HW1b Numpy (Total Points - 10)

You have to submit two files for this part of the HW

1. FirstNameLastName_Hw1b.ipynb (colab notebook)
2. FirstNameLastName_Hw1b.pdf pdf file**

```
#import the package
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
np.__version__
```

```
'1.23.5'
```

Linear Regression

1. Linear regression is a linear approach to modelling the relationship between a **dependent variable** and one or more **independent variables**.
2. Representation of Linear regression

   $y \approx \theta_0 + \theta_1 x_1$

   where,

   y is a dependent variable

   x_1, x_2 are independent variables

   $\theta_0, \theta_1 x_1, \theta_2$ are parameters
3. We are given y, x_1, x_2. We need to estimate the parameters θ_0, θ_1 .

Gradient descent

1. Gradient descent is the backbone of the machine learning.
2. It is an optimization process which is used to train the models in the machine learning.
3. Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).
4. A gradient simply measures the change in all weights with regard to the change in error.
5. Let's check the following explaination of the gradient descent and how we use it as follows:

Create a dataset

Task1: Generate independent variable (X) - 1 Point

- Generate X by darwing samples from unifrom distribution
- The shape of X should be (100, 2). This means we will have 100 observations and 2 variables. Column 1 represents $\mathbf{x_1}$ and column 2 represents $\mathbf{x_2}$
- All values in matrix $\mathbf{X}$ should lie between 0 and 5 (Hint if we use random.rand(), it will give you values from uniform distribution. However the values will lie between 0 and 1

```
np.random.seed(seed=123) # pleasse do not change this

# genrate X
```

```
X = np.random.uniform(0,5,(100,2))

# Print first five values of X
print(X[:5])
print(X.shape)
```

```
[[3.48234593 1.43069667]
 [1.13425727 2.75657385]
 [3.59734485 2.1155323 ]
 [4.90382099 3.42414869]
 [2.40465951 1.96058759]]
(100, 2)
```

Task2: Generate Dependent Varianble (Y) - 1 Points

Assume : $\theta_0 = 2$, $\theta_1 = 5$, $\theta_2 = 3$
Generate y using following : $\mathbf{y} = \theta_0 + \theta_1 \mathbf{x_1} + \theta_2 \mathbf{x_2} + noise$

where noise is a random normal array of shape (100, 1)

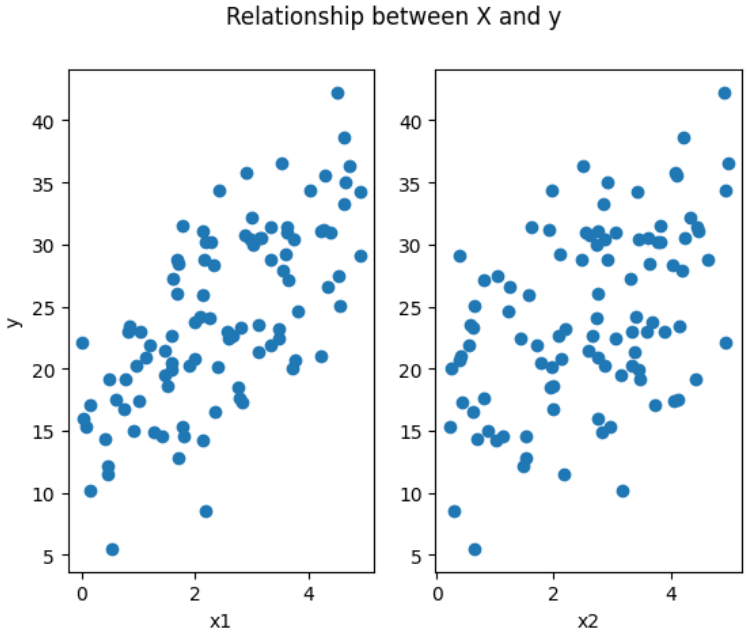
```
np.random.seed(seed=123) # pleasse do not change this
theta0 = 2
theta1 = 5
theta2 = 3
# use np.random.randn to get 100 values from nornal distribution. The array should have the shape (100, 1)
noise = np.random.normal(2,3,(100,1))

# use array slicing to get x1 and x2 from X; x1 is the first column and x2 is the second column of  X

# check the shape of x1 and x2. The shape should be (100, 1) i.e. it should have two dimensions
# If teh shape is not (100, 1) then reshape x1 and x2 to have shape (100, 1)

x1 = X[:,0].reshape(100,1)
x2 = X[:,1].reshape(100,1)


# use formulae y = theta0 + theta1 * x1 + theta2 * x2 + noise
y = theta0+(theta1*x1)+(theta2*x2)+noise

# Print first five values of y
print(y[:5])
print(y.shape)
```

```
[[22.44692785]
 [20.93304422]
 [29.18225664]
 [34.2726669 ]
 [20.16925955]]
(100, 1)
```

Visualize the Data

- Here we are using matplotlib's package pyplot module to visualize the data points. We will further discuss about this package in our course.
- The main aim for visualizing a data is to understand the data to check the outliers, spread, correlation of the data etc. which will be discussed later on.

```
# Let's plot the values of X and y
# Let's use the scatter plot graph
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Relationship between X and y')
ax1.scatter(x1, y)
ax2.scatter(x2, y)
ax1.set_xlabel('x1')
ax1.set_ylabel('y')
ax2.set_xlabel('x2')
#It is used to show the graph
plt.show();
```



Predicted Value of Dependent Variable

We can estimate the predicted value of y for a particular instance (observation) using following equation:

$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

In this equation:
$\hat{y}$ is the predicted value of y.
n is the number of features (independent variables).
x_i is the ith feature value.
θ_j is the j^{th} model parameter.

We can write this in vectorized form as follows

$$\hat{y} = h_\theta(\mathbf{x}) = \theta^T x = \begin{bmatrix} \theta_0 \theta_1 \cdots \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ . \\ . \\ . \\ x_n \end{bmatrix} = \theta \cdot \mathbf{x}$$

Here
θ is vector of model parameters
$\mathbf{x}$ is the instance's feature vector, containing x_0 to x_n, with x_0 always equal to 1.
$\theta \cdot \mathbf{x}$ is the dot product of the vectors θ and $\mathbf{x}$ which is equal to $\theta_0 x_0 + \theta_1 x_1 + \cdots \theta_n x_n$

The training examples are stored in matrix X row-wise

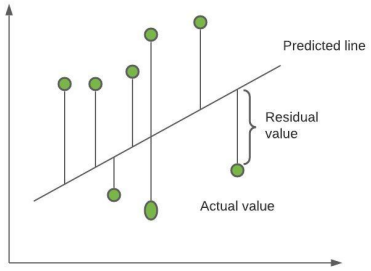
$$X = \begin{bmatrix} x_0^{(1)} x_1^{(1)} \cdots x_n^{(1)} \\ x_0^{(2)} x_1^{(2)} \cdots x_n^{(2)} \\ . \\ . \\ . \\ x_0^{(m)} x_1^{(m)} \cdots x_n^{(m)} \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ . \\ . \\ . \\ \theta_n \end{bmatrix}.$$

We can calculale the predicted values of y for all m observations using follwoing:

$h_\theta(X) = X\theta$ where $X\theta$ is matrix multiplication of X and θ

Task3: Cost Function - 2 Points

- The cost function is defined as sum of the squares of the difference between predicted values and avtual values of y.
- The equation for calculating cost function is shown below.
- The cost function for other algorithm will be different and the gradients are always derived from the cost functions.
- Our main aim is to get the predicted line ($\hat{y}$) such that the total distance from alll the points (actual y values) is minimized i.e. we want to find a line tha best fits our dataset.



Cost function Formulae

$$J(\theta) = 1/2m \sum_{i=1}^{m} (y^{\hat{i}} - y^{(i)})^2$$

here m is number of observations

- Let's define a cost function by following the mentioned steps below.
- It's fine if we don't know the concept completely. Its just for numpy practice exercise.
- We will be discussing about cost functions and gradient descents in our future lectures.

```
def calculate_cost(X, y, theta,):
  """
  In this function we will calculate the cost value by using the above formula
  J()=1/2m∑(prediction-y)^(2)
  where m = number of y values (number of observations)
  we know that,
  prediction = xθ (matrix multiplication of X and θ )

  """
  # Find the length of the y by using np.size() method and equate it to m
  # calculate 1/(2*m) and assign it to 'a'.
  # calculate Xθ (matrix multiplication of X and theta) and assign it to prediction
  # Find the difference between predictions and y and assign it to error
  # Square the error by using np.square() method and assign it to square_error
  # Use sum() method of numpy to caculate the sum of square_error. Assign it to b
  # Now multiply a, b and assign it to J.

  m = y.size
  a = 1/(2*m)
  prediction = np.dot(X,theta)
  error = prediction - y
  square_error = np.square(error)
  b = square_error.sum()
  J = a*b

  return J
```

Task4: Feature Scaling - Scaling at absolute maximum - 2 Points

1. Feature sacling is one of the important step in the many machine learning models.
2. We actually compress our input variable into smaller and similar magnitude for faster calculations.
3. Scaling is a technique to seggregate the date between 0 to 1.

$$X_{scaled} = \frac{X}{max(abs(X))}$$

```
# Use the above formulae to calculate X_scaled
# Hint (Since X is a matrix, we will need to be careful in specifying axis for max opeartion)

X_scaled =  (X/(np.max(np.abs(X),axis=0)))
# print first five values of X_Scaled
print(X_scaled[:5])
```

```
[[0.70813816 0.28747365]
 [0.23065223 0.55388564]
 [0.7315233  0.42507947]
 [0.99719639 0.68802321]
 [0.48898966 0.39394603]]
```

Task5: Initialize the parameter- 2 Points

Since, we have created the cost function. Now we will be assigning some values to the parameter and call the cost function.

Double-click (or enter) to edit

```
# initializing parameter
# check the shape of the X_scaled (X Scaled)
print(f'Shape of X_scaled is {X_scaled.shape}')
# if the shape of the X_scaled is (100,2) then we won't reshape
# If the shape of the X_scaled is not (100,2) then Reshape the X_scaled variable as (100, 2)
# We take an additional value in X i.e a column of ones
# Create a numpy array of ones with shape (100, 1) and assign it to X_ones
X_ones = np.random.uniform(0,1,(100,1))
print(f'The shape of X_ones is {X_ones.shape}')
# Use hstack to combine the X_ones and X_scaled and assign it to X_b
X_b = np.hstack((X_scaled,X_ones))
# Create a numpy array of zeros with shape (3, 1) and assign it to theta
theta=np.zeros((3,1),dtype=float)
# this represents the initial value of theta1, theta2 and theta3
print(f'The first 5 values of X_b are {X_b[:5]}')
# print first five rows of X_b
print(f'The values of theta are {theta}')
# print theta
print(f'The shape of X_b is {X_b.shape}')
# print shape of X_b (shape should be (100, 3)
print(f'The shape of theta is {theta.shape}')
# print shape of theta (shape should be (3, 1))

# your code here # Print shape of xscaled (the sjape should be (100, 2))
print(f'The shape of X_scaled is {X_scaled.shape}')


# your code here # print first five rows of X_b and
# your code here # print theta
# your code here # print shape of X_b (shape should be (100, 3)
# your code here # print shape of theta (shape should be (3, 1))
```

```
Shape of X_scaled is (100, 2)
The shape of X_ones is (100, 1)
The first 5 values of X_b are [[0.70813816 0.28747365 0.33867085]
 [0.23065223 0.55388564 0.55237008]
 [0.7315233  0.42507947 0.57855147]
 [0.99719639 0.68802321 0.52153306]
 [0.48898966 0.39394603 0.00268806]]
The values of theta are [[0.]
```

```
 [0.]
 [0.]]
The shape of X_b is (100, 3)
The shape of theta is (3, 1)
The shape of X_scaled is (100, 2)
```

Now our aim is to reduce this initial cost value further, so that we can achieve the optimal linear fit for our data. This initial cost value corresponds to initial zero value for θ_0, θ_1 and θ_2. We need to find values of θ_0, θ_1 and θ_2 for which the cost function is minimized.

Task6: Gradient descent - 2 Points

Gradient descend is a one such algorithm used to find the optimal parameter 'theta' using the given parameters ,

x — Input values

y — output values

Initial_theta — in most cases consdiered as NULL theta

alpha — alpha is the learning rate.

iteration — setting how many iteration it should take

Understanding "Gradinet Desecnd" may require bit of calculus , but it is not necessary for this exercise. We will provide you exact formulae to implement gradient descent using numpy.

Further explaination of the gradient descent is given below :-

Gradient descent for the cost function

First we calculate the partial derivative of parameter (θ_j) with respect to cost (J). This is called the gradient. For our cost finction the partial derivative (gradient is as follows):

$$\frac{\partial J(\theta)}{\partial \theta_j} = 1/m \sum_{i=1}^{m} (y^{\hat{}i)} - y^{(i)}) \cdot x_j^{(i)}$$

Gradient Descent Algorithm: Repeat until convergence (usually we will use finite number of iterations):

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial J(\theta)}{\partial \theta_j}$$

So basically we start with some initial values of θ_j and use the above equation to update the value of θ_j until convergence.

In our case we will repeatedly (= number of interations) update the values of θ_0 , θ_1 , and θ_2 using the following equations:

The algorithm starts with some "initial guess" for θ, and then repeatedly changes θ to make J(θ) smaller, until we converge to a value of θ that minimizes J(θ).

$$\theta_0 := \theta_0 - \frac{\alpha}{m}(\sum_{i=1}^{m} (y^{\hat{}i)} - y^{(i)}).x_0^{(i)})$$

$$\theta_1 := \theta_1 - \frac{\alpha}{m}(\sum_{i=1}^{m} (y^{\hat{}i)} - y^{(i)}).x_1^{(i)})$$

$$\theta_2 := \theta_2 - \alpha.(1/m.\sum_{i=1}^{m} (y\hat{i}) - y^{(i)}).x_2^{(i)})$$

We can write these equations using following vector form:

$$\theta := \theta - \frac{\alpha}{m} X^T(X\theta - y)$$

here

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

X^T is the transpose of X

```
def gradient_descent(X, y, theta):
    '''
```

```
    We can use the above formula of gradient descent and deduce the steps as follows:
    We will initialize the alpha and iteration values as 0.2, 500 respectively.
    '''
    # Find the length of the y by using size() method and equate it to m
    # calculate Xθ (matrix multiplication of X and theta) and assign it to prediction
    # calculate prediction - y and assign it to error
    # Perfrom the matrix multiplication of X^T (transpose of X) and error and assign it to b
    # multiply b with 1/m and assign it to gradient
    # update theta using : theta = theta - alpha* gradient


    alpha = 0.2
    iterations = 500
    m = y.size


    # The for loop will execute till 500 iterations
    for iter in range(0, iterations):

          prediction = np.dot(X,theta)
          error =  prediction - y
          b =   np.dot((X.T) , error)
          gradient =  b * 1/m
          theta = theta - alpha * gradient

    return theta
```

Let's use gradient function for our data :

```
# Call the gardient descent function using (X_b, y and theta) as inputs
# make sure you use X_b as the input and not X
theta_optimized = gradient_descent(X_b,y,theta)

# Call the cost function - (X_b, y and theta_optimized) as inputs
min_cost = calculate_cost(X_b,y,theta_optimized)
# print the optimal values of theta (theta_optimized)
print(f" The optimal values for theta is : \n {theta_optimized}")

# Print the minimum value of teh ocst
print(f"The minimum cost by using gradient descent is \n {min_cost}")
```

```
 The optimal values for theta is :
 [[24.71825108]
 [20.33246046]
 [ 1.37534575]]
The minimum cost by using gradient descent is
 6.566255187800536
```

Optional Bonus question:

We used gardient descent to find the parameters in linear regression. Gradient descent is a genric algorithn which can be applied to other cost finctins as well. The solution to linear regression can also be obtained directly using following normal equation:

$$\theta = (X^T X)^{-1} X^T y$$

Here X^T is transpose of X, and $(X^T X)^{-1}$ is inverse of matrix $(X^T X)$

```
# Let us check the solutions using normal equation:
# When implementing the above qeuation use X_b and not X

theta_normal_equation = np.dot(np.linalg.inv(np.dot((X.T),X)),np.dot((X.T),y))
# print theta_normal_equation
print(theta_normal_equation)
```

```
[[5.1476528 ]
 [4.21578793]]
```

✓ 0s completed at 2:36 PM