Task-03

August 4, 2022

0.0.1 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

[1]:
```
# import the necessary libraries
import torch
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
/home/koushik/anaconda3/envs/kpb/lib/python3.9/site-
packages/torch/cuda/__init__.py:83: UserWarning: CUDA initialization: CUDA
unknown error - this may be due to an incorrectly set up environment, e.g.
changing env variable CUDA_VISIBLE_DEVICES after program start. Setting the
available devices to be zero. (Triggered internally at
../c10/cuda/CUDAFunctions.cpp:109.)
  return torch._C._cuda_getDeviceCount() > 0
```

0.1 Preliminaries - Pytorch Basics

[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([-0.6173,  0.8696, -0.0281,  0.3251, -0.8105])
The size of the to_tensor:  torch.Size([5])
The size of the to_tensor:  torch.Size([5])
```

[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([[-1.1051, -0.9893, -0.9709],
        [ 1.1245, -1.9685, -0.2387],
        [ 0.7444,  0.8782, -1.9104]])
tensor([[[ 1.1167,  0.7769,  1.2348],
         [-0.8657, -0.7370, -0.6928],
         [ 0.8738,  0.1863,  0.0636]],

        [[-0.9268,  1.0652,  0.3860],
         [-0.9666,  0.2789,  0.5441],
         [-0.1008,  1.4361,  0.1923]],

        [[-0.3839,  0.0357, -1.1384],
         [-0.7892,  0.6098, -1.4819],
         [ 1.0663,  0.4981,  1.0752]]])
```

[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([-1.1051,  1.1245,  0.7444])
tensor([-1.1051, -0.9893, -0.9709])
```

[5]:
```
print(two_dim[:, 0:2])
print(two_dim[0:2, :])
```

```
tensor([[-1.1051, -0.9893],
        [ 1.1245, -1.9685],
        [ 0.7444,  0.8782]])
tensor([[-1.1051, -0.9893, -0.9709],
        [ 1.1245, -1.9685, -0.2387]])
```

[6]:
```
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 $x = 1$

[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.)
```

0.1.1 Task 03 - a

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

[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()
```

[9]:
```
# 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

[10]:
```
#sample-dataset
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]
```

0.2 Task: 03 - b

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

[11]:
```
# 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
```

[12]:
```
#forward function to calculate y_pred for a given x according to the linear␣
 ↪model defined above
def forward(x):
    #implement the forward model to compute y_pred as w*x + b
    ## YOUR CODE STARTS HERE
```

```
    return ((w*x) + b)

    ## 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
    return ((y_pred- y_actual)**2)

    ## YOUR CODE ENDS HERE
```

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

[13]:
```
y_pred_without_train = forward(4)
```

Begin Training

[14]:
```
# In this method, we learn the dataset multiple times (called epochs)
# Each time, the weight (w) gets updates using the graident decent algorithm␣
 ↪based 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
        yPred = forward(x)            #forward
        currentLoss = loss(yPred,y)     #loss
        total_loss += currentLoss
        currentLoss.backward()    #back

        w.data = w.data - alpha * (w.grad.item()) #iterate/update weight
        ## YOUR CODE ENDS HERE

        # Manually zero the gradients after updating weights
        w.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

[15]:
```
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
```

0.3 Task: 03 - c

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

Using backward propagation for quadratic model $\hat{y} = x^2 * w_2 + x * w_1$

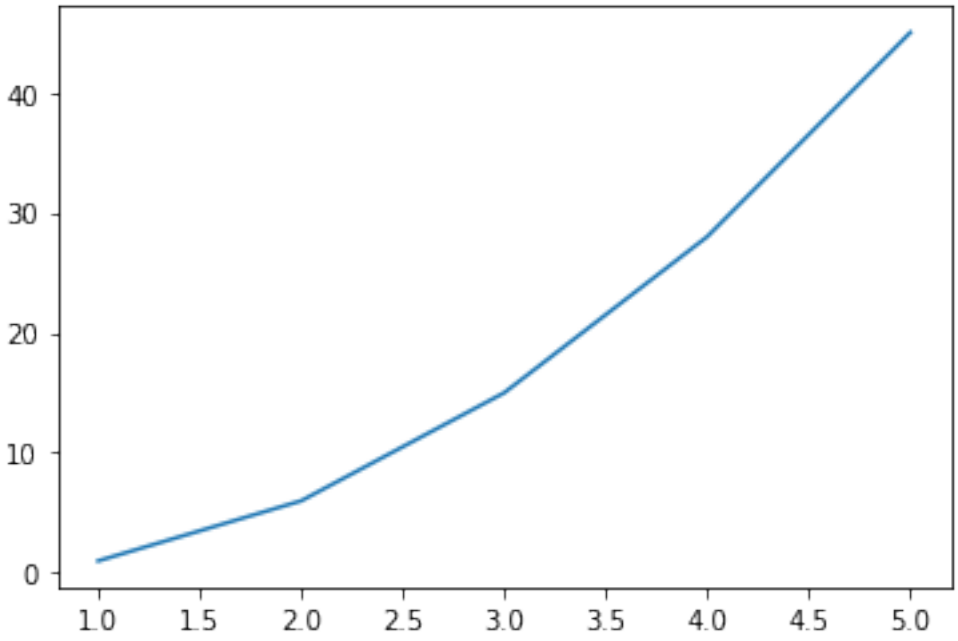
$loss = (\hat{y} - y)^2$

- Using Dummy values of x and y

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

[16]:
```
x_data = [1.0, 2.0, 3.0, 4.0, 5.0]
y_data = [1.0, 6.0, 15.0, 28, 45]
```

[17]:
```
# Visualize the given dataset
plt.plot(x_data,y_data)
plt.show()
```



[18]:
```
# 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)
```

[19]:
```
# Quadratic forward pass based on the function above. Taking b as zero for now
def quad_forward(x):
    return w_2*(x**2)+w_1*x

# Loss fucntion as per the defination above
def loss(y_pred,y_actual):
    return (y_pred-y_actual)**2
```

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

[20]:
```
y_pred_without_train = quad_forward(6)
```

Begin Training

[21]:
```
# In this method, we learn the dataset multiple times (called epochs)
# Each time, the weight (w) gets updates using the graident decent algorithm␣
 ↪based on weights of the previous epoch

alpha = 0.001 # Let us set learning rate as 0.01
weight_list_1 = []
weight_list_2 = []
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)
        total_loss += current_loss
        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()
        ## 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_1: {w_1.item()} | w_2: {w_2.item()}")
    weight_list_1.append(w_1)
    weight_list_2.append(w_2)
    loss_list.append(avg_mse)
```

```
Epoch: 1 | Loss: 23.789419174194336 | w_1: 1.1494112014770508 | w_2:
1.6618750095367432
Epoch: 2 | Loss: 5.293725967407227 | w_1: 1.1156214475631714 | w_2:
1.5936779975891113
Epoch: 3 | Loss: 4.067604064941406 | w_1: 1.101760745048523 | w_2:
1.6043506860733032
Epoch: 4 | Loss: 4.0941009521484375 | w_1: 1.0858769416809082 | w_2:
1.606476068496704
Epoch: 5 | Loss: 4.023508071899414 | w_1: 1.0703413486480713 | w_2:
1.6095004081726074
Epoch: 6 | Loss: 3.9647536277770996 | w_1: 1.0548969507217407 | w_2:
1.612403392791748
Epoch: 7 | Loss: 3.90571928024292 | w_1: 1.0395704507827759 | w_2:
1.6152952909469604
```

```
Epoch: 8 | Loss: 3.8476860523223877 | w_1: 1.0243579149246216 | w_2:
1.6181646585464478
Epoch: 9 | Loss: 3.7905020713806152 | w_1: 1.0092588663101196 | w_2:
1.6210126876831055
Epoch: 10 | Loss: 3.734168291091919 | w_1: 0.9942724704742432 | w_2:
1.6238394975662231
Epoch: 11 | Loss: 3.6786715984344482 | w_1: 0.9793977737426758 | w_2:
1.6266450881958008
Epoch: 12 | Loss: 3.6239993572235107 | w_1: 0.9646340608596802 | w_2:
1.629429817199707
Epoch: 13 | Loss: 3.570141315460205 | w_1: 0.9499804377555847 | w_2:
1.632193922996521
Epoch: 14 | Loss: 3.517085313796997 | w_1: 0.9354360699653625 | w_2:
1.6349371671676636
Epoch: 15 | Loss: 3.4648163318634033 | w_1: 0.9210003614425659 | w_2:
1.6376601457595825
Epoch: 16 | Loss: 3.4133172035217285 | w_1: 0.9066721796989441 | w_2:
1.6403625011444092
Epoch: 17 | Loss: 3.362590789794922 | w_1: 0.8924509882926941 | w_2:
1.6430450677871704
Epoch: 18 | Loss: 3.312619686126709 | w_1: 0.8783357739448547 | w_2:
1.645707607269287
Epoch: 19 | Loss: 3.263388156890869 | w_1: 0.8643258213996887 | w_2:
1.6483498811721802
Epoch: 20 | Loss: 3.2148895263671875 | w_1: 0.8504204750061035 | w_2:
1.6509729623794556
Epoch: 21 | Loss: 3.1671102046966553 | w_1: 0.8366188406944275 | w_2:
1.653576135635376
Epoch: 22 | Loss: 3.1200389862060547 | w_1: 0.8229200839996338 | w_2:
1.6561598777770996
Epoch: 23 | Loss: 3.07366943359375 | w_1: 0.8093234896659851 | w_2:
1.6587246656417847
Epoch: 24 | Loss: 3.0279903411865234 | w_1: 0.7958282828330994 | w_2:
1.661270022392273
Epoch: 25 | Loss: 2.98298978805542 | w_1: 0.7824338674545288 | w_2:
1.6637965440750122
Epoch: 26 | Loss: 2.9386563301086426 | w_1: 0.7691393494606018 | w_2:
1.666304111480713
Epoch: 27 | Loss: 2.8949832916259766 | w_1: 0.7559439539909363 | w_2:
1.6687930822372437
Epoch: 28 | Loss: 2.8519604206085205 | w_1: 0.7428469657897949 | w_2:
1.671263337135315
Epoch: 29 | Loss: 2.809572696685791 | w_1: 0.7298476696014404 | w_2:
1.673715353012085
Epoch: 30 | Loss: 2.767817974090576 | w_1: 0.716945230960846 | w_2:
1.6761490106582642
Epoch: 31 | Loss: 2.726682662963867 | w_1: 0.7041390538215637 | w_2:
1.678564429283142
```

```
Epoch: 32 | Loss: 2.686161756515503 | w_1: 0.6914284229278564 | w_2:
1.6809619665145874
Epoch: 33 | Loss: 2.646240472793579 | w_1: 0.678812563419342 | w_2:
1.6833417415618896
Epoch: 34 | Loss: 2.6069116592407227 | w_1: 0.666290819644928 | w_2:
1.6857032775878906
Epoch: 35 | Loss: 2.568166494369507 | w_1: 0.6538625955581665 | w_2:
1.6880476474761963
Epoch: 36 | Loss: 2.530001163482666 | w_1: 0.6415269374847412 | w_2:
1.6903743743896484
Epoch: 37 | Loss: 2.4924018383026123 | w_1: 0.6292833685874939 | w_2:
1.6926838159561157
Epoch: 38 | Loss: 2.4553608894348145 | w_1: 0.6171311140060425 | w_2:
1.6949760913848877
Epoch: 39 | Loss: 2.418872833251953 | w_1: 0.6050694584846497 | w_2:
1.6972512006759644
Epoch: 40 | Loss: 2.3829216957092285 | w_1: 0.5930976867675781 | w_2:
1.6995090246200562
Epoch: 41 | Loss: 2.347506523132324 | w_1: 0.5812153220176697 | w_2:
1.7017505168914795
Epoch: 42 | Loss: 2.3126184940338135 | w_1: 0.5694215297698975 | w_2:
1.703974962234497
Epoch: 43 | Loss: 2.2782483100891113 | w_1: 0.557715654373169 | w_2:
1.706182837486267
Epoch: 44 | Loss: 2.2443912029266357 | w_1: 0.5460972785949707 | w_2:
1.7083745002746582
Epoch: 45 | Loss: 2.2110342979431152 | w_1: 0.5345653891563416 | w_2:
1.7105494737625122
Epoch: 46 | Loss: 2.1781725883483887 | w_1: 0.5231196284294128 | w_2:
1.7127083539962769
Epoch: 47 | Loss: 2.145801544189453 | w_1: 0.5117591619491577 | w_2:
1.7148512601852417
Epoch: 48 | Loss: 2.1139135360717773 | w_1: 0.5004834532737732 | w_2:
1.7169781923294067
Epoch: 49 | Loss: 2.0824952125549316 | w_1: 0.48929181694984436 | w_2:
1.7190889120101929
Epoch: 50 | Loss: 2.051546096801758 | w_1: 0.47818368673324585 | w_2:
1.721184253692627
Epoch: 51 | Loss: 2.0210583209991455 | w_1: 0.4671584367752075 | w_2:
1.7232639789581299
Epoch: 52 | Loss: 1.9910224676132202 | w_1: 0.45621535181999207 | w_2:
1.7253278493881226
Epoch: 53 | Loss: 1.961432695388794 | w_1: 0.44535401463508606 | w_2:
1.7273768186569214
Epoch: 54 | Loss: 1.9322824478149414 | w_1: 0.43457353115081787 | w_2:
1.7294100522994995
Epoch: 55 | Loss: 1.9035654067993164 | w_1: 0.42387351393699646 | w_2:
1.7314285039901733
```

```
Epoch: 56 | Loss: 1.8752772808074951 | w_1: 0.4132533073425293 | w_2:
1.7334316968917847
Epoch: 57 | Loss: 1.847405195236206 | w_1: 0.40271222591400146 | w_2:
1.7354196310043335
Epoch: 58 | Loss: 1.819947600364685 | w_1: 0.3922499120235443 | w_2:
1.7373932600021362
Epoch: 59 | Loss: 1.7929010391235352 | w_1: 0.381865531206131 | w_2:
1.739351749420166
Epoch: 60 | Loss: 1.7662551403045654 | w_1: 0.37155869603157043 | w_2:
1.7412960529327393
Epoch: 61 | Loss: 1.7400058507919312 | w_1: 0.3613286316394806 | w_2:
1.7432254552841187
Epoch: 62 | Loss: 1.7141444683074951 | w_1: 0.3511749505996704 | w_2:
1.7451406717300415
Epoch: 63 | Loss: 1.6886718273162842 | w_1: 0.341096967458725 | w_2:
1.7470417022705078
Epoch: 64 | Loss: 1.663575530052185 | w_1: 0.3310941755771637 | w_2:
1.748928427696228
Epoch: 65 | Loss: 1.6388533115386963 | w_1: 0.3211659789085388 | w_2:
1.7508010864257812
Epoch: 66 | Loss: 1.6144946813583374 | w_1: 0.311311811208725 | w_2:
1.7526596784591675
Epoch: 67 | Loss: 1.5904988050460815 | w_1: 0.3015311658382416 | w_2:
1.7545044422149658
Epoch: 68 | Loss: 1.5668623447418213 | w_1: 0.29182347655296326 | w_2:
1.7563356161117554
Epoch: 69 | Loss: 1.543575406074524 | w_1: 0.28218814730644226 | w_2:
1.758152961730957
Epoch: 70 | Loss: 1.5206345319747925 | w_1: 0.2726247012615204 | w_2:
1.7599568367004395
Epoch: 71 | Loss: 1.4980374574661255 | w_1: 0.2631326913833618 | w_2:
1.7617474794387817
Epoch: 72 | Loss: 1.4757730960845947 | w_1: 0.2537113428115845 | w_2:
1.7635241746902466
Epoch: 73 | Loss: 1.453840970993042 | w_1: 0.24436034262180328 | w_2:
1.765288233757019
Epoch: 74 | Loss: 1.432237148284912 | w_1: 0.2350790649652481 | w_2:
1.7670389413833618
Epoch: 75 | Loss: 1.4109532833099365 | w_1: 0.22586700320243835 | w_2:
1.768776535987854
Epoch: 76 | Loss: 1.389981746673584 | w_1: 0.21672362089157104 | w_2:
1.7705010175704956
Epoch: 77 | Loss: 1.369321584701538 | w_1: 0.20764845609664917 | w_2:
1.7722127437591553
Epoch: 78 | Loss: 1.3489738702774048 | w_1: 0.19864103198051453 | w_2:
1.773911952972412
Epoch: 79 | Loss: 1.3289241790771484 | w_1: 0.18970070779323578 | w_2:
1.7755980491638184
```

```
Epoch: 80 | Loss: 1.3091728687286377 | w_1: 0.1808270961046219 | w_2:
1.7772717475891113
Epoch: 81 | Loss: 1.2897166013717651 | w_1: 0.17201969027519226 | w_2:
1.778933048248291
Epoch: 82 | Loss: 1.2705495357513428 | w_1: 0.16327796876430511 | w_2:
1.7805818319320679
Epoch: 83 | Loss: 1.25166654586792 | w_1: 0.15460145473480225 | w_2:
1.7822184562683105
Epoch: 84 | Loss: 1.233066201210022 | w_1: 0.14598968625068665 | w_2:
1.783842921257019
Epoch: 85 | Loss: 1.2147419452667236 | w_1: 0.13744212687015533 | w_2:
1.7854552268981934
Epoch: 86 | Loss: 1.1966872215270996 | w_1: 0.12895826995372772 | w_2:
1.787055253982544
Epoch: 87 | Loss: 1.1789003610610962 | w_1: 0.12053774297237396 | w_2:
1.788643479347229
Epoch: 88 | Loss: 1.1613801717758179 | w_1: 0.11218006908893585 | w_2:
1.790220022201538
Epoch: 89 | Loss: 1.1441218852996826 | w_1: 0.10388471186161041 | w_2:
1.7917847633361816
Epoch: 90 | Loss: 1.1271164417266846 | w_1: 0.09565116465091705 | w_2:
1.7933375835418701
Epoch: 91 | Loss: 1.1103665828704834 | w_1: 0.0874791145324707 | w_2:
1.7948791980743408
Epoch: 92 | Loss: 1.093865990638733 | w_1: 0.07936801016330719 | w_2:
1.796409249305725
Epoch: 93 | Loss: 1.0776091814041138 | w_1: 0.07131728529930115 | w_2:
1.7979276180267334
Epoch: 94 | Loss: 1.0615931749343872 | w_1: 0.06332667917013168 | w_2:
1.799434781074524
Epoch: 95 | Loss: 1.0458157062530518 | w_1: 0.055395711213350296 | w_2:
1.8009308576583862
Epoch: 96 | Loss: 1.0302761793136597 | w_1: 0.04752388596534729 | w_2:
1.8024157285690308
Epoch: 97 | Loss: 1.0149630308151245 | w_1: 0.03971068188548088 | w_2:
1.803889274597168
Epoch: 98 | Loss: 0.9998787045478821 | w_1: 0.03195581212639809 | w_2:
1.805351972579956
Epoch: 99 | Loss: 0.9850174784660339 | w_1: 0.024258777499198914 | w_2:
1.806803822517395
Epoch: 100 | Loss: 0.970378041267395 | w_1: 0.016619160771369934 | w_2:
1.8082448244094849
```

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

[22]:
```
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 :  65.196533203125
```