Neural Network:

- Y = wX + b(1) => NN with 1 Input, 1 Output
  - Known as **'Linear Unit'** or **'Perceptron'**
  - Y: Dependent Variable; X: Independent Variable; b: Bias; w: Weights
  - Y = w1X1 + w2X2 + w3X3 + ... + b(1) => NN with n Inputs, 1 Output
  - Generalize Form: Y = sumation(WiXi) + b

- We introduce Bias so that model can take decisions independent of inputs
- In NN every layer has it's own Bias

- **Acivation Function:** We introduce Activation Function to Achieve Non-Linearity, without them our model will be limited to Lines, Planes & HyperPlanes
  - Rectifier Function => y = max(0, x); Rectify or Nulify Negative Values
  - A Linear Unit(NN with 1 Input, 1 Output) with Rectifier as Activation Function is known as Rectifier Linear Unit or ReLU

- **Loss Function:** Defines how a Model Performed
  - Loss Function = Yactual - Ypredicted

- **Optimizer:** Is an optimization function that tells model how to change it's weights
  - Example: SGD: Stochastic Gradient Descent
    - Stochastic = By Choice; Gradient = Slope of Loss Function; Descent = Get to Global Minima
  - Steps for Optimization:
    - Sample some training data(MiniBatch) and run it through the network to make predictions.
    - Measure the loss between the predictions and the true values.
    - Finally, adjust the weights in a direction that makes the loss smaller.
  - **MiniBatch:** For each Iteration of Optimization a random set of datapoints are selected called as **MiniBatch** or **Batch**
  - **Epoch:** Complete round of training data is called an **Epoch**
    - Number of Epochs determine how many times your model will look at a Training Sample
  - **Learning Rate & Batch Size**
    - A smaller Learning Rate means the network needs to see more Minibatches

- **Overfitting:** When your model remember your training data.
  - Model does very well for training data but does not perform well for test data
  - It can be seen with a Graph of Loss vs Training for both Training Data and Testing Data

- **Underfitting:** When your model is weak and learns nothing.

- **Capacity of Model:** A model's capacity refers to the size and complexity of the patterns it is able to learn.
  - If Model is Underfitting the data, then increase its Capacity.
  - To increase Capacity do one or both of following:
    - Make Model Wider: Increase number of Neurons in Hidden Layers
    - Make Model Denser: Increase number of Hidden Layers

- **Early Stopping:** Interrupting the Training when Validation Loss is not Decreasing any more is known as Early Stopping
  - min_delta: Minimium amount of change to count as an Improvement
  - patience: How many epochs to wait before stopping
  - Example:

    -
      ```
      early_stopping = EarlyStopping(
      min_delta=0.001, # minimium amount of change to count as an improvement
      patience=20, # how many epochs to wait before stopping
      restore_best_weights=True,
      )
      ```
    - This means If there hasn't been at least an improvement of 0.001 in the validation loss over the

    - This means,if there hasn't been at least an improvement of 0.001 in the validation loss over the previous 20 epochs, then stop the training and keep the best model you found.

- **Dropout Layer:** Simple way to prevent model from overfitting
    - It's not an actual Layer with Neurons, but it's a preprocessing layer
    - Can add Dropout Layer to every layer of model even for Input Layer but not for Output Layer
    - As name suggents it randomly drop units in a NN so that Unit has to find more General Patterns rather than Robust Patterns
    - Examaple:
        - layer.Dropout(rate=0.3), # apply 30% dropout to the next layer

- **Batch Normalization Layer or BatchNorm Layer:** Help correct a training that is slow or unstable.
    - SGD changes weights with proportion to activation produced by data, it means, large value data will produce larger activation and hence larger shift in weights, whereas, small value data will produce smaller activation and small shift in weights, hence making the learning unstable
    - To avoid this, it's a best practice to use Normalization (to put all your data on common scale or squize the data into 0 & 1 for example)

```
In [ ]:
```