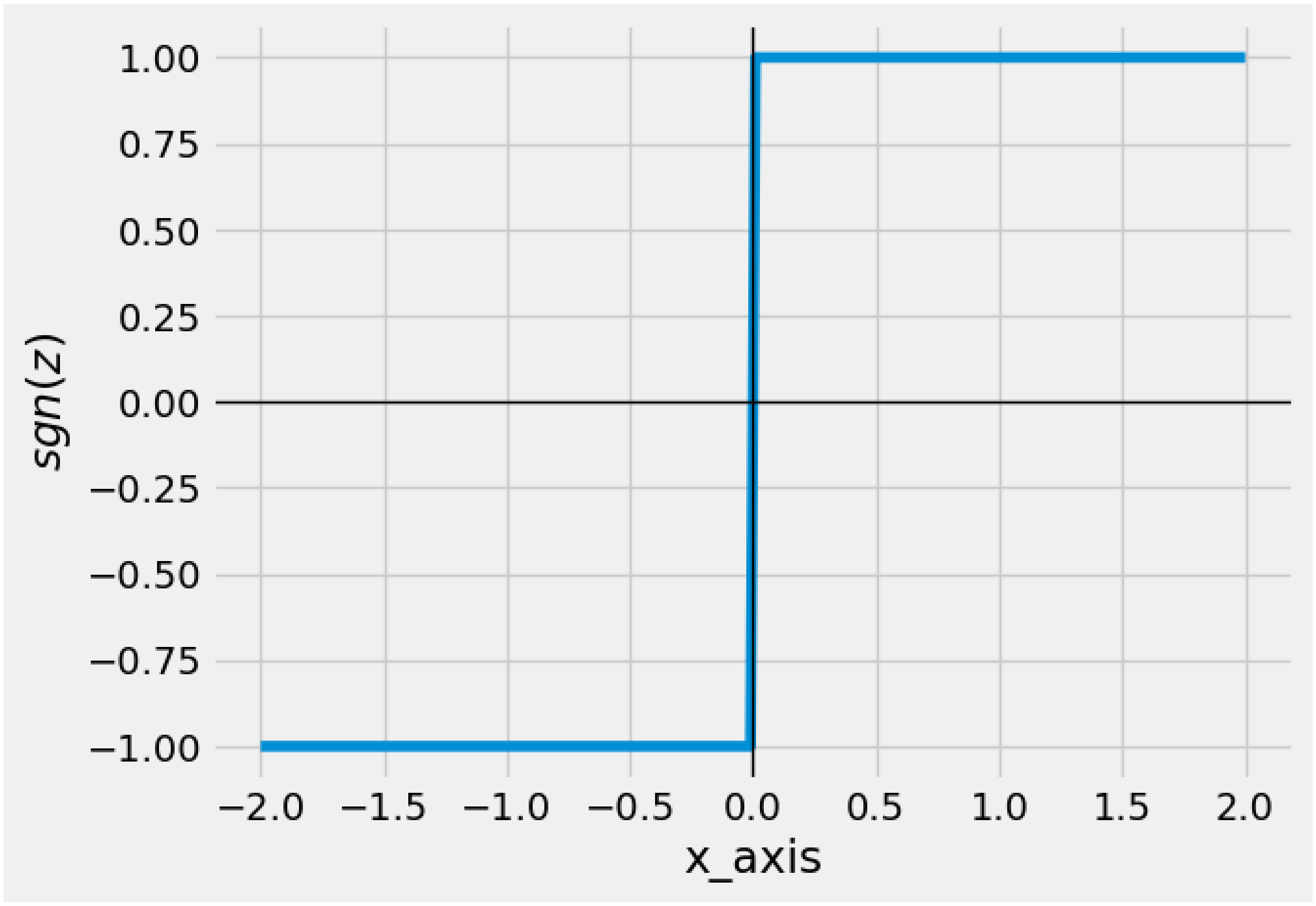
- In the further tutorials we will study more about activation functions, for now we can understand **activation functions** are mathematical equations that are applied to the output of a neural network node, in order to introduce non-linearity into the output. These functions help the neural network to learn and model complex patterns in the data. Activation functions are a key component of artificial neural networks and are used to determine the output of each neuron based on the input it receives. There are many types of activation functions, including **sigmoid, ReLU, tanh, and softmax**, each with their own unique properties and use cases.

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
plt.style.use("fivethirtyeight")

x_axis = np.linspace(-2,2,200)
step = np.where(x_axis < 0, 0, 1)

plt.plot(x_axis, step)
plt.xlabel("x_axis")
plt.ylabel(r"$step(z)$")
plt.axhline(0, color='k', lw=1);
plt.axvline(0, color='k', lw=1);
```



- $$sgn(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$$

In [2]:
```
def sgn(x):
    if x < 0:
        return -1
    elif x > 0:
        return 1
    return 0

sgn = np.array(list(map(sgn, x_axis)))

plt.plot(x_axis, sgn)
plt.xlabel("x_axis")
plt.ylabel(r"$sgn(z)$")
plt.axhline(0, color='k', lw=1);
plt.axvline(0, color='k', lw=1);
```



- These activation function are **analogous to firing of neurotransmitter** at a specific threshold
- Perceptron learning Rule:

$$w_{i,j} \leftarrow w_{i,j} + \eta(y_j - \hat{y}_j)x_i$$

  where,
  - $w_{i,j}$ : connection weight between $i^{th}$ input neuron and $j^{th}$ output neuron
  - $x_i$ : $i^{th}$ input value.
  - $\hat{y}_j$ : output of $j^{th}$ output neuron
  - $y_j$ : target output of $j^{th}$ output neuron
  - $\eta$ : learning rate
- It can also be written as for jth element of w vector

$$w_j = w_j + \triangle w_j$$

- Single TLUs (Threshold Logic Unit) are **simple linear binary classifier** hence not suitable for non linear operation.
- Rosenblatt proved that if the data is **linearly separable** then only this algorithm will converge which is known as **Perceptron learning theorem**

- Some serious weaknesses of Perceptrons was revealed In 1969 by Marvin Minsky and Seymour Papert. Not able to solve some simple logic operations like XOR, EXOR etc.
- But above mentioned problem were solved by implementing multiplayer perceptron.

## Derivation:-

Let's assume that you are doing a binary classification with class +1 and -1

Let there be decision function $\phi(z)$

which takes linear combination of certain inputs "$\mathbf{x}$"

corresponding weights "$\mathbf{w}$" and net input $\mathbf{z} = w_1x_1 + w_2x_2 + ... + w_nx_n$

So in vector form we have

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

so, $\mathbf{z = w^T x}$

Now, if

for a sample $\mathbf{x}$

$$\phi(z) = \begin{cases} +1 & \text{if } z \geq \theta \\ -1 & \text{if } z < \theta \end{cases}$$

Lets simplify the above equation -

$$\phi(z) = \begin{cases} +1 & \text{if } z - \theta \geq 0 \\ -1 & \text{if } z - \theta < 0 \end{cases}$$

Suppose $w_0 = -\theta$ and $x_0 = 1$

Then,

$$\mathbf{z}' = w_0x_0 + w_1x_1 + w_2x_2 + ... + w_nx_n$$

and

$$\phi(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

here $w_0 x_0$ is usually known as bias unit

---

## Implementation of Perceptron with Python

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import joblib
import pandas as pd
plt.style.use("fivethirtyeight")
```

In [2]:
```
class Perceptron:
  def __init__(self, eta, epochs):
    self.weights = np.random.randn(3) * 1e-4
    print(f"self.weights: {self.weights}")
    self.eta = eta
    self.epochs = epochs

  def activationFunction(self, inputs, weights):
    z = np.dot(inputs, weights)
    return np.where(z > 0 , 1, 0)

  def fit(self, X, y):
    self.X = X
    self.y = y

    X_with_bias = np.c_[self.X, -np.ones((len(self.X), 1))] # concactination
    print(f"X_with_bias: \n{X_with_bias}")

    for epoch in range(self.epochs):
      print(f"for epoch: {epoch}")
      y_hat = self.activationFunction(X_with_bias, self.weights)
      print(f"predicted value: \n{y_hat}")
      error = self.y - y_hat
      print(f"error: \n{error}")
      self.weights = self.weights + self.eta * np.dot(X_with_bias.T, error)
      print(f"updated weights: \n{self.weights}")
      print("#############\n")

  def predict(self, X):
    X_with_bias = np.c_[X, -np.ones((len(self.X), 1))]
    return self.activationFunction(X_with_bias, self.weights)
```

## AND Operation:

In [3]:
```
data = {"x1": [0,0,1,1], "x2": [0,1,0,1], "y": [0,0,0,1]}

AND = pd.DataFrame(data)
AND
```

Out[3]:
```
   x1  x2  y
0   0   0  0
1   0   1  0
2   1   0  0
3   1   1  1
```

In [4]:
```
X = AND.drop("y", axis=1)
X
```

Out[4]:
```
   x1  x2
0   0   0
1   0   1
2   1   0
3   1   1
```

In [5]:
```
y = AND['y']
y.to_frame()
```

Out[5]:
```
   y
0  0
1  0
2  0
3  1
```

In [6]:

```python
model = Perceptron(eta = 0.5, epochs=10)
model.fit(X,y)
```

```
self.weights: [ 4.49529150e-05  1.85654482e-04 -3.14676580e-05]
X_with_bias:
[[ 0.  0. -1.]
 [ 0.  1. -1.]
 [ 1.  0. -1.]
 [ 1.  1. -1.]]
for epoch: 0
predicted value:
[1 1 1 1]
error:
0   -1
1   -1
2   -1
3    0
Name: y, dtype: int64
updated weights:
[-0.49995505 -0.49981435  1.49996853]
#############

```

In [7]:

```python
model.predict(X)
```

Out[7]:

```
array([0, 0, 0, 1])
```

In [8]:

```python
model.weights
```

Out[8]:

```
array([0.50004495, 0.50018565, 0.99996853])
```

## Saving and Loading model

In [9]:

```python
import os

# saving model
dir_ = "Perceptron_model"
os.makedirs(dir_, exist_ok=True)
filename = os.path.join(dir_, 'AND_model.model')
joblib.dump(model, filename)
```

Out[9]:

```
['Perceptron_model\\AND_model.model']
```

In [10]:

```python
# Load the model from drive
loaded_model = joblib.load(filename)
result = loaded_model.predict(X)
print(result)
```

```
[0 0 0 1]
```

---

## OR Operation:

```
In [12]: data = {"x1": [0,0,1,1], "x2": [0,1,0,1], "y": [0,1,1,1]}

         OR = pd.DataFrame(data)
         OR
```

Out[12]:

| | x1 | x2 | y |
|---|---|---|---|
| **0** | 0 | 0 | 0 |
| **1** | 0 | 1 | 1 |
| **2** | 1 | 0 | 1 |
| **3** | 1 | 1 | 1 |

```
In [13]: X = OR.drop("y", axis=1)
         X
```

Out[13]:

| | x1 | x2 |
|---|---|---|
| **0** | 0 | 0 |
| **1** | 0 | 1 |
| **2** | 1 | 0 |
| **3** | 1 | 1 |

```
In [14]: y = OR['y']
         y.to_frame()
```

Out[14]:

| | y |
|---|---|
| **0** | 0 |
| **1** | 1 |
| **2** | 1 |
| **3** | 1 |

```
In [15]: model = Perceptron(eta = 0.5, epochs=10)
         model.fit(X,y)
```

```
self.weights: [-4.24035454e-05  1.13286067e-04  4.02537022e-05]
X_with_bias:
[[ 0.  0. -1.]
 [ 0.  1. -1.]
 [ 1.  0. -1.]
 [ 1.  1. -1.]]
for epoch: 0
predicted value:
[0 1 0 1]
error:
0    0
1    0
2    1
3    0
Name: y, dtype: int64
updated weights:
[ 4.99957596e-01  1.13286067e-04 -4.99959746e-01]
#############
```

---

## XOR Operation:

```
In [16]: data = {"x1": [0,0,1,1], "x2": [0,1,0,1], "y": [0,1,1,0]}

         XOR = pd.DataFrame(data)
         XOR
```

Out[16]:

|   | x1 | x2 | y |
|---|---|---|---|
| **0** | 0 | 0 | 0 |
| **1** | 0 | 1 | 1 |
| **2** | 1 | 0 | 1 |
| **3** | 1 | 1 | 0 |

```
In [17]: X = XOR.drop("y", axis=1) # axis = 1 >>> dropping across column
         X
```

Out[17]:

|   | x1 | x2 |
|---|---|---|
| **0** | 0 | 0 |
| **1** | 0 | 1 |
| **2** | 1 | 0 |
| **3** | 1 | 1 |

In [18]:
```
y = XOR['y']
y.to_frame()
```

Out[18]:

| | y |
|---|---|
| **0** | 0 |
| **1** | 1 |
| **2** | 1 |
| **3** | 0 |

In [19]:
```
model = Perceptron(eta = 0.5, epochs=50)
model.fit(X,y)
```

```
self.weights: [-1.51414242e-04  1.54706413e-04  4.27810246e-05]
X_with_bias:
[[ 0.  0. -1.]
 [ 0.  1. -1.]
 [ 1.  0. -1.]
 [ 1.  1. -1.]]
for epoch: 0
predicted value:
[0 1 0 0]
error:
0    0
1    0
2    1
3    0
Name: y, dtype: int64
updated weights:
[ 4.99848586e-01  1.54706413e-04 -4.99957219e-01]
#############
```

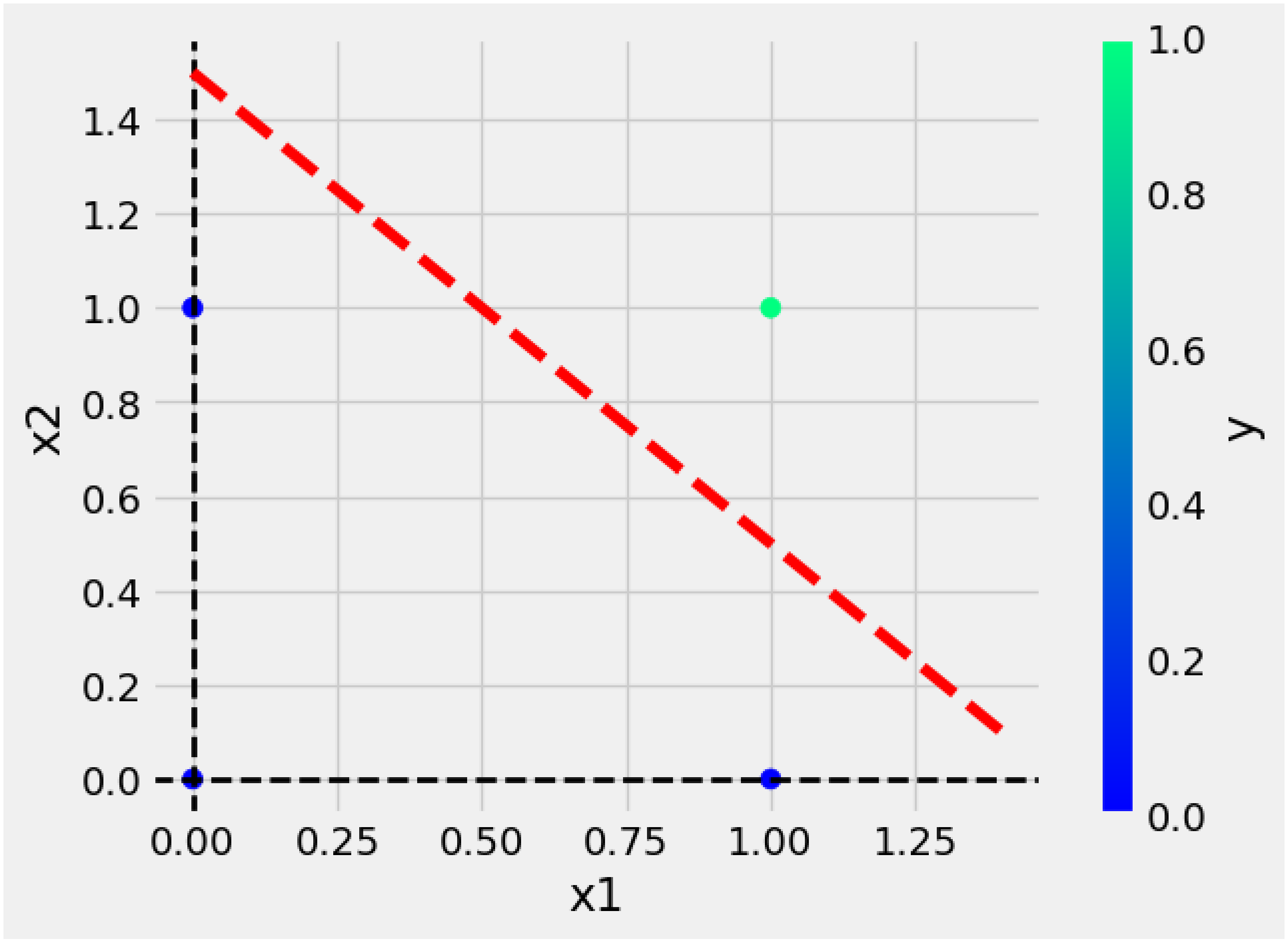
## Conclusion:

Here we can see Perceptron can only classify the linear problem like AND, OR operation because they were linear problem. But in the case of XOR it couldn't classify correctly because it was a non-linear problem. Lets see graphically.

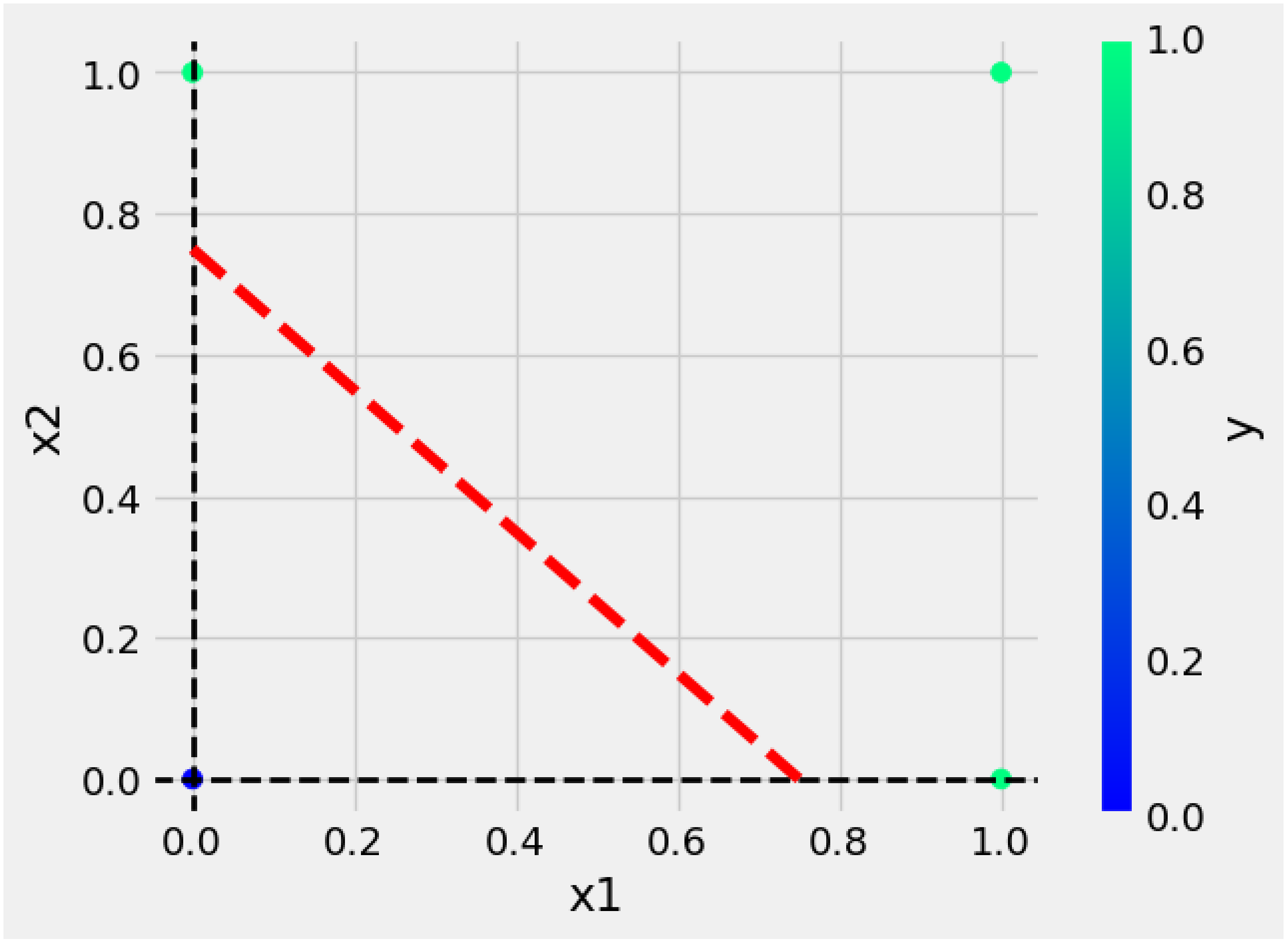
## Analysis with the graph

In [20]:
```
AND.plot(kind="scatter", x="x1", y="x2", c="y", s=50, cmap="winter")
plt.axhline(y=0, color="black", linestyle="--", linewidth=2)
plt.axvline(x=0, color="black", linestyle="--", linewidth=2)

x = np.linspace(0, 1.4) # >>> 50
y = 1.5 - 1*np.linspace(0, 1.4) # >>> 50

plt.plot(x, y, "r--")
```

Out[20]:
```
[<matplotlib.lines.Line2D at 0x22459924790>]
```

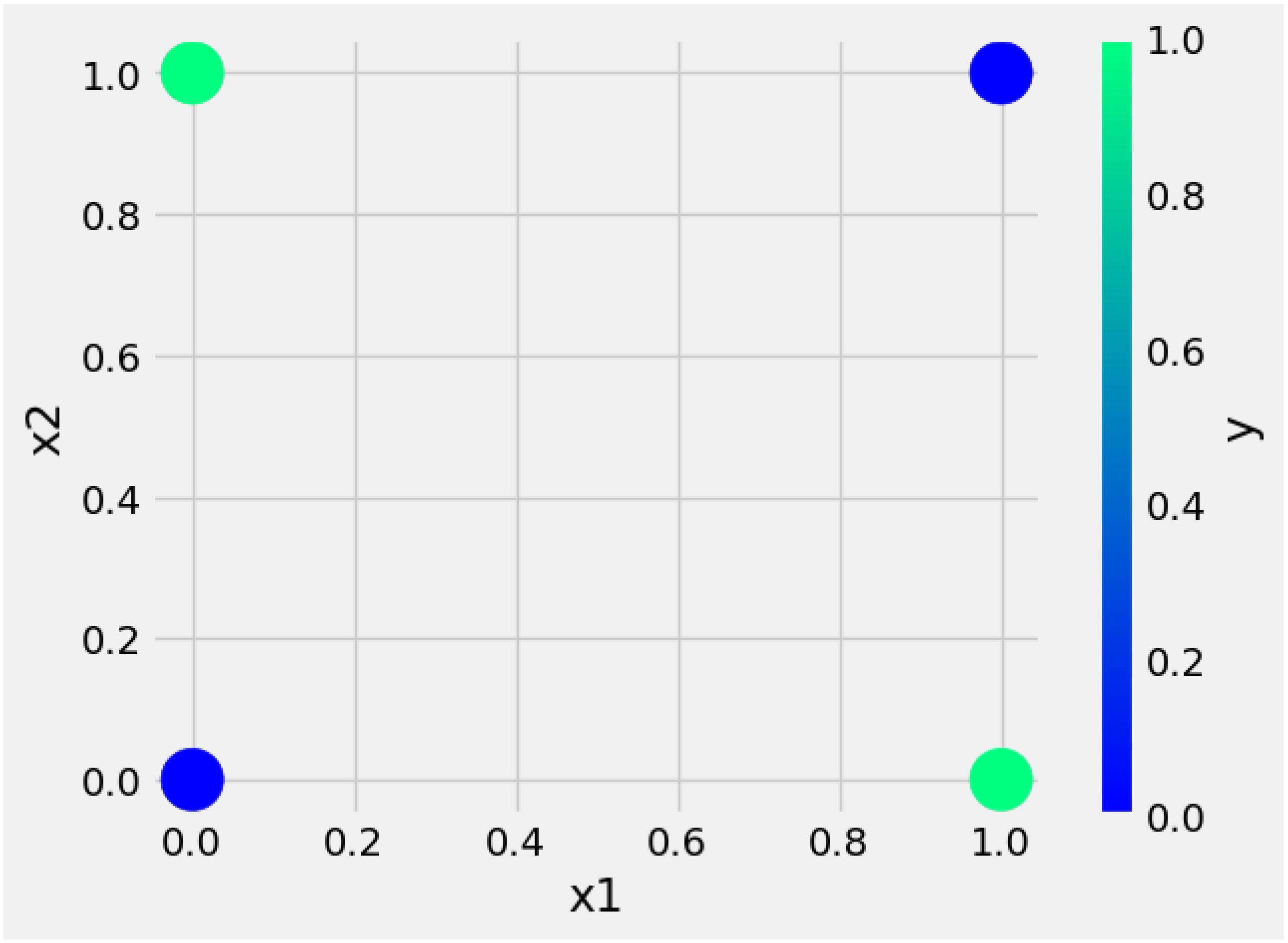


In [21]:
```python
OR.plot(kind="scatter", x="x1", y="x2", c="y", s=50, cmap="winter")
plt.axhline(y = 0, color ="black", linestyle ="--", linewidth=2)
plt.axvline(x = 0, color ="black", linestyle ="--", linewidth=2)
plt.plot(np.linspace(0,0.75), 0.75 - 1*np.linspace(0,0.75), 'r--');
```



```
In [22]: XOR.plot(kind="scatter", x="x1", y="x2", c="y", s=500, cmap="winter")
```

```
Out[22]: <AxesSubplot:xlabel='x1', ylabel='x2'>
```



### What is learning then?

**forward pass + backward pass**, we will learn more about this later

### When does the prediction happen?

The prediction happens during the forward pass.

### Drawbacks of Perceptron:-

- It cannot be used if the data is non linear.

### Solution to this problem:-

- This can be solved by stacking the layers of perceptron, Also called as Multilayer Perceptron (MLP).

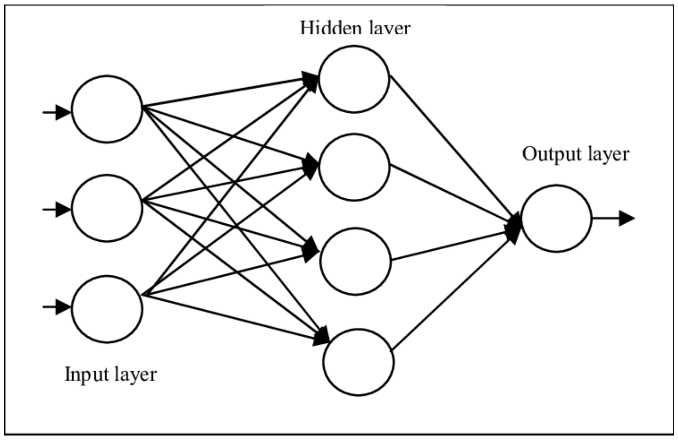
## Deep Learning With Computer Vision And Advanced NLP (DL_CV_NLP)

## Basic Understanding of ANN (Artificial Neural Network)

The solution to fitting more complex (*i.e.* non-linear) models with neural networks is to use a more complex network that consists of more than just a single perceptron. The take-home message from the perceptron is that all of the learning happens by adapting the synapse weights until prediction is satisfactory. Hence, a reasonable guess at how to make a perceptron more complex is to simply **add more weights**.

There are two ways to add complexity:

1. Add backward connections, so that output neurons feed back to input nodes, resulting in a **recurrent network**
2. Add neurons between the input nodes and the outputs, creating an additional ("hidden") layer to the network, resulting in a **multi-layer perceptron**



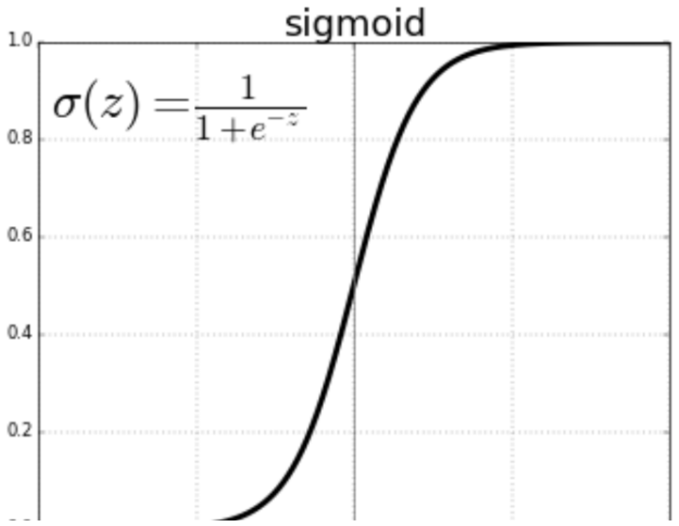
How to train a multilayer network is not intuitive. Propagating the inputs forward over two layers is straightforward, since the outputs from the hidden layer can be used as inputs for the output layer. However, the process for updating the weights based on the prediction error is less clear, since it is difficult to know whether to change the weights on the input layer or on the hidden layer in order to improve the prediction.

Updating a multi-layer perceptron (MLP) is a matter of:

1. moving forward through the network, calculating outputs given inputs and current weight estimates
2. moving backward updating weights according to the resulting error from forward propagation(using backpropagation method).

In this sense, it is similar to a single-layer perceptron, except it has to be done twice, once for each layer.

##Activation Function of ANN: In ANN we use sigmoid as an activation function in each layer instead of step function.Because ANN can solve non-linear problem so the output can be varied. Sigmoid outputs numbers 0 to 1. On the other hand step function outputs just 0 or 1.



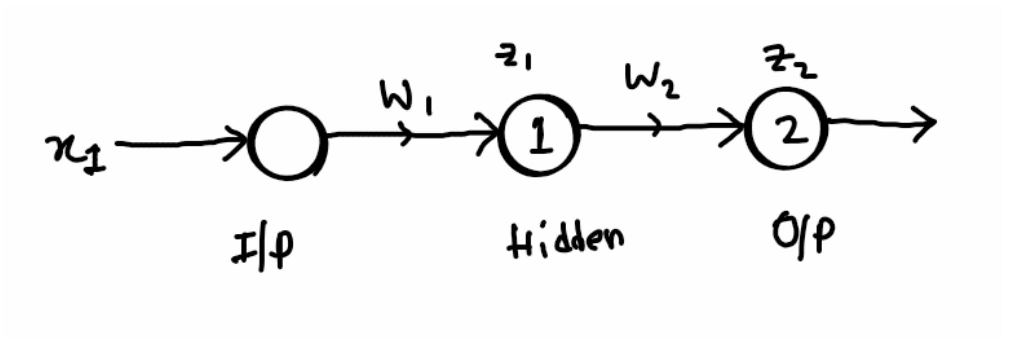
- Formula of Sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- It can take number between $-\infty, +\infty$ and gives output 0 to 1.

### Need of Activation Function:

- An activation function added into an artificial neural network in order to help the network learn complex patterns in the data.
- It also scales the data
- It filters out the important portion of data
- Without activation function Deep stack of network will behave like a single linear transformation.
  - **Example:** without activation function



$z_1 = x_1 . w_1 , z_2 = w_2 . z_1$

$$z_2 = w_2 . z_1$$
$$z_2 = w_2 . x_1 w_1$$
$$z_2 = W x_1$$

  - So, you can see it has been a single neuron.Behave like a single linear transformation.
- Without activation function all the continuous function cannot be approximated.

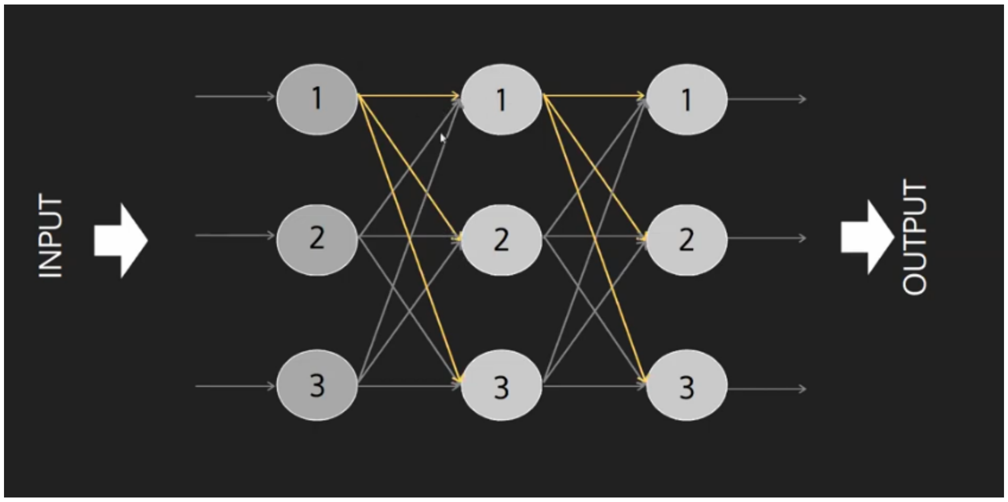
## Weight update of ANN (Backpropagation intiution):

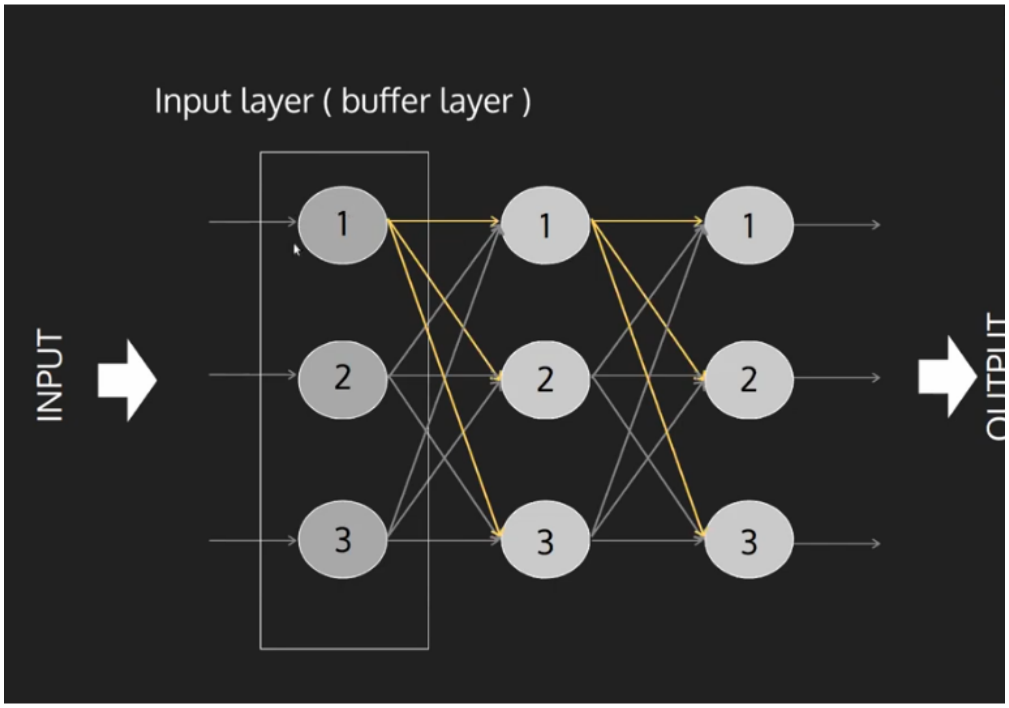
- In the year 1986 a groundbreaking paper "Learning Internal Representation by Error Propagation" was published by -

    - David Rumelhart,
    - Geoffrey Hinton, &
    - Ronald Williams
- It depicted an efficient way to update weights and biases of the network based on the error/loss function by passing twice through the network i.e forward and backward pass.
    - forward pass: data is passed through the input layer to the hidden layer and it calculates ouput. Its nothing but making prediction.
    - error calculation: Based on loss function error is calculated to check how much deviation is there from the ground truth or actual value and predicted value.
    - error contribution from the each connection of the output layer is calculated.
    - Then algo goes a layer deep and calculates how much previous layer contributed into the error of present layer and this way it propagates till the input layer.
    - This reverse pass measures the error gradient accross all the connection.
    - At last by using these error gradients a gradient step is performed to update the weights.
- In MLP key changes were to introduce a sigmoid activation function

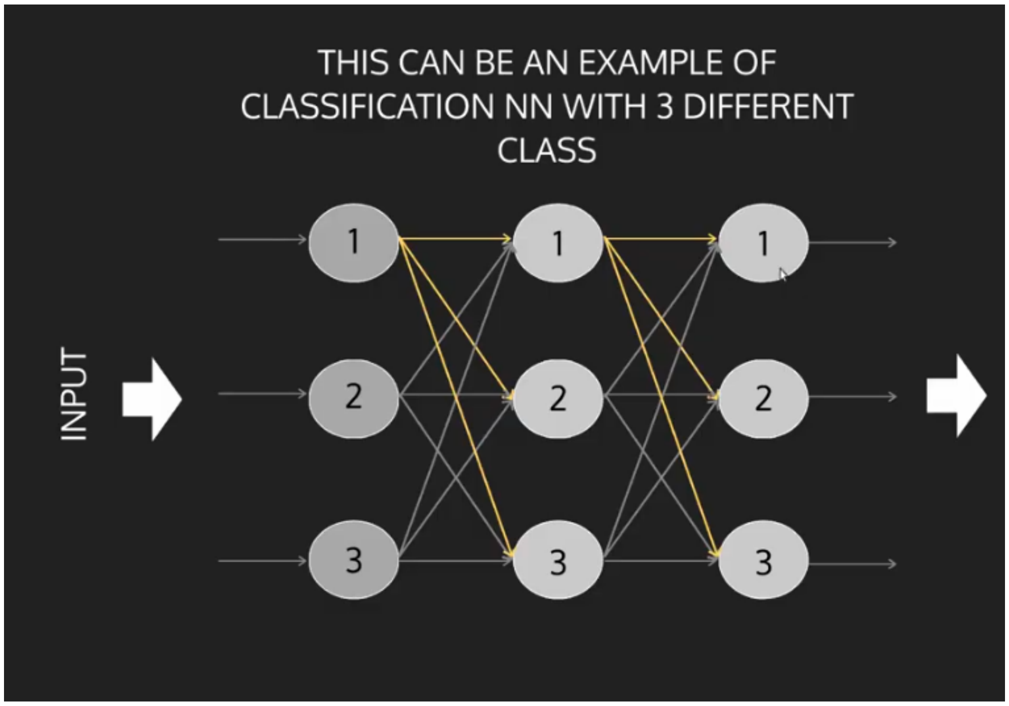
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

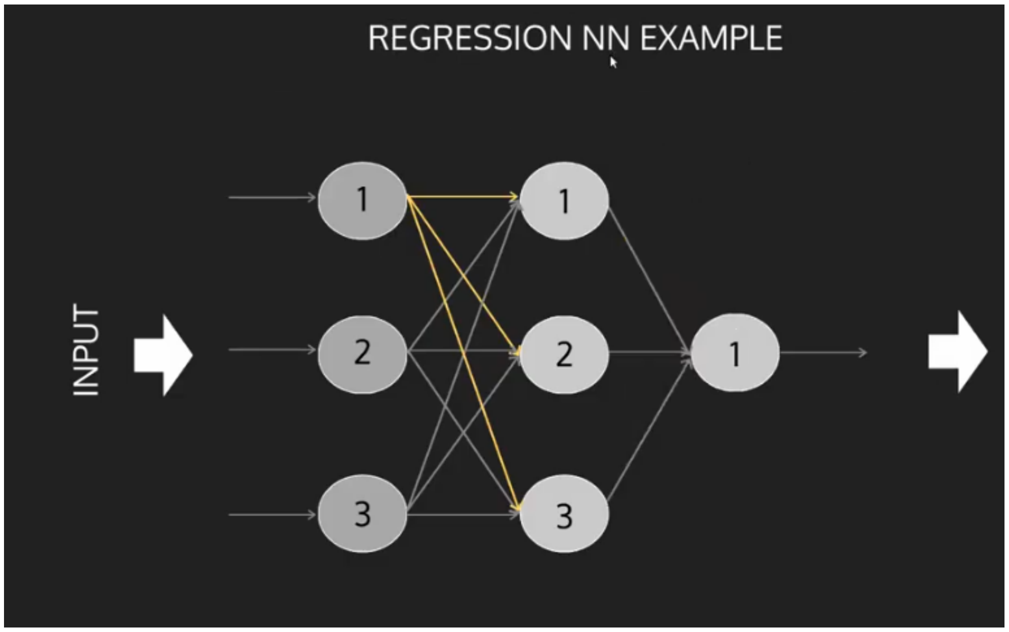
#Now lets get some intuition of ANN.





At the first layer is Input or Buffer layer. Second layer called hidden layer & the 3rd layer called output layer.

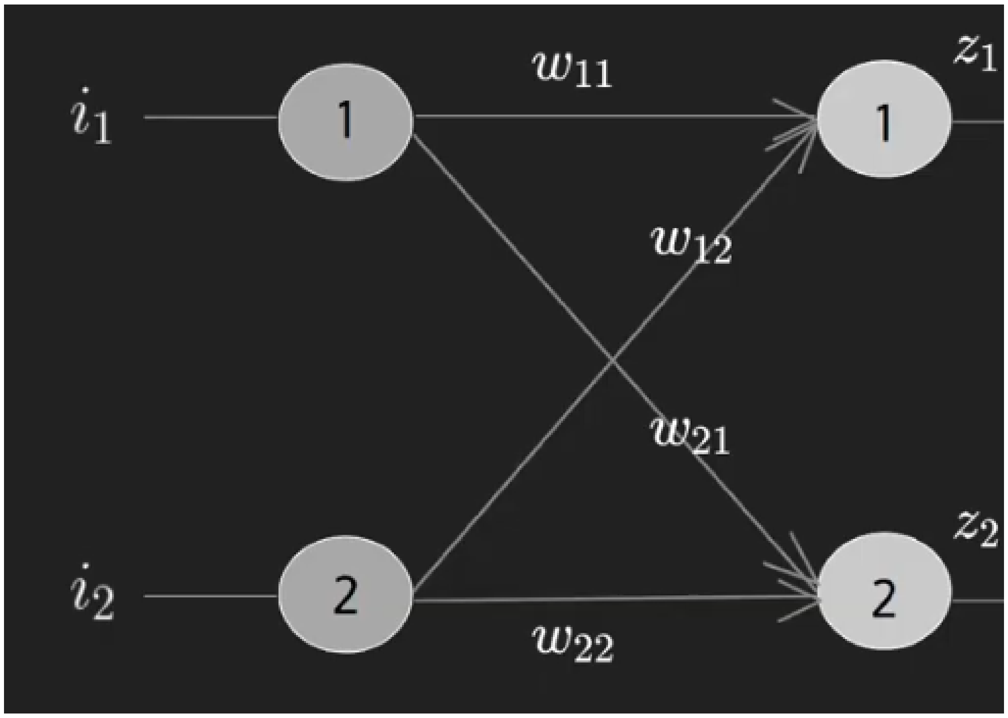

THIS CAN BE AN EXAMPLE OF CLASSIFICATION NN WITH 3 DIFFERENT CLASS
INPUT
1
2
3
1
2
3
1
2
3



In the classification outputs neuron can be multiple but in the case of Regression output neuron might be one.

## Simple Example:

Lets take a simple neuron network , Here consider bias = 0



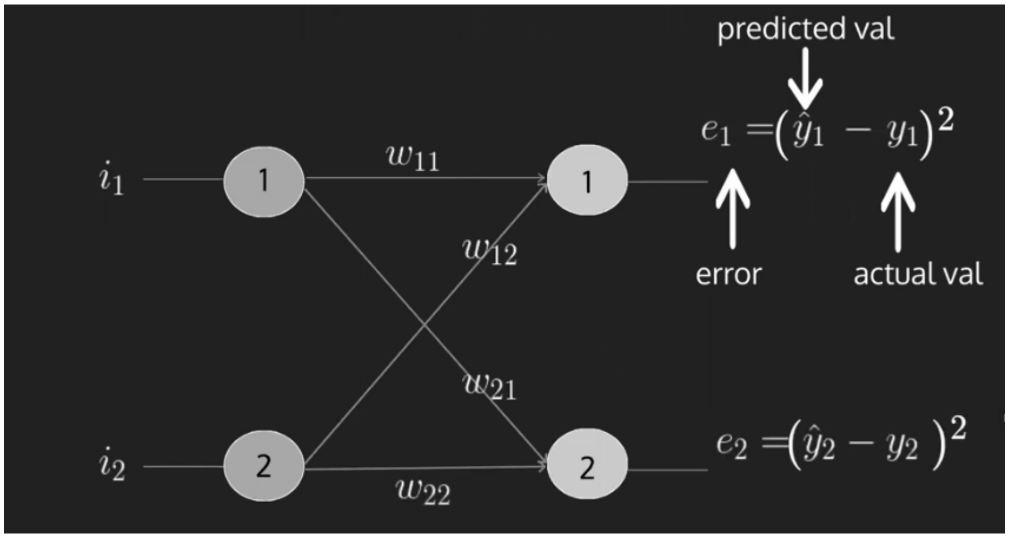
- So,

$$\begin{aligned} z_1 &= (w_{11}.\, i_1) + (w_{12}.\, i_2) \\ \therefore \hat{y_1} &= act(z_1) \\ z_2 &= (w_{21}.\, i_1) + (w_{22}.\, i_2) \\ \therefore \hat{y_2} &= act(z_2) \end{aligned}$$

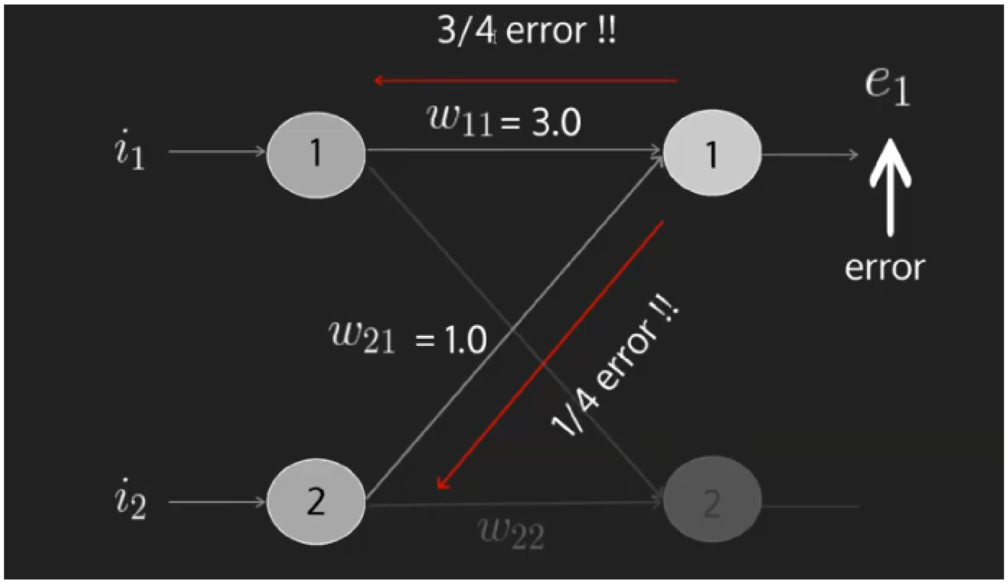
Let's define it as a metrix form,

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} * \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} (w_{11}.\, i_1) + (w_{12}.\, i_2) \\ (w_{21}.\, i_1) + (w_{22}.\, i_2) \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

## Error Calculation:



Now weight will be updated based on the proportional error!



In order to do that weight update rule (Backpropagration) can be used, that will be discuss further.

## Difference between Perceptron & Neural Network:

### Perceptron:

- Perceptron is a single layer neural network, It can be also multi layer.
- Perceptron is a linear classifier (binary).
- It cann't solve non-linear problem.
- Perceptron use step function as an activation function.

###Neural Network:

- A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.
- A neuron network is consisted by single layer or multiple layer
- It can be very depth
- It can solve non-linear problems.
- It can have many hidden layers.
- It use sigmoid, ReLu, softmax etc. as an activation function.

```
In [ ]:
```