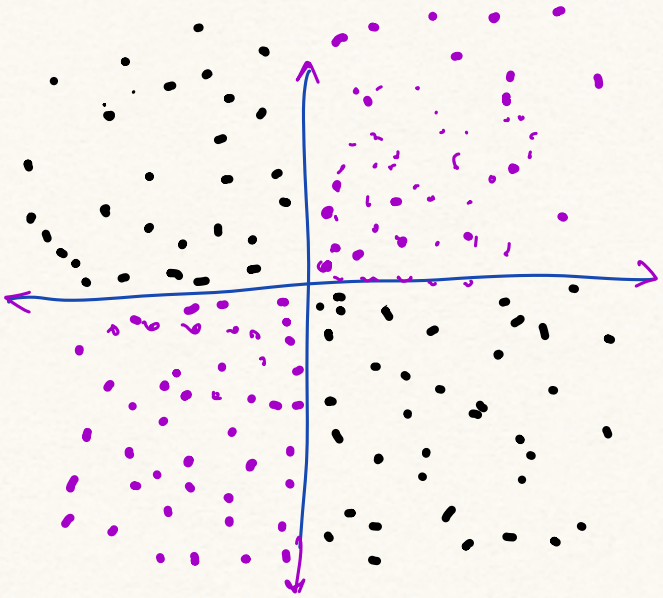
Neural Network



Non linear Classification problem because a linear function cannot separate all the blue dots from pink dots.

Hence Non-linear means you cannot perfectly predict a label with a model of the form $b + w_1 x_1 + w_2 x_2$, means decision surface is not a line

if we perform feature cross on our features x_1 & x_2 we can then represent non-linear relationship between two features using linear Model

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

where $x_3 = x_1 * x_2$

↳ feature cross between feature x_1, x_2

⇒ By adding feature cross x_1, x_2, model can learn hyperbolic shape that separates the blue dot & orange dots.

Hence → Neural Networks are family of model ~~who~~

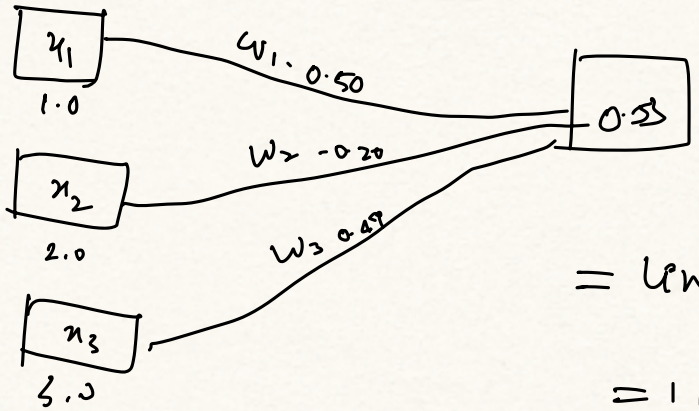
architecture designed to get non-linear patterns in data.
During training Model get the optimal values for features used to minimize loss given input data

⇒ Neural Networks learns non-linearity

$$\underbrace{y'}_{\text{O/P}} = b + w_1 x_1 + w_2 x_2 + w_3 x_3$$

(Input da)

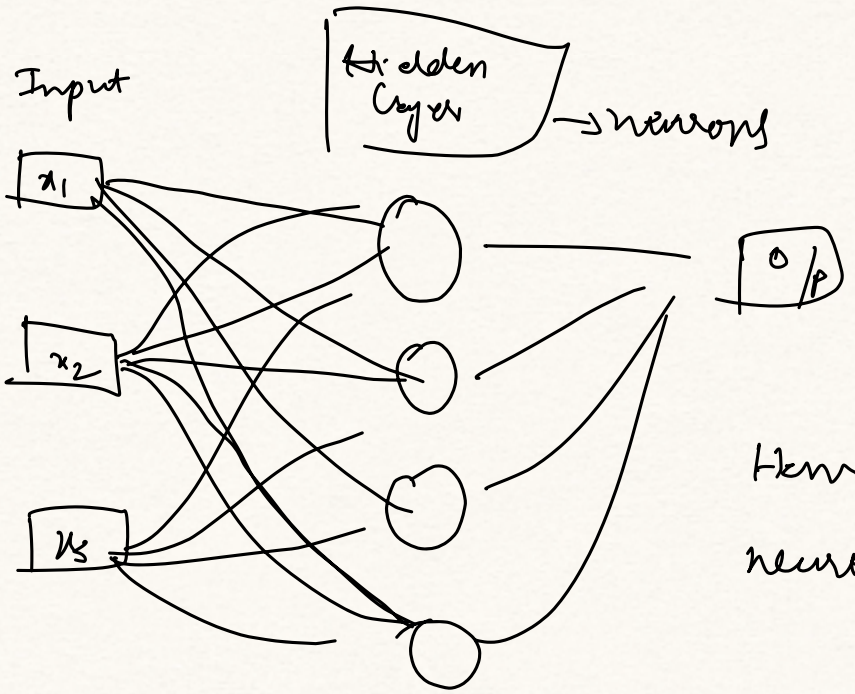
Input O/P



$= \text{Linear}(w_1 x_1 + w_2 x_2 + w_3 x_3 + b)$

$= 1(0.50 + -0.41 + -1.46 + 0.83)$

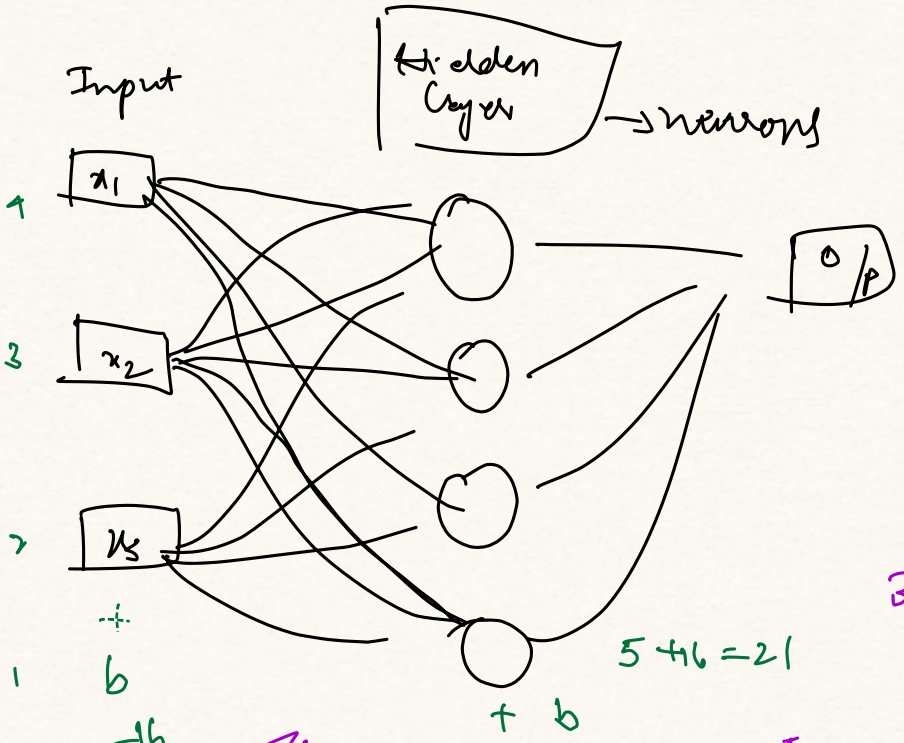
$= 0.53$



Adding one more layer to network between I/P & O/P. Hence this additional layer is called neurons

* The value of each neuron is calculated as the

Same way as the output of a linear Model, Sum of the product of each of its input (neurons of previous layer) × unique weight parameter plus the bias, In the same manner neurons in next layer calculates the hidden layer's neuron value as input



Too many parameters (weights and bias) does this neural network have

4 parameters used to calculate value of 1 in the hidden layer. i.e 4 node values.

3 weights & bias

Summs = 16

There are 5 parameters used to calculate the O/p value

4 weights & bias.

→ All this linear calculations i.e multiplications & additions will result a linear, Hence this model still doesn't calculate any non-linearities

Activation functions

To make linear network to learn non-linearity we have to introduce something like sigmoid function like we did in logistic regression

Sigmoid $\left(\begin{matrix} b \\ w_1 x_1 + w_2 x_2 + w_3 x_3 \end{matrix}\right)$

Here Here if we differentiate Sigmoid function to get the value of the derivative

Here sigmoid serves as activation function for neural network, on non linear transform of a neurons output value before the value is passed as input to activation of the next layer of the neural network

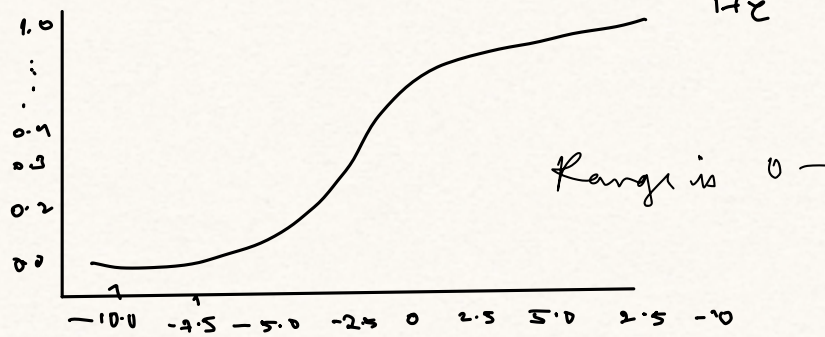
Common activation functions

1. Sigmoid
2. Tanh
3. ReLU

* Sigmoid function → we use sigmoid is called as sigmoid as it is S-shaped, which squeezes value between 0 – 1

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

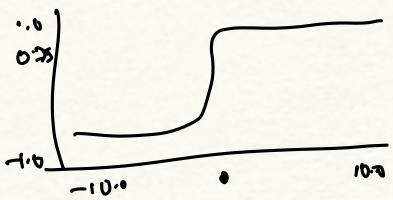
basically it is called as logistic function $F(x) = \frac{1}{1+e^{-x}}$



Range is 0 → 1

* The tanh (short for "hyperbolic tangent") function transforms input x to produce an output value between −1 & 1

$$F(x) = \tanh(x)$$



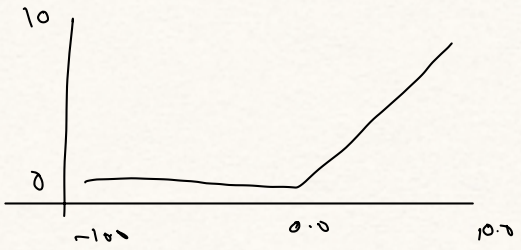
Range of tanh function is − 1 to 1

* Relu – Rectified linear Unit activation function (or ReLU)

→ if the input value 'x' is less than 0, returns 0.

* value 'x' is greater than or equal to 0, returns the input

$$F(x) = \max(0, x)$$



ReLU often works a little better as an activation function than smooth functions like sigmoid or tanh

because it is less susceptible to vanishing gradient problem during neural network training

Other functions

→ Mathematical function can serve as an activation function

$\sigma(w \cdot x + b)$

Keras provides out of box support for many activation functions

recommended is (ReLU)

Back Propagation

It is most common { algorithm / technique } used to train neural network, it makes ~~neural network~~ gradient descent feasible for multilayer neural network

Best Practices

Back propagation failure cases & most common ways to regularize a neural network

Vanishing gradients

gradients in neural network layers can become very small, when gradient values approach 0 for the lower layers, the gradient are said to vanish

Layers with vanishing gradients train very slowly or not at all.

ReLU activation function can help

Exploding Gradients

If weight are very large, then the gradients for the layer involve

product of many large terms, gradients that get too large to converge

Batch normalization → can help exploding gradients

Dead ReLU Units

When weighted sum falls under zero, ReLU can stuck and contributes nothing to network o/p, Hence gradients can no longer flow through it during back propagation

Learning Rate can solve or help ReLU units from dying.

Dropout Regularization

→ another regularization is useful for neural networks, It works randomly "dropping out" Unit activations in a network for single gradient step. The more you drop the more regularization

0.0 = No dropout regularization
1.0 = Drop out all nodes → Model learns nothing
Values between 0.0 and 1.0 → Most useful

Multi class Classification

In Binary Classification

Spam ← not Spam

Tumor is malignant or not malignant. (benign)

Multi class

→ Is this dog beagle, basset hound or blood hound

flower → Siberian Iris, Dutch Iris, Blue flag iris or Dwarf crested Iris

plane → Boing, Airbus etc

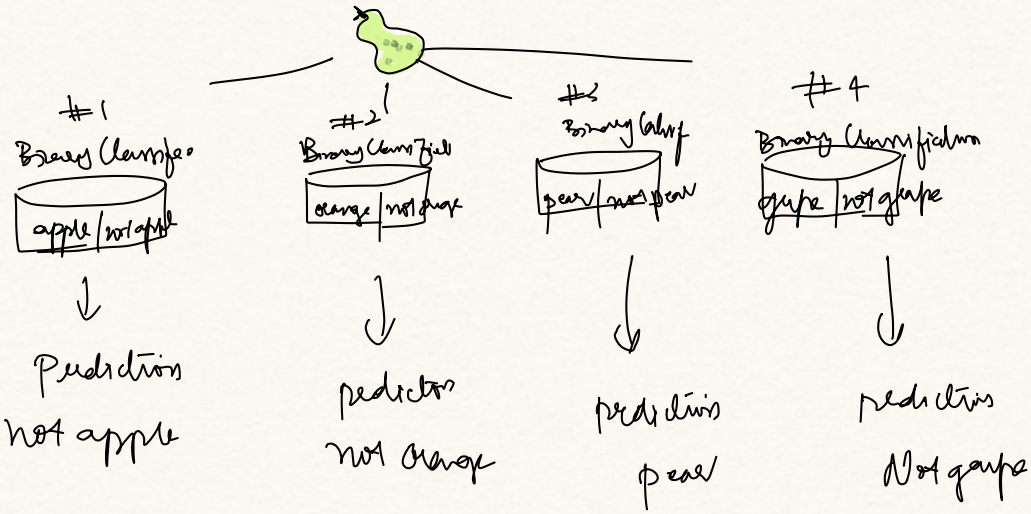
Image → apple, banana, candy, dog etc.

Variants of Multiclass

1. One-vs-all
2. One-vs-one — also called as softmax.

1. One-vs-all uses binary classification for a series of yes or no predictions across multiple possible labels

Eg: given classification problem with N solutions a one-vs-all N-separate binary classifications → one binary classifier for each problem & possible outcome



One Vs One (Softmax)

In binary classification probability is compared are determined & dependently from other.

⇒ When we want to predict probability of every point to each other item relatively is One Vs One

⇒ Here one vs one we apply function called softmax which assigns decimal probabilities to each class in multiclass problem. Such that all probabilities add up to 1.0.

$$\frac{e^{(w_i^T x + b_i)}}{}$$

$$p(y=j/x) = \frac{e^{(w_j^T x + b_j)}}{\sum_{k \in K} e^{(w_k^T x + b_k)}}$$

"it basically extension of the formula = $\frac{1}{1-e^x}$

Softmax options

- full softmax → Softmax calculating probability for every possible class
- Candidate Sampling → Softmax calculate probability for all positive labels with only sample of negative labels

Eg: if we want to determine beagle or hound we don't need to give all the non-doggy stuff.

Softmax can be used to come class to one class and it expects to be similar class.