Machine Learning

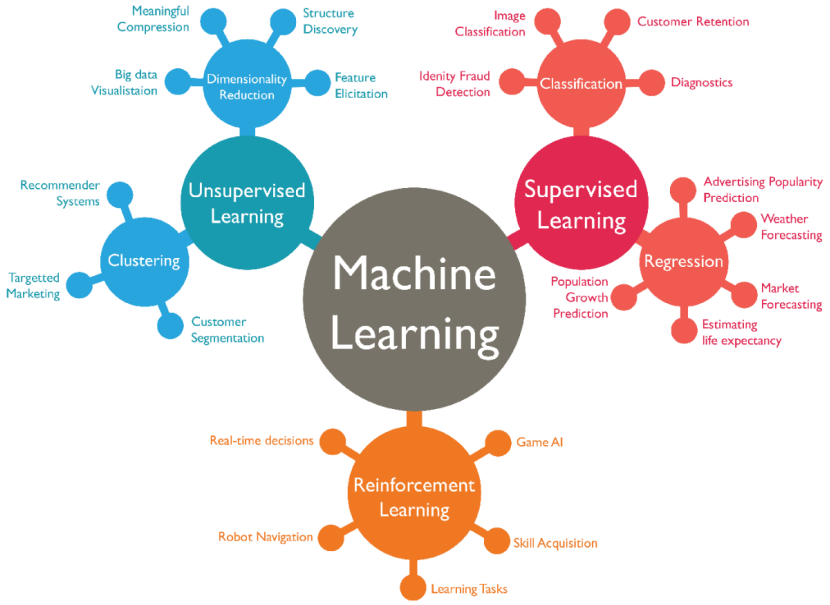
Machine learning is a branch of Artificial Intelligence (AI) that is the study of algorithms and models that train on data sets to improve their performance in predicting values for new data. Machine learning can be implemented through algorithms and neural networks, both of which can be used for supervised and unsupervised learning.

Branches of Machine Learning:

- **Supervised Learning:** The model is trained with label data. This means that in addition to receiving features/attributes from the data, the model also receives the expected output. Then, when given new data the model can output its prediction.
    - **Branches:**
        - **Classification**: Given data with labels of the class, the model trains to classify new data. **The outputs are discrete values.**
            - **Examples:** Given 5 pictures of 5 students each along with their name, classify a new picture of a student.
            - **Common Algorithms:**
                - **K Nearest Neighbors (KNN)**
                - **Naive Bayes (NB)**
                - **Support Vector Classifier (SVC)**
                - **Decision Trees (DT)**
                - **Random Forests**
                - **Logistic Regression**
        - **Regression**: The model is given data with the expected outputs and trains on the data to predict output for new data. **The outputs are continuous values.**
            - **Examples:** Given data of bitcoin market share, trade volume, etc... and price, predict the price for new data.
            - **Common Algorithms:**
                - **Linear Regression**
                - **Logistic Regression**
                - **Polynomial Regression**
                - **Support Vector Regression (SVR)**
                - **K Nearest Neighbors Regression (KNNR)**
    - **Purpose:** Supervised learning is used for training a model on data that contains the target values of the attributes. It can be used for predicting house price values given relevant data or for classifying plants.

- **Unsupervised Learning:** The model is trained with unlabeled data. What this means is that it is not told the class of the data or the expected values. It has to determine on its own what to output.
    - **Branches**:
        - **Clustering**: The model discovers inherent groupings in the data on its own.
            - **Example:** Given 25 pictures with 1 student in each of them, match the face in a new picture with a face group from the 25 pictures. *The model is not told how many faces there are or which faces are similar.*
        - **Association**: The model discovers rules that describe large portions of data, such as people that buy X also tend to buy Y.
    - **Purpose:** Unsupervised Learning is used on large data sets where it is very time-consuming for a human to go through and label the data for the model to learn from. Instead, the model predicts the values.

- **Reinforcement Learning:** Aims at using observations from training to maximize the reward while minimizing the risk. The model learns from its experiences to explore a full range of possible states that it uses to make optimal decisions:
    - **Applications**: Used for training bots that play games. The model studies the possible moves that it can make and uses its past experience to determine which move will generate the highest reward.



Source: askdatascience.com

General Terminology:

- **Underfitting** - The model is too simple and makes too many assumptions about the data and cannot find the underlying trend. **High variance, low bias.**
- **Overfitting** - The model is trained too well on the data and "memorizes" it. Therefore, it captures the noise in the dataset, making it less accurate. **Low bias, high variance.**
- **Generalization -** Refers to an algorithm's ability to be effective across a range of inputs.
- **Bias -** Difference between the expected output and the average prediction of the model.
- **Variance** - The amount by which the model's predictions for a data point vary.

Algorithms:

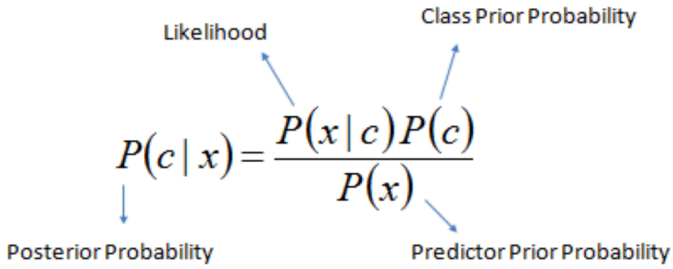
One of the methods of doing machine learning is through using algorithms. Many of the popular machine learning algorithms come from the scikit learn module which can be installed through the pip package manager.

Using an algorithm:

1. A classifier must be created and have its hyperparameters optimized.
2. A classifier must be trained on the data.
3. A classifier must be tested on the test data.

On the following page are some of the popular algorithms for supervised learning:

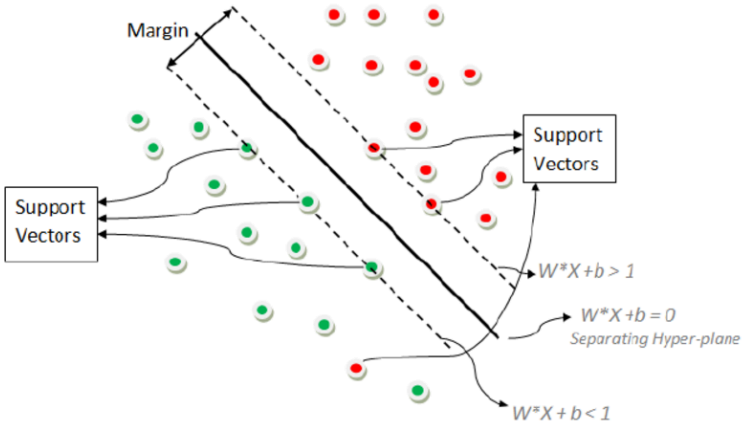
- **Naive Bayes:** It is a classifier based on Bayes' theorem and classifies values independently of other values. It uses probability to predict a class.



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

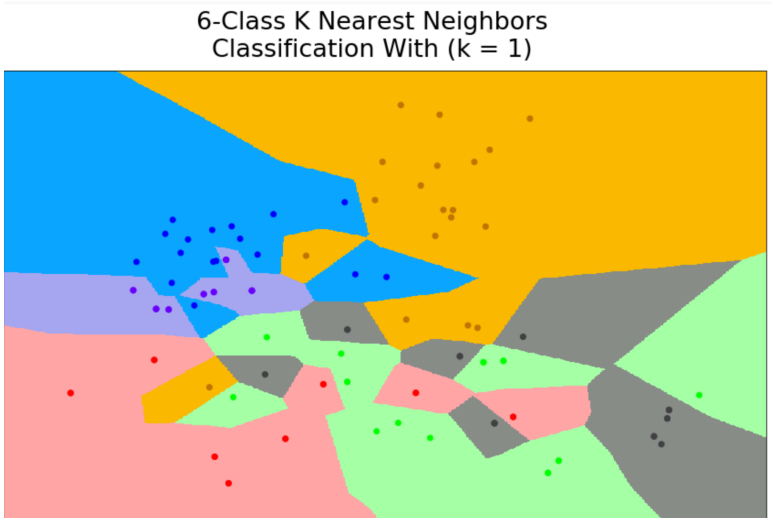
  - **Pros:** Easy and fast prediction of classes.
  - **Cons:** Unable to make predictions for a class that was not observed in the training set. Makes an assumption that data values are mutually exclusive.
  - **Types:**
    - **Gaussian Naive Bayes:** Used in classification. Assumes that features follow normal distribution. Data is continuous.
    - **Multinomial:** Used when data has discrete values. *Example: Movie ratings can either have 1, 2, 3, 4, or 5 stars.*
    - **Bernoulli:** Assumes that all features have binary values.

- **Support Vector Classifier:** Finds a hyperplane that separates the classes with the maximum possible perpendicular distance between the nearest points from each class. This *distance is called the **margin*** and the *nearest points are called **support vectors***.

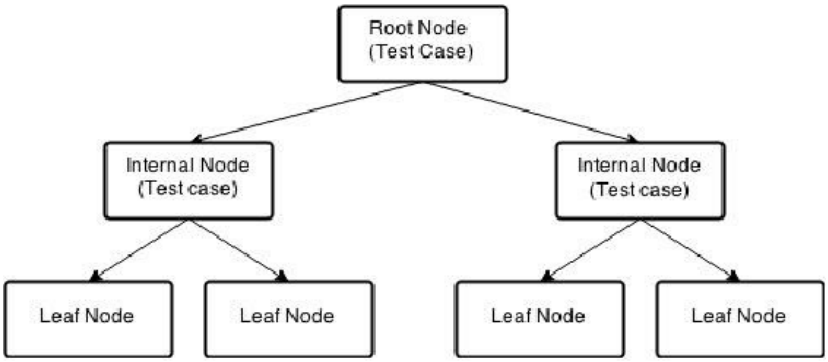


  - **Pros:** It has a hyperparameter for regularization that helps with avoiding overfitting.
  - **Cons:** Have to find the right hyperparameters such as the kernel type.
  - **Kernel Types:** Kernels define the shape of the hyperplane.
    - **Linear:** The hyperplane is *linear*.
    - **Poly:** The hyperplane is in the shape of a *polynomial figure.*
    - **RBF:** Space of Gaussian distributions.
  - **Comparison:**
    - **Learning time**: Linear < Poly < RBF
    - **Fitting Ability:** Linear < Poly < RBF
    - **Overfitting Risk**: Linear < Poly < RBF
    - **Underfitting Risk**: RBF < Poly < Linear
    - **# of Hyperparameters**: Linear(0) < RBF(2) < Poly(3)

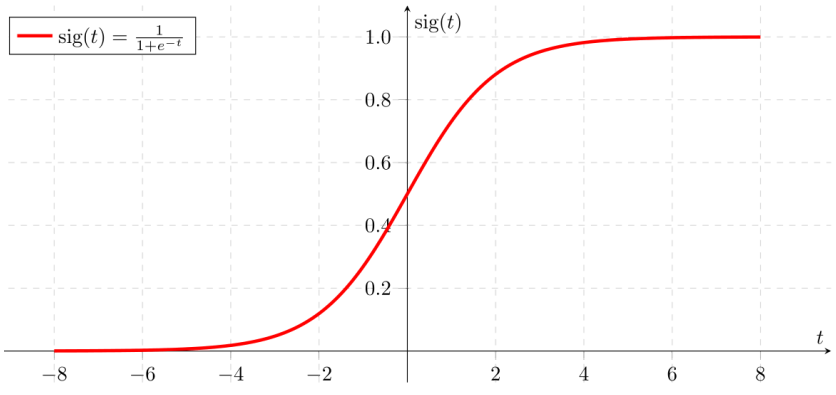
- **K Nearest Neighbors:** Calculates the Euclidean distance between a point of data and the k nearest points of pre-classified data. It classifies the new point as the class of the majority of its neighbors.
    - **Pros:** Simple algorithm. No assumptions. Evolves with new data. One hyperparameter to adjust.
    - **Cons:** Very sensitive to outliers. Sensitive to an imbalance in the number of data points per class.



- **Decision Trees:** Flowchart-like tree structure that branches to illustrate outcomes for different decisions.
    - **Pros:** Easy to understand. Time Complexity: O(n) where n is depth.
    - **Cons:** Unstable with low variance in the data.
    - **Metrics for Splitting the Tree:**
        - **Gini** - Measures the probability of a random sample being classified incorrectly.
        - **Entropy** - Splits the tree based on the information gain.



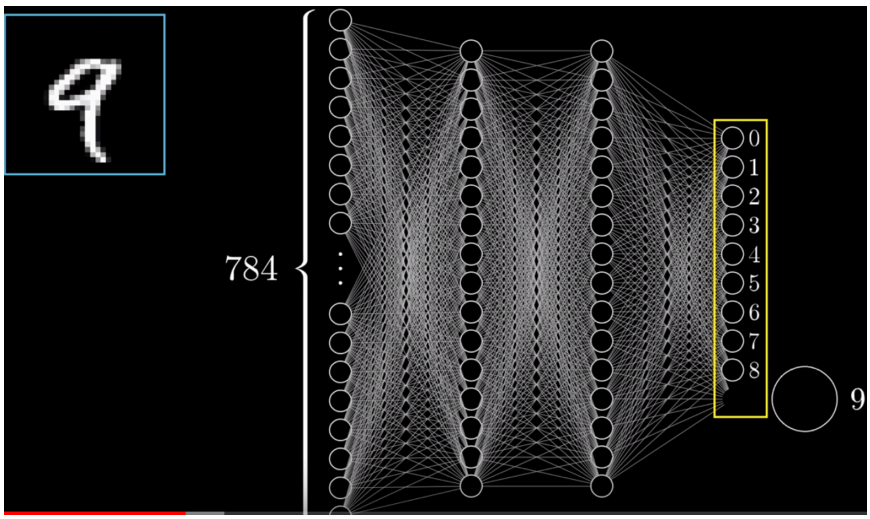
- **Logistic Regression:** Uses the logit function, also called sigmoid function, to calculate probabilities in order to make predictions.
    - **Pros:** Easy and fast prediction of classes.
    - **Cons:** Probability for predicting classes - may not work efficiently for multiple classes.
    - **Types:**
        - **Binary Logistic Regression:** Two classes, hence binary.
        - **Multinomial Logistic Regression:** Used for classifying multiple classes.



Neural Networks:

Artificial Neural Networks (ANNs) are a type of machine learning model loosely inspired by the biological structure of the brain. They are used in machine learning to be trained on data in order to analyze new data.

A neural network is made up of multiple layers of neurons. The first layer is known as the input layer, the last layer as the output layer, and the layers in between as the hidden layers. An example of the purpose of the neural network layers is with digit recognition from the mnist data set. The first layer of the network will receive the 784 pixels of the digit's image as input. In the following hidden layers, the shapes and lines making up the digit will be broken up continuously and passed on to the following layer where the model will update its weights and biases for each neuron in the layer. Finally, based on the input from the last hidden layer, the output layer will output the digit that has been recognized.



Source: 3Blue1Brown

Algorithms vs Neural Networks:

Neural networks take longer to train than algorithms do and are better for unsupervised learning because of their more robust structures. Additionally, it is optimal to train neural networks with fewer attributes. **Principle Component Analysis (PCA)** is a method that reduces the dimensionality of a data set consisting of many attributes whilst retaining the variation in the dataset. This method is used to limit a dataset to only a few features for training.

TensorFlow and Keras for Deep Learning - TensorFlow is one of the most famous machine learning libraries and it provides access to many deep learning models. Keras is a higher-level API built on top of TensorFlow which makes it simpler to build, compile, train, and test a robust neural network. On the other hand, TensorFlow provides more functionality for its models.

Keras's Sequential Model - Keras's most basic model is the Sequential model which is essentially a neural network. It has many different hyperparameters such as learning rate, batch size, epoche size, and more in addition to being optimized based on different types of losses

On the following page is information about the steps for using a neural network.

Parameters - Parameters are the data values that are being given to the model to train on.

Hyperparameters - Hyperparameters are the features of a model that can be tuned to provide optimal results.

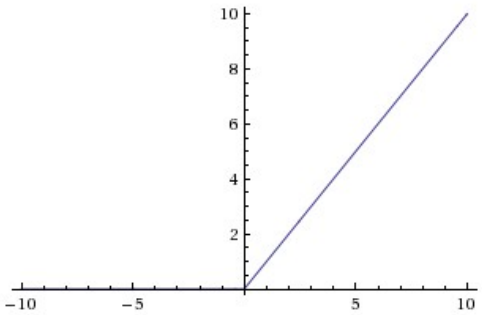
Steps for training and testing a neural network:

1. **Model Creation -** First the model must be created by instantiating an object of the Sequential model through its default constructor: `model = Sequential()`. In the creation process, the *density* of the model and type of *activation function* for each layer is defined.
    a. **Sample Code:**

    ```
    #Create the Sequential model
    model = Sequential()
    #Add 8 layers to the model with the softmax activation.
    model.add(Dense(8, activation = "softmax", input_dim = 4)) #input_dim is # of feature
    ```

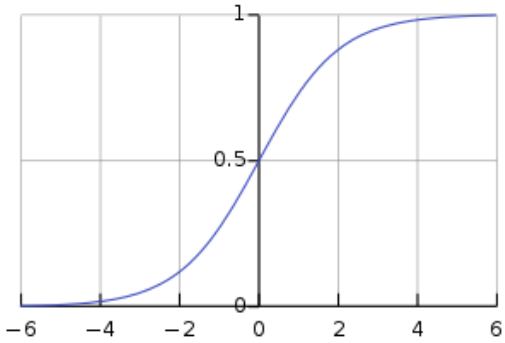
    b. **Density:** The density is the number of layers in a model.
    c. **Activation:** The activation parameter in the Dense() specifies how input from one layer will be processed to give output to the next layer which will then be received as input.

    Examples: Common examples of activation functions are -



Rectified Linear Units (ReLU)

Formulas: $y = max(0, x)$



Sigmoid

$y = \frac{1}{1+e^{-x}}$

    d. **Vanishing Gradient Problem -** Gradient-based activation function methods understand the values of parameters by "learning" how a change in the parameters' values can affect the neural network's output. However, if the change in the parameters' values is very insignificant, then the neural network can't understand the parameter's effectively. This is known as the **Vanishing Gradient Problem**.

    Cause - The cause of this problem is due to the choice of the activation functions. Activation functions such as tanh and sigmoid can be considered to "squash" their input into a very limited range when compared to activation functions like ReLU, *(the graphs are shown above).* While sigmoid has a range of 0 to 1and tanh has a range of -1 to 1, ReLU's range is from 0 to infinity because it maps x to $max(0, x)$.

2. **Model Compilation -** After creating the model, it must be compiled. In this stage, the type of *loss function* that the neural network will train based off of and the *optimizer method* will be declared. Additionally, other hyperparameters such as the *momentum* and *learning rate* can be specified.
    a. **Sample Code:**

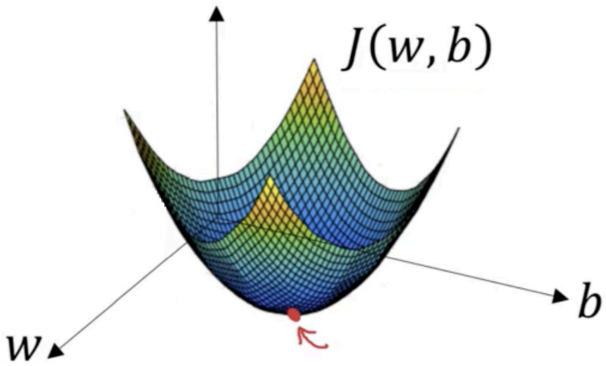
    ```
    #The optimizer being used is Adamax. The learning rate is 0.002. The momentum hasn't been declared since it
    #has a default value.
    opt = optimizers.Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0)
    ```

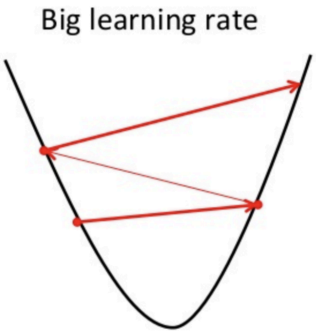
#compile the model with the optimizer, metrics, and loss function of "mean_absolute_error".
model.compile(optimizer = opt, loss = "mean_absolute_error", metrics = ['accuracy'])

b. **Optimizer:** The optimizer is a method that is used to minimize the loss function during the training process of a model. **Examples:** Adam, Adagrad, RMSProp.
   i. **Learning Rate:** One of the parameters that can be passed into the optimizer method is the learning rate of the model. The learning rate specifies how quickly a model is going to update its weights when given new data.

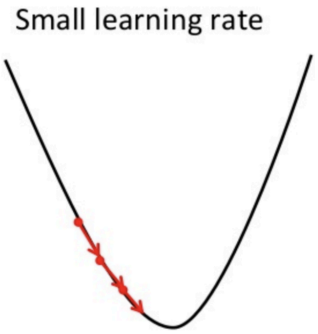
c. **Loss Function:** While going through iterations of data (aka epochs), the model also works on reducing its loss. There are many ways in which it can calculate its loss for minimizing, one of which is Mean Squared Error (MSE). Some others include Root Mean Squared Error (RMSE) and Mean Absolute Error (MEA).

d. **Gradient Descent** - Gradient descent is a popular optimization method that can be used for every type of neural network. It is an optimization algorithm used while training a model and it tweaks its parameters repeatedly to minimize the given loss function to its local minima. It can be thought of as the process of climbing down to the bottom of the valley (which represents minimizing the loss). The diagram below represents the gradient and the local minima.

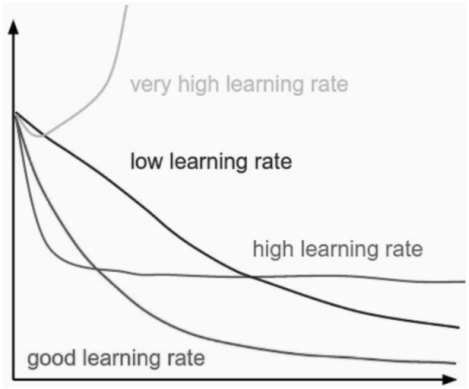




The **importance of the learning rate in gradient descent** is that it determines how big the "steps" are to reaching the minima. In order for gradient descent to reach the minima, it is imperative to find the right learning rate. If the learning rate is **too high**, then the descent may bounce off the other sides of the convex gradient. If it's **too low** it may take too much time to reach the minima.



The optimal learning rate is one which is not too high or too low. It should be able to minimize the loss faster than a slower learning rate, but not be so fast that it cannot efficiently minimize the loss:



3. **Training -** Now that the model has been created, with its multiple layers, and compiled, with the optimizer and loss function, it is time to train the model on the dataset. The below code just represents an example of how to train the model.
    a. **Sample Code:**

    #The fit method is used to train the model on the data. X_train contains the data values while y_train contains #the expected outputs.

    model.fit(X_train, y_train, epochs = 2500, batch_size = 10, class_weight = None)

    b. **Epochs:** The epoch is the **number of iterations** of the dataset that the neural network will train on. The goal of multiple epochs is to minimize the loss of the model.

    c. **Batch Size:** The batch size is the number of samples per gradient update. It specifies how many rows to train the model on at a time. A batch size of 1 leads to what is known as stochastic gradient descent.

    d. **Class Weight:** Optional dictionary mapping indices (integers) to a weight (floats) used for weighting the loss function during training. It can be useful for telling the model to "pay more attention" to certain attributes by giving them a higher weight.

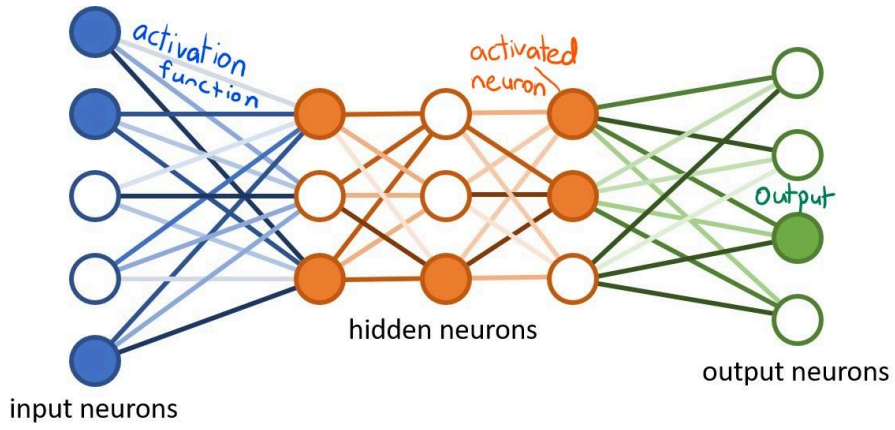
4. **Evaluation -** Finally, the model has been successfully trained on the data. It is now time to feed the model data to test on. In this process, the model does not learning from the testing data, it simply makes predictions and calculates its testing accuracy by dividing the number of correctly classified instances by the number of instances.
    a. **Sample Code:**

    #The evaluate method is use for the model to evaluate itself on the testing data.

    scores = model.evaluate(X_test, y_test)

    #Print the accuracy of the model on the testing data.

    *print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))*



Source: Medium