Week 3 Questions (220962018)

January 25, 2025

0.1 Week 3 Lab Exercise

[67]:
```
import torch
from matplotlib import pyplot as plt
from torch import nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np
device = "cuda" if torch.cuda.is_available() else "cpu"
torch.cuda.device_count() , device
```

[67]:
```
(1, 'cuda')
```

0.2 Lab Exercise

0.2.1 1) For the following training data, build a linear regression model. Assume w and b are initialized with 1 and learning parameter is set to 0.001.

x = torch.tensor([12.4, 14.3, 14.5, 14.9, 16.1, 16.9, 16.5, 15.4, 17.0, 17.9, 18.8, 20.3, 22.4, 19.4, 15.5, 16.7, 17.3, 18.4, 19.2, 17.4, 19.5, 19.7, 21.2])

y = torch.tensor([11.2, 12.5, 12.7, 13.1, 14.1, 14.8, 14.4, 13.4, 14.9, 15.6, 16.4, 17.7, 19.6, 16.9, 14.0, 14.6, 15.1, 16.1, 16.8, 15.2, 17.0, 17.2, 18.6])

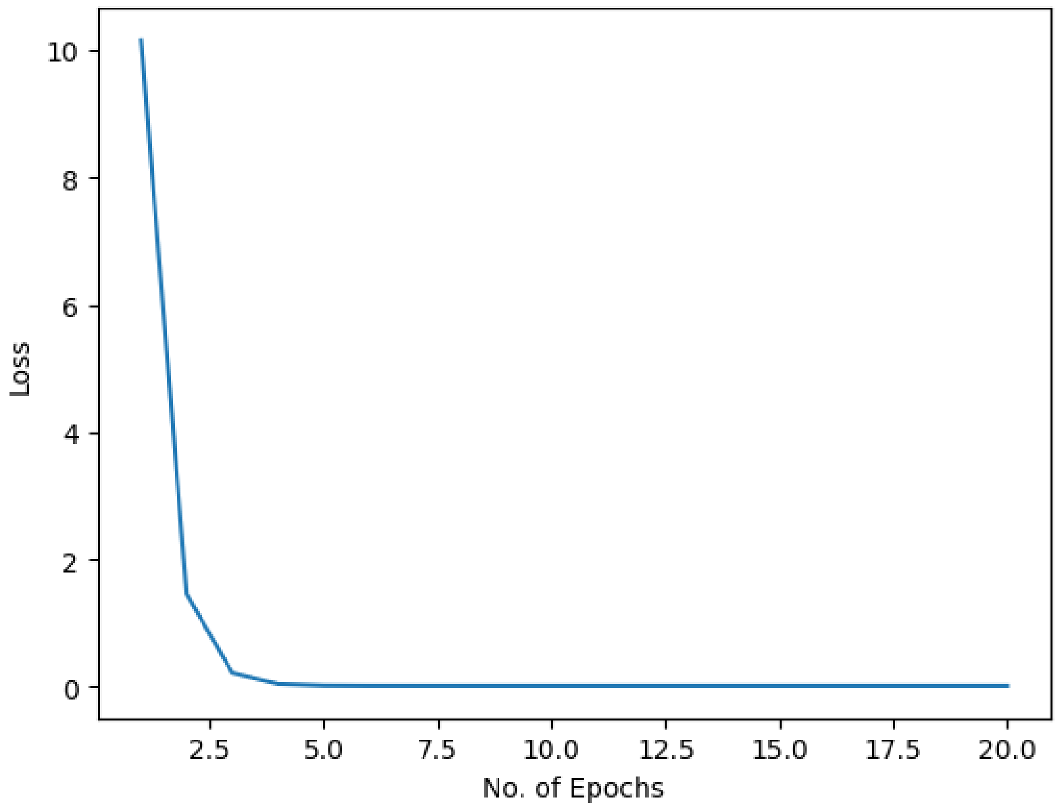
Assume learning rate =0.001. Plot the graph of epoch in x axis and loss in y axis.

[68]:
```
x=torch.tensor([12.4, 14.3, 14.5, 14.9, 16.1, 16.9, 16.5, 15.4, 17.0, 17.9, 18.
 ↪8, 20.3, 22.4,
19.4, 15.5, 16.7, 17.3, 18.4, 19.2, 17.4, 19.5, 19.7, 21.2])
y=torch.tensor([11.2, 12.5, 12.7, 13.1, 14.1, 14.8, 14.4, 13.4, 14.9, 15.6, 16.
 ↪4, 17.7, 19.6,
16.9, 14.0, 14.6, 15.1, 16.1, 16.8, 15.2, 17.0, 17.2, 18.6])
```

[69]:
```
def mse(y, y_pred):
    return torch.mean((y - y_pred)**2)

lr = 0.001
epochs = 20
```

```python
w, b = torch.tensor(1., requires_grad=True), torch.tensor(1.,␣
↪requires_grad=True)
losses = []

for i in range(epochs):
    y_pred = w * x + b

    loss = mse(y, y_pred)

    loss.backward()

    with torch.no_grad():
        w -= lr * w.grad
        b -= lr * b.grad
        losses.append(loss.detach())
        print(loss)
    w.grad.zero_()
    b.grad.zero_()

plt.plot(list(range(1,epochs+1)), losses)
plt.xlabel('No. of Epochs')
plt.ylabel('Loss')
plt.show()
```

```
tensor(10.1539, grad_fn=<MeanBackward0>)
tensor(1.4567, grad_fn=<MeanBackward0>)
tensor(0.2217, grad_fn=<MeanBackward0>)
tensor(0.0464, grad_fn=<MeanBackward0>)
tensor(0.0215, grad_fn=<MeanBackward0>)
tensor(0.0179, grad_fn=<MeanBackward0>)
tensor(0.0174, grad_fn=<MeanBackward0>)
tensor(0.0174, grad_fn=<MeanBackward0>)
tensor(0.0173, grad_fn=<MeanBackward0>)
tensor(0.0173, grad_fn=<MeanBackward0>)
tensor(0.0173, grad_fn=<MeanBackward0>)
tensor(0.0173, grad_fn=<MeanBackward0>)
tensor(0.0173, grad_fn=<MeanBackward0>)
tensor(0.0173, grad_fn=<MeanBackward0>)
tensor(0.0173, grad_fn=<MeanBackward0>)
tensor(0.0173, grad_fn=<MeanBackward0>)
tensor(0.0173, grad_fn=<MeanBackward0>)
tensor(0.0173, grad_fn=<MeanBackward0>)
tensor(0.0173, grad_fn=<MeanBackward0>)
tensor(0.0173, grad_fn=<MeanBackward0>)
```



0.2.2 2) Find the value of w.grad, b.grad using analytical solution for the given linear regression problem. Initial value of w = b =1. Learning parameter is set to 0.001. Implement the same and verify the values of w.grad , b.grad and updated parameter values for two epochs. Consider the difference between predicted and target values of y is defined as (yp-y).

[70]:
```
lr = 0.001
epochs = 2
w, b = torch.tensor(1., requires_grad=True), torch.tensor(1.,␣
 ↪requires_grad=True)

x = torch.tensor([2, 4])
y = torch.tensor([20, 40])

for i in range(epochs):
    print(w, b)
    y_pred = w * x + b

    loss = mse(y, y_pred)

    loss.backward()
```

```
    print(f"Epoch {i+1}: w_grad={w.grad.item()}, b_grad={b.grad.item()}")

    with torch.no_grad():
        w -= lr * w.grad
        b -= lr * b.grad

    w.grad.zero_()
    b.grad.zero_()

    print(f"Epoch {i+1}: Loss={loss.item()}\n")

w, b
```

```
tensor(1., requires_grad=True) tensor(1., requires_grad=True)
Epoch 1: w_grad=-174.0, b_grad=-52.0
Epoch 1: Loss=757.0

tensor(1.1740, requires_grad=True) tensor(1.0520, requires_grad=True)
Epoch 2: w_grad=-170.20799255371094, b_grad=-50.85199737548828
Epoch 2: Loss=724.3797607421875
```

[70]:
```
(tensor(1.3442, requires_grad=True), tensor(1.1029, requires_grad=True))
```

0.2.3 3) Revise the linear regression model by defining a user defined class titled RegressionModel with two parameters w and b as its member variables. Define a constructor to initialize w and b with value 1. Define four member functions namely forward(x) to implement wx+b, update() to update w and b values, reset_grad() to reset parameters to zero, criterion(y, yp) to implement MSE Loss given the predicted y value yp and the target label y. Define an object of this class named model and invoke all the methods. Plot the graph of epoch vs loss by varying epoch to 100 iterations.

x = torch.tensor([5.0, 7.0, 12.0, 16.0, 20.0])

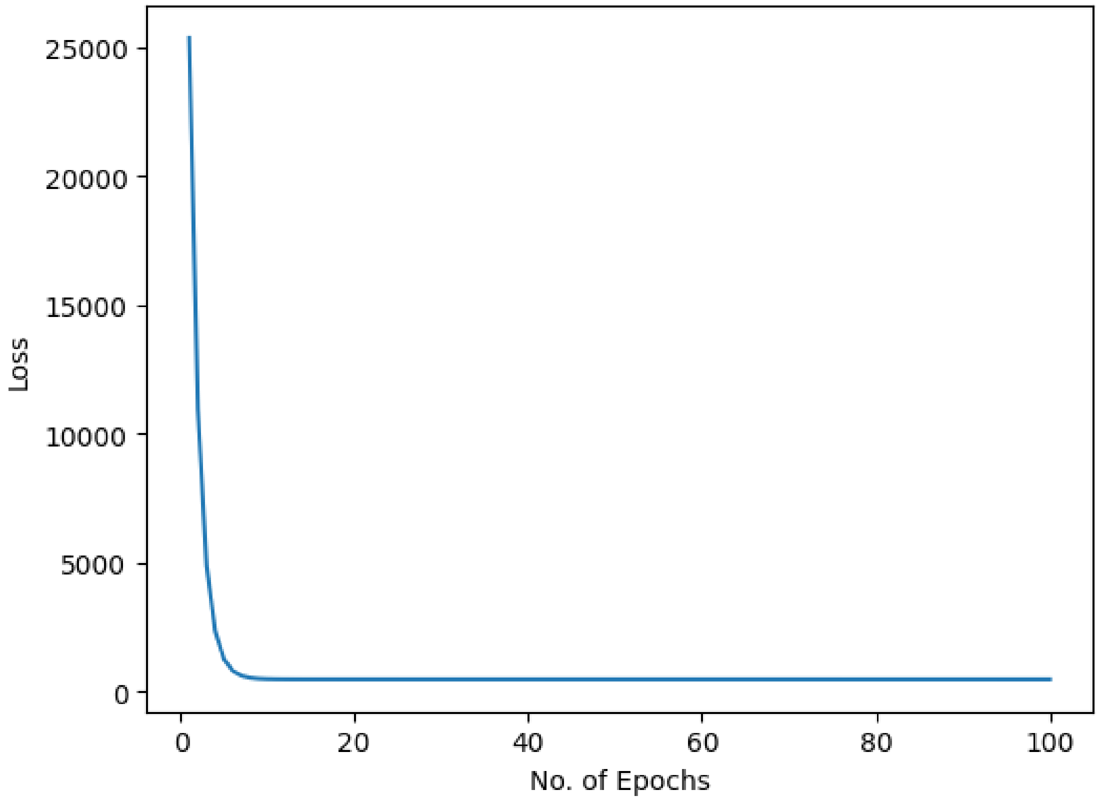
y = torch.tensor([40.0, 120.0, 180.0, 210.0, 240.0])

learning_rate = torch.tensor(0.001)

[71]:
```
class RegressionModel:
    def __init__(self, w, b, lr=0.001):
        self.w = torch.tensor(w, requires_grad=True)
        self.b = torch.tensor(b, requires_grad=True)
        self.lr = lr
        self.y = None
        self.loss = None
        self.losses = []
```

```python
    def forward(self, x):
        self.y = self.w * x + self.b
        return self.y

    def update(self):
        self.loss.backward()
        with torch.no_grad():
            self.w -= self.lr * self.w.grad
            self.b -= self.lr * self.b.grad

    def reset_grad(self):
        self.w.grad.zero_()
        self.b.grad.zero_()

    def criterion(self, y, yp):
        self.loss = torch.mean((y - y_pred)**2)
        self.losses.append(self.loss.detach())
        return self.loss
```

[72]:
```python
model = RegressionModel(1., 1.)
epochs = 100

x = torch.tensor([5.0, 7.0, 12.0, 16.0, 20.0])
y = torch.tensor([40.0, 120.0, 180.0, 210.0, 240.0])

for i in range(epochs):
    y_pred = model.forward(x)
    loss = model.criterion(y, y_pred)
    model.update()
    model.reset_grad()

plt.plot(np.arange(1, epochs+1), model.losses)
plt.xlabel('No. of Epochs')
plt.ylabel('Loss')
plt.show()
```



0.2.4 4) Convert your program written in Qn 3 to extend nn.module in your model. Also override the necessary methods to fit the regression line. Illustrate the use of Dataset and DataLoader from torch.utils.data in your implementation. Use the SGD Optimizer torch.optim.SGD()

[73]:
```
def criterion(yp, y):
    return torch.mean((y - y_pred)**2)

x = torch.tensor([5.0, 7.0, 12.0, 16.0, 20.0])
y = torch.tensor([40.0, 120.0, 180.0, 210.0, 240.0])

w = 1.
b = 1.

# Define the RegressionModel class inheriting from nn.Module
class RegressionModel(nn.Module):
    def __init__(self, w, b):
        super(RegressionModel, self).__init__()
        self.w = nn.Parameter(torch.tensor(w, requires_grad=True, dtype=torch.
↪float))
```

```python
        self.b = nn.Parameter(torch.tensor(b, requires_grad=True, dtype=torch.
↪float))

    def forward(self, x):
        return self.w * x + self.b

# Custom Dataset to handle input and output data
class LinearDataset(Dataset):
    def __init__(self, x_data, y_data):
        self.x_data = x_data
        self.y_data = y_data

    def __len__(self):
        return len(self.x_data)

    def __getitem__(self, idx):
        return self.x_data[idx], self.y_data[idx]

# Create the dataset and data loader
dataset = LinearDataset(x, y)
dataloader = DataLoader(dataset, batch_size=16, shuffle=False)

# Initialize the model, loss function, and optimizer
model = RegressionModel(w, b)
#criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=1e-4)

# Training loop
num_epochs = 100
for epoch in range(num_epochs):
    for batch_x, batch_y in dataloader:
        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass
        y_pred = model(batch_x)

        # Compute loss
        loss = criterion(y_pred, batch_y)

        # Backward pass
        loss.backward()

        # Update weights
        optimizer.step()

    if (epoch + 1) % 10 == 0:
```

```
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

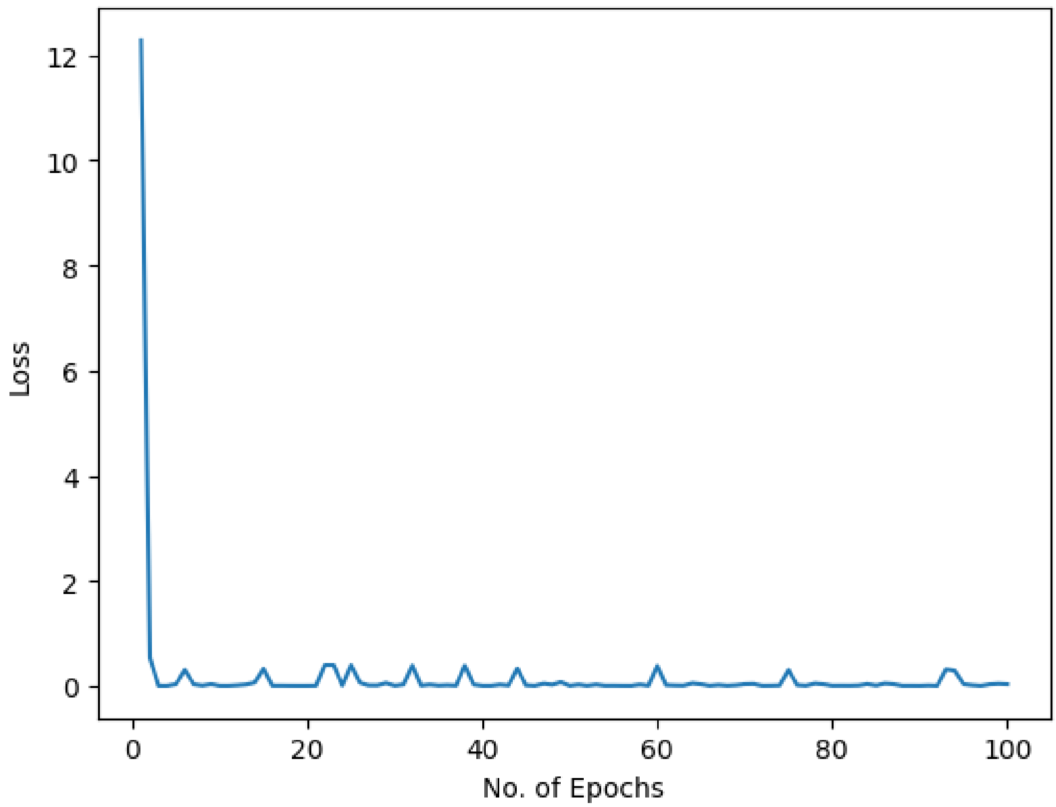
```
Epoch [10/100], Loss: 13552.6689
Epoch [20/100], Loss: 6875.4150
Epoch [30/100], Loss: 3609.3821
Epoch [40/100], Loss: 2011.8713
Epoch [50/100], Loss: 1230.4796
Epoch [60/100], Loss: 848.2745
Epoch [70/100], Loss: 661.3213
Epoch [80/100], Loss: 569.8713
Epoch [90/100], Loss: 525.1349
Epoch [100/100], Loss: 503.2474
```

0.2.5 5) Use PyTorch's nn.Linear() in your implementation to perform linear regression for the data provided in Qn. 1. Also plot the graph.

[74]:
```
x = torch.tensor( [12.4, 14.3, 14.5, 14.9, 16.1, 16.9, 16.5, 15.4, 17.0, 17.9,␣
 ↪18.8, 20.3, 22.4,
19.4, 15.5, 16.7, 17.3, 18.4, 19.2, 17.4, 19.5, 19.7, 21.2])

y = torch.tensor( [11.2, 12.5, 12.7, 13.1, 14.1, 14.8, 14.4, 13.4, 14.9, 15.6,␣
 ↪16.4, 17.7, 19.6,
16.9, 14.0, 14.6, 15.1, 16.1, 16.8, 15.2, 17.0, 17.2, 18.6])

losses = []

# Create the dataset and data loader
dataset = LinearDataset(x, y)
dataloader = DataLoader(dataset, batch_size=1, shuffle=True)

# Initialize the model, loss function, and optimizer
model = nn.Linear(1,1)
#criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.0001)

# Training loop
num_epochs = 100
for epoch in range(num_epochs):
    for batch_x, batch_y in dataloader:
        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass
        y_pred = model(batch_x)

        # Compute loss
        loss = criterion(y_pred, batch_y)
```

```python
        # Backward pass
        loss.backward()

        # Update weights
        optimizer.step()

    losses.append(loss.item())

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

plt.plot(np.arange(1, num_epochs + 1), losses)
plt.xlabel('No. of Epochs')
plt.ylabel('Loss')
plt.show()
```

```
Epoch [10/100], Loss: 0.0006
Epoch [20/100], Loss: 0.0000
Epoch [30/100], Loss: 0.0025
Epoch [40/100], Loss: 0.0008
Epoch [50/100], Loss: 0.0007
Epoch [60/100], Loss: 0.3828
Epoch [70/100], Loss: 0.0341
Epoch [80/100], Loss: 0.0024
Epoch [90/100], Loss: 0.0002
Epoch [100/100], Loss: 0.0321
```



0.2.6 6) Implement multiple linear regression for the data provided below

Subject	X1	X2	Y
1	3	8	-3.7
2	4	5	3.5
3	5	7	2.5
4	6	3	11.5
5	2	1	5.7

Verify your answer for the data point X1=3, X2=2.

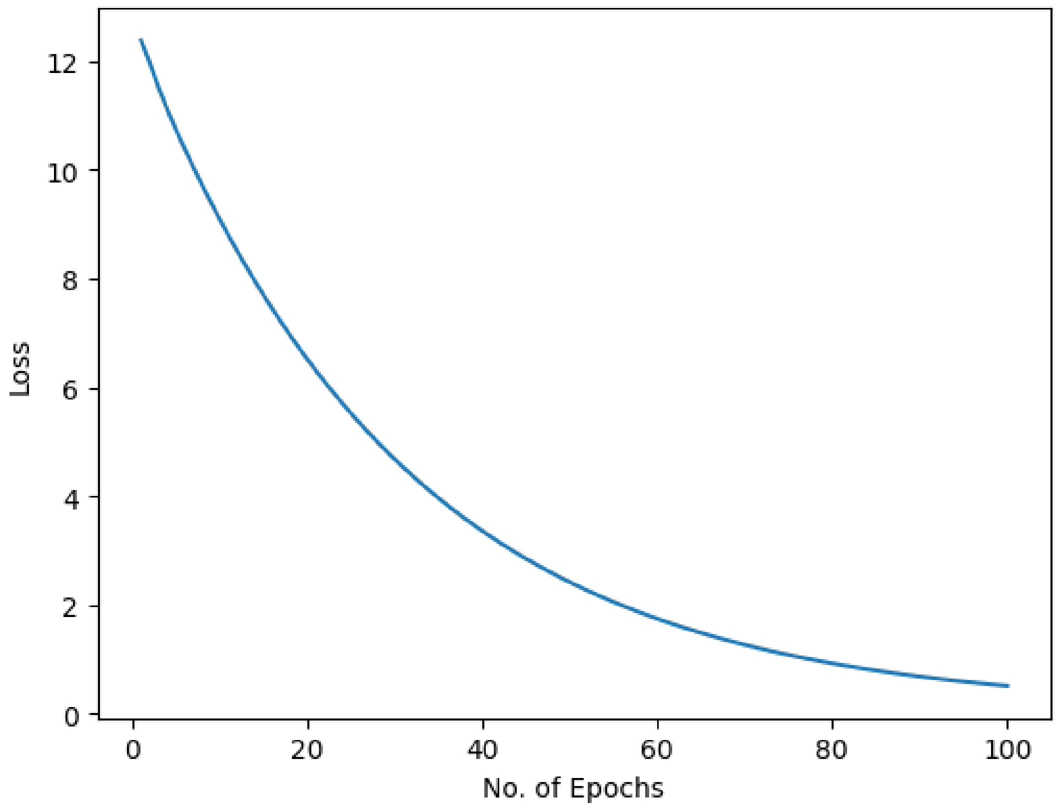
[75]:
```
x1 = torch.tensor([3, 4, 5, 6, 2])
x2 = torch.tensor([8, 5, 7, 3, 1])
x = torch.stack((x1, x2), dim=1)
y = torch.tensor([-3.7, 3.5, 2.5, 11.5, 5.7])

losses = []

epochs = 20
w, b = [1., 1.], 1.
```

```python
# Create the dataset and data loader
dataset = LinearDataset(x, y)
dataloader = DataLoader(dataset, batch_size=1, shuffle=False)

# Initialize the model, loss function, and optimizer
model = RegressionModel(w, b)
#criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=1e-2)

# Training loop
num_epochs = 100
for epoch in range(num_epochs):
    for batch_x, batch_y in dataloader:
        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass
        y_pred = model(batch_x)

        # Compute loss
        loss = criterion(y_pred, batch_y)

        # Backward pass
        loss.backward()

        # Update weights
        optimizer.step()

    losses.append(loss.detach())
    if (epoch + 1) % 1 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

plt.plot(np.arange(1, num_epochs+1), losses)
plt.xlabel('No. of Epochs')
plt.ylabel('Loss')
plt.show()
```

```
Epoch [1/100], Loss: 12.3834
Epoch [2/100], Loss: 11.9640
Epoch [3/100], Loss: 11.5118
Epoch [4/100], Loss: 11.1089
Epoch [5/100], Loss: 10.7362
Epoch [6/100], Loss: 10.3819
Epoch [7/100], Loss: 10.0413
Epoch [8/100], Loss: 9.7126
Epoch [9/100], Loss: 9.3948
Epoch [10/100], Loss: 9.0875
```

```
Epoch [11/100], Loss: 8.7903
Epoch [12/100], Loss: 8.5028
Epoch [13/100], Loss: 8.2247
Epoch [14/100], Loss: 7.9558
Epoch [15/100], Loss: 7.6957
Epoch [16/100], Loss: 7.4440
Epoch [17/100], Loss: 7.2007
Epoch [18/100], Loss: 6.9653
Epoch [19/100], Loss: 6.7377
Epoch [20/100], Loss: 6.5175
Epoch [21/100], Loss: 6.3045
Epoch [22/100], Loss: 6.0985
Epoch [23/100], Loss: 5.8993
Epoch [24/100], Loss: 5.7067
Epoch [25/100], Loss: 5.5203
Epoch [26/100], Loss: 5.3401
Epoch [27/100], Loss: 5.1658
Epoch [28/100], Loss: 4.9972
Epoch [29/100], Loss: 4.8342
Epoch [30/100], Loss: 4.6766
Epoch [31/100], Loss: 4.5241
Epoch [32/100], Loss: 4.3766
Epoch [33/100], Loss: 4.2340
Epoch [34/100], Loss: 4.0961
Epoch [35/100], Loss: 3.9627
Epoch [36/100], Loss: 3.8338
Epoch [37/100], Loss: 3.7090
Epoch [38/100], Loss: 3.5884
Epoch [39/100], Loss: 3.4718
Epoch [40/100], Loss: 3.3590
Epoch [41/100], Loss: 3.2500
Epoch [42/100], Loss: 3.1445
Epoch [43/100], Loss: 3.0425
Epoch [44/100], Loss: 2.9439
Epoch [45/100], Loss: 2.8485
Epoch [46/100], Loss: 2.7563
Epoch [47/100], Loss: 2.6672
Epoch [48/100], Loss: 2.5810
Epoch [49/100], Loss: 2.4976
Epoch [50/100], Loss: 2.4170
Epoch [51/100], Loss: 2.3391
Epoch [52/100], Loss: 2.2638
Epoch [53/100], Loss: 2.1909
Epoch [54/100], Loss: 2.1205
Epoch [55/100], Loss: 2.0524
Epoch [56/100], Loss: 1.9866
Epoch [57/100], Loss: 1.9229
Epoch [58/100], Loss: 1.8614
```

```
Epoch [59/100], Loss: 1.8019
Epoch [60/100], Loss: 1.7444
Epoch [61/100], Loss: 1.6888
Epoch [62/100], Loss: 1.6350
Epoch [63/100], Loss: 1.5831
Epoch [64/100], Loss: 1.5329
Epoch [65/100], Loss: 1.4843
Epoch [66/100], Loss: 1.4374
Epoch [67/100], Loss: 1.3920
Epoch [68/100], Loss: 1.3481
Epoch [69/100], Loss: 1.3057
Epoch [70/100], Loss: 1.2648
Epoch [71/100], Loss: 1.2252
Epoch [72/100], Loss: 1.1869
Epoch [73/100], Loss: 1.1499
Epoch [74/100], Loss: 1.1141
Epoch [75/100], Loss: 1.0796
Epoch [76/100], Loss: 1.0462
Epoch [77/100], Loss: 1.0139
Epoch [78/100], Loss: 0.9827
Epoch [79/100], Loss: 0.9525
Epoch [80/100], Loss: 0.9234
Epoch [81/100], Loss: 0.8952
Epoch [82/100], Loss: 0.8680
Epoch [83/100], Loss: 0.8417
Epoch [84/100], Loss: 0.8163
Epoch [85/100], Loss: 0.7918
Epoch [86/100], Loss: 0.7681
Epoch [87/100], Loss: 0.7451
Epoch [88/100], Loss: 0.7230
Epoch [89/100], Loss: 0.7016
Epoch [90/100], Loss: 0.6809
Epoch [91/100], Loss: 0.6610
Epoch [92/100], Loss: 0.6417
Epoch [93/100], Loss: 0.6230
Epoch [94/100], Loss: 0.6050
Epoch [95/100], Loss: 0.5876
Epoch [96/100], Loss: 0.5708
Epoch [97/100], Loss: 0.5546
Epoch [98/100], Loss: 0.5389
Epoch [99/100], Loss: 0.5238
Epoch [100/100], Loss: 0.5092
```



0.2.7 7) Implement logistic regression

x = [1, 5, 10, 10, 25, 50, 70, 75, 100,]

y = [0, 0, 0, 0, 0, 1, 1, 1, 1]

[76]:
```
class LogisticModel(nn.Module):
    def __init__(self, w, b):
        super(LogisticModel, self).__init__()
        self.w = nn.Parameter(torch.tensor(w, requires_grad=True, dtype=torch.
 ↪float))
        self.b = nn.Parameter(torch.tensor(b, requires_grad=True, dtype=torch.
 ↪float))

    def forward(self, x):
        z = self.w * x + self.b
        #print("stuff:", self.w, x, self.b)
        return torch.sigmoid(z)    # 1 / (1 + torch.exp(-z))

x = torch.tensor([1, 5, 10, 10, 25, 50, 70, 75, 100], dtype=torch.float)
y = torch.tensor([0, 0, 0, 0, 0, 1, 1, 1, 1], dtype=torch.float)
```

```python
losses = []

epochs = 20
w, b = 1., 1.

# Create the dataset and data loader
dataset = LinearDataset(x, y)
dataloader = DataLoader(dataset, batch_size=1, shuffle=False)

# Initialize the model, loss function, and optimizer
model = LogisticModel(w, b)
criterion = nn.BCELoss()
optimizer = optim.SGD(model.parameters(), lr=0.005)

# Training loop
num_epochs = 200
for epoch in range(num_epochs):
    epoch_loss = []
    for batch_x, batch_y in dataloader:
        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass
        y_pred = model(batch_x)

        # Compute loss
        loss = criterion(y_pred, batch_y)

        # Backward pass
        loss.backward()
        epoch_loss.append(loss.detach())

        # Update weights
        optimizer.step()

    losses.append(sum(epoch_loss) / len(epoch_loss))
    if (epoch+1) % 10 == 0:
        print(f"Loss for Epoch {epoch+1}/{num_epochs}: {losses[-1]}")

plt.plot(np.arange(1, num_epochs+1), losses)
plt.xlabel('No. of Epochs')
plt.ylabel('Loss')
plt.show()
```

```
Loss for Epoch 10/200: 1.3185741901397705
Loss for Epoch 20/200: 1.2531628608703613
Loss for Epoch 30/200: 1.1903891563415527
```

```
Loss for Epoch 40/200: 1.1303671598434448
Loss for Epoch 50/200: 1.0731796026229858
Loss for Epoch 60/200: 1.0188769102096558
Loss for Epoch 70/200: 0.9674785137176514
Loss for Epoch 80/200: 0.9189727306365967
Loss for Epoch 90/200: 0.8733193278312683
Loss for Epoch 100/200: 0.8304525017738342
Loss for Epoch 110/200: 0.7902848124504089
Loss for Epoch 120/200: 0.7527111768722534
Loss for Epoch 130/200: 0.7176131010055542
Loss for Epoch 140/200: 0.6848623752593994
Loss for Epoch 150/200: 0.6543242931365967
Loss for Epoch 160/200: 0.6258624792098999
Loss for Epoch 170/200: 0.5993388295173645
Loss for Epoch 180/200: 0.5746187567710876
Loss for Epoch 190/200: 0.5515710711479187
Loss for Epoch 200/200: 0.5300692915916443
```