IQRA University

CSC 471 Artificial Intelligence

Lab# 12
Predictions Using Regression and Neural Networks

Objective:

The lab first introduces Neural Networks as a classification model. Later the students are familiarized to the concept of prediction problems within supervised learning. Regression and Neural Network models are implemented as prediction algorithms.

Name of Student: **Faraz Alam**

Roll No: **13948** **Sec.** **Saturday (12:00pm-14:00pm)**

Date of Experiment: **13/01/24**

Lab 12: Regression and Neural Networks

Linear and Logistic Regression

Linear Regression

There are two types of supervised machine learning algorithms: Regression and classification. The former predicts continuous value outputs while the latter predicts discrete outputs. For instance, predicting the price of a house in dollars is a regression problem whereas predicting whether a tumor is malignant or benign is a classification problem.

Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. There are different regression models based on the kind of relationship between dependent and independent variables, and the number of independent variables being used.

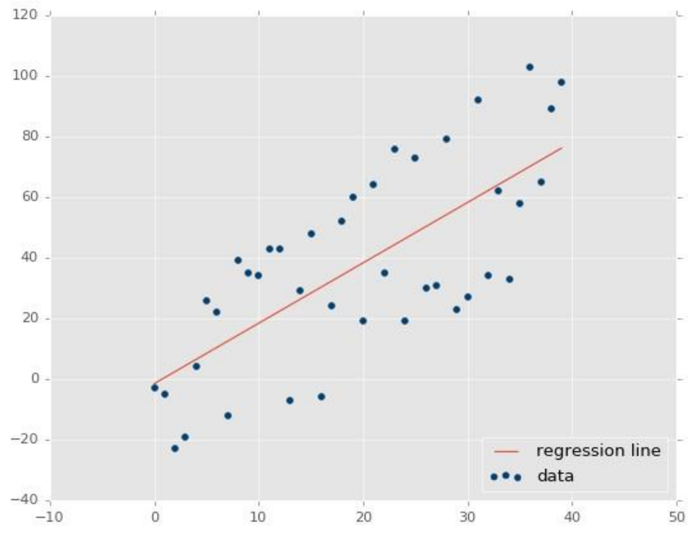
Uni-variate and Multivariate:

When there is only feature it is called **Uni-variate** Linear Regression and if there are multiple features, it is called **Multiple** Linear Regression.

Remember, a linear regression model in two dimensions is a straight line; in three dimensions it is a plane, and in more than three dimensions, a hyperplane.

Linear Regression

The term "linearity" in algebra refers to a linear relationship between two or more variables. Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y (output). Hence, the name is Linear Regression. If we plot the independent variable (x) on the xaxis and dependent variable (y) on the y-axis, linear regression gives us a straight line that best fits the data points, as shown in the figure below.



s

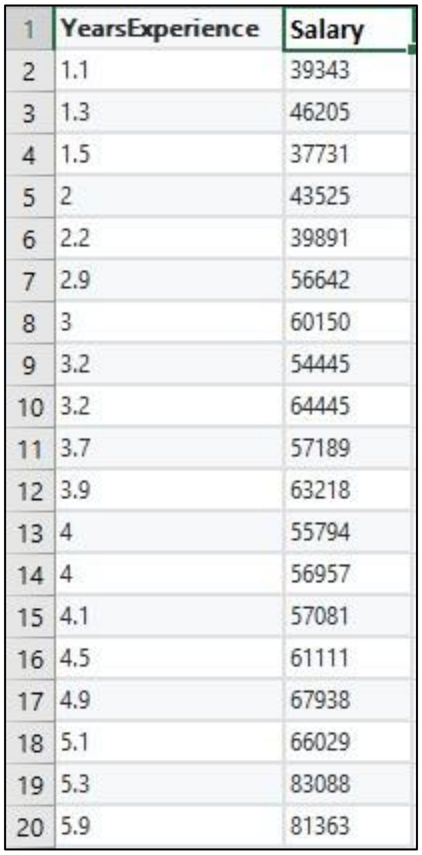
The equation of the above line is:

$$Y = mx + b$$

Where **b** is the intercept and **m** is the slope of the line. So basically, the linear regression algorithm gives us the most optimal value for the intercept and the slope (in two dimensions). The y and x variables remain the same, since they are the data features and cannot be changed. The values that we can control are the intercept(b) and slope(m). There can be multiple straight lines depending upon the values of intercept and slope. Basically, what the linear regression algorithm does is, it fits multiple lines on the data points and returns the line that results in the least error.

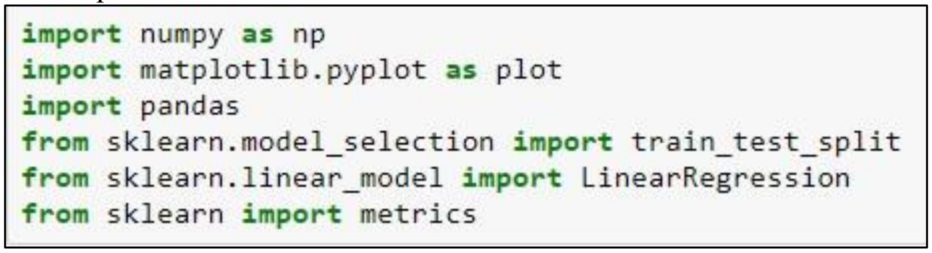
Linear regression is a very powerful technique and can be used to understand the factors that influence profitability. It can be used to forecast sales in the coming months by analyzing the sales data for previous months. It can also be used to gain various insights about customer behavior.

Let us apply linear regression on an example dataset as shown below:



1	YearsExperience	Salary
2	1.1	39343
3	1.3	46205
4	1.5	37731
5	2	43525
6	2.2	39891
7	2.9	56642
8	3	60150
9	3.2	54445
10	3.2	64445
11	3.7	57189
12	3.9	63218
13	4	55794
14	4	56957
15	4.1	57081
16	4.5	61111
17	4.9	67938
18	5.1	66029
19	5.3	83088
20	5.9	81363

Importing all the required libraries:



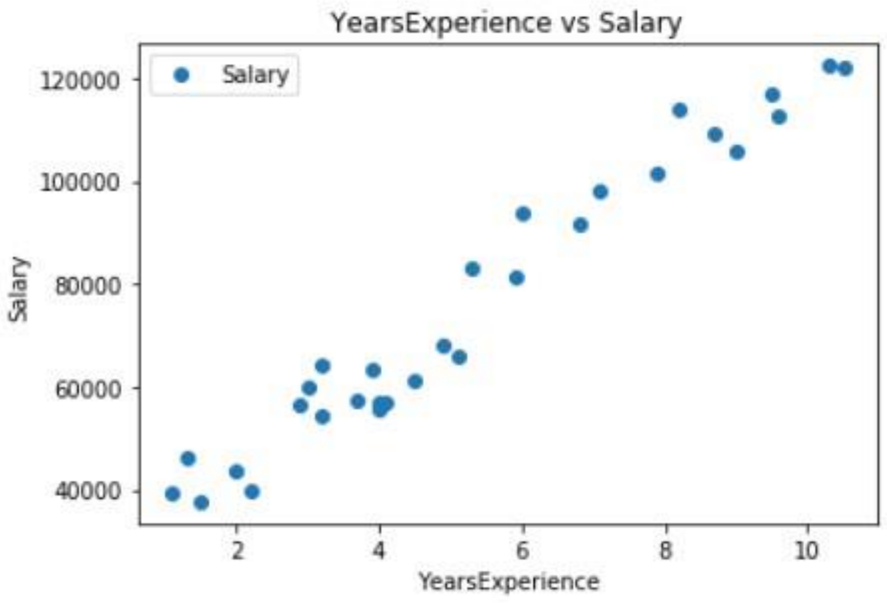
```
import numpy as np
import matplotlib.pyplot as plot
import pandas
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

The next step is to import the data and divide it into "attributes" and "labels". Attributes are the independent variables while labels are dependent variables whose values are to be predicted. In our dataset, we only have two columns. We want to predict the Salary depending upon the Year of experience recorded. Therefore, our attribute set will consist of the "YearExperience" column which is stored in the x variable, and the label will be the "Salary" column which is stored in y variable. The following command imports the dataset using pandas:

```
# Import the dataset
dataset = pandas.read_excel('SalaryAge.xlsx')
x = dataset['YearsExperience'].values.reshape(-1,1)
y = dataset['Salary'].values.reshape(-1,1)
```

let’s plot our data points on a 2-D graph to look at our dataset and see if we can manually find any relationship between the data using the below script:

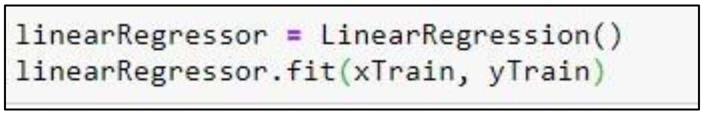
```
dataset.plot(x='YearsExperience', y='Salary', style='o')
plot.title('YearsExperience vs Salary')
plot.xlabel('YearsExperience')
plot.ylabel('Salary')
plot.show()
```



Next, we split 80% of the data to the training set while 20% of the data to test set using below code. The test_size variable is where we actually specify the proportion of the test set.

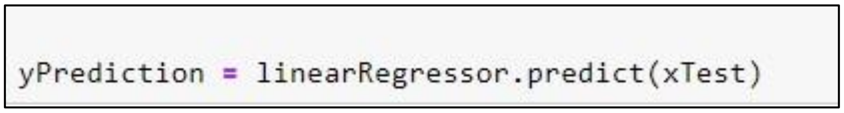
```
# Split the dataset into the training set and test set
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

After splitting the data into training and testing sets, finally, the time is to train our algorithm. For that, we need to import LinearRegression class, instantiate it, and call the fit() method along with our training data.



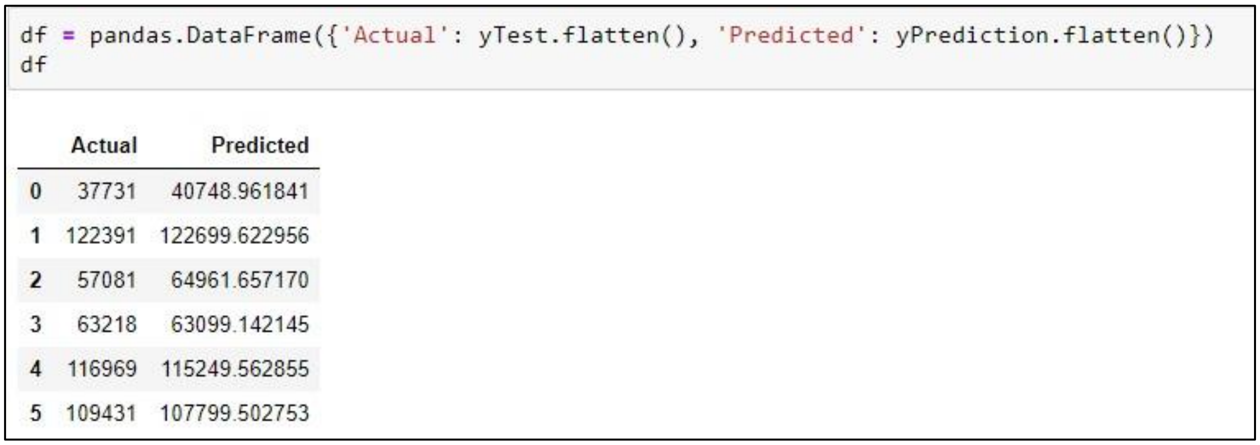
```
linearRegressor = LinearRegression()
linearRegressor.fit(xTrain, yTrain)
```

Now that we have trained our algorithm, it’s time to make some predictions. To do so, we will use our test data and see how accurately our algorithm predicts the percentage score. To make predictions on the test data, execute the following script:



```
yPrediction = linearRegressor.predict(xTest)
```

Now compare the actual output values for X_test with the predicted values, execute the following script:

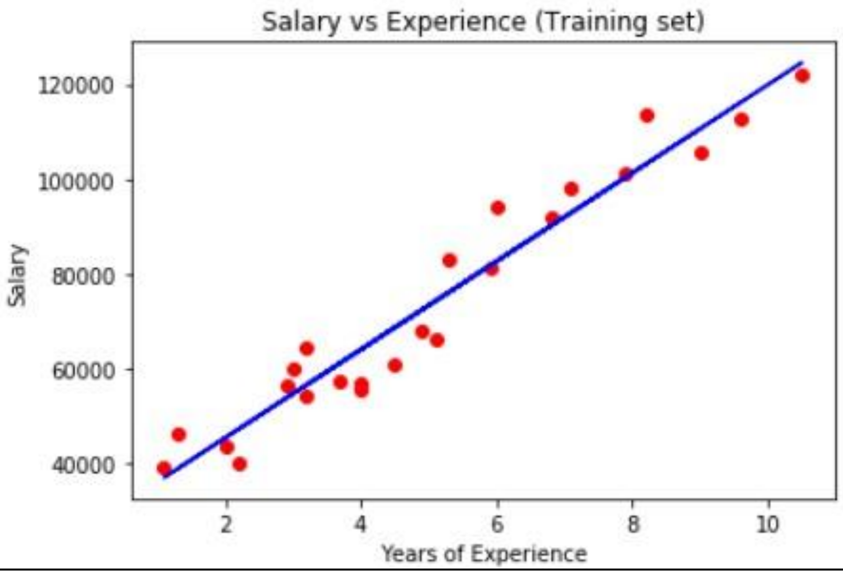


```
df = pandas.DataFrame({'Actual': yTest.flatten(), 'Predicted': yPrediction.flatten()})
df
```

```
   Actual      Predicted
0   37731   40748.961841
1  122391  122699.622956
2   57081   64961.657170
3   63218   63099.142145
4  116969  115249.562855
5  109431  107799.502753
```

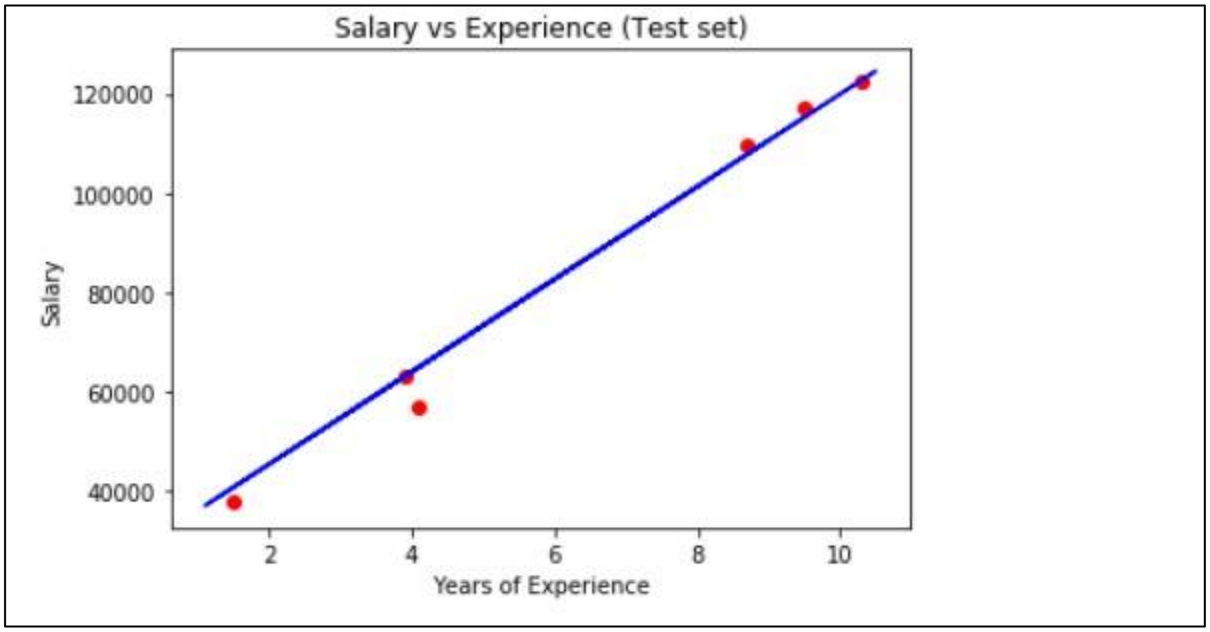
Let's plot our regression line with training data:

```python
plot.scatter(xTrain, yTrain, color = 'red')
plot.plot(xTrain, linearRegressor.predict(xTrain), color = 'blue')
plot.title('Salary vs Experience (Training set)')
plot.xlabel('Years of Experience')
plot.ylabel('Salary')
plot.show()
```



Now plot it with test data:

```python
plot.scatter(xTest,yTest,color='red')
plot.plot(xTest, linearRegression.predict(xTest), color='blue')
plot.title("salary vs Exp (Training set)")
plot.xlabel("years of exp")
plot.ylabel("salary")
plot.show()
```



The final step is to evaluate the performance of the algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For regression algorithms, three evaluation metrics are commonly used:

1. **Mean Absolute Error** (MAE) is the mean of the absolute value of the errors. It is calculated as:

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n} |y_j - y_j|$$

Mean Absolute Error

2. **Mean Squared Error** (MSE) is the mean of the squared errors and is calculated as:

$$MSE = \frac{1}{N}\sum_{i}^{n}(Y_i - y_i)^2$$

Mean Squared Error

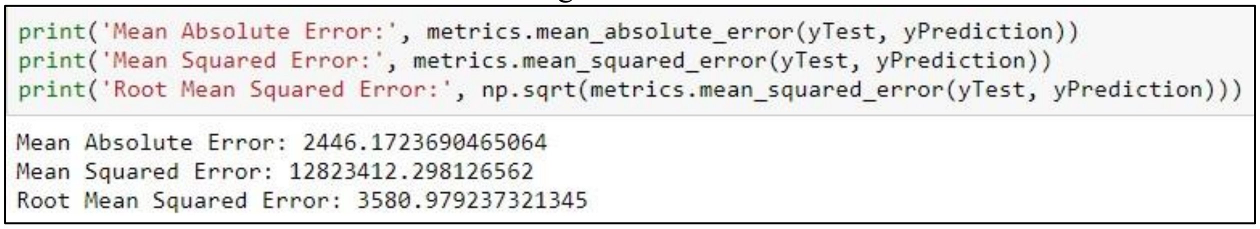
3. **Root Mean Squared Error** (RMSE) is the square root of the mean of the squared errors:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Root Mean Squared Error

The Scikit-Learn library comes with pre-built functions that can be used to find out these values for us.

Let us find the values for these metrics using our test data.



```
print('Mean Absolute Error:', metrics.mean_absolute_error(yTest, yPrediction))
print('Mean Squared Error:', metrics.mean_squared_error(yTest, yPrediction))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(yTest, yPrediction)))
```

```
Mean Absolute Error: 2446.1723690465064
Mean Squared Error: 12823412.298126562
Root Mean Squared Error: 3580.979237321345
```

The higher values of the error shows that the algorithm is not very accurate but can still make predictions.

Neural Network

Neural networks (NN), also called artificial neural networks (ANN) are subsets of learning algorithms within the machine learning field that are loosely based on the concept of biological neural networks. Neural Networks can be defined as mathematical functions that map a given input to a desired output.

Now, how do we model artificial neurons?

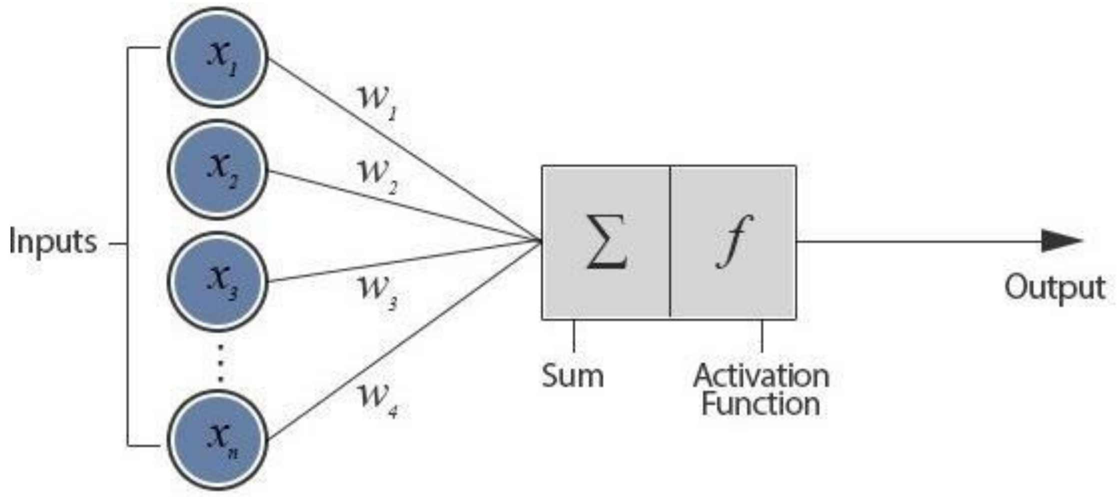


Figure 1 Model of Artificial Neural Network

Figure 1 depicts a neuron connected with n other neurons and thus receives n inputs (x_1, x_2, ….. x_n). This configuration is called a Perceptron. The inputs (x_1, x_2, …. X_n) and weights (w_1, w_2, …. W_n) are real numbers and can be positive or negative. The perceptron consists of weights, summation processor and an activation function. It also contains a threshold processor (known as bias) but we will talk about that later!

All the inputs are individually weighted, added together and passed into the activation function. There are many different types of activation function but one of the simplest would be step function. A step function will typically output a 1 if the input is higher than a certain threshold, otherwise it's output will be 0.

There are other activation functions too such as sigmoid, etc which are used in practice.

An example would be,

Input 1 (x1) = 0.6

Input 2 (x2) = 1.0

Weight 1 (w1) = 0.5

Weight 2 (w2) = 0.8

Threshold = 1.0

Weighing the inputs and adding them together gives,

$x_1w_1 + x_2w_2 = (0.6 \times 0.5) + (1 \times 0.8) = 1.1$ **Training**

in perceptrons!

Try teaching a child to recognize a bus? You show her examples, telling her, "This is a bus. That is not a bus," until the child learns the concept of what a bus is. Furthermore, if the child sees new objects that she hasn't seen before, we could expect her to recognize correctly whether the new object is a bus or not. This is exactly the idea behind the perceptron.

Similarly, input vectors from a training set are presented to the perceptron one after the other and weights are modified according to the following equation,

For all inputs i,

W(i) = W(i) + a*(T-A)*P(i), where a is the learning rate

Actually the equation is W(i) = W(i) + a*g'(sum of all inputs)*(T-A)*P(i), where g' is the derivative of the activation function. Since it is problematic to deal with the derivative of step function, we drop that out of the equation here.

Here, W is the weight vector. P is the input vector. T is the correct output that the perceptron should have known and A is the output given by the perceptron. When an entire pass through all of the input training vectors is completed without an error, the perceptron has learnt!

At this time, is an input vector P (already in the training set) is given to the perceptron, it will output the correct value. If P is not in the training set, the network will respond with an output similar to other training vectors close to P.

What is a Perceptron doing?

The perceptron is adding all the inputs and separating them into 2 categories, those that cause it to fire and those that don't. That is, it is drawing the line:

$\mathbf{x_1w_1 + x_2w_2 = t}$, where t is the threshold

and looking at where the input point lies. Points on one side of the line fall into 1 category, points on the other side fall into the other category. And because the weights and thresholds can be anything, this is just any line across the 2 dimensional input space.

Limitation of Perceptrons

Not every set of inputs can be divided by a line like this. Those that can be are called linearly separable. If the vectors are not linearly separable, learning will never reach a point where all vectors are classified properly.

What is a multilayer neural network?

Here, the total input is higher than the threshold and thus the neuron fires.

What are multi-layered neural networks?

Figure 02 shows the architecture of a 2-layer Neural Network (note that the input layer is typically excluded when counting the number of layers in a Neural Network)

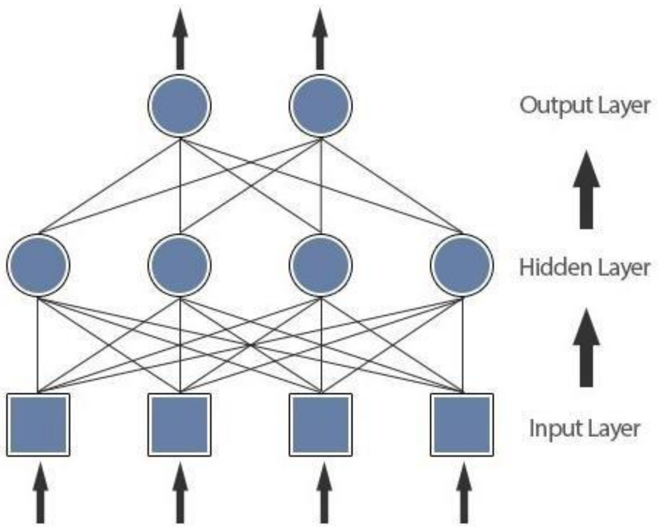


Figure 2 Two layer Neural Network

Each input from the input layer is fed up to each node in the hidden layer, and from there to each node on the output layer. We should note that there can be any number of nodes per layer and there are usually multiple hidden layers to pass through before ultimately reaching the output layer.

Now we will consider a multilayer Neural Network(NN), with respective python code. For this we will consider a toy problem, based on table 01. We will design an NN with Back-propagation.
Back-propagation is the essence of neural net training. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration). Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization. Other than this, there are other concepts that are required to know how a multi layer NN work, which are also discussed in this manual.

Student Exercise

Task 1

Thoroughly go through and attempt the notebooks provided with this lab.

Task 2

Apply Linear Regression on the dataset provided with the manual (weather.xlsx) and predict the minimum temperature based on the maximum temperature. Elaborate the results.

Task3

Apply a Neural Network on the dataset provided with the manual (weather.xlsx) and predict the minimum temperature based on the maximum temperature. Elaborate the results.