Task 03:

In this task, you have to implement the Backpropagation method using Pytorch. This is particularly useful when the hypothesis function contains several weights.

Backpropagation: Algorithm to caculate gradient for all the weights in the network with several weights.

- It uses the `Chain Rule` to calcuate the gradient for multiple nodes at the same time.
- In pytorch this is implemented using a `variable` data type and `loss.backward()` method to get the gradients

In [1]:

```
# import the necessary libraries
import torch
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Preliminaries - Pytorch Basics

In [2]:

```
# creating a tensor

# zero tensor
zeros = torch.zeros(5)
print(zeros)
# ones
ones = torch.ones(5)
print(ones)
# random normal
random = torch.randn(5)
print(random)


# creating tensors from list and/or numpy arrays
my_list = [0.0, 1.0, 2.0, 3.0, 4.0]
to_tensor = torch.Tensor(my_list)
print("The size of the to_tensor: ", to_tensor.size())

my_array = np.array(my_list) # or
to_tensor = torch.tensor(my_array)
to_tensor = torch.from_numpy(my_array)
print("The size of the to_tensor: ", to_tensor.size())
```

```
tensor([0., 0., 0., 0., 0.])
tensor([1., 1., 1., 1., 1.])
tensor([ 1.7497,  0.8596, -0.4556, -0.7131,  0.3751])
The size of the to_tensor:  torch.Size([5])
The size of the to_tensor:  torch.Size([5])
```

In [3]:

```
# multi dimenstional tensors

# 2D
two_dim = torch.randn((3, 3))
print(two_dim)
# 3D
three_dim = torch.randn((3, 3, 3))
print(three_dim)
```

```
tensor([[-1.0055, -1.2573, -2.0889],
        [ 1.9444,  1.3892,  0.5272],
        [ 0.0321, -0.6079,  1.4815]])
tensor([[[-1.0585e+00,  7.2191e-01, -2.1606e+00],
         [ 1.2057e+00, -3.7276e-01,  5.7974e-01],
         [-3.1539e-01,  1.0682e+00, -2.7281e-01]],

        [[-7.5735e-01, -1.0275e+00,  4.0275e-01],
         [-4.6469e-01,  6.1583e-01,  7.5642e-01],
         [-3.1483e-01, -1.1823e+00,  2.2491e-01]],

        [[ 2.5763e-01, -1.2086e+00,  1.2257e+00],
         [-1.1847e+00, -1.3737e+00, -2.1316e-03],
         [-1.9128e-02, -1.2176e+00, -6.6626e-02]]])
```

In [4]:

```
# tensor shapes and axes

print(zeros.shape)
print(two_dim.shape)
print(three_dim.shape)

# zeroth axis - rows
print(two_dim[:, 0])
# first axis - columns
print(two_dim[0, :])
```

```
torch.Size([5])
torch.Size([3, 3])
torch.Size([3, 3, 3])
tensor([-1.0055,  1.9444,  0.0321])
tensor([-1.0055, -1.2573, -2.0889])
```

In [5]:

```
print(two_dim[:, 0:2])
print(two_dim[0:2, :])
```

```
tensor([[-1.0055, -1.2573],
        [ 1.9444,  1.3892],
        [ 0.0321, -0.6079]])
tensor([[-1.0055, -1.2573, -2.0889],
        [ 1.9444,  1.3892,  0.5272]])
```

In [6]:

```
rand_tensor = torch.randn(2,3)
print("Tensor Shape : " , rand_tensor.shape, rand_tensor)
resized_tensor = rand_tensor.reshape(3,2)
print("Resized Tensor Shape : " , resized_tensor.shape,resized_tensor) # or
resized_tensor = rand_tensor.reshape(3,-1)
print("Resized Tensor Shape : " , resized_tensor.shape,resized_tensor)
flattened_tensor = rand_tensor.reshape(-1)
print("Flattened Tensor Shape : " , flattened_tensor.shape, flattened_tensor)
```

```
Tensor Shape :  torch.Size([2, 3]) tensor([[ 0.3659,  0.2436, -1.094
2],
        [ 0.0098, -0.3362, -0.6905]])
Resized Tensor Shape :  torch.Size([3, 2]) tensor([[ 0.3659,  0.243
6],
        [-1.0942,  0.0098],
        [-0.3362, -0.6905]])
Resized Tensor Shape :  torch.Size([3, 2]) tensor([[ 0.3659,  0.243
6],
        [-1.0942,  0.0098],
        [-0.3362, -0.6905]])
Flattened Tensor Shape :  torch.Size([6]) tensor([ 0.3659,  0.2436,
-1.0942,  0.0098, -0.3362, -0.6905])
```

Determine the derivative of $y = 2x^3 + x$ at $x = 1$

In [7]:

```
x = torch.tensor(1.0, requires_grad = True)
y = 2 * (x ** 3) + x
y.backward()
print("Value of Y at x=1 : " , y)
print("Derivative of Y wrt x at x=1 : " , x.grad)
```

```
Value of Y at x=1 :  tensor(3., grad_fn=<AddBackward0>)
Derivative of Y wrt x at x=1 :  tensor(7.)
```

Task 03 - a

Determine the partial derivative of $y = uv + u^2$ at $u = 1$ and $v = 2$ with respect to u and v.

In [8]:

```
# YOUR CODE STARTS HERE
u = torch.tensor(1.0, requires_grad = True)
v = torch.tensor(2.0, requires_grad = True)
y = u*v + u**2
y.backward()

# YOUR CODE ends HERE
print("Value of y at u=1, v=2 : " , y)
print("Partial Derivative of y wrt u : " , u.grad)
print("Partial Derivative of y wrt v : " , v.grad)
```

```
Value of y at u=1, v=2 :  tensor(3., grad_fn=<AddBackward0>)
Partial Derivative of y wrt u :  tensor(4.)
Partial Derivative of y wrt v :  tensor(1.)
```

Hypothesis Function and Loss Function

$y = x * w + b$

$loss = (\hat{y} - y)^2$

Let us make use of a randomly-created sample dataset as follows

In [9]:

```
#sample-dataset
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]
```

Task: 03 - b

Declare pytorch tensors for weight and bias and implement the forward and loss function of our model

In [10]:

```
# Define w = 1 and b = -1 for y = wx + b
# Note that w,b are learnable paramteter
# i.e., you are going to take the derivative of the tensor(s).
# YOUR CODE STARTS HERE
w = torch.tensor([1.0], requires_grad = True)
b = torch.tensor([-1.0], requires_grad = True)
# YOUR CODE ENDS HERE

assert w.item() == 1
assert b.item() == -1
assert w.requires_grad == True
assert b.requires_grad == True
```

In [11]:

```
#forward function to calculate y_pred for a given x according to the linear mode
l defined above
def forward(x):
    #implement the forward model to compute y_pred as w*x + b
    ## YOUR CODE STARTS HERE
    y_predicted= w*x + b
    return y_predicted

    ## YOUR CODE ENDS HERE

#loss-function to compute the mean-squared error between y_pred and y_actual
def loss(y_pred, y_actual):
    #calculate the mean-squared-error between y_pred and y_actual
    ## YOUR CODE STARTS HERE
    loss_=(y_actual-y_pred)**2
    #loss_.backward()
    return loss_

    ## YOUR CODE ENDS HERE
```

Calculate y_{pred} for $x = 4$ without training the model

In [12]:

```
y_pred_without_train = forward(4)
```

Begin Training

In [13]:

```
# In this method, we learn the dataset multiple times (called epochs)
# Each time, the weight (w) gets updates using the graident decent algorithm bas
ed on weights of the previous epoch

alpha = 0.01 # Let us set learning rate as 0.01
weight_list = []
loss_list=[]

# Training loop
for epoch in range(10):
    total_loss = 0
    count = 0

    for x, y in zip(x_data, y_data):

        #implement forward pass, compute loss and gradients for the weights and
 update weights
        ## YOUR CODE STARTS HERE
        y_pred=forward(x)
        current_loss= loss(y_pred,y)
        #loss_fxn = 0.5*(y-(w*x+b))**2
        current_loss.backward()
        w.data = w.data - alpha* w.grad.item()
        #b.data = b.data - alpha* b.grad
        total_loss+=current_loss
        ## YOUR CODE ENDS HERE

        # Manually zero the gradients after updating weights
        w.grad.data.zero_()
        #b.grad.data.zero_()
        count += 1

    avg_mse = total_loss / count
    print(f"Epoch: {epoch+1} | Loss: {avg_mse.item()} | w: {w.item()}")
    weight_list.append(w)
    loss_list.append(avg_mse)
```

```
Epoch: 1 | Loss: 8.32815933227539 | w: 1.368575930595398
Epoch: 2 | Loss: 4.635132312774658 | w: 1.641068696975708
Epoch: 3 | Loss: 2.6127521991729736 | w: 1.842525839805603
Epoch: 4 | Loss: 1.5045195817947388 | w: 1.991465449333191
Epoch: 5 | Loss: 0.8966817855834961 | w: 2.1015784740448
Epoch: 6 | Loss: 0.5628984570503235 | w: 2.182986259460449
Epoch: 7 | Loss: 0.3793121576309204 | w: 2.2431719303131104
Epoch: 8 | Loss: 0.2781200110912323 | w: 2.287667751312256
Epoch: 9 | Loss: 0.22218374907970428 | w: 2.3205642700195312
Epoch: 10 | Loss: 0.19114667177200317 | w: 2.3448848724365234
```

Calculate y_{pred} for $x = 4$ after training the model

In [14]:

```
y_pred_with_train = forward(4)

print("Actual Y Value for x=4 : 8")
print("Predicted Y Value before training : " , y_pred_without_train.item())
print("Predicted Y Value after training : " , y_pred_with_train.item())
```

```
Actual Y Value for x=4 : 8
Predicted Y Value before training :  3.0
Predicted Y Value after training :  8.379539489746094
```

Task: 03 - c

Repeat **Task:03 - b** for the quadratic model defined below

Using backward propagation for quadratic model

$\hat{y} = x^2 * w_2 + x * w_1$

$loss = (\hat{y} - y)^2$

- Using Dummy values of x and y

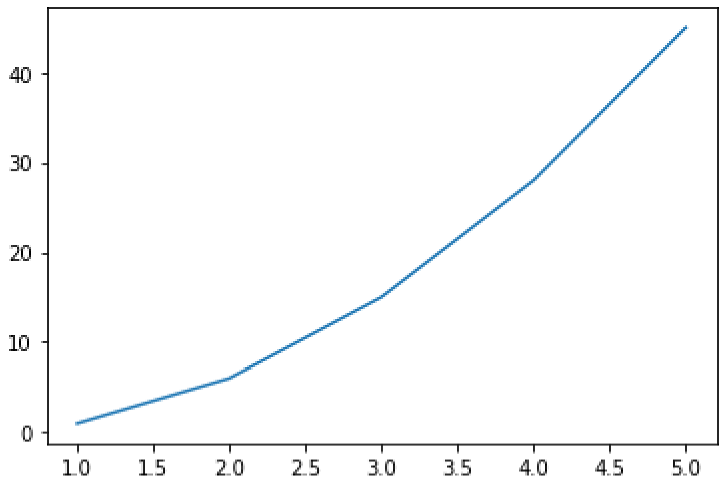
`x = 1,2,3,4,5  y = 1,6,15,28,45`

In [15]:

```
x_data = [1.0, 2.0, 3.0, 4.0, 5.0]
y_data = [1.0, 6.0, 15.0, 28, 45]
```

In [16]:

```
# Visualize the given dataset
plt.plot(x_data,y_data)
plt.show()
```



In [17]:

```
# Initialize w2 and w1 with random values
w_1 = torch.tensor([1.0], requires_grad=True)
w_2 = torch.tensor([1.0], requires_grad=True)

assert w_1.item() == 1
assert w_2.item() == 1
assert w_1.requires_grad == True
assert w_2.requires_grad == True
```

In [18]:

```
#quadratic-forward function to calculate y_pred for a given x according to the q
uadratic model defined above
def quad_forward(x):
    #implement the forward model to compute y_pred as w1*x + w2*(x^2)
    ## YOUR CODE STARTS HERE
    y_predicted= w_1*x + w_2*(x**2)
    return y_predicted
    ## YOUR CODE ENDS HERE

#loss-function to compute the mean-squared error between y_pred and y_actual
def loss(y_pred, y_actual):
    #calculate the mean-squared-error between y_pred and y_actual
    ## YOUR CODE STARTS HERE
    loss_= (y_actual-y_pred)**2
    #loss_.backward()
    return loss_
    ## YOUR CODE ENDS HERE
```

Calculate y_{pred} for $x = 6$ without training the model

In [19]:

```
y_pred_without_train = quad_forward(6)
```

Begin Training

In [20]:

```
# In this method, we learn the dataset multiple times (called epochs)
# Each time, the weight (w) gets updates using the graident decent algorithm bas
ed on weights of the previous epoch


alpha = 0.0012 # Let us set learning rate as 0.01
weight_list = []
loss_list=[]

# Training loop
for epoch in range(100):
    total_loss = 0
    count = 0

    for x, y in zip(x_data, y_data):

        #implement forward pass, compute loss and gradients for the weights and
 update weights
        ## YOUR CODE STARTS HERE
        y_pred=quad_forward(x)
        current_loss=loss(y_pred,y)
        #loss_fxn= (y-(w_1*x + w_2*(x**2)))**2
        current_loss.backward()
        w_1.data= w_1.data - alpha* w_1.grad.item()
        w_2.data= w_2.data - alpha* w_2.grad.item()
        total_loss+=current_loss
        ## YOUR CODE ENDS HERE

        # Manually zero the gradients after updating weights
        w_1.grad.data.zero_()
        w_2.grad.data.zero_()

        count += 1

    avg_mse = total_loss / count
    print(f"Epoch: {epoch+1} | Loss: {avg_mse.item()} | w: {w.item()}")
    weight_list.append(w)
    loss_list.append(avg_mse)
```

```
Epoch: 1 | Loss: 19.670841217041016 | w: 2.3448848724365234
Epoch: 2 | Loss: 6.430711269378662 | w: 2.3448848724365234
Epoch: 3 | Loss: 4.341702938079834 | w: 2.3448848724365234
Epoch: 4 | Loss: 4.463932037353516 | w: 2.3448848724365234
Epoch: 5 | Loss: 4.349801063537598 | w: 2.3448848724365234
Epoch: 6 | Loss: 4.273285865783691 | w: 2.3448848724365234
Epoch: 7 | Loss: 4.193112373352051 | w: 2.3448848724365234
Epoch: 8 | Loss: 4.115159511566162 | w: 2.3448848724365234
Epoch: 9 | Loss: 4.038552284240723 | w: 2.3448848724365234
Epoch: 10 | Loss: 3.9633827209472656 | w: 2.3448848724365234
Epoch: 11 | Loss: 3.8896148204803467 | w: 2.3448848724365234
Epoch: 12 | Loss: 3.817220687866211 | w: 2.3448848724365234
Epoch: 13 | Loss: 3.7461726665496826 | w: 2.3448848724365234
Epoch: 14 | Loss: 3.6764445304870605 | w: 2.3448848724365234
Epoch: 15 | Loss: 3.6080145835876465 | w: 2.3448848724365234
Epoch: 16 | Loss: 3.5408616065979004 | w: 2.3448848724365234
Epoch: 17 | Loss: 3.4749622344970703 | w: 2.3448848724365234
Epoch: 18 | Loss: 3.4102847576141357 | w: 2.3448848724365234
Epoch: 19 | Loss: 3.3468120098114014 | w: 2.3448848724365234
Epoch: 20 | Loss: 3.2845165729522705 | w: 2.3448848724365234
Epoch: 21 | Loss: 3.2233805656433105 | w: 2.3448848724365234
Epoch: 22 | Loss: 3.163391590118408 | w: 2.3448848724365234
Epoch: 23 | Loss: 3.1045126914978027 | w: 2.3448848724365234
Epoch: 24 | Loss: 3.04672908782959 | w: 2.3448848724365234
Epoch: 25 | Loss: 2.990025758743286 | w: 2.3448848724365234
Epoch: 26 | Loss: 2.934373140335083 | w: 2.3448848724365234
Epoch: 27 | Loss: 2.8797547817230225 | w: 2.3448848724365234
Epoch: 28 | Loss: 2.8261539936065674 | w: 2.3448848724365234
Epoch: 29 | Loss: 2.773554563522339 | w: 2.3448848724365234
Epoch: 30 | Loss: 2.7219345569610596 | w: 2.3448848724365234
Epoch: 31 | Loss: 2.6712708473205566 | w: 2.3448848724365234
Epoch: 32 | Loss: 2.621551990509033 | w: 2.3448848724365234
Epoch: 33 | Loss: 2.572758436203003 | w: 2.3448848724365234
Epoch: 34 | Loss: 2.5248758792877197 | w: 2.3448848724365234
Epoch: 35 | Loss: 2.4778826236724854 | w: 2.3448848724365234
Epoch: 36 | Loss: 2.4317634105682373 | w: 2.3448848724365234
Epoch: 37 | Loss: 2.3865020275115967 | w: 2.3448848724365234
Epoch: 38 | Loss: 2.342083692550659 | w: 2.3448848724365234
Epoch: 39 | Loss: 2.298491954803467 | w: 2.3448848724365234
Epoch: 40 | Loss: 2.2557120323181152 | w: 2.3448848724365234
Epoch: 41 | Loss: 2.2137269973754883 | w: 2.3448848724365234
Epoch: 42 | Loss: 2.1725239753723145 | w: 2.3448848724365234
Epoch: 43 | Loss: 2.1320881843566895 | w: 2.3448848724365234
Epoch: 44 | Loss: 2.092405319213867 | w: 2.3448848724365234
Epoch: 45 | Loss: 2.053462028503418 | w: 2.3448848724365234
Epoch: 46 | Loss: 2.0152411460876465 | w: 2.3448848724365234
Epoch: 47 | Loss: 1.977731704711914 | w: 2.3448848724365234
Epoch: 48 | Loss: 1.9409242868423462 | w: 2.3448848724365234
Epoch: 49 | Loss: 1.9047966003417969 | w: 2.3448848724365234
Epoch: 50 | Loss: 1.869340181350708 | w: 2.3448848724365234
Epoch: 51 | Loss: 1.8345476388931274 | w: 2.3448848724365234
Epoch: 52 | Loss: 1.8004045486450195 | w: 2.3448848724365234
Epoch: 53 | Loss: 1.766897201538086 | w: 2.3448848724365234
Epoch: 54 | Loss: 1.7340103387832642 | w: 2.3448848724365234
Epoch: 55 | Loss: 1.701735258102417 | w: 2.3448848724365234
Epoch: 56 | Loss: 1.6700595617294312 | w: 2.3448848724365234
Epoch: 57 | Loss: 1.6389776468276978 | w: 2.3448848724365234
Epoch: 58 | Loss: 1.6084731817245483 | w: 2.3448848724365234
Epoch: 59 | Loss: 1.578535795211792 | w: 2.3448848724365234
Epoch: 60 | Loss: 1.549156665802002 | w: 2.3448848724365234
Epoch: 61 | Loss: 1.520322322845459 | w: 2.3448848724365234
```

```
Epoch: 62 | Loss: 1.492027997970581 | w: 2.3448848724365234
Epoch: 63 | Loss: 1.4642534255981445 | w: 2.3448848724365234
Epoch: 64 | Loss: 1.4370014667510986 | w: 2.3448848724365234
Epoch: 65 | Loss: 1.4102556705474854 | w: 2.3448848724365234
Epoch: 66 | Loss: 1.3840065002441406 | w: 2.3448848724365234
Epoch: 67 | Loss: 1.358245849609375 | w: 2.3448848724365234
Epoch: 68 | Loss: 1.3329664468765259 | w: 2.3448848724365234
Epoch: 69 | Loss: 1.3081586360931396 | w: 2.3448848724365234
Epoch: 70 | Loss: 1.2838106155395508 | w: 2.3448848724365234
Epoch: 71 | Loss: 1.2599154710769653 | w: 2.3448848724365234
Epoch: 72 | Loss: 1.2364667654037476 | w: 2.3448848724365234
Epoch: 73 | Loss: 1.2134513854980469 | w: 2.3448848724365234
Epoch: 74 | Loss: 1.190863847732544 | w: 2.3448848724365234
Epoch: 75 | Loss: 1.1687015295028687 | w: 2.3448848724365234
Epoch: 76 | Loss: 1.1469495296478271 | w: 2.3448848724365234
Epoch: 77 | Loss: 1.1256020069122314 | w: 2.3448848724365234
Epoch: 78 | Loss: 1.1046502590179443 | w: 2.3448848724365234
Epoch: 79 | Loss: 1.0840911865234375 | w: 2.3448848724365234
Epoch: 80 | Loss: 1.0639140605926514 | w: 2.3448848724365234
Epoch: 81 | Loss: 1.0441125631332397 | w: 2.3448848724365234
Epoch: 82 | Loss: 1.0246798992156982 | w: 2.3448848724365234
Epoch: 83 | Loss: 1.0056073665618896 | w: 2.3448848724365234
Epoch: 84 | Loss: 0.9868906736373901 | w: 2.3448848724365234
Epoch: 85 | Loss: 0.9685209393501282 | w: 2.3448848724365234
Epoch: 86 | Loss: 0.9504930377006531 | w: 2.3448848724365234
Epoch: 87 | Loss: 0.9328028559684753 | w: 2.3448848724365234
Epoch: 88 | Loss: 0.9154413342475891 | w: 2.3448848724365234
Epoch: 89 | Loss: 0.8984050750732422 | w: 2.3448848724365234
Epoch: 90 | Loss: 0.8816840052604675 | w: 2.3448848724365234
Epoch: 91 | Loss: 0.8652714490890503 | w: 2.3448848724365234
Epoch: 92 | Loss: 0.8491684198379517 | w: 2.3448848724365234
Epoch: 93 | Loss: 0.8333638906478882 | w: 2.3448848724365234
Epoch: 94 | Loss: 0.8178545832633972 | w: 2.3448848724365234
Epoch: 95 | Loss: 0.802628219127655 | w: 2.3448848724365234
Epoch: 96 | Loss: 0.7876907587051392 | w: 2.3448848724365234
Epoch: 97 | Loss: 0.773030161857605 | w: 2.3448848724365234
Epoch: 98 | Loss: 0.7586407661437988 | w: 2.3448848724365234
Epoch: 99 | Loss: 0.7445210218429565 | w: 2.3448848724365234
Epoch: 100 | Loss: 0.7306647300720215 | w: 2.3448848724365234
```

Calculate y_{pred} for $x = 6$ after training the model

In [21]:

```
y_pred_with_train = quad_forward(6)

print("Actual Y Value for x=4 : 66")
print("Predicted Y Value before training : " , y_pred_without_train.item())
print("Predicted Y Value after training : " , y_pred_with_train.item())
```

```
Actual Y Value for x=4 : 66
Predicted Y Value before training :  42.0
Predicted Y Value after training :  65.66741180419922
```

In [ ]:

```
!pip install nbconvert
!sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-gene
ric
!jupyter nbconvert --to html "/content/drive/MyDrive/Colab Notebooks/Task_03_EE2
1S060.ipynb"
```