Task 9:

# Calculate the derivative of y = 2x^3 + x at x = 1

# Type your code here

x = torch.tensor(1.0, requires\_grad=True)

y = 2\*x \*\* 3 + x

y.backward()

print("derivative of x at 1 = ", x.grad)

Task 10:

# Calculate the derivative of f = u \* v + (u \* v) \*\* 2 at u = 2, v = 1

# Type the code here

u = torch.tensor(2.0,requires\_grad=True)

v = torch.tensor(1.0,requires\_grad=True)

f = u\*v + (u\*v) \*\* 2

f.backward()

print("df/du = ", u.grad)

Task 11:

# Create a new object with length 50, and print the length of object out.

# Type your code here

eleven\_dataset = ToySet(50)

print("len(eleven\_dataset): ", len(eleven\_dataset))

Task 13:

# Make a compose as Mult() execute first and then AddMult(). Apply the compose on ToySet dataset. Print out the first 3 elements in the transformed dataset.

# Type your code here.

thirteen\_transform = transforms.Compose([Mult(), AddMult()])

thirteen\_data\_set = ToySet(transform = thirteen\_transform)

for i in range(3):

x, y = thirteen\_data\_set[i]

print('Index: ', i, 'x: ', x, 'y: ', y)

Task 15:

x = torch.tensor([[1.0], [2.0], [3.0]])

def lab1\_task15(x):

# Type your code here to return the prediction

yhat = w \* x + b

return yhat

print("The prediction: ", lab1\_task15(x)

Task18:

def lab1\_task18\_train\_model(iter):

# Type your lab02\_task04\_train\_model function definiition here

for epoch in range (iter):

Yhat = forward\_without\_bias(X)

loss = criterion(Yhat,Y)

LOSS2.append(loss)

loss.backward()

w.data = w.data - learning\_rate \* w.grad.data

w.grad.data.zero\_()

lab1\_task18\_train\_model(4)

Task19:

def lab1\_task19\_train\_model(iter):

for epoch in range(iter):

Yhat = forward(X)

loss = criterion(Yhat, Y)

LOSS2.append(loss)

loss.backward()

w.data = w.data - learning\_rate \* w.grad.data

b.data = b.data - learning\_rate \* b.grad.data

w.grad.data.zero\_()

b.grad.data.zero\_()

lab1\_task19\_train\_model(15)

Task20:

def lab1\_task20\_train\_model(epochs):

# Type your code here

for epoch in range(epochs):

Yhat = forward(X)

LOSS.append(criterion(Yhat, Y).tolist())

for x, y in trainloader:

yhat = forward(x)

loss = criterion(yhat, y)

loss.backward()

w.data = w.data - learning\_rate \* w.grad.data

b.data = b.data - learning\_rate \* b.grad.data

w.grad.data.zero\_()

b.grad.data.zero\_()

lab1\_task20\_train\_model(10)

Task21: (*Only the hilighted is missing from the answer of some students*)

# Task 21

my\_dataset = Data()

LOSS20 = []

w = torch.tensor(-15.0, requires\_grad = True)

b = torch.tensor(-10.0, requires\_grad = True)

learning\_rate = 0.1

# Type your code here to define the trainloader

**trainloader = DataLoader(dataset = my\_dataset, batch\_size = 20)**

def lab1\_task21\_train\_model(epochs):

# Type your code here

for epoch in range(epochs):

Yhat = forward(X)

error\_surfaces.set\_para\_loss(w.data.tolist(), b.data.tolist(), criterion(Yhat, Y).tolist())

error\_surfaces.plot\_ps()

LOSS20.append(criterion(forward(X),Y).tolist())

for x, y in trainloader:

yhat = forward(x)

loss = criterion(yhat, y)

error\_surfaces.set\_para\_loss(w.data.tolist(), b.data.tolist(), loss.tolist())

loss.backward()

w.data = w.data - learning\_rate \* w.grad.data

b.data = b.data - learning\_rate \* b.grad.data

w.grad.data.zero\_()

b.grad.data.zero\_()

lab1\_task21\_train\_model(10)