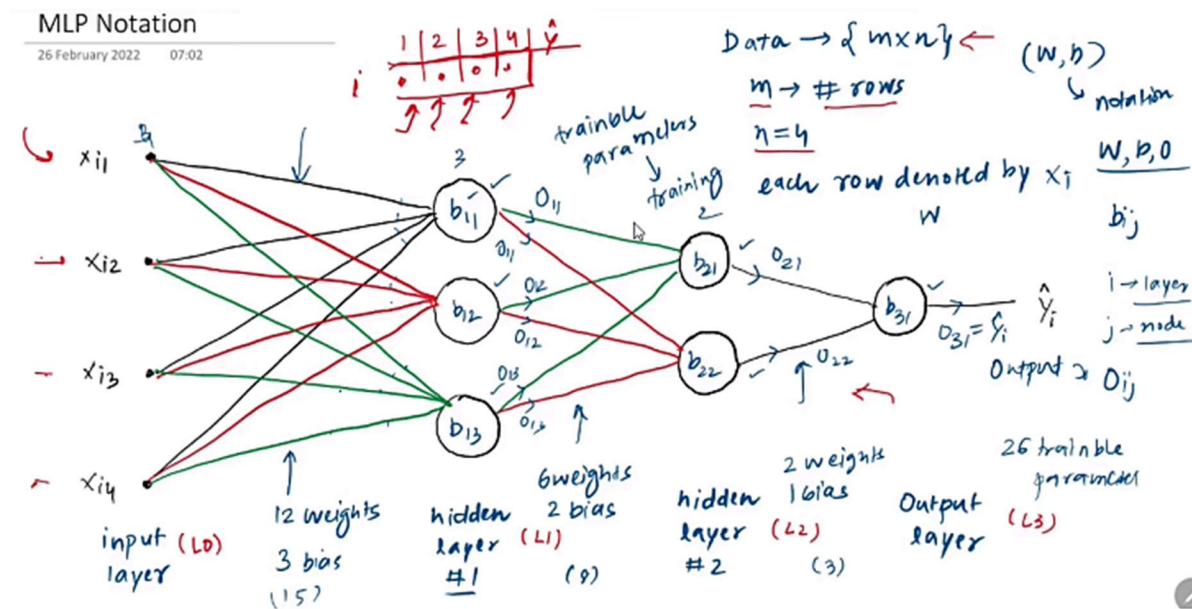
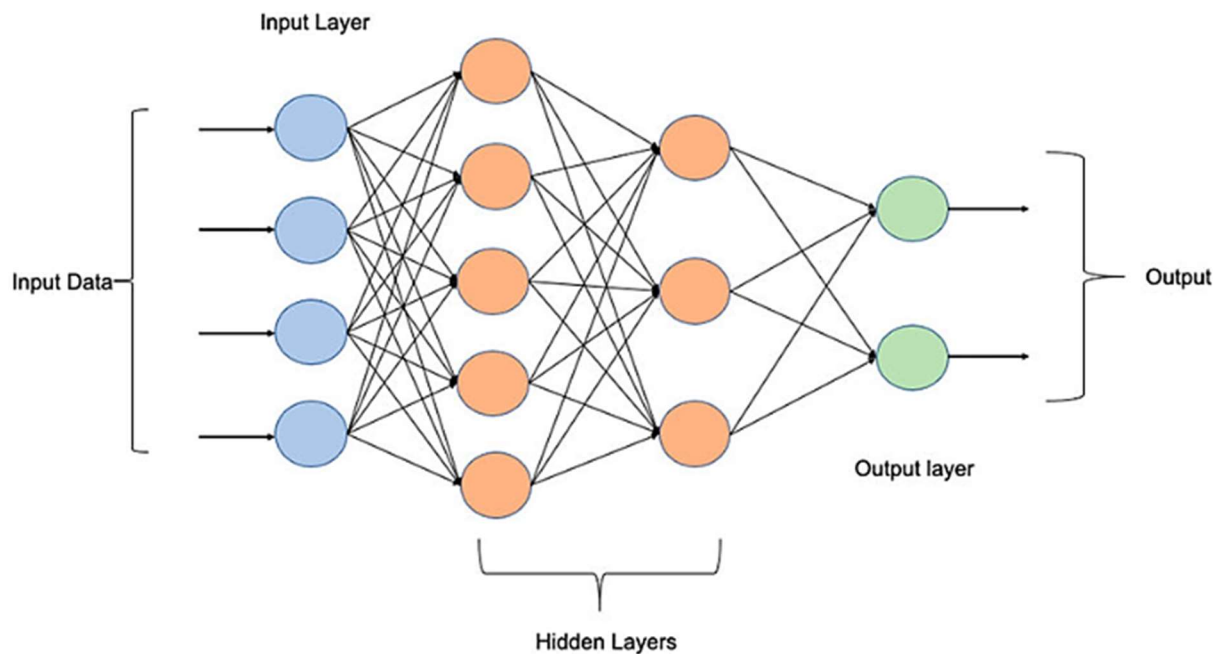


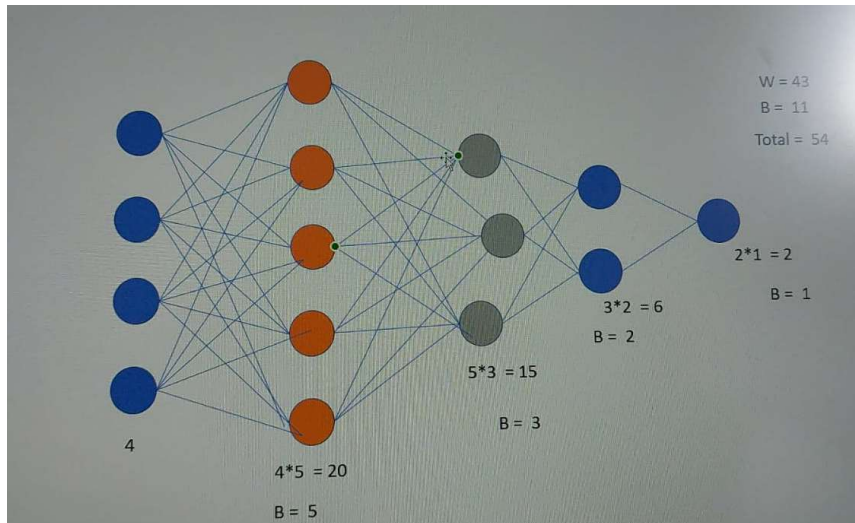
MLP Notation



Trainable parameter



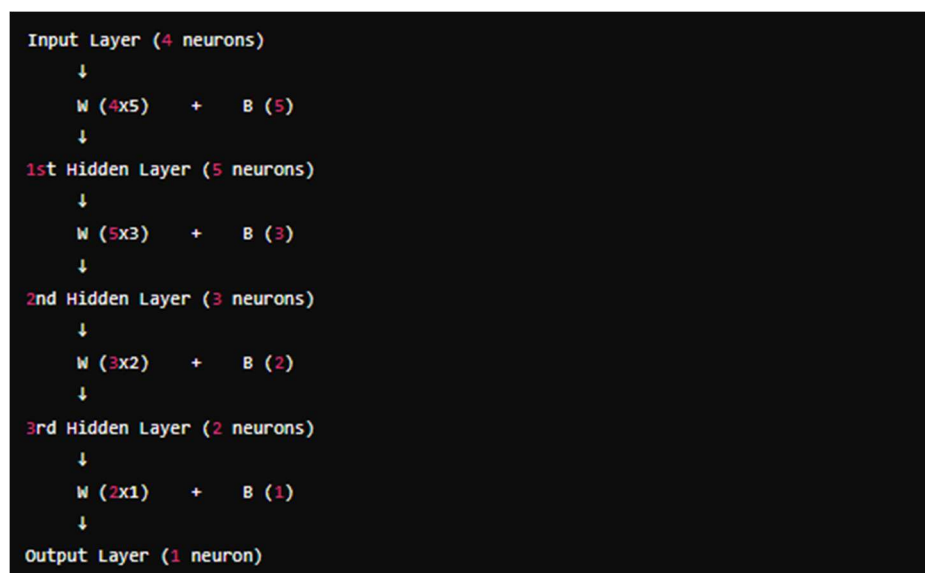
Calculate Trainable parameters and label it? Ans= 54



- Input layer = 4
- 1st Hidden layer = 5
- 2nd Hidden layer = 3
- 3rd Hidden layer = 2
- Output layer = 1

Now,

- Trainable parameters between input layer and first hidden layer: $4 \times 5 = 20 + 5 \Rightarrow 25$.
- Trainable parameters between first and second hidden layers: $5 \times 3 = 15 + 3 \Rightarrow 18$.
- Trainable parameters between second hidden layer and third layer: $3 \times 2 = 6 + 2 \Rightarrow 8$.
- Trainable parameters between third hidden layer and output layer: $2 \times 1 = 2 + 1 \Rightarrow 3$.
- Total number of trainable parameters of the neural net: $25 + 18 + 8 + 3 = 54$.



Here's how you can visualize it in a simplified manner:

1. **Input Layer:** 4 neurons
2. **1st Hidden Layer:** 5 neurons
 - Connections: 4 (input) x 5 (hidden) = 20 weights
 - Biases: 5
3. **2nd Hidden Layer:** 3 neurons
 - Connections: 5 (hidden) x 3 (hidden) = 15 weights
 - Biases: 3
4. **3rd Hidden Layer:** 2 neurons
 - Connections: 3 (hidden) x 2 (hidden) = 6 weights
 - Biases: 2
5. **Output Layer:** 1 neuron
 - Connections: 2 (hidden) x 1 (output) = 2 weights
 - Biases: 1

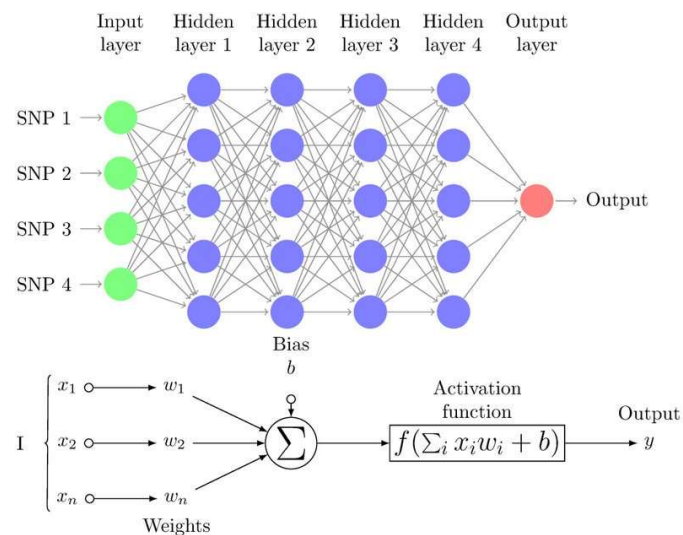
Total Parameters:

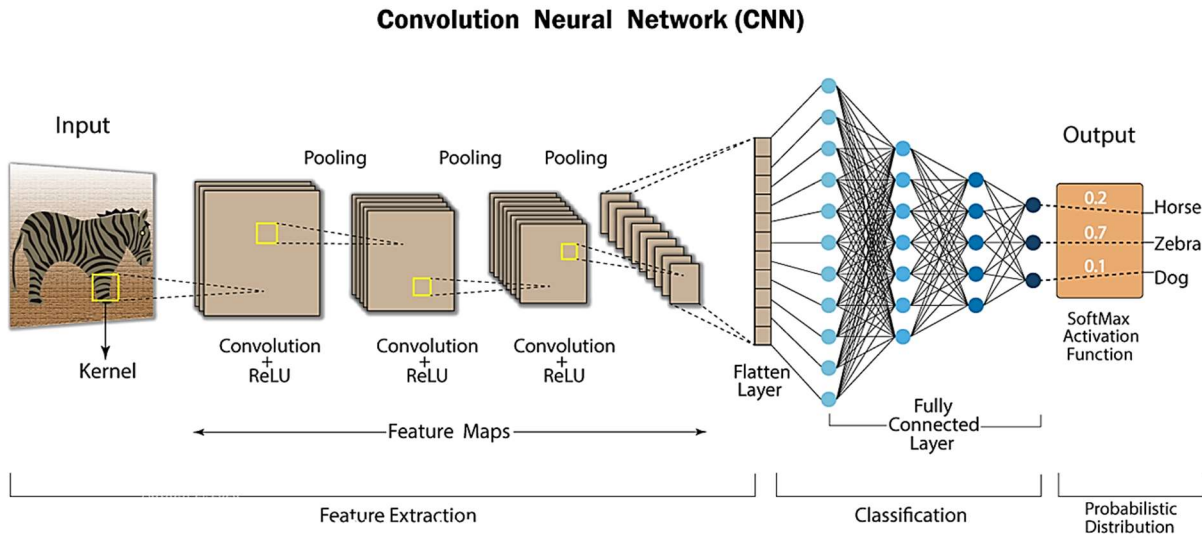
- Weights: 20 (Input to 1st Hidden) + 15 (1st Hidden to 2nd Hidden) + 6 (2nd Hidden to 3rd Hidden) + 2 (3rd Hidden to Output) = 43
- Biases: 5 (1st Hidden) + 3 (2nd Hidden) + 2 (3rd Hidden) + 1 (Output) = 11

Grand Total: 43 (Weights) + 11 (Biases) = **54** trainable parameters.

For a graphical representation, imagine:

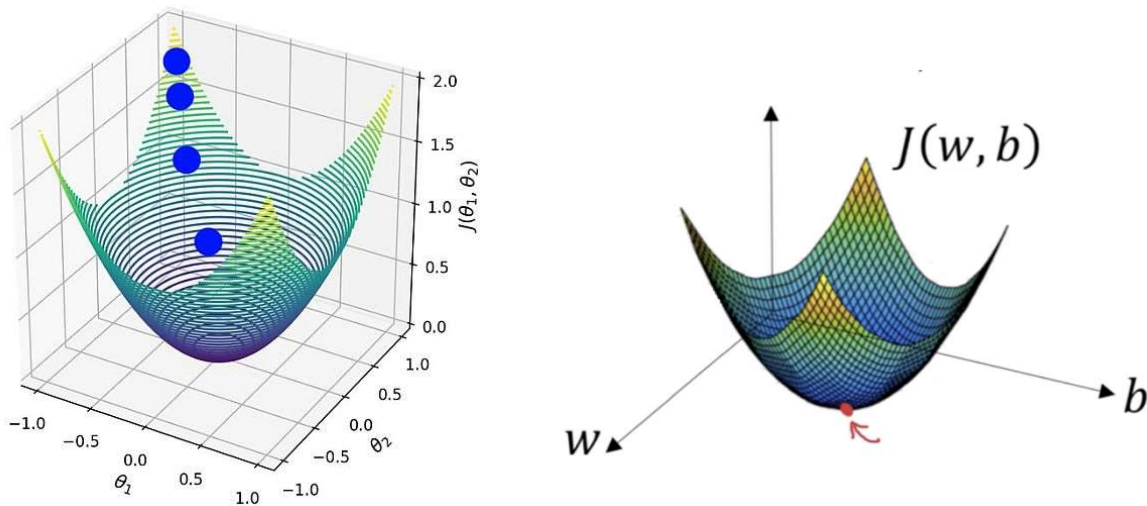
- Each layer is depicted as a group of circles (neurons).
- Arrows between the layers represent the weights.
- Each neuron in a layer connects to every neuron in the next layer, with weights and biases added for each connection.





What Is Gradient Descent in Machine Learning?

Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent in machine learning is simply used to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible.

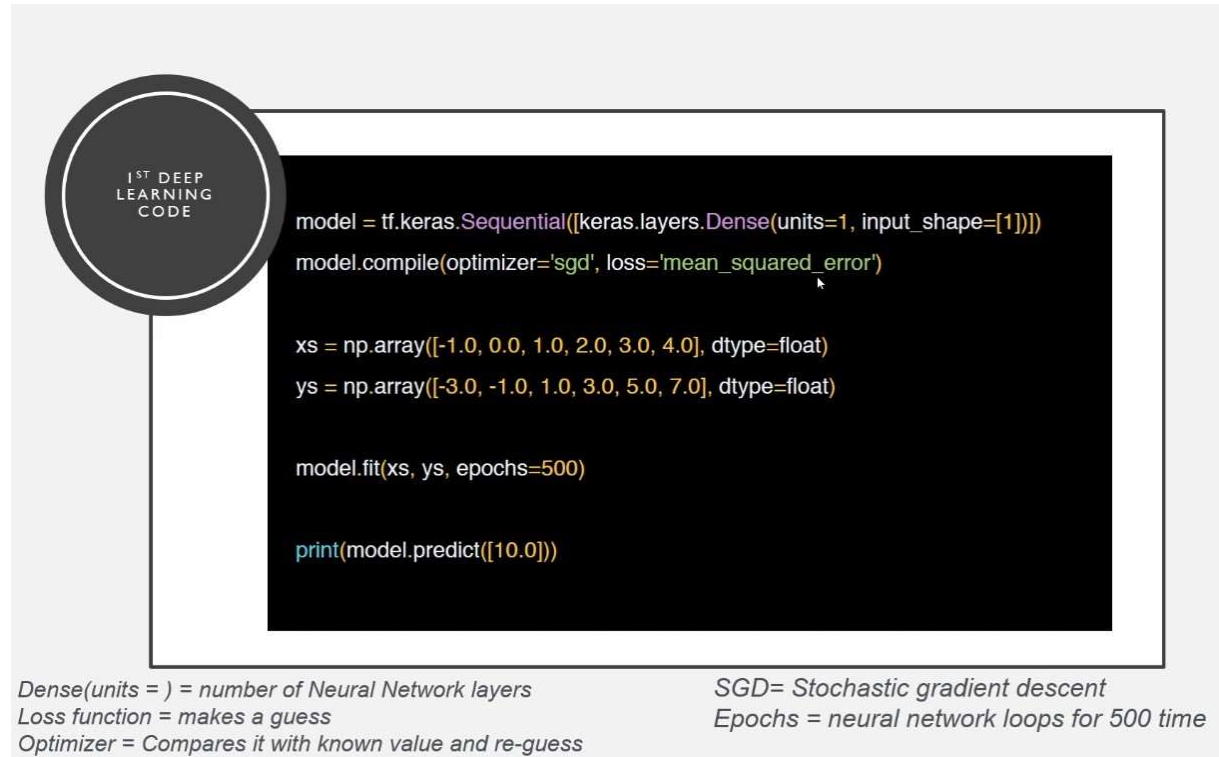


There are two types of gradient descent mainly used in deep learning:

- Batch gradient descent
- Stochastic gradient descent

Batch gradient descent takes longer to converge since it computes the gradient using the entire training dataset in each iteration. Stochastic gradient descent, on the other hand, can converge faster since it updates the model parameters after processing each example, which can lead to faster convergence

Neural Network



```
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')

xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model.fit(xs, ys, epochs=500)

print(model.predict([10.0]))
```

Dense(units =) = number of Neural Network layers
Loss function = makes a guess
Optimizer = Compares it with known value and re-guess

SGD= Stochastic gradient descent
Epochs = neural network loops for 500 time

For model layers:

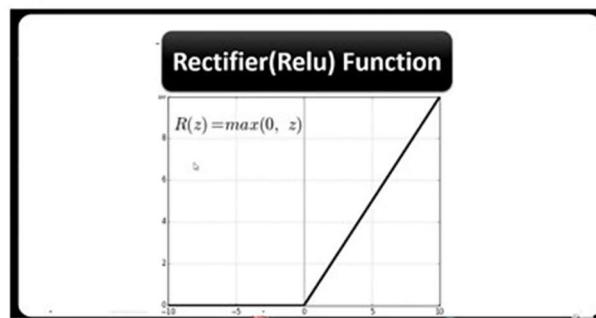
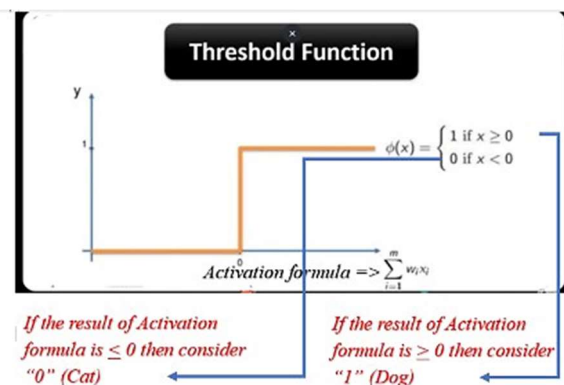
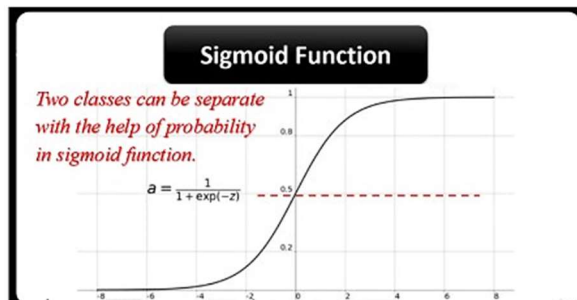
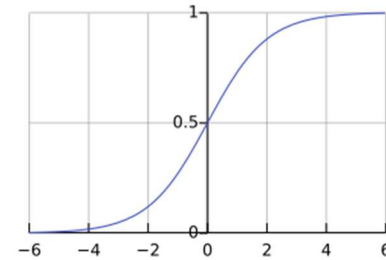
- Create sequential building
- Use optimizer and loss
- Epoch means iteration

Sigmoid function

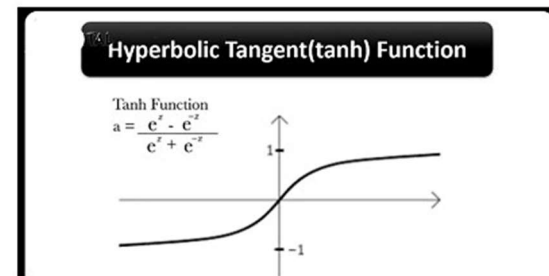
A sigmoid function is any mathematical function whose graph has a characteristic S-shaped or sigmoid curve.

A common example of a sigmoid function is the **logistic function** shown in the first figure and defined by the formula:^[1]

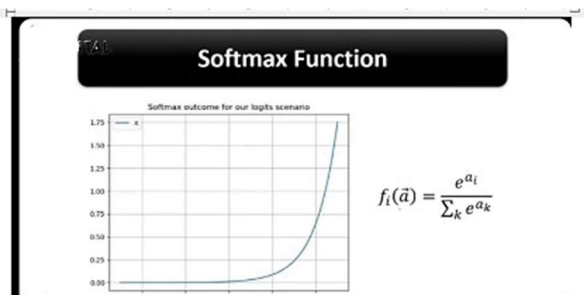
$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x} = 1 - \sigma(-x).$$



By default activation function is used is called Rectifier (Relu) Function. 1st we will find out the value of "z"



All above functions are use for differentiate b/w two classes of an object



If we have more than two classifications of an object then we will use "Softmax Function"