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([ 0.7762, -0.4039,  1.0991, -0.4072,  0.6939])
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
```

```python
three_dim = torch.randn((3, 3, 3))
print(three_dim)
```

```
tensor([[-2.0047, -0.5585, -0.3240],
        [-0.2907,  0.7649, -1.0998],
        [-1.0089, -0.9874, -2.6865]])
tensor([[[ 0.2957, -0.1774,  1.6920],
         [ 0.7589, -0.8556,  0.7433],
         [-0.2026, -1.3267, -0.0660]],

        [[ 0.0788,  1.0962,  0.9155],
         [ 0.8615,  0.3445, -0.1930],
         [ 0.9122,  0.9303, -1.0900]],

        [[ 0.4070,  0.1347,  1.1328],
         [-0.1610, -1.2374,  1.0245],
         [ 1.8327,  2.3215,  0.8277]]])
```

In [4]:

```python
# 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([-2.0047, -0.2907, -1.0089])
tensor([-2.0047, -0.5585, -0.3240])
```

In [5]:

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

```
tensor([[-2.0047, -0.5585],
        [-0.2907,  0.7649],
        [-1.0089, -0.9874]])
tensor([[-2.0047, -0.5585, -0.3240],
        [-0.2907,  0.7649, -1.0998]])
```

In [6]:

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

```
Tensor Shape :  torch.Size([2, 3])
Resized Tensor Shape :  torch.Size([3, 2])
Resized Tensor Shape :  torch.Size([3, 2])
Flattened Tensor Shape :  torch.Size([6])
```

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

In [7]:

```python
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 model de
def forward(x):
    #implement the forward model to compute y_pred as w*x + b
    ## YOUR CODE STARTS HERE
    y = w*x + b
    return y

    ## 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_pred - y_actual)**2
    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 based o

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 upda
        ## YOUR CODE STARTS HERE
        y_pred = forward(x)
        current_loss = loss(y_pred , y)
        current_loss.backward()
        w.data -= (alpha * (w.grad))
        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: 8.035331726074219 | w: 1.361407995223999
Epoch: 2 | Loss: 3.9245269298553467 | w: 1.6126642227172852
Epoch: 3 | Loss: 1.9271574020385742 | w: 1.7872123718261719
```

```
Epoch: 4 | Loss: 0.9558445811271667 | w: 1.9083435535430908
Epoch: 5 | Loss: 0.48289188742637634 | w: 1.9922776222229004
Epoch: 6 | Loss: 0.2521496117115021 | w: 2.0503103733062744
Epoch: 7 | Loss: 0.1392294019460678 | w: 2.090308427810669
Epoch: 8 | Loss: 0.08369665592908859 | w: 2.1177496910095215
Epoch: 9 | Loss: 0.05616690218448639 | w: 2.1364498138427734
Epoch: 10 | Loss: 0.04233689606189728 | w: 2.1490650177001953
```

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

In [14]:

```python
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 :  8.151883125305176
```

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

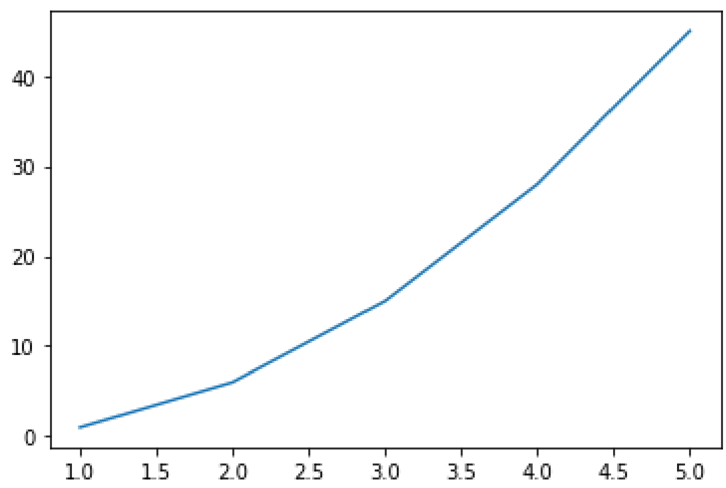
```
x = 1,2,3,4,5  y = 1,6,15,28,45
```

In [15]:

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

In [16]:

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



In [17]:

```python
# Initialize w2 and w1 with randon values
```

```
w_1 = torch.tensor([1.0], requires_grad=True)
w_2 = torch.tensor([1.0], requires_grad=True)
```

In [18]:

```
#wuadratic-forward function to calculate y_pred for a given x according to the quadr
def quad_forward(x):
    #implement the forward model to compute y_pred as w1*x + w2*(x^2)
    ## YOUR CODE STARTS HERE
    y=(w_1*x) + (w_2*x*x)
    return y

    ## 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_pred - y_actual)**2
    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)
print(y_pred_without_train)
```

```
tensor([42.], grad_fn=<AddBackward0>)
```

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 based o

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 upda
        ## YOUR CODE STARTS HERE
        y_pred = quad_forward(x)
        current_loss = loss(y_pred, y)
        current_loss.backward()
        w_1.data -= (alpha * (w_1.grad))
        w_2.data -= (alpha * (w_2.grad))
        ## YOUR CODE ENDS HERE

        # Manually zero the gradients after updating weights
        w_1.grad.data.zero_()
        w_2.grad.data.zero_()
        total_loss += current_loss
        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: 2.1490650177001953
Epoch: 2 | Loss: 6.430716514587402 | w: 2.1490650177001953
Epoch: 3 | Loss: 4.341702938079834 | w: 2.1490650177001953
Epoch: 4 | Loss: 4.463935852050781 | w: 2.1490650177001953
Epoch: 5 | Loss: 4.349801063537598 | w: 2.1490650177001953
Epoch: 6 | Loss: 4.273284435272217 | w: 2.1490650177001953
Epoch: 7 | Loss: 4.1931023597717285 | w: 2.1490650177001953
Epoch: 8 | Loss: 4.115151405334473 | w: 2.1490650177001953
Epoch: 9 | Loss: 4.038548469543457 | w: 2.1490650177001953
Epoch: 10 | Loss: 3.9633827209472656 | w: 2.1490650177001953
Epoch: 11 | Loss: 3.8896117210388184 | w: 2.1490650177001953
Epoch: 12 | Loss: 3.8172192573547363 | w: 2.1490650177001953
Epoch: 13 | Loss: 3.746173858642578 | w: 2.1490650177001953
Epoch: 14 | Loss: 3.6764469146728516 | w: 2.1490650177001953
Epoch: 15 | Loss: 3.6080188751220703 | w: 2.1490650177001953
Epoch: 16 | Loss: 3.540867567062378 | w: 2.1490650177001953
Epoch: 17 | Loss: 3.4749627113342285 | w: 2.1490650177001953
Epoch: 18 | Loss: 3.410282850265503 | w: 2.1490650177001953
Epoch: 19 | Loss: 3.3468079566955566 | w: 2.1490650177001953
Epoch: 20 | Loss: 3.2845168113708496 | w: 2.1490650177001953
Epoch: 21 | Loss: 3.223386287689209 | w: 2.1490650177001953
Epoch: 22 | Loss: 3.1633896827697754 | w: 2.1490650177001953
Epoch: 23 | Loss: 3.104512929916382 | w: 2.1490650177001953
Epoch: 24 | Loss: 3.0467324256896973 | w: 2.1490650177001953
Epoch: 25 | Loss: 2.990025758743286 | w: 2.1490650177001953
Epoch: 26 | Loss: 2.934375762939453 | w: 2.1490650177001953
Epoch: 27 | Loss: 2.879761219024658 | w: 2.1490650177001953
Epoch: 28 | Loss: 2.8261590003967285 | w: 2.1490650177001953
Epoch: 29 | Loss: 2.773557424545288 | w: 2.1490650177001953
Epoch: 30 | Loss: 2.721935272216797 | w: 2.1490650177001953
Epoch: 31 | Loss: 2.6712708473205566 | w: 2.1490650177001953
Epoch: 32 | Loss: 2.621551990509033 | w: 2.1490650177001953
Epoch: 33 | Loss: 2.572758436203003 | w: 2.1490650177001953
Epoch: 34 | Loss: 2.5248782634735107 | w: 2.1490650177001953
Epoch: 35 | Loss: 2.477883815765381 | w: 2.1490650177001953
Epoch: 36 | Loss: 2.4317591190338135 | w: 2.1490650177001953
Epoch: 37 | Loss: 2.386500597000122 | w: 2.1490650177001953
Epoch: 38 | Loss: 2.342085599899292 | w: 2.1490650177001953
Epoch: 39 | Loss: 2.298488140106201 | w: 2.1490650177001953
Epoch: 40 | Loss: 2.255709648132324 | w: 2.1490650177001953
Epoch: 41 | Loss: 2.2137296199798584 | w: 2.1490650177001953
Epoch: 42 | Loss: 2.1725244522094727 | w: 2.1490650177001953
Epoch: 43 | Loss: 2.1320881843566895 | w: 2.1490650177001953
Epoch: 44 | Loss: 2.092407703399658 | w: 2.1490650177001953
Epoch: 45 | Loss: 2.053462505340576 | w: 2.1490650177001953
Epoch: 46 | Loss: 2.0152413845062256 | w: 2.1490650177001953
Epoch: 47 | Loss: 1.9777324199676514 | w: 2.1490650177001953
Epoch: 48 | Loss: 1.9409221410751343 | w: 2.1490650177001953
Epoch: 49 | Loss: 1.9047987461090088 | w: 2.1490650177001953
Epoch: 50 | Loss: 1.8693469762802124 | w: 2.1490650177001953
Epoch: 51 | Loss: 1.834551453590393 | w: 2.1490650177001953
Epoch: 52 | Loss: 1.800405502319336 | w: 2.1490650177001953
Epoch: 53 | Loss: 1.7668958902359009 | w: 2.1490650177001953
Epoch: 54 | Loss: 1.7340103387832642 | w: 2.1490650177001953
Epoch: 55 | Loss: 1.7017333507537842 | w: 2.1490650177001953
Epoch: 56 | Loss: 1.6700608730316162 | w: 2.1490650177001953
Epoch: 57 | Loss: 1.638979196548462 | w: 2.1490650177001953
Epoch: 58 | Loss: 1.6084703207015991 | w: 2.1490650177001953
Epoch: 59 | Loss: 1.5785328149795532 | w: 2.1490650177001953
Epoch: 60 | Loss: 1.5491578578948975 | w: 2.1490650177001953
Epoch: 61 | Loss: 1.5203224420547485 | w: 2.1490650177001953
```

```
Epoch: 62 | Loss: 1.4920268058776855 | w: 2.1490650177001953
Epoch: 63 | Loss: 1.464254379272461 | w: 2.1490650177001953
Epoch: 64 | Loss: 1.4370005130767822 | w: 2.1490650177001953
Epoch: 65 | Loss: 1.4102575778961182 | w: 2.1490650177001953
Epoch: 66 | Loss: 1.384007215499878 | w: 2.1490650177001953
Epoch: 67 | Loss: 1.3582470417022705 | w: 2.1490650177001953
Epoch: 68 | Loss: 1.3329681158065796 | w: 2.1490650177001953
Epoch: 69 | Loss: 1.3081576824188232 | w: 2.1490650177001953
Epoch: 70 | Loss: 1.2838081121444702 | w: 2.1490650177001953
Epoch: 71 | Loss: 1.2599149942398071 | w: 2.1490650177001953
Epoch: 72 | Loss: 1.2364667654037476 | w: 2.1490650177001953
Epoch: 73 | Loss: 1.2134500741958618 | w: 2.1490650177001953
Epoch: 74 | Loss: 1.1908642053604126 | w: 2.1490650177001953
Epoch: 75 | Loss: 1.168701410293579 | w: 2.1490650177001953
Epoch: 76 | Loss: 1.1469495296478271 | w: 2.1490650177001953
Epoch: 77 | Loss: 1.1256003379821777 | w: 2.1490650177001953
Epoch: 78 | Loss: 1.104649543762207 | w: 2.1490650177001953
Epoch: 79 | Loss: 1.084090232849121 | w: 2.1490650177001953
Epoch: 80 | Loss: 1.0639140605926514 | w: 2.1490650177001953
Epoch: 81 | Loss: 1.0441116094589233 | w: 2.1490650177001953
Epoch: 82 | Loss: 1.02467679977417 | w: 2.1490650177001953
Epoch: 83 | Loss: 1.005606770515442 | w: 2.1490650177001953
Epoch: 84 | Loss: 0.9868909120559692 | w: 2.1490650177001953
Epoch: 85 | Loss: 0.9685211181640625 | w: 2.1490650177001953
Epoch: 86 | Loss: 0.9504938125610352 | w: 2.1490650177001953
Epoch: 87 | Loss: 0.9328027963638306 | w: 2.1490650177001953
Epoch: 88 | Loss: 0.9154437780380249 | w: 2.1490650177001953
Epoch: 89 | Loss: 0.8984071016311646 | w: 2.1490650177001953
Epoch: 90 | Loss: 0.8816855549812317 | w: 2.1490650177001953
Epoch: 91 | Loss: 0.8652728199958801 | w: 2.1490650177001953
Epoch: 92 | Loss: 0.8491679430007935 | w: 2.1490650177001953
Epoch: 93 | Loss: 0.8333643674850464 | w: 2.1490650177001953
Epoch: 94 | Loss: 0.8178556561470032 | w: 2.1490650177001953
Epoch: 95 | Loss: 0.8026320338249207 | w: 2.1490650177001953
Epoch: 96 | Loss: 0.7876909971237183 | w: 2.1490650177001953
Epoch: 97 | Loss: 0.7730289697647095 | w: 2.1490650177001953
Epoch: 98 | Loss: 0.7586439847946167 | w: 2.1490650177001953
Epoch: 99 | Loss: 0.7445234060287476 | w: 2.1490650177001953
Epoch: 100 | Loss: 0.7306660413742065 | w: 2.1490650177001953
```

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

In [21]:

```python
y_pred_with_train = 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 :  12.450013160705566
```

In [ ]: