Task-03

August 5, 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

[24]:
```
# import the necessary libraries
import torch
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

0.1 Preliminaries - Pytorch Basics

[25]:
```
# 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([-0.0123,  0.2534,  0.8436, -0.8071, -0.1570])
The size of the to_tensor:  torch.Size([5])
The size of the to_tensor:  torch.Size([5])
```

[26]:
```
# 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([[ 0.1920, -0.9857,  0.0202],
        [-0.4803,  0.4148, -0.1485],
        [ 1.8382, -0.8884,  0.8270]])
tensor([[[ 0.5306, -0.7220,  0.5482],
         [-0.2700, -1.2972, -0.8821],
         [-0.1850, -0.7852, -0.8710]],

        [[-1.6633,  0.3835, -0.5511],
         [ 0.2636, -0.2619,  0.6295],
         [ 1.0871, -1.3955, -0.8907]],

        [[ 1.2143,  0.7296,  0.0511],
         [-0.2637,  0.1045,  0.5383],
         [-0.4690, -0.6278, -0.7048]]])
```

[27]:
```
# 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([ 0.1920, -0.4803,  1.8382])
tensor([ 0.1920, -0.9857,  0.0202])
```

[28]:
```
print(two_dim[:, 0:2])
print(two_dim[0:2, :])
```

```
tensor([[ 0.1920, -0.9857],
        [-0.4803,  0.4148],
        [ 1.8382, -0.8884]])
tensor([[ 0.1920, -0.9857,  0.0202],
        [-0.4803,  0.4148, -0.1485]])
```

[29]:
```
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 $x = 1$

[30]:
```
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.

[31]:
```
# 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()
```

[32]:
```
# 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

[33]:
```
#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

[34]:
```
# 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
```

[35]:
```
#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

[36]:
```
y_pred_without_train = forward(4)
```

Begin Training

[37]:
```
# 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

[38]:
```
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 $\hat{y} = x^2 * w_2 + x * w_1$

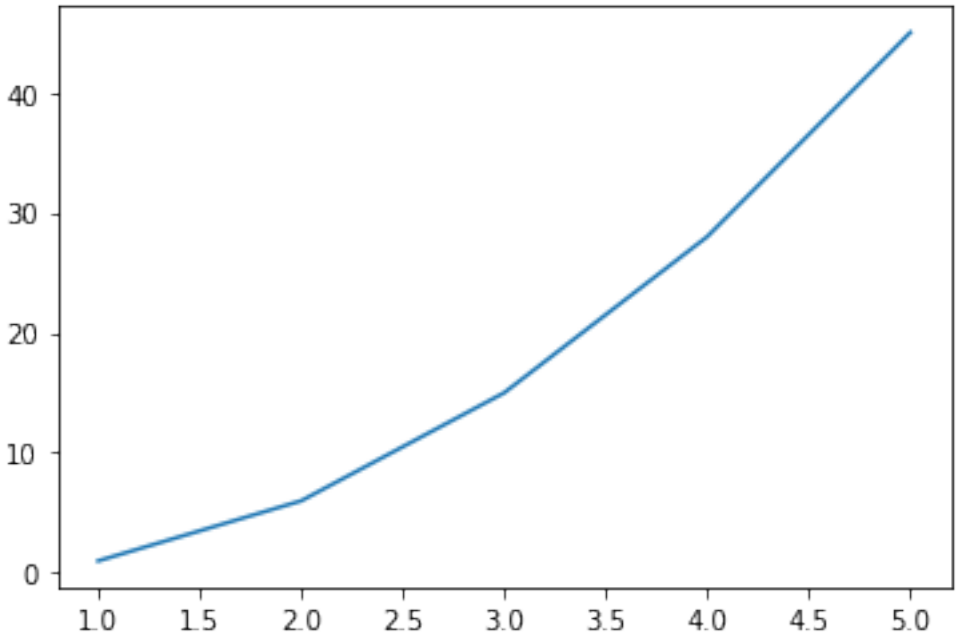
$loss = (\hat{y} - y)^2$

- Using Dummy values of x and y

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

[39]:
```
x_data = [1.0, 2.0, 3.0, 4.0, 5.0]
y_data = [1.0, 6.0, 15.0, 28, 45]
```

[40]:
```
# Visualize the given dataset
plt.plot(x_data,y_data)
plt.show()
```



[41]:
```
# 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)
```

[42]:
```
# 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

[43]:
```
y_pred_without_train = quad_forward(6)
```

Begin Training

[44]:
```
# 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 = 0.0012 # 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)         #forward prop
        current_loss = loss(y_pred, y)    #loss estimate
        total_loss += current_loss        ##total loss
        current_loss.backward()            #gradient

        w_1.data = w_1.data - (alpha * w_1.grad.item())  #update the weights
        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: 19.670841217041016 | w_1: 1.1621068716049194 | w_2:
1.7099769115447998
Epoch: 2 | Loss: 6.430711269378662 | w_1: 1.115532636642456 | w_2:
1.610548496246338
Epoch: 3 | Loss: 4.341702938079834 | w_1: 1.099595308303833 | w_2:
1.629080057144165
Epoch: 4 | Loss: 4.463932037353516 | w_1: 1.0794039964675903 | w_2:
1.6303775310516357
Epoch: 5 | Loss: 4.349801063537598 | w_1: 1.060043454170227 | w_2:
1.6341499090194702
Epoch: 6 | Loss: 4.273285865783691 | w_1: 1.0407702922821045 | w_2:
1.6375246047973633
Epoch: 7 | Loss: 4.193112373352051 | w_1: 1.0216909646987915 | w_2:
1.6409205198287964
```

```
Epoch: 8 | Loss: 4.115159511566162 | w_1: 1.0027880668640137 | w_2:
1.644276738166809
Epoch: 9 | Loss: 4.038552284240723 | w_1: 0.9840622544288635 | w_2:
1.6476027965545654
Epoch: 10 | Loss: 3.9633827209472656 | w_1: 0.9655115008354187 | w_2:
1.65089750289917
Epoch: 11 | Loss: 3.8896148204803467 | w_1: 0.9471341967582703 | w_2:
1.654161810874939
Epoch: 12 | Loss: 3.817220687866211 | w_1: 0.928928554058075 | w_2:
1.6573951244354248
Epoch: 13 | Loss: 3.7461726665496826 | w_1: 0.9108932614326477 | w_2:
1.660598635673523
Epoch: 14 | Loss: 3.6764445304870605 | w_1: 0.8930264711380005 | w_2:
1.6637717485427856
Epoch: 15 | Loss: 3.6080145835876465 | w_1: 0.8753268718719482 | w_2:
1.6669154167175293
Epoch: 16 | Loss: 3.5408616065979004 | w_1: 0.8577927947044373 | w_2:
1.670029878616333
Epoch: 17 | Loss: 3.4749622344970703 | w_1: 0.8404226303100586 | w_2:
1.6731152534484863
Epoch: 18 | Loss: 3.4102847576141357 | w_1: 0.8232148885726929 | w_2:
1.6761715412139893
Epoch: 19 | Loss: 3.3468120098114014 | w_1: 0.8061680793762207 | w_2:
1.6791993379592896
Epoch: 20 | Loss: 3.2845165729522705 | w_1: 0.7892804741859436 | w_2:
1.6821985244750977
Epoch: 21 | Loss: 3.2233805656433105 | w_1: 0.7725509405136108 | w_2:
1.6851699352264404
Epoch: 22 | Loss: 3.163391590118408 | w_1: 0.7559778690338135 | w_2:
1.688113808631897
Epoch: 23 | Loss: 3.1045126914978027 | w_1: 0.7395595908164978 | w_2:
1.6910297870635986
Epoch: 24 | Loss: 3.04672908782959 | w_1: 0.7232949137687683 | w_2:
1.6939187049865723
Epoch: 25 | Loss: 2.990025758743286 | w_1: 0.7071822285652161 | w_2:
1.6967805624008179
Epoch: 26 | Loss: 2.934373140335083 | w_1: 0.6912202835083008 | w_2:
1.6996155977249146
Epoch: 27 | Loss: 2.8797547817230225 | w_1: 0.6754074692726135 | w_2:
1.7024239301681519
Epoch: 28 | Loss: 2.8261539936065674 | w_1: 0.659742534160614 | w_2:
1.7052063941955566
Epoch: 29 | Loss: 2.773554563522339 | w_1: 0.6442241668701172 | w_2:
1.7079628705978394
Epoch: 30 | Loss: 2.7219345569610596 | w_1: 0.6288508772850037 | w_2:
1.7106932401657104
Epoch: 31 | Loss: 2.6712708473205566 | w_1: 0.6136212944984436 | w_2:
1.7133980989456177
```

```
Epoch: 32 | Loss: 2.621551990509033 | w_1: 0.5985341668128967 | w_2:
1.7160779237747192
Epoch: 33 | Loss: 2.572758436203003 | w_1: 0.5835880637168884 | w_2:
1.7187325954437256
Epoch: 34 | Loss: 2.5248758792877197 | w_1: 0.5687816739082336 | w_2:
1.7213623523712158
Epoch: 35 | Loss: 2.4778826236724854 | w_1: 0.5541138052940369 | w_2:
1.7239676713943481
Epoch: 36 | Loss: 2.4317634105682373 | w_1: 0.5395830273628235 | w_2:
1.7265485525131226
Epoch: 37 | Loss: 2.3865020275115967 | w_1: 0.5251880884170532 | w_2:
1.7291052341461182
Epoch: 38 | Loss: 2.342083692550659 | w_1: 0.510927677154541 | w_2:
1.7316380739212036
Epoch: 39 | Loss: 2.298491954803467 | w_1: 0.4968006908893585 | w_2:
1.7341471910476685
Epoch: 40 | Loss: 2.2557120323181152 | w_1: 0.4828057885169983 | w_2:
1.736633062362671
Epoch: 41 | Loss: 2.2137269973754883 | w_1: 0.46894168853759766 | w_2:
1.7390953302383423
Epoch: 42 | Loss: 2.1725239753723145 | w_1: 0.4552072286605835 | w_2:
1.74153470993042
Epoch: 43 | Loss: 2.1320881843566895 | w_1: 0.4416012763977051 | w_2:
1.743951439857483
Epoch: 44 | Loss: 2.092405319213867 | w_1: 0.4281224012374878 | w_2:
1.7463454008102417
Epoch: 45 | Loss: 2.053462028503418 | w_1: 0.41476961970329285 | w_2:
1.7487173080444336
Epoch: 46 | Loss: 2.0152411460876465 | w_1: 0.4015416204929352 | w_2:
1.7510664463043213
Epoch: 47 | Loss: 1.977731704711914 | w_1: 0.3884373605251312 | w_2:
1.7533941268920898
Epoch: 48 | Loss: 1.9409242868423462 | w_1: 0.3754555881023407 | w_2:
1.755699872970581
Epoch: 49 | Loss: 1.9047966003417969 | w_1: 0.36259517073631287 | w_2:
1.757983922958374
Epoch: 50 | Loss: 1.869340181350708 | w_1: 0.34985506534576416 | w_2:
1.7602466344833374
Epoch: 51 | Loss: 1.8345476388931274 | w_1: 0.3372340798377991 | w_2:
1.7624883651733398
Epoch: 52 | Loss: 1.8004045486450195 | w_1: 0.32473108172416687 | w_2:
1.764709234237671
Epoch: 53 | Loss: 1.766897201538086 | w_1: 0.3123449683189392 | w_2:
1.766909122467041
Epoch: 54 | Loss: 1.7340103387832642 | w_1: 0.3000746965408325 | w_2:
1.7690885066986084
Epoch: 55 | Loss: 1.701735258102417 | w_1: 0.28791916370391846 | w_2:
1.7712475061416626
```

```
Epoch: 56 | Loss: 1.6700595617294312 | w_1: 0.2758772373199463 | w_2:
1.7733861207962036
Epoch: 57 | Loss: 1.6389776468276978 | w_1: 0.2639479637145996 | w_2:
1.7755051851272583
Epoch: 58 | Loss: 1.6084731817245483 | w_1: 0.2521301805973053 | w_2:
1.777604103088379
Epoch: 59 | Loss: 1.578535795211792 | w_1: 0.24042290449142456 | w_2:
1.7796835899353027
Epoch: 60 | Loss: 1.549156665802002 | w_1: 0.22882501780986786 | w_2:
1.7817434072494507
Epoch: 61 | Loss: 1.520322322845459 | w_1: 0.21733567118644714 | w_2:
1.78378427028656
Epoch: 62 | Loss: 1.492027997970581 | w_1: 0.2059536725282669 | w_2:
1.7858058214187622
Epoch: 63 | Loss: 1.4642534255981445 | w_1: 0.19467811286449432 | w_2:
1.7878084182739258
Epoch: 64 | Loss: 1.4370014667510986 | w_1: 0.1835079789161682 | w_2:
1.7897924184799194
Epoch: 65 | Loss: 1.4102556705474854 | w_1: 0.17244228720664978 | w_2:
1.7917578220367432
Epoch: 66 | Loss: 1.3840065002441406 | w_1: 0.16148003935813904 | w_2:
1.7937047481536865
Epoch: 67 | Loss: 1.358245849609375 | w_1: 0.15062032639980316 | w_2:
1.7956336736679077
Epoch: 68 | Loss: 1.3329664468765259 | w_1: 0.13986217975616455 | w_2:
1.7975444793701172
Epoch: 69 | Loss: 1.3081586360931396 | w_1: 0.12920458614826202 | w_2:
1.7994375228881836
Epoch: 70 | Loss: 1.2838106155395508 | w_1: 0.11864663660526276 | w_2:
1.8013126850128174
Epoch: 71 | Loss: 1.2599154710769653 | w_1: 0.10818740725517273 | w_2:
1.8031704425811768
Epoch: 72 | Loss: 1.2364667654037476 | w_1: 0.09782599657773972 | w_2:
1.8050107955932617
Epoch: 73 | Loss: 1.2134513854980469 | w_1: 0.08756142854690552 | w_2:
1.8068337440490723
Epoch: 74 | Loss: 1.190863847732544 | w_1: 0.07739287614822388 | w_2:
1.8086398839950562
Epoch: 75 | Loss: 1.1687015295028687 | w_1: 0.067319355905056 | w_2:
1.8104290962219238
Epoch: 76 | Loss: 1.1469495296478271 | w_1: 0.05734003335237503 | w_2:
1.8122016191482544
Epoch: 77 | Loss: 1.1256020069122314 | w_1: 0.04745401814579964 | w_2:
1.8139575719833374
Epoch: 78 | Loss: 1.1046502590179443 | w_1: 0.0376603789627552 | w_2:
1.8156967163085938
Epoch: 79 | Loss: 1.0840911865234375 | w_1: 0.02795841544866562 | w_2:
1.817420244216919
```

```
Epoch: 80 | Loss: 1.0639140605926514 | w_1: 0.018347082659602165 | w_2:
1.8191272020339966
Epoch: 81 | Loss: 1.0441125631332397 | w_1: 0.008825712837278843 | w_2:
1.820818543434143
Epoch: 82 | Loss: 1.0246798992156982 | w_1: -0.0006067296490073204 | w_2:
1.8224937915802002
Epoch: 83 | Loss: 1.0056073665618896 | w_1: -0.009950952604413033 | w_2:
1.8241534233093262
Epoch: 84 | Loss: 0.9868906736373901 | w_1: -0.019207805395126343 | w_2:
1.8257975578308105
Epoch: 85 | Loss: 0.9685209393501282 | w_1: -0.028378114104270935 | w_2:
1.8274261951446533
Epoch: 86 | Loss: 0.9504930377006531 | w_1: -0.03746265918016434 | w_2:
1.8290398120880127
Epoch: 87 | Loss: 0.9328028559684753 | w_1: -0.046462297439575195 | w_2:
1.8306381702423096
Epoch: 88 | Loss: 0.9154413342475891 | w_1: -0.05537775158882141 | w_2:
1.8322217464447021
Epoch: 89 | Loss: 0.8984050750732422 | w_1: -0.06420981138944626 | w_2:
1.83379065990448
Epoch: 90 | Loss: 0.8816840052604675 | w_1: -0.07295937091112137 | w_2:
1.8353445529937744
Epoch: 91 | Loss: 0.8652714490890503 | w_1: -0.08162709325551987 | w_2:
1.8368840217590332
Epoch: 92 | Loss: 0.8491684198379517 | w_1: -0.09021376073360443 | w_2:
1.838409185409546
Epoch: 93 | Loss: 0.8333638906478882 | w_1: -0.09872013330459595 | w_2:
1.8399202823638916
Epoch: 94 | Loss: 0.8178545832633972 | w_1: -0.10714709758758545 | w_2:
1.8414167165756226
Epoch: 95 | Loss: 0.802628219127655 | w_1: -0.11549517512321472 | w_2:
1.8428994417190552
Epoch: 96 | Loss: 0.7876907587051392 | w_1: -0.12376518547534943 | w_2:
1.8443684577941895
Epoch: 97 | Loss: 0.773030161857605 | w_1: -0.13195794820785522 | w_2:
1.8458234071731567
Epoch: 98 | Loss: 0.7586407661437988 | w_1: -0.14007404446601868 | w_2:
1.8472650051116943
Epoch: 99 | Loss: 0.7445210218429565 | w_1: -0.14811421930789948 | w_2:
1.8486931324005127
Epoch: 100 | Loss: 0.7306647300720215 | w_1: -0.15607918798923492 | w_2:
1.850108027458191
```

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

[45]:
```
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
```