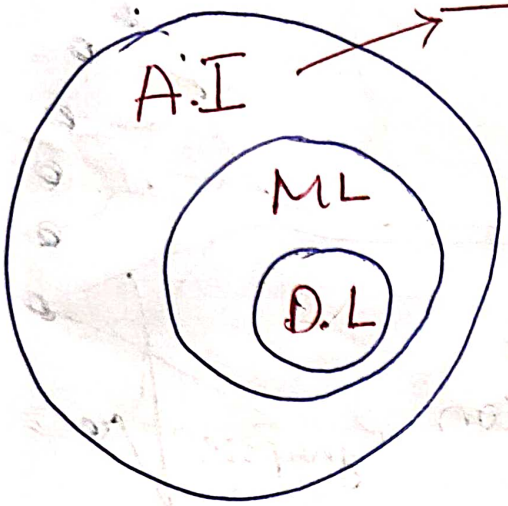
DEEP LEARNING

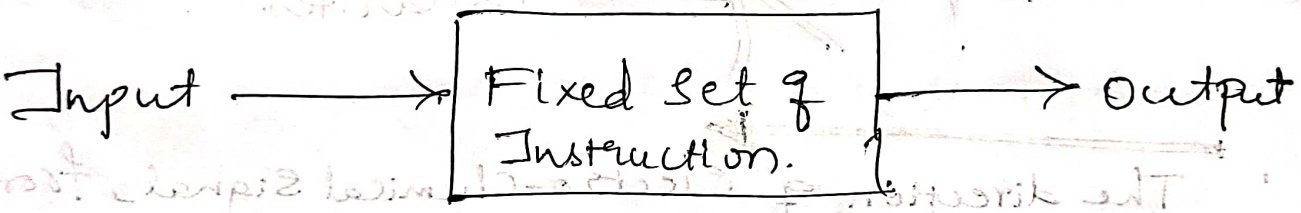
Neural Networks

RPA



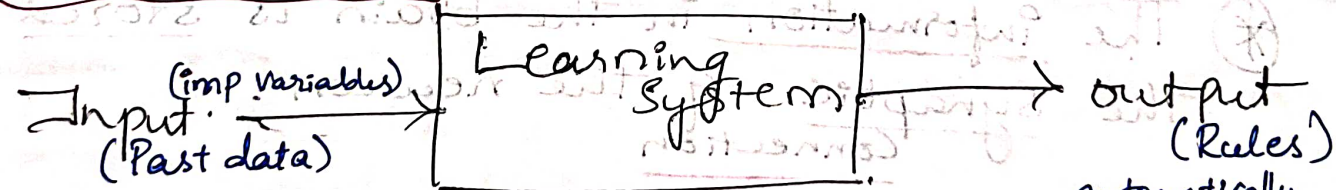
* Deep Learning is the subset of M. Learning which is a subset of Artificial Intelligence

A. Intelligence :-



* Rules based decision Making (Robotic Process Automation)

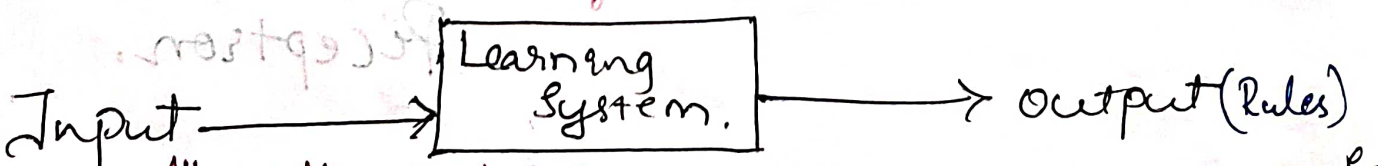
Machine Learning :-



automatically

* Machine learning involves figuring out the rules from the Past data

Deep Learning :-

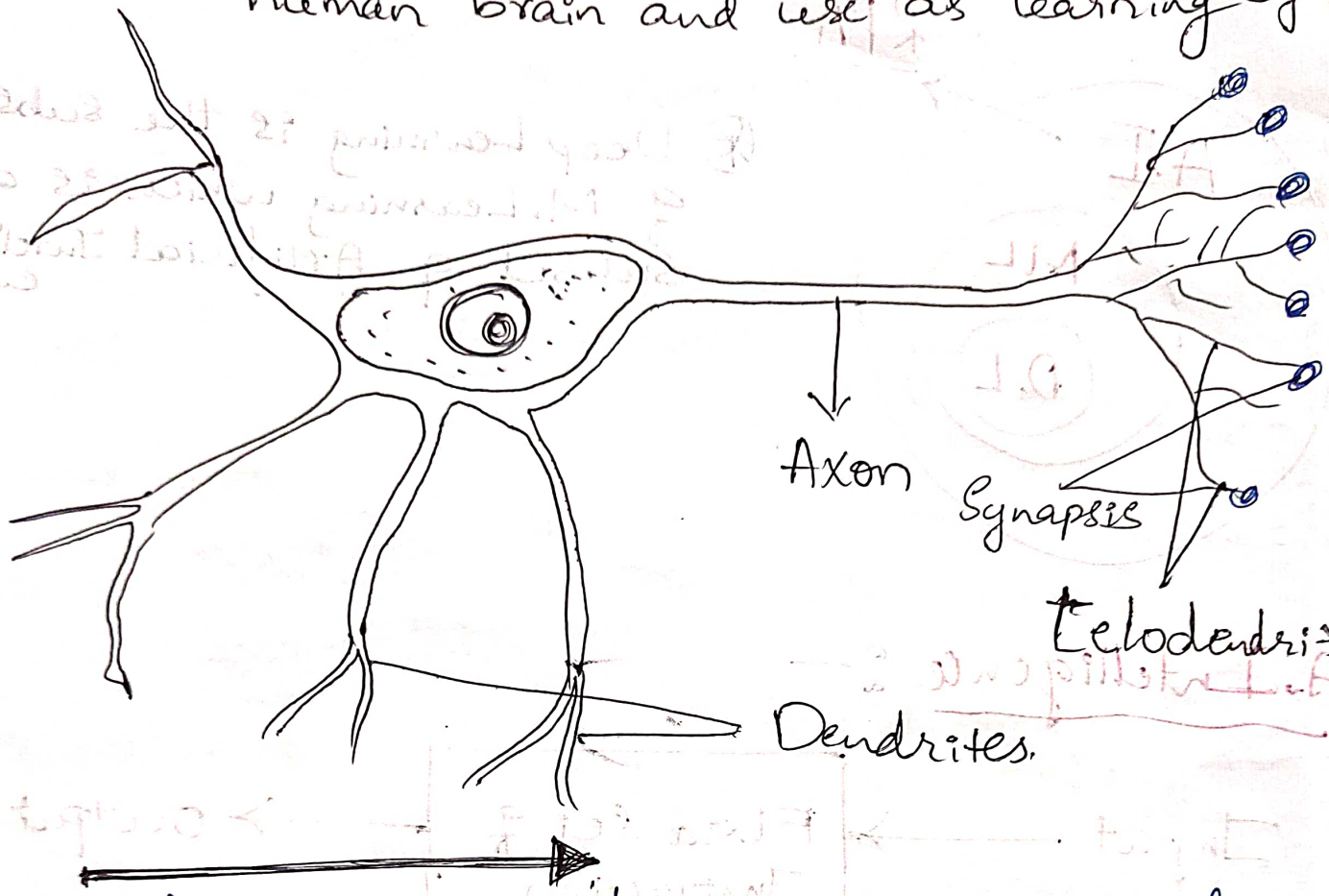


All variables, not only imp variables

(Raw data)

* The algorithm automatically figures out the imp variables and create the Rules.

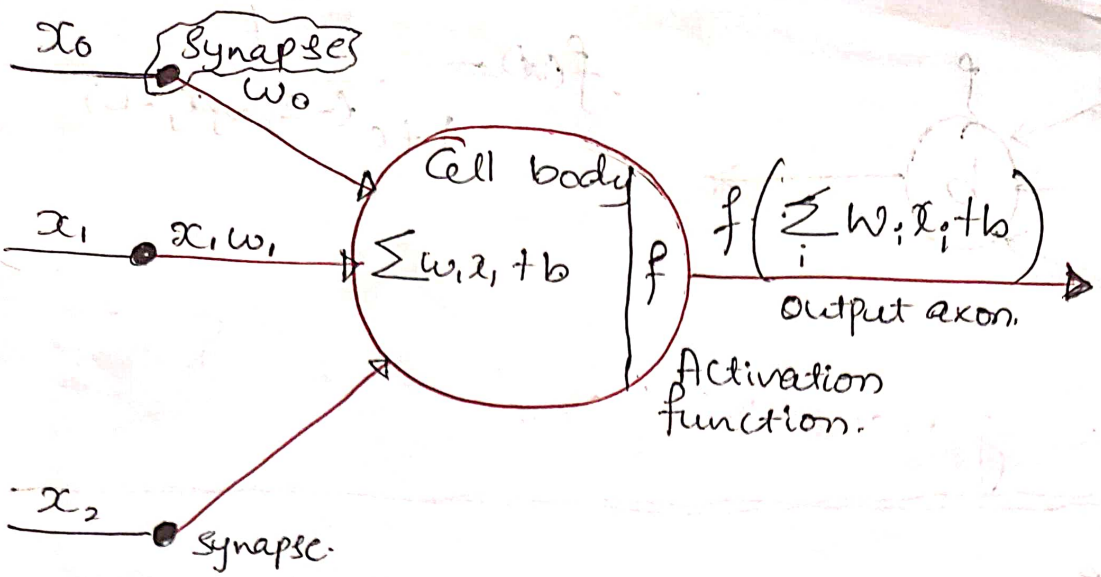
* The Best Learning System is Human Brain
  - Scientist were in the urge to replicate human brain and use as learning System



The direction of electro-chemical signal, from Dendsites to telodendsites by the diff in Na^+ & K^+ ions

* The information in the brain is stored in the Synaptic connection of the neuron.

* In 1949, Donald Hebb tried to create the Mathematical model of Neuron called Perceptron.



* The input from the Synapse is Attenuated (adds weight) and sent into the body.

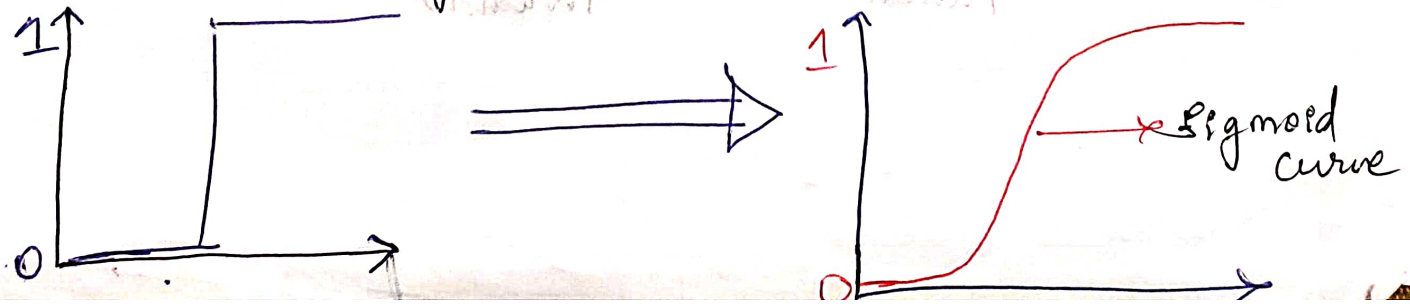
* The output will be 1 if the signal coming in meets the threshold, otherwise zero(0).

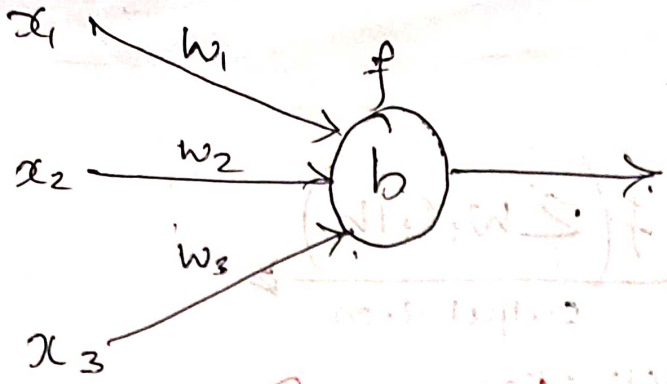
$$\text{Output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold.} \end{cases}$$

* For each problem neurons will have different set of weights, for figuring out the right set of weight, tuning of the neuron must be done.

This tuning process is difficult if we have digital output (0, 1)

∴ we modify this neuron to a Sigmoid neuron which has output from '0' to '1' in gradually increasing manner





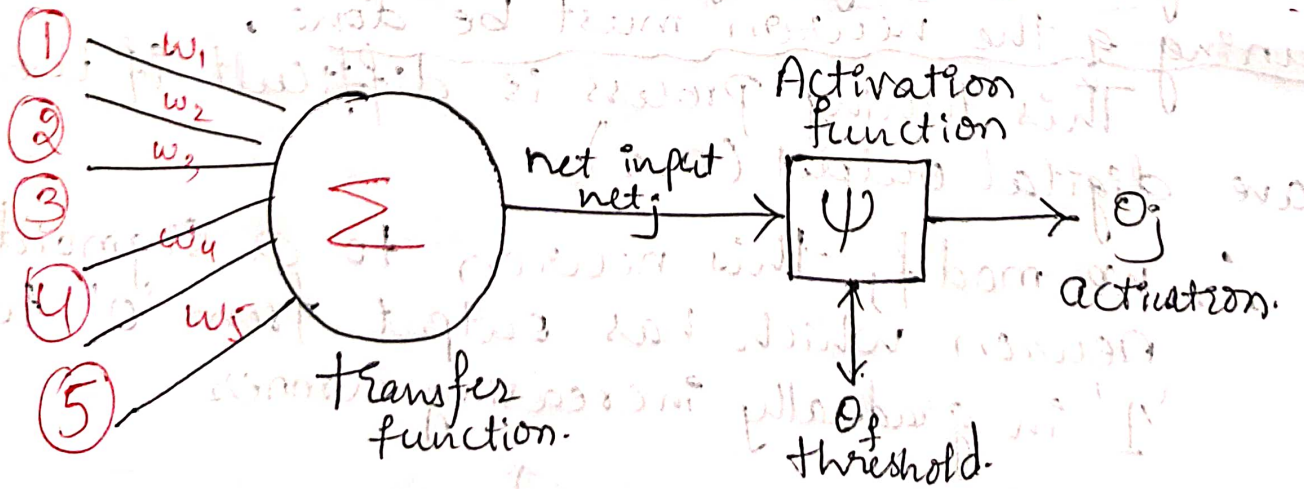
$$f(x) = \frac{1}{1+e^{(-\sum_j w_j x_j - b)}}$$

Sigmoid Neuron.

Example :-

	Independent Variable	weight Coeff
1	Salary	w_1
2	Credit rating	w_2
3	Age	w_3
4	Edu level	w_4
5	Account balance	w_5

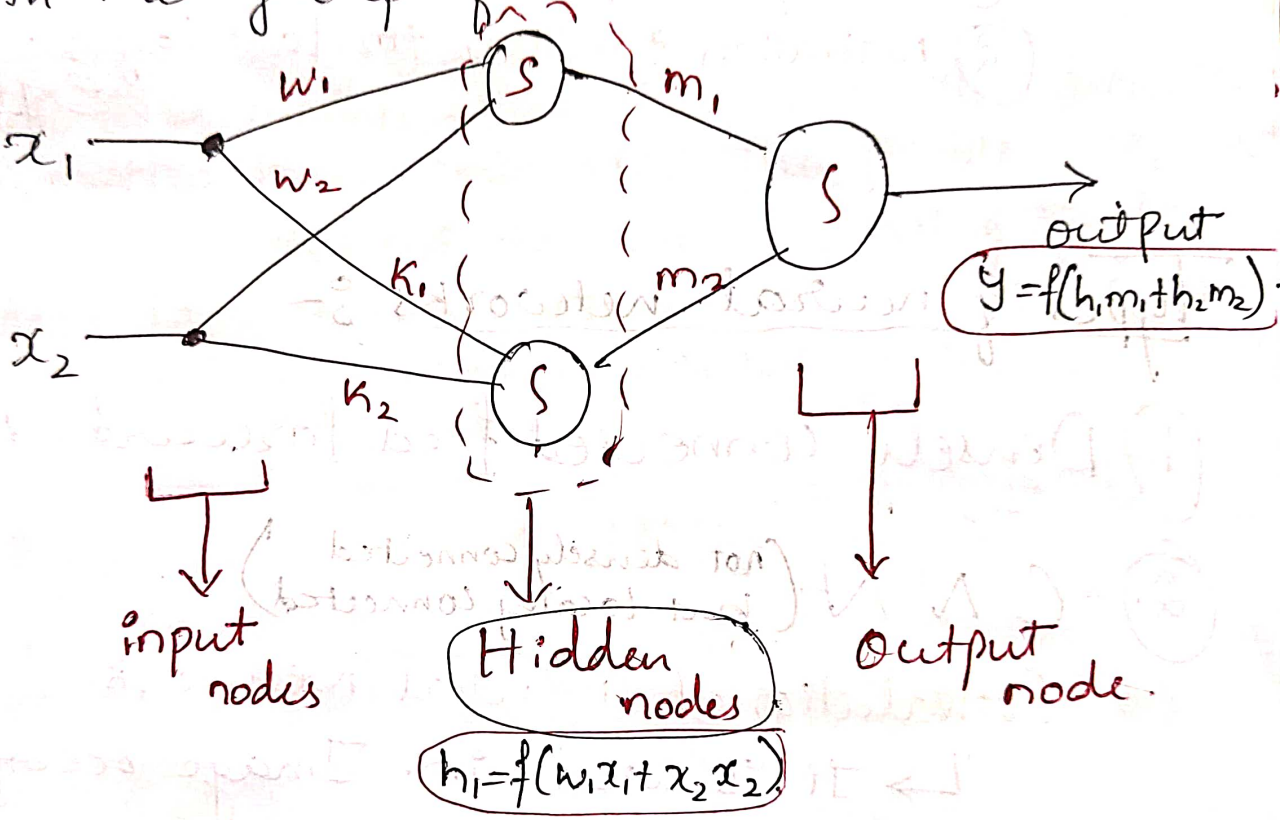
To determine loan approval



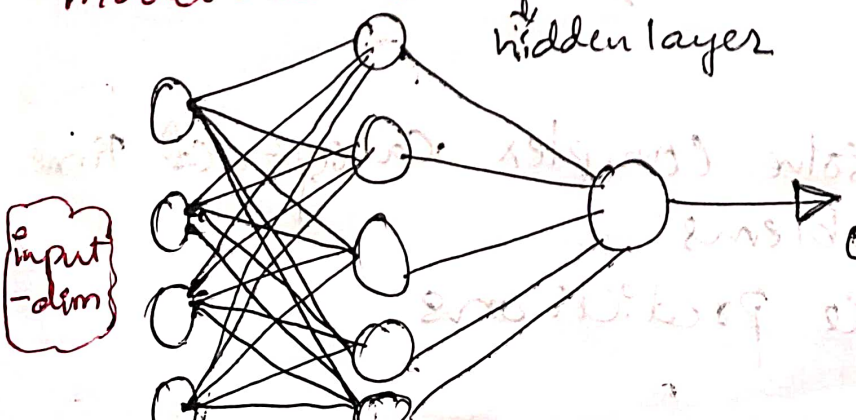
(*) This artificial neuron does the same job of Log. regression

∴ Sigmoid neuron failed to be different
↳ it also failed in classifying XOR outputs

(*) Due to the failure of perceptron, Scientists ~~thinked in the world~~ thought of trying with the group of neurons



```
model = Sequencial()
model.add(Dense(5, input_dim = 4, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
```



Output layer

(*) Activation function in output layer
- (1) Linear → for regressions.
- (2) sigmoid → classification problems.

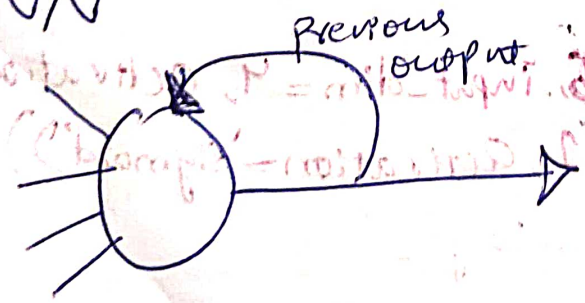
(*) Multiclass classification problems
- (1) Multiple output neurons
- (2) Activation function for last layer → Softmax.

Types of neural networks :-

(1) Densely connected feed forward N.N.

(2) CNN (not densely connected but locally connected)
Convalutional Neural Network
↳ It is used in Image recognition

(3) Recurrent Neural Network
RNN.



(*) used to solve complex catagorical time series problems

(*) sentence predictions...

Hyperparameters in N.N.

1. Network structure.
2. Activation function.
3. Number of hidden layers.
4. Number of neurons → through Cross Validation.
5. Cost function.

(*) There is a mathematical theorem which garuntees no matter how complicated the problem is, we can achive desired accuracy by using single hidden layer neural network (by setting the hyper Parameter → no. of neurons in hid. layer)

But why we need more layers ??

→ learning will become slower in One layer.

→ One layer learn one patter and other will learn other patterns so learning becomes easy.

→ The power of backprop will be prominent in many layers!

(*) How to Assign the weights to the neuron

↳ we use Gradient decent method to assign the correct set of weights

$$w_1^0 = w_1^0 - \delta \left[\frac{\partial SSE}{\partial w_1}\right]$$

↳ If this values is positive, then that means if weight increases the error will increase and vice versa

$$w_1^0 =$$

↳ Firstly we assume the random set of weights and get the error... After this we try to increase or decrease the weights accordingly by using above formula till we reach the least error and consider the corresponding weights

$$\vec{G}_1 = \vec{G}_0 - \delta \nabla SSE$$

ANN → Artificial Neural Network.

Multiple method of updating weights :—

Full batch → Regular Training as done for all other ML Algos. Entire input is used to train & update weights using Grad Descent.

Online → Show one input row at a time... adjust weights, show another input and adjust weights ....., once the input is all over, start with row 1 if needed.

Mini-Batch → Pick a Small random Sample of inputs perform batch update. Then pick another set of sample inputs randomly.. & perform update.

↓

most preffered.

```
import keras
from keras import regularizers, optimizers
from keras.layers import Dense
from keras.callbacks import EarlyStopping,
                             ReduceLROnPlateau

model_sig = Sequential()
model_sig.add(Dense(10, input_shape=(21,)
                    , activation='sigmoid'))
model_sig.add(Dense(18, activation='sigmoid'))
model_sig.add(Dense(1, activation='sigmoid',
              kernel_regularizer=regularizers.
                                 l2(0.04)))

from keras.optimizers import SGD
opt = SGD()
model_sig.compile(loss='binary_crossentropy',
                  optimizer=opt, metrics=['accuracy'])
model_sig.fit(X_train, y_train, epochs=30,
              batch_size=100)
```

→ minimize error

→ 1st hidden

2nd hidden

output layer

(*) What is Deep Learning?

Less data
computatio
Powe

↳ Deep learning is a Neural Network which has more than 2 hidden layer.

$N > 2$ is Deep

→ Since we know that from math. theorem a single layer is enough with 'n' neuron to achive desired Accuracy, but in this case the learning in the neurons is like memorising, but in deep learning it will need shallow learning to understand the dataset

∴ Therefore we use greater than 2 hidden layer which enables neurons to learn general patterns
Eg understand the dataset better

③ Overfitting :-

→ Problems in Deep learning :

(1) Vanishing gradients :-

↳ As we add more hidden layers, back-propagation become less and less useful in passing information to the Lower layers.

It is the method to compute the gradient

→ Relu Activation Function will solve this problem.

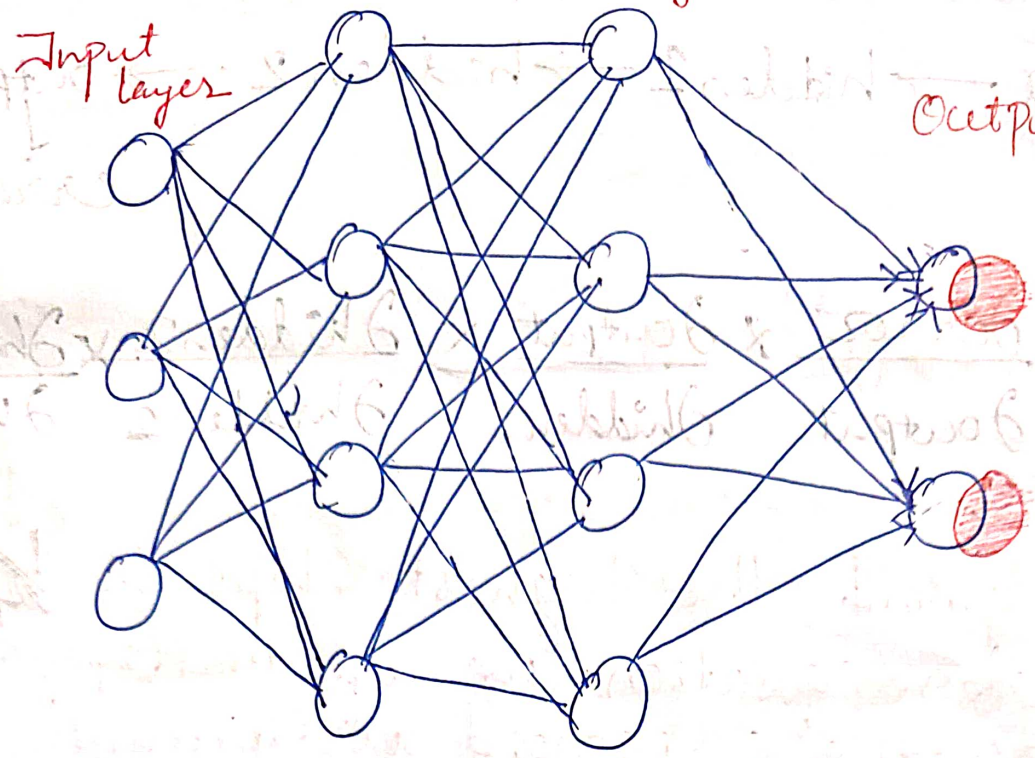
(2) Difficulty in optimization :-

In deeper networks, the parameters space is large, and surface being optimized has complex structures. Gradient descent becomes very inefficient

(3) Overfitting :-

→ training accuracy becomes High

→ test accuracy becomes Low



* Compute the error in the output layer

ldle
Point
ep grad
allow grad

Understanding the Vanishing Gradient

$W_1 . x \rightarrow$ hidden 1 $\rightarrow$ hidden 2 $\rightarrow$ output $\rightarrow$ error

$$\frac{\partial error}{\partial W_1} = \frac{\partial error}{\partial output} \times \frac{\partial output}{\partial hidden_2} \times \frac{\partial hidden_2}{\partial hidden_1} \times \frac{\partial hidden_1}{\partial W_1}$$

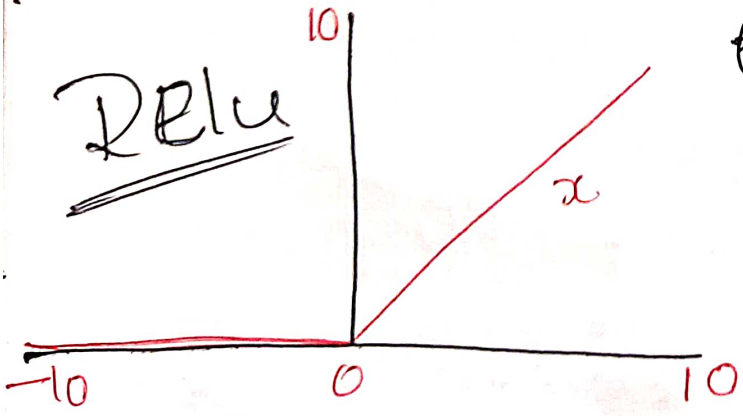
(*) For Sigmoid the highest slope is 1/4

∵ in above equation we are multiplying one derivative of Sigmoid with another derivative of Sigmoid, therefore the Product goes on decreasing since the slope is 1/4th.

∴ If we add more & more layers the derivative values are getting Smaller & Smaller ∴ termed as Vanishing Gradient

---

ReLu



(*) The derivative of the Relu slope (x) is 1

∴ the derivative will not become Smaller.

∴ The hidden layer must be built using Relu Activation function.

→ Instance name = Class name()
S1 = Student()

* Optimization Difficulty :-

↳ The surface over which we need to find minimum can be very complicated.

* Saddle points → The point which is min for one and max for another layer
* Steep gradients
* Extremely shallow gradients

↳ Since in deep neural network we will be searching for minima in million dimensions ∴ this simple Gradient decent will not solve the problem of optimization.

In normal G.D.
↳ Below shown is conture plot to find the minima

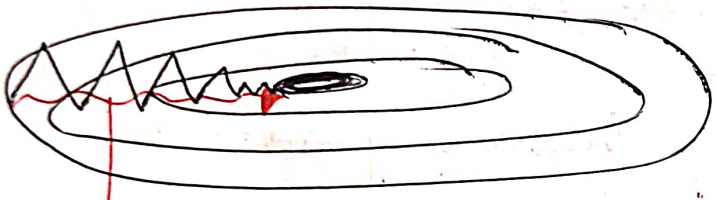
* we take tangent from each circle and measure gradient. After measuring we move in opp direction of gradient to find the minima.

* Since heights of the slope varies, we move in bit diff direction and finally reach the minima which takes much longer duration.

∴ Idea of momentum came into consideration

(*) Vector addition is done using parallelogram & triangular law of vector addition.

(*) ∴ Instead of taking the latest gradient, we take the average of past few gradients.



Gradient decent with Momentum

```
from keras.optimizers import SGD, RMSprop
                             Adam
model = Sequential()
model.add(Dense(10, input_dim=5, activation='relu',
          kernel_initializer='normal'))
model.add(Dense(2, activation='sigmoid', kernel_act
          ='normal'))
opt=SGD(momentum=0.9)
model.compile(loss='binary_crossentropy',
    optimizer = opt, metrics=['accuracy'])
model.fit(X_train, ...
```

(*) G. Helton thought of taking the step depending on the gradient i.e, at steeper gradient, taking smaller step size and at larger gradient, taking larger step size and therefore combining both to get resultant step size direction vector.
This idea lead to get appropriately correct direction and named it as RMSprop

↳ RMSprop work well at Saddle point

```
opt = RMSprop(learning_rate=0.1)
model.compile(loss='binary_crossentropy',
              optimizer=opt, metrics=['accuracy'])
model.fit(X_train, y_train, epochs=30, batch_size
          = 100)
```

(*) The combined momentum (i.e both momentum & RMS prop) is called Adam (Adaptive moment)

↳ It has two hyper parameters.

(i) momentum (beta1)

(ii) Squared momentum (beta2)

Adam with beta1 = 0.9, beta2 = 0.999, learning_rate = $1e^{-3}$ or $5e^{-4}$ is great starting point for many models.

```
opt = Adam(learning_rate=0.001, beta_1=0.9,
           beta_2=0.999, epsilon=1.e-07, amsgrad=False)
```

It is good to use Adam to overcome difficulties in optimization

(*) Weight initialization (techniques) can do better instead of choosing random set of weights

↳ It is the active area of research.

↳ Xavier/Glorot initialization

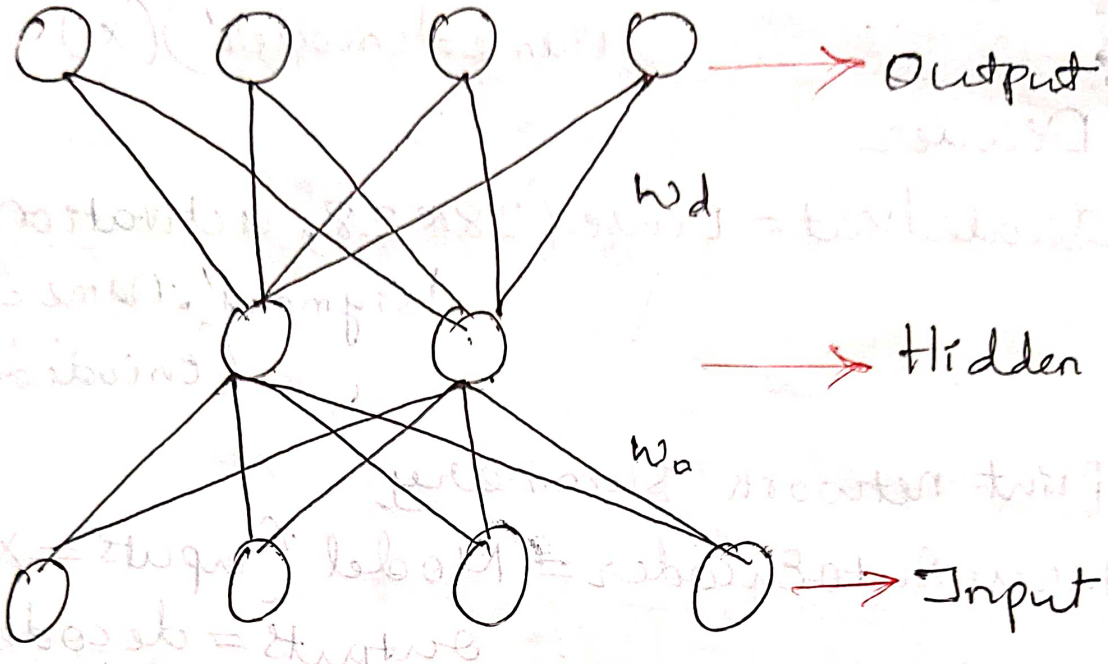
In this weight initialization method, if the neuron is getting inputs from many input nodes, then its weights should be smaller and if inputs are less, then the weights must be larger.

```
model = Sequential()
model.add(Dense(10, input_dim=5, activation='relu',
          kernel_initializer='glorot_normal'))
model.add(Dense(2, activation='sigmoid',
          kernel_initializer='glorot_normal'))
```

AutoEncoders :-

Restricted Boltzmann Machines (RBM)

→ AutoEncoder is a neural network which is used to solve unsupervised data. has same number of input & output nodes.



→ In Auto Encoder the Output is same as input. In hidden layer, reduced Outputs are sent to Output neurons which are same as input
⟹ It says that the number of truly required inputs are two (in above case) since we reconstructed back from those.
↳ It has done the dimensionality reduction (non-linear)

→ Therefore if input variable are corelated non-linearly, Auto Encoders can be used for dimensionality reduction

Input

```
X = Input(name='inputs', shape = 28*28,
          dtype='float32')
```

Encoder

```
Encodedvector = Dense(32, activation='relu',
                      name='encoder')(x)
```

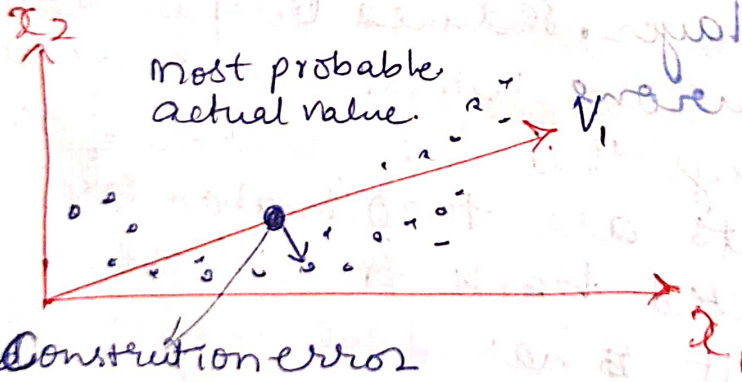
Decoder

```
decodedvect = Dense(28*28, activation='sigmoid', name='decoder')
                    (encodedvector)
```

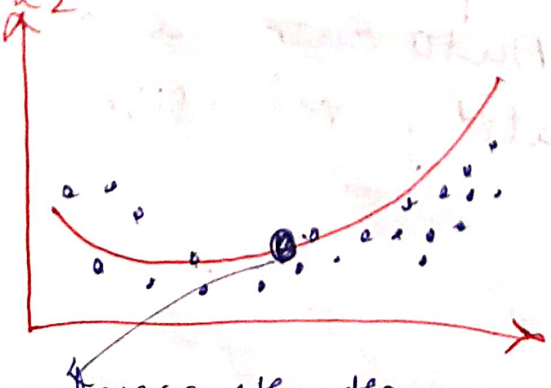
Print network Summary

```
MNist AutoEncoder = Model(inputs = X,
                          outputs = decodedvect)
MNist AutoEncoder.Summary()
```

PCA



Auto Encoder



A normal auto-encoder with multiple sigmoid/Relu layers will map the input data into a non-linear subspace and have lower reconstruction error

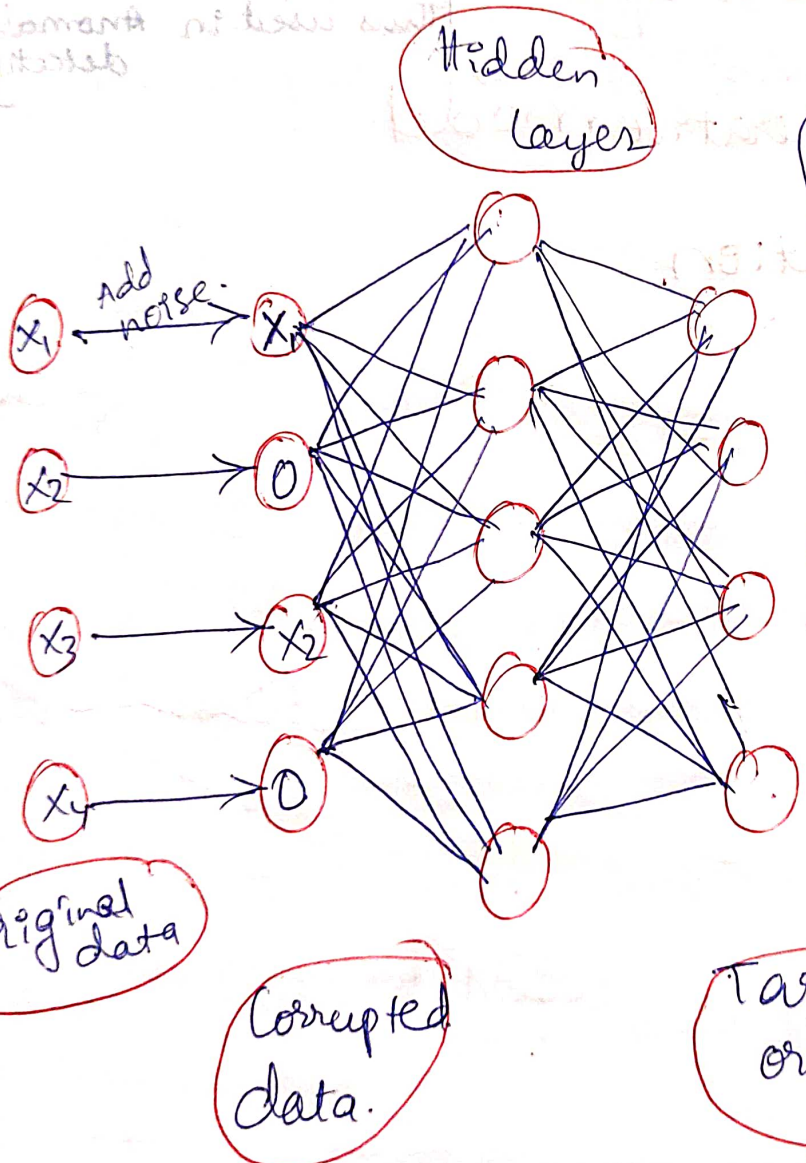
(*) Auto Encoder can be used as replacement of PCA.

(*) Auto Encoder is used to clean the noise in the image

↳ In the each pixel the intensity are stored ranging from 0-255 (255 is most dark)

↳ Noise is the incorrect pixel intensity stored

(*)



(*) When the output matches the original image error is low & output matches the corrupted image. then the error is high

This idea is used to train the model i.e we get correct original image

Denoising AutoEncoder

Autoencoder application:-

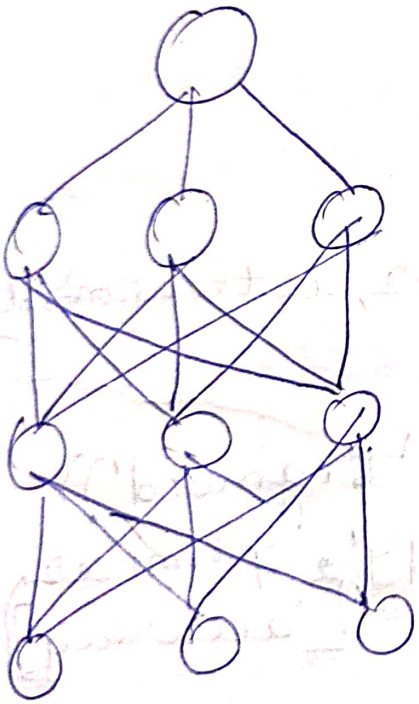
1. Image processing application
   - → De-noising
   - → Auto filling missing value.
2. Anomaly detection → Autoencoder is trained with same input & output tuning. If any data containing other variable as input, then the output will be diff. Thus used in Anomaly detection
3. Feature generation.
4. Learning generative model
5. Text translation

(*) Dropout :-

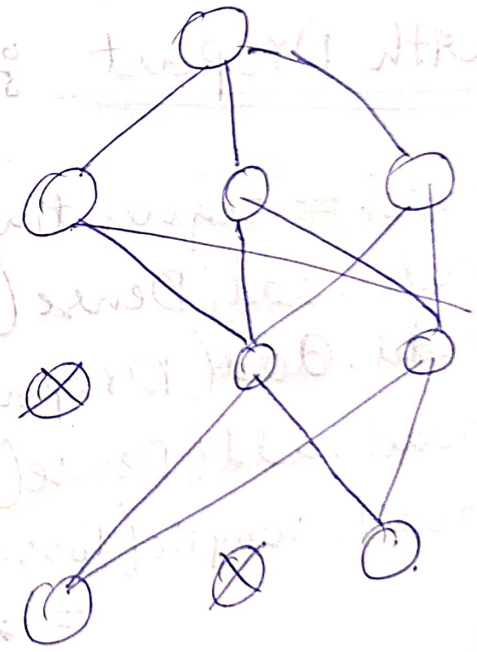
(Controlling overfit.)

↳ In this method, start with the complex neural network and after which cutting out the neurons randomly is done which helps neurons not to memorise but learn.

↳ This step is done using dropout technique during training the model.



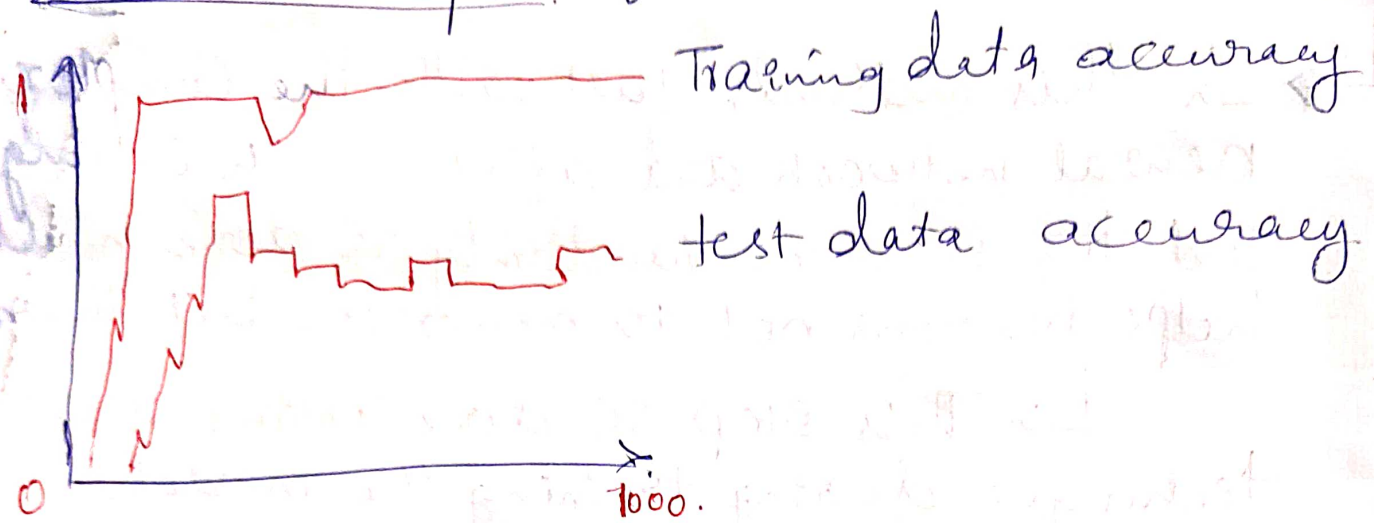
(a) Standard Neural Net



(b) After applying dropout

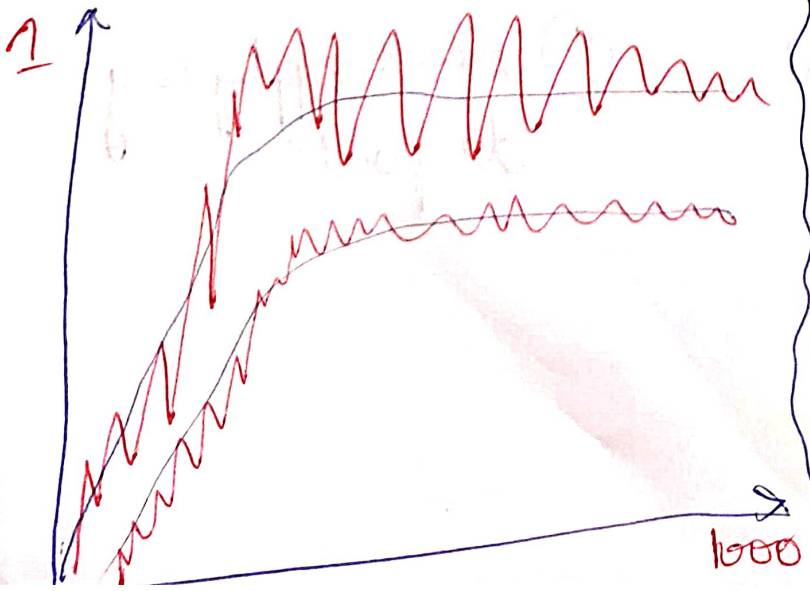
dropout implementation.

without dropout :-



with Dropout :-

```
model = Sequential()
model.add(Dense(500, input_dim=2, activation='relu'))
model.Add(Dropout(0.4))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer
    ='adam', metrics=['accuracy'])
```



(*) Training will be lot more noisy because diff set of neurons are dissappearing

∴ on an average there will be less diff b/w train & test accuracy.

∴ more accurate

Normalization :-

(Batch normalization)

* usually scaling of the data is done to train the model faster although scaling is optional.

* likewise in neural networks Batch normalization is done after every layer to train the model faster and for better accuracy.

```
model=Sequencial().
Dense(64, input_Shape=(4,),activation='relu')
BatchNormalization()
Dense(128, activation='relu')
BatchNormalization()
Dense(128, activation='relu')
BatchNormalization(),
Dense(64,activation='relu')
BatchNormalization()
Dense(1, activation='relu')
```

* Can be applied to a state-of-the-art image classification model.
* Achives same accuracy with 14 times fewer training steps
* Acts as a regularizer too.