Deep Learning

Machine Learning :

Here we give the computer a bunch of examples and tell it to follow some rules. Then, it uses those rules to make predictions or decisions about new instances it hasn't seen before.

Ex: Teaching a robot to sort toys into different boxes based on their colors. We tell the robot what colors are important and how to tell them apart

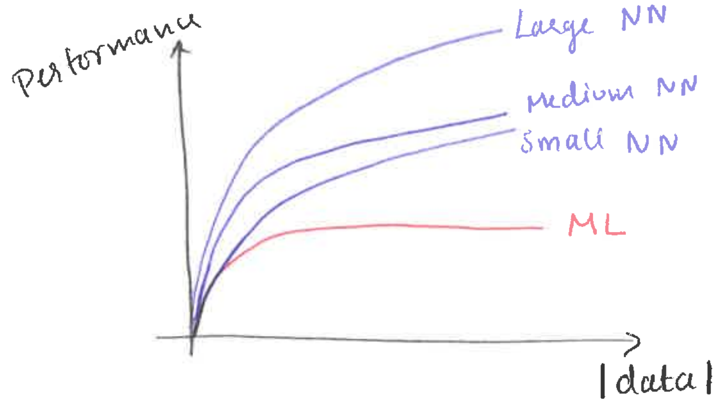
Deep Learning :

Instead of giving the computer strict rules, we give it lots and lots of examples and let it figure out the pattern on its own.

Ex: Teaching some one how to do magic tricks by showing it how to do them over and over again. The person learns by watching and practicing, with out telling every little detail.

ML is like following a recipe, while deep learning is more like learning by watching and doing.

Aspect	Machine Learning	Deep Learning
1. Architecture & Feature. representation	Algorithms learns from data, often manually engineered or selected features	Algorithms learns from data, where the features are learned automatically
2. Performance	Performance plateaus with large datasets	Often scales with large datasets.
3. Computational Requirements	Requires less	Requires powerful resources : GPUs TPUs
4. Interpretability	Models are often more interpretable	Models can be black boxes, challenging to interpret



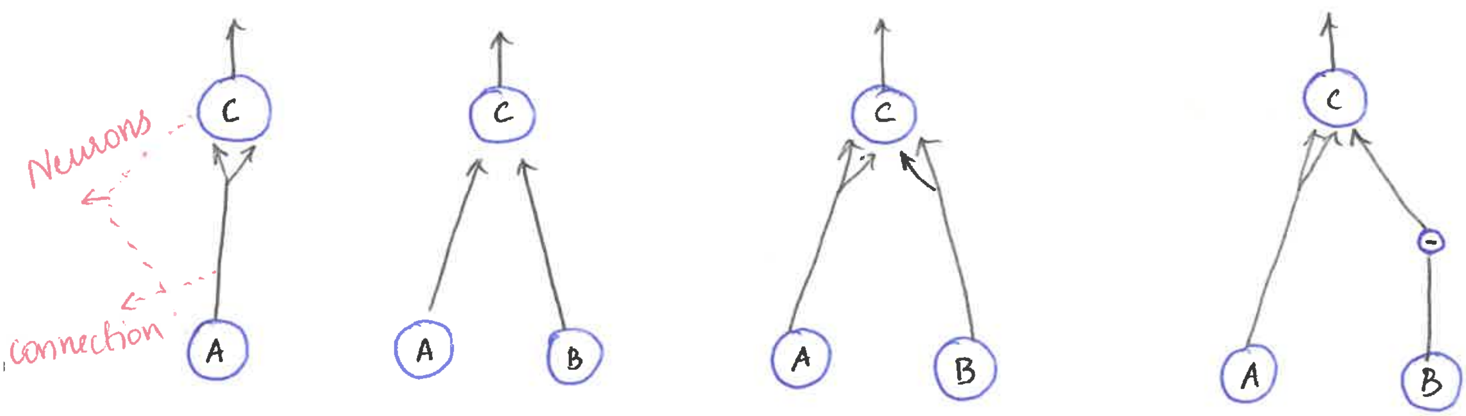
Artificial Neurons

McCulloch & Pitts proposed a very simple model of biological neuron, which later became known as an artificial neuron.

It has one or more binary inputs and one binary output. It activates its output when more than a certain number of its inputs are active.

Simple logical computations:

Note: Here, a neuron is activated when atleast two of its input connections are active.



Buffer $C = A$

AND $C = A \,\&\, B$

OR $C = A \mid B$

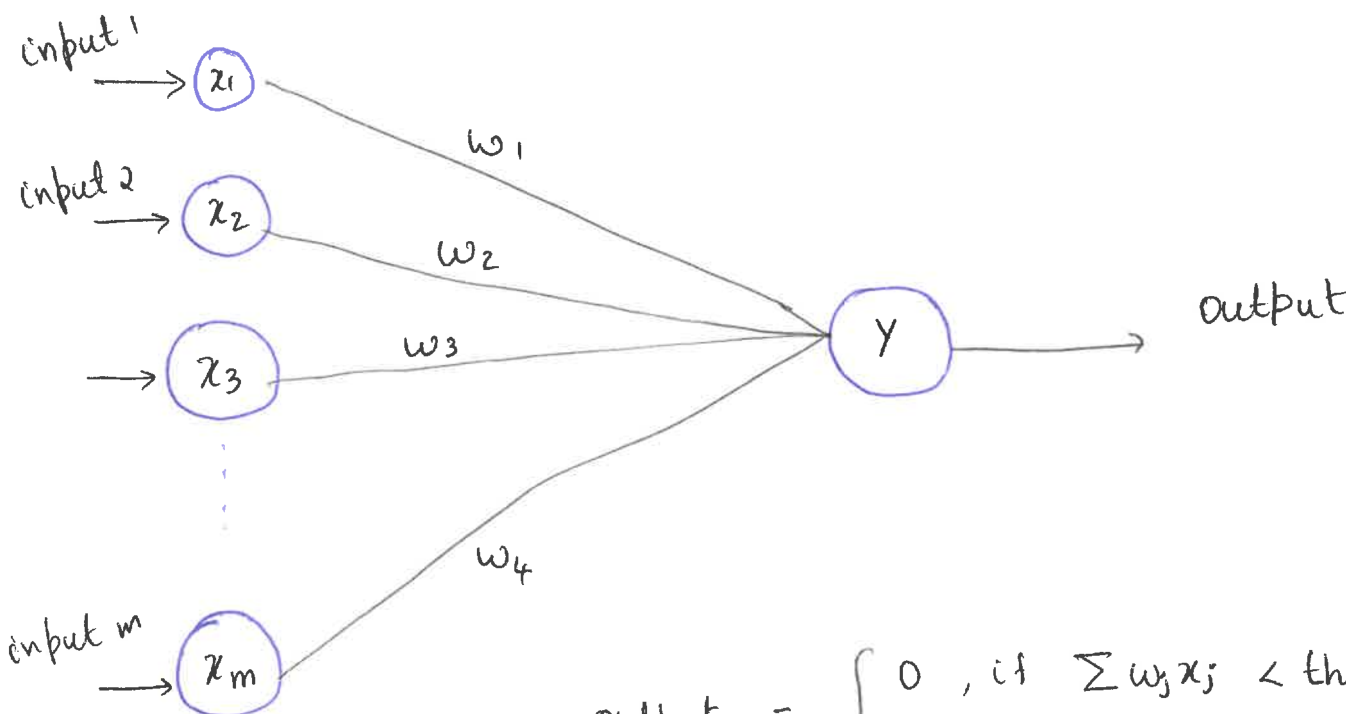
$C = A \mid \sim B$

Perceptron :

A perceptron is a simple computational unit built on artificial neurons. that

1. takes inputs
2. applies weights to inputs
3. computes a weighted sum
4. passes it through an activation function
5. produces an output

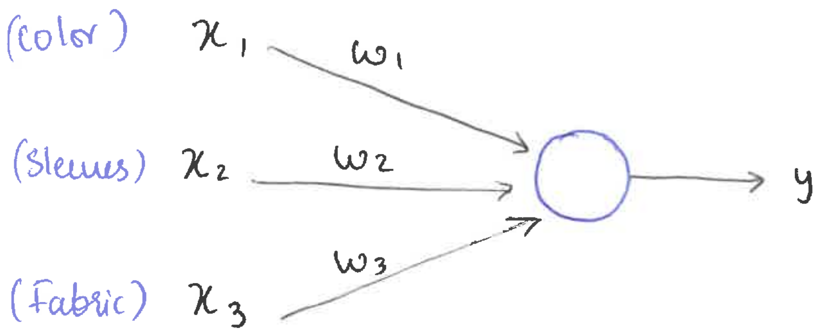
Note: A multi-layer perceptron is called Neural Networks



$$\text{output} = \begin{cases} 0, & \text{if } \sum w_j x_j < \text{threshold} \\ 1, & \text{if } \sum w_j x_j \geq \text{threshold} \end{cases}$$

Example : Purchasing a shirt

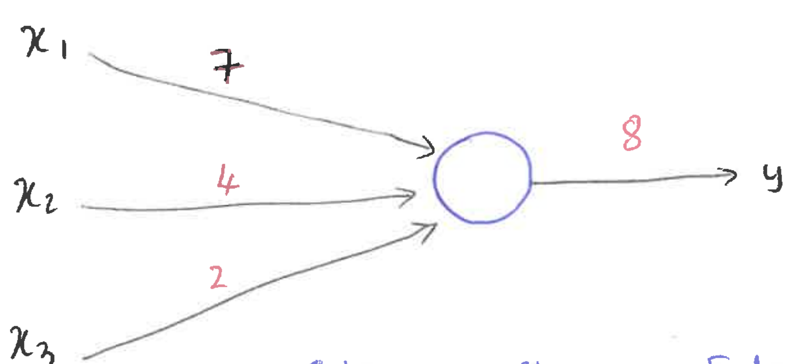
inputs		output
Color :	Black or Not	buy a shirt ?
Sleeves :	Full or half	
Fabric :	Cotton or not	



$$\text{weighted\_sum} = \sum_{j=1}^{3} x_j w_j$$

$$= x_1 w_1 + x_2 w_2 + x_3 w_3$$

buys a shirt if weighted-sum $\geq$ threshold



if you want only black full sleeve cotton shirt, try

$w_1 = 8$, $w_2 = 6$, $w_3 = 3$

threshold = 16

Color	Sleeves	Fabric	w. sum	th	Buy ?
Black	Half	Not cotton	7	8	Not buy
Black	Full	Not cotton	11	8	Buy
Not Black	Full	cotton	6	8	Not buy

Activation function :

It is a mathematical function that determines the output of a neuron.

$$\sum w_j x_j < \text{threshold}$$

$$\sum w_j x_j - \text{threshold} < 0$$

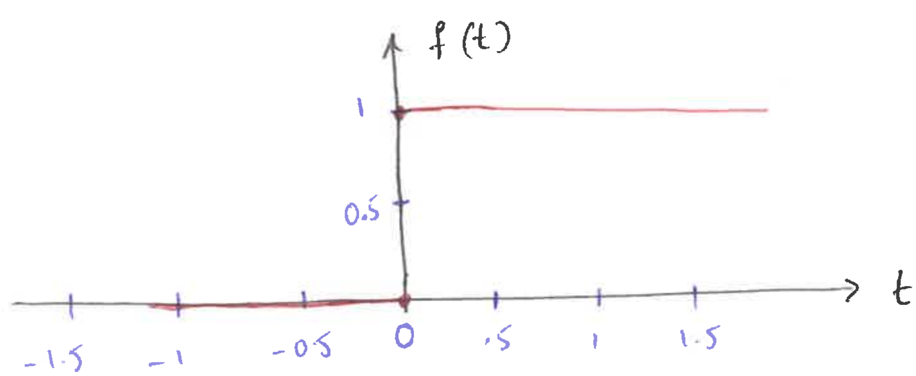
$$\sum w_j x_j + (-\text{threshold}) < 0$$

$$\sum w_j x_j + b < 0 \qquad \sum w_j x_j + b \geq 0$$

b is called bias

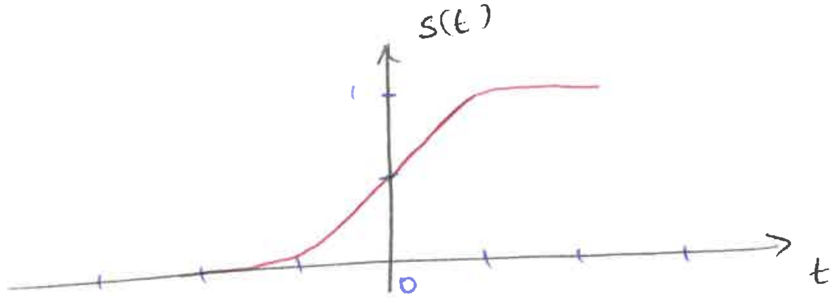
$$\text{output} = \begin{cases} 0, & \sum w_j x_j + b < 0 \\ 1, & \sum w_j x_j + b \geq 0 \end{cases}$$

Step Activation Function



Note: the change is not gradual here.

Sigmoid Activation Function :



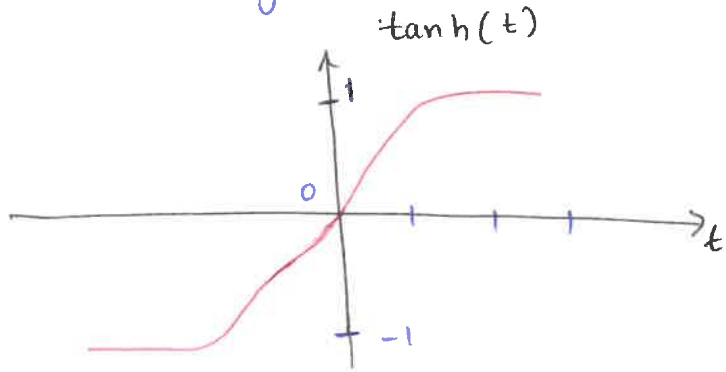
$$s(t) = \frac{1}{1 + e^{-t}}$$

It is a non-linear function and bounds the value of a neuron in the small range (0, 1)

$$\text{output} = \frac{1}{1 + e^{-(\Sigma w_j x_j + b)}}$$

when output is close to 1, neuron is active

Hyperbolic Tangent Function



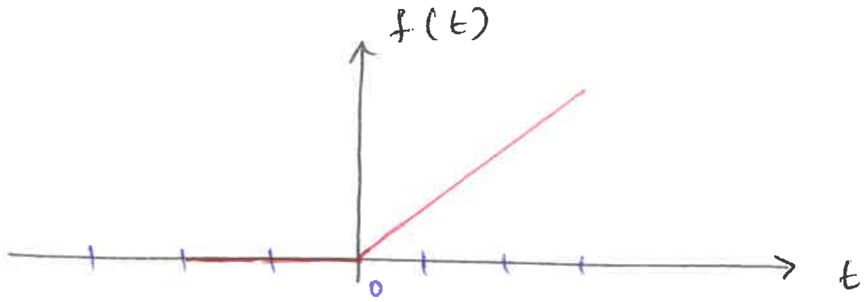
It is a shifted & stretched version of the sigmoid, where the output range is (-1, 1)

$$\tanh(t) = \frac{e^{t} - e^{-t}}{e^{t} + e^{-t}} = \frac{2}{1 + e^{-2t}} - 1$$

$$\text{Output} = \frac{2}{1 + e^{-2(\Sigma w_j x_j + b)}} - 1$$

Rectified Linear Unit (ReLU)

$$f(t) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$



It sets the negative values to zero and leaves the positive values unchanged

Multi-Layer Perceptron : Stacking cells to create Network

Types of Stacking :

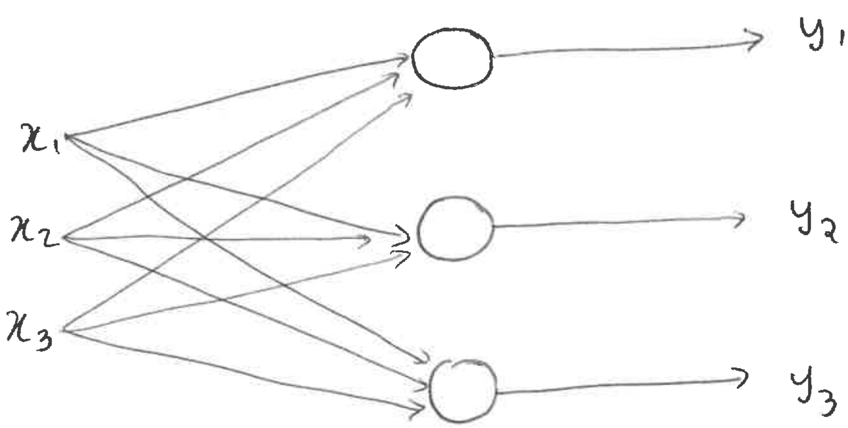
1. Parallel
2. Sequential

Stacking : A technique that combines multiple layers to create a complex ~~kee~~ model capable of learning complex patterns and representations from the data.

Parallel Stacking :

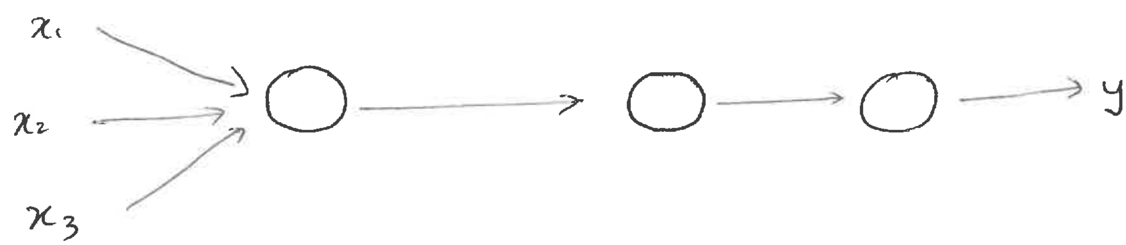
* Multiple layers are combined in parallel and each layer operate independently on the input data.
* Each layer receives the same input data independently & produces its own output.
* It is used to extract diverse features from the input data.

Example : In Image recognition, find the face + co-ordinates (x, y)



Sequential Stacking:

* Multiple layers are combined sequentially
* The output of one layer serves as input to the next layer.
* Each layer in the sequence typically performs a different transformation or feature extraction operation.



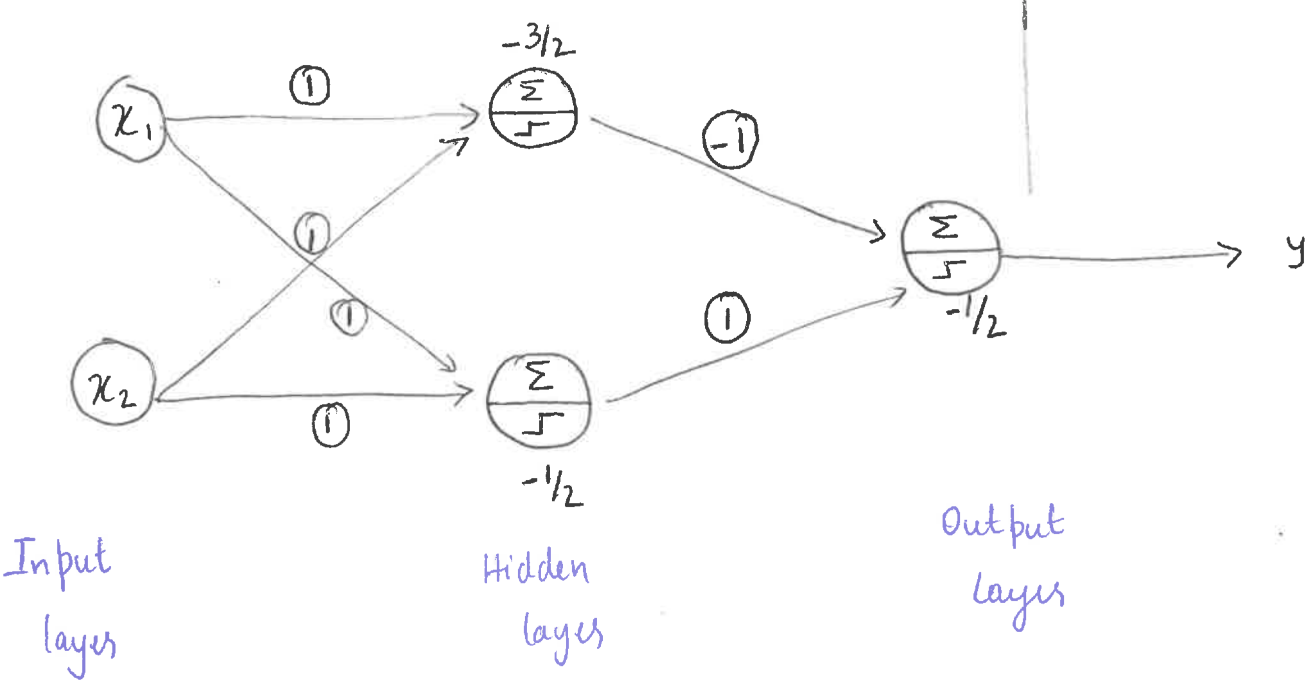
Stacking perceptrons in parallel and sequential manner results in a multi-layer perceptron (MLP)

Example: An MLP solving XOR problem

$$A \oplus B = A\bar{B} + \bar{A}B$$

Variables to establish NN

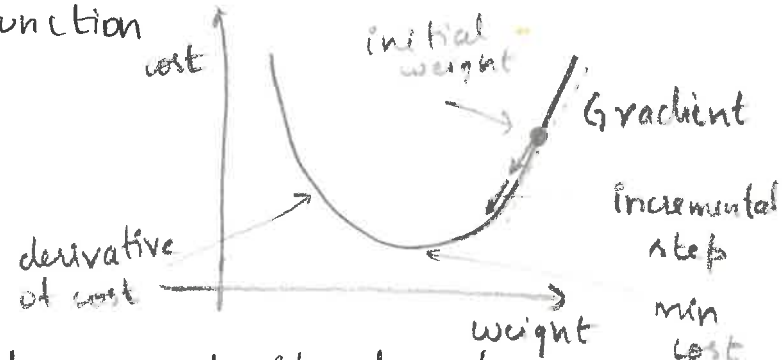
Weights = 6 , Biases = 3



Gradient Descent :

It is an optimization algorithm used to minimize the loss function of a NN by iteratively updating the parameters (weights & biases). of the network in the direction of the steepest descent of the loss function

$$\theta_{t+1} = \theta_t - \eta \frac{\partial L}{\partial \theta_t}$$



θ_t : parameters of the network at iteration t,

η : (eta) learning rate, which controls the size of the step taken in the direction of the gradient.

$\frac{\partial L}{\partial \theta_t}$: partial derivative of the loss function L wrt parameter (θ_t)

Process :

Step 1 - Assign random W and B values — Initialization

Step 2 - Calculate final output using these values Forward propagation

Step 3 - Estimate error using error function

Step 4 - Find those W and B which can reduce error — Backward propagation

Step 5 - Update W and B, and repeat from step 2 — Implementation of GD

Coding : Regression

$$W_{t+1} = \omega_t - \eta \frac{\partial L}{\partial W}$$

$$\frac{\partial L}{\partial \omega} = \frac{\partial \left(\frac{1}{2} (y - y')^2 \right)}{\partial \omega}$$

$$= \frac{\partial}{\partial \omega} \left(\tfrac{1}{2} (y - \cancel{w}^T x_n)^2 \right)$$

$$= \frac{1}{2} * 2 \; (y - w^T x_n) * (0 - \cancel{w}\, x_n)$$

$$= -(y - w^T x)(x_n)$$

$$= -(\text{error}) * \text{input})$$

$$\omega_{t+1} = \omega_t - \eta \left(-\text{error} * \text{input} \right)$$

$$= \omega_t + \eta * \text{error} * \text{input}.$$

Regularization : Neural Networks
DROPOUT

similar to L_1 & L_2

* It is a regularization technique used in neural networks to prevent over fitting.

* Overfitting occurs when a model learns to memorize the training data rather than generalize from it, leading to perform poorly on unseen data.

* It was proposed in a research paper by Geoffery Hinton et al. in 2012 & further detailed in a 2014 paper by Nitish Srivastava et al.

Working :

1. During Training :

* In each iteration of the back propagation algorithm, individual neurons are randomly dropped out or deactivated by deleting all in coming and out going links.

* ~~The neural network is trained on this dropout network~~

* Standard forward and backward propagation is applied on this dropout network and the model parameters are updated.

* In this step, the model parameters of a deactivated neuron are unchanged.

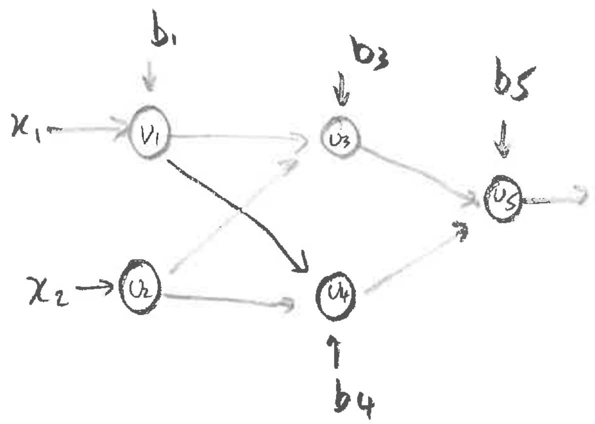
Note:

1. Neurons are deactivated or dropped out according to a dropout rate value in range (0, 1)

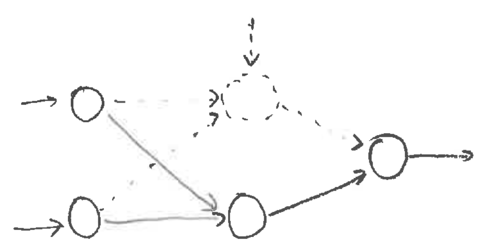
   i.e., if drop out rate is 0.2, then

   20% of the neurons in that dense layer is dropped out.

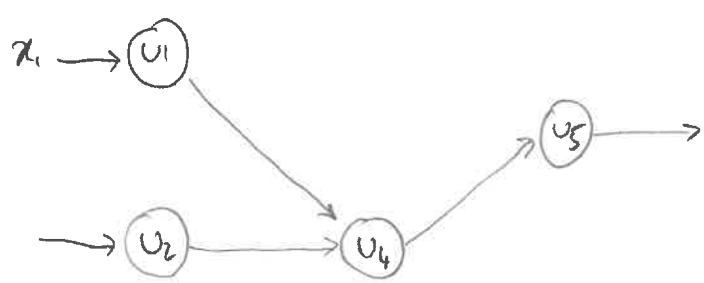
2. Output neurons are never dropped.
3. Input neurons may be dropped.
4. There must exist a link from inputs to output
5. Given a network with N droppable neurons, there are a total of 2^N possible networks.



original network



drop out U_3



Drop out network

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
model = keras.Sequential([
        keras.layers.Dense(64, activation='relu', input_shape =
                                                    x_train.shape[1],)),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dropout(0.3),
        Dense(10, activation='softmax')
    ])
```

Dropout prevents co-adaptation

Co-adaptation in NN refers to a situation where neurons within a network becomes overly dependent on each other to make predictions.

Autoencoder

* It is a type of Neural Network used for unsupervised learning.

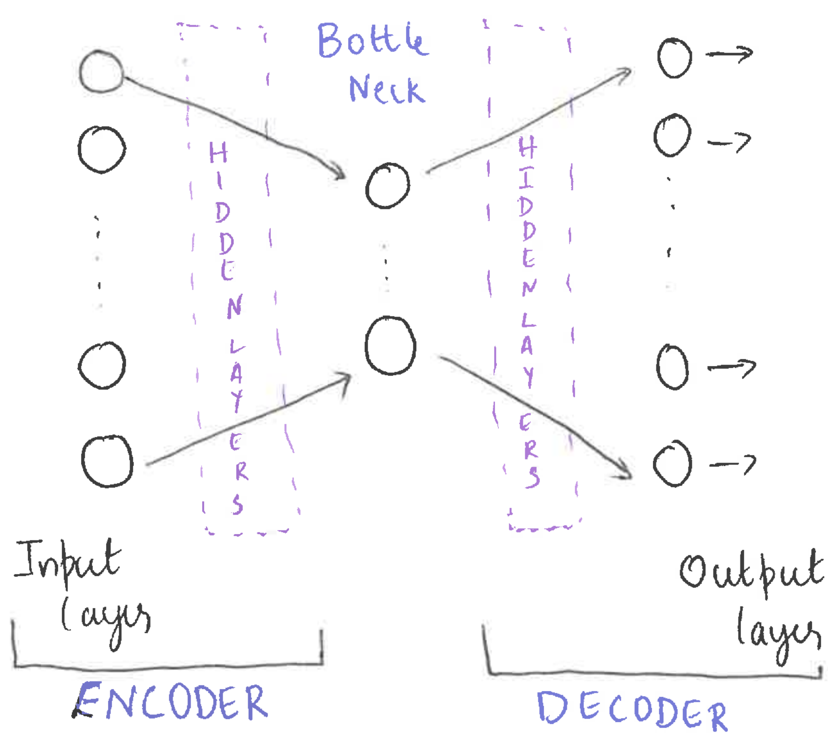
* The main purpose of an autoencoder is to learn a representation for a set of data.

* They work by compressing the input into a lower-dimensional code and then reconstructing the output from this representation

Uses :

1. Dimensionality reduction
2. Data denoising
3. Data generation

Components of an autoencoder



1. Encoder

Input layer : This layer receives the input data.

Hidden layers : These layers progressively reduce the dimensions of the input data to form an encoded representation.

2. Bottleneck

Latent Space Representation : This is the compressed representation of the input. It captures the most important features of the input data.

3. Decoder

Hidden Layers : These layers take the encoded representation and try to reconstruct the original input data

Output layer : This layer produces the output data, attempting to reconstruct the input data.

Note : Autoencoders are trained using the input data as the target

The model adjusts its weights and biases to minimize the reconstruction error.

Autoencoders for dimensionality reduction :

* Number of units (neurons) in the hidden layers are less than the input units.

* If the decoder uses a linear activation function and mean-squared error as the loss function, the autoencoder is equivalent to PCA

Types of auto encoders :

1. Under complete Autoencoders :

These have a lower dimensionality in the hidden layers compared to the input layer.

2. Over complete Autoencoders :

These have a higher dimensionality in the hidden layers compared to the input layer.

3. Denoising Autoencoders :

These add noise to the input data and the model is trained to recover the input data without noise

4 Sparse auto encoders :

These have a higher number of Units (neurons) in the hidden layers but activate a small number of neurons at a time.

5. Variational Autoencoders (VAEs) :

These map inputs to a normal distribution in the latent space.