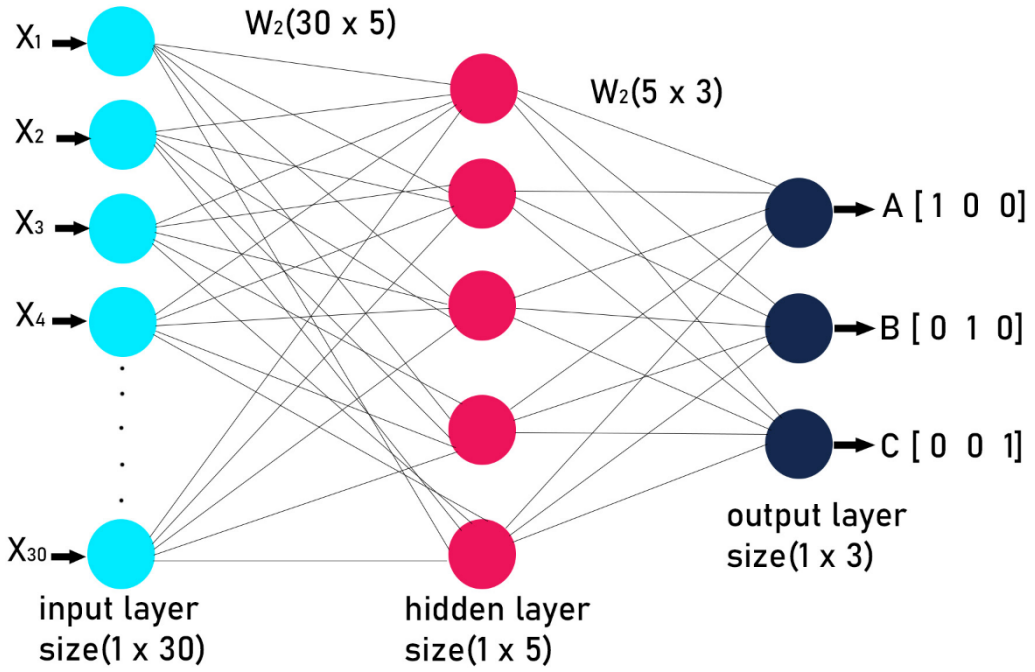
Week 3: Lab 3 Machine Learning Part II

In this part, we will practice apply Artificial Neural Network and K-means on simple data by filling in the given code template.

Task 1. Artificial Neural Network:

In this problem, we will develop a very simple and easy neural network for the classification of three only three alphabets A, B, or C. The neural network consists of three layers including input layer, hidden layer, and output layer as shown in figure below. We will write a variety of function to achieve this task based on mean square root error loss function.



Step 1: Creating a dataset

```
# A
a =[0, 0, 1, 1, 0, 0,
    0, 1, 0, 0, 1, 0,
    1, 1, 1, 1, 1, 1,
    1, 0, 0, 0, 0, 1,
    1, 0, 0, 0, 0, 1]

# B
b =[0, 1, 1, 1, 1, 0,
    0, 1, 0, 0, 1, 0,
    0, 1, 1, 1, 1, 0,
    0, 1, 0, 0, 1, 0,
    0, 1, 1, 1, 1, 0]

# C
c =[0, 1, 1, 1, 1, 0,
```

```
   0, 1, 0, 0, 0, 0,
   0, 1, 0, 0, 0, 0,
   0, 1, 0, 0, 0, 0,
   0, 1, 1, 1, 1, 0]

# Creating labels
y =[[1, 0, 0],
    [0, 1, 0],
    [0, 0, 1]]
```

Step 2: Dataset visualization

```
import numpy as np
import matplotlib.pyplot as plt
# visualizing the data, ploting A.
plt.imshow(np.array(a).reshape(5, 6))
plt.show()
```

Step 3: Pre-processing on dataset

```
x =[np.array(a).reshape(1, 30), np.array(b).reshape(1, 30),
np.array(c).reshape(1, 30)]
# Labels are also converted into NumPy array
y = np.array(y)
print(x, "\n\n", y)
```

Step 4: Defining ANN architecture

1st layer: Input layer(1, 30)
2nd layer: Hidden layer (1, 5)
3rd layer: Output layer(3, 3)

Step 5: Writing a Python Custom functions to Train and Test ANN

```
def sigmoid(x):
    return(1/(1 + np.exp(-x)))
# Creating the Feed forward neural network
# 1 Input layer(1, 30)
# 1 hidden layer (1, 5)
# 1 output layer(3, 3)


def f_forward(x, w1, w2):
    # hidden
    z1 = x.dot(w1)# input from layer 1
    a1 = sigmoid(z1)# out put of layer 2

    # Output layer
    z2 = a1.dot(w2)# input of out layer
    a2 = sigmoid(z2)# output of out layer
    return(a2)

# initializing the weights randomly
def generate_wt(x, y):
    l =[]
    for i in range(x * y):
        l.append(np.random.randn())
    return(np.array(l).reshape(x, y))
```

```
# for loss we will be using mean square error(MSE)
def loss(out, Y):
    s =(np.square(out-Y))
    s = np.sum(s)/len(Y)
    return(s)


# Back propagation of error
def back_prop(x, y, w1, w2, alpha):
    # hiden layer
    z1 = x.dot(w1)# input from layer 1
    a1 = sigmoid(z1)# output of layer 2
    # Output layer
    z2 = a1.dot(w2)# input of out layer
    a2 = sigmoid(z2)# output of out layer
    # error in output layer
    d2 =(a2-y)
    d1 = np.multiply((w2.dot((d2.transpose()))).transpose(),
                                    (np.multiply(a1, 1-a1)))

    # Gradient for w1 and w2
    w1_adj = x.transpose().dot(d1)
    w2_adj = a1.transpose().dot(d2)

    # Updating parameters
    w1 = w1- (alpha*(w1_adj))
    w2 = w2- (alpha*(w2_adj))
    return(w1, w2)


 def train(x, Y, w1, w2, alpha = 0.01, epoch = 10):
    acc =[]
    losss =[]

    for j in range(epoch):
        l =[]
        for i in range(len(x)):
            out = f_forward(x[i], w1, w2)
            l.append((loss(out, Y[i])))
            w1, w2 = back_prop(x[i], Y[i], w1, w2, alpha)
        print("epochs:", j + 1, "======== acc:", (1- (sum(l)/len(x)))*100)
        acc.append((1- (sum(l)/len(x)))*100)
        losss.append(sum(l)/len(x))

    return(acc, losss, w1, w2)


def predict(x, w1, w2):
    Out = f_forward(x, w1, w2)
    maxm = 0
    k = 0
```

```
for i in range(len(Out[0])):
    if(maxm<Out[0][i]):
        maxm = Out[0][i]
        k = i

if(k == 0):
    print("Image is of letter A.")
elif(k == 1):
    print("Image is of letter B.")
else:
    print("Image is of letter C.")
plt.imshow(x.reshape(5, 6))
plt.show()
```

Step 6: Train ANN for Alphabet Classification
Please write a python code here to first initialize the NN weights and then train.

Step 7: ANN Training visualization

```
# ploting accuraccy
plt1.plot(acc)
plt1.ylabel('Accuracy')
plt1.xlabel("Epochs:")
plt1.show()
# plotting Loss
plt1.plot(losss)
plt1.ylabel('Loss')
plt1.xlabel("Epochs:")
plt1.show()
print(w1, "\n", w2)
```

Step 8: Inference Phase
Write a python code here to test the testing dataset for A, B, and C

Task 2. K-means cluster:
In this lab, we will implement k-means clustering algorithm. K-means is a clustering algorithm typically used for unsupervised machine learning task. In unsupervised learning, the learner is provided data with no labels and its goal is to learn some useful representation from this data. K-means uses K centroids to represent clusters. A point in the data belongs to a particular cluster if it is closer to that cluster's centroid than any other centroid.

K-Means finds the best centroids by alternating between two steps. The first step assigns data points to clusters based on the current centroids. The second step updates the centroids based on the current assignment of data points to clusters.

You are required to implement the following functions without using any pre-built libraries. The following exercises require implementation in code skeleton provided in lab3.py.

T2-Part A: Understanding K-Means Clustering Algorithm

In this exercise, you will be manually running k-means algorithm on a toy dataset. Consider that you are given a 8 data samples as: R1 (185, 72), R2 (170,56), R3 (168,60), R4 (179,68), R5 (182,72), R6 (188,77), R7 (180,71), and R8 (180,70) and two initial clusters as C1 (185,72) and C2 (170,56).

Q 1: Assign each of data sample (e.g., data rows) to one of the two clusters.
Q 2: Estimate the cluster centroids after each row assignment.

T2-Part B: Implementing K-Means Clustering Algorithm

Write a code for k-means clustering algorithm in the provided file. Initialize random points, and then write three major functions for KMeans clustering including *initializing random centroids*, *assign each data sample to initialized centroids*, and then *update centroids*.

Step 1: Initialize Random Points

```
import numpy as np
points = np.vstack(((np.random.randn(150, 2) * 0.75 + np.array([1, 0])),
                    (np.random.randn(50, 2) * 0.25 + np.array([-0.5, 0.5])),
                    (np.random.randn(50, 2) * 0.5 + np.array([-0.5, -0.5]))))
```

Step 2: Initializing random centroids

Write your code in the provided file.

Step 3: Assign each data sample to initialized centroids

Write your code in the provided file.

Step 4: Update centroids

Write your code in the provided file.

Step 5: show the points and the learned centroids after 50 iteration.

```
plt.scatter(points[:, 0], points[:, 1])
plt.scatter(c[:, 0], c[:, 1], marker='^')
plt.show()
```