In [1]:
```python
import torch
```

In [2]:
```python
torch.__version__
```

Out[2]:
```
'2.2.2'
```

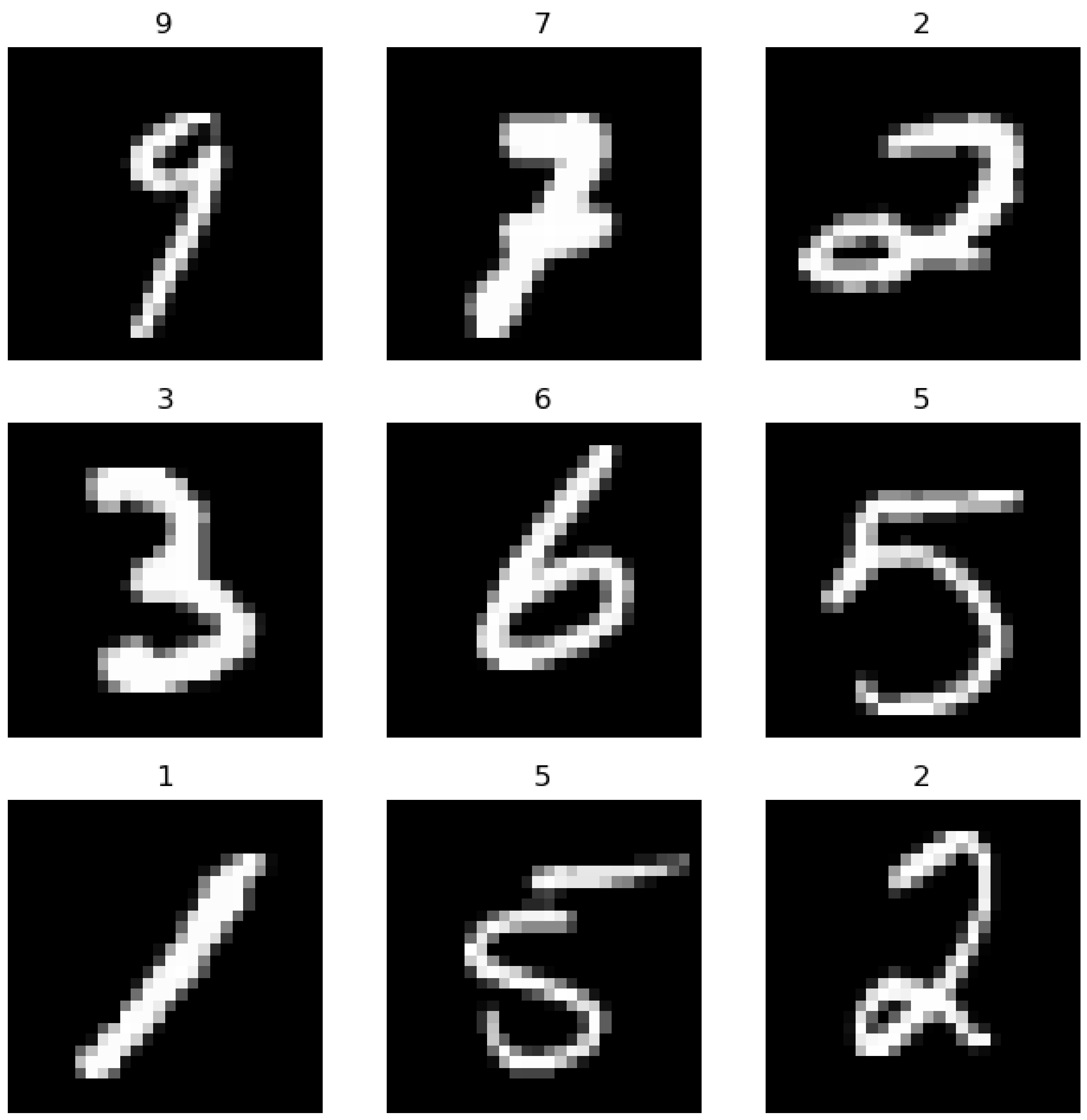
Following Tutorial

In [3]:
```python
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
```

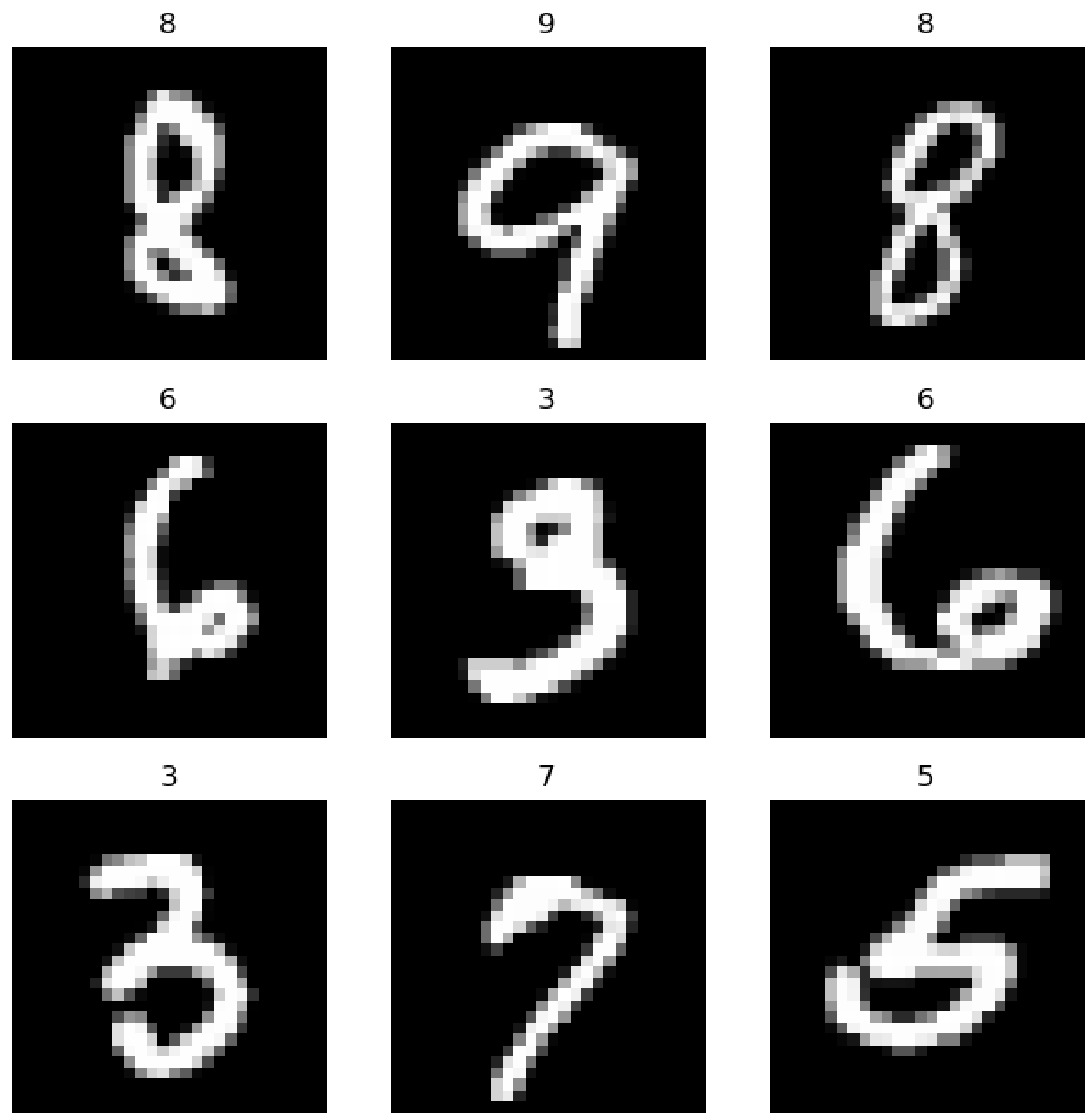
In [4]:
```python
# Load QMNIST dataset
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5
train_dataset = torchvision.datasets.QMNIST(root='./data', train=True, download='True',
test_dataset = torchvision.datasets.QMNIST(root='./data', train=False, download='True',

# Create Data Loaders
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

Visualize the Dataset

In [5]:
```python
labels_map = {
    0: "0",
    1: "1",
    2: "2",
    3: "3",
    4: "4",
    5: "5",
    6: "6",
    7: "7",
    8: "8",
    9: "9",
}

figure = plt.figure(figsize=(8,8))
cols, rows = 3, 3
for i in range(1,cols * rows + 1):
    sample_idx = torch.randint(len(train_dataset), size=(1,)).item()
    img, label = train_dataset[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(labels_map[label])
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap='gray')
plt.show()
```



```
In [6]: # test Dataset

        figure = plt.figure(figsize=(8,8))
        cols, rows = 3, 3

        for i in range(1, cols * rows + 1):
            sample_idx = torch.randint(len(test_dataset), size=(1,)).item()
            img, label = test_dataset[sample_idx]
            figure.add_subplot(rows, cols, i)
            plt.title(labels_map[label])
            plt.axis("off")
            plt.imshow(img.squeeze(), cmap='gray')
        plt.show()
```



Modeling NN

In [7]:

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

model = MLP()
```

In [8]:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```python
# Training the model
num_epochs = 5
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outpus = model(inputs)
        loss = criterion(outpus, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

        if i % 100 == 99:
            print(f'Epoch {epoch+1}, Batch {i+1}, Loss: {running_loss / 100}')
            running_loss = 0.0
            correct = 0
            total = 0

print('Finished Training')
```

```
Epoch 1, Batch 100, Loss: 1.0556920623779298
Epoch 1, Batch 200, Loss: 0.44912069872021676
Epoch 1, Batch 300, Loss: 0.3752118667960167
Epoch 1, Batch 400, Loss: 0.35129957914352417
Epoch 1, Batch 500, Loss: 0.3467361940443516
Epoch 1, Batch 600, Loss: 0.3209052826464176
Epoch 1, Batch 700, Loss: 0.2789001079648733
Epoch 1, Batch 800, Loss: 0.2676535963267088
Epoch 1, Batch 900, Loss: 0.25787740416824817
Epoch 2, Batch 100, Loss: 0.22970901150256395
Epoch 2, Batch 200, Loss: 0.21640905916690825
Epoch 2, Batch 300, Loss: 0.20729080080986023
Epoch 2, Batch 400, Loss: 0.20601989228278397
Epoch 2, Batch 500, Loss: 0.1918893662840128
Epoch 2, Batch 600, Loss: 0.19008899740874768
Epoch 2, Batch 700, Loss: 0.1624915297329426
Epoch 2, Batch 800, Loss: 0.17488610912114383
Epoch 2, Batch 900, Loss: 0.1631122048571706
Epoch 3, Batch 100, Loss: 0.1517372542992234
Epoch 3, Batch 200, Loss: 0.13904143542051314
Epoch 3, Batch 300, Loss: 0.14733837608247996
Epoch 3, Batch 400, Loss: 0.13561398176476358
Epoch 3, Batch 500, Loss: 0.14350079905241728
Epoch 3, Batch 600, Loss: 0.1395022723451257
Epoch 3, Batch 700, Loss: 0.14385301157832145
Epoch 3, Batch 800, Loss: 0.12824302963912487
Epoch 3, Batch 900, Loss: 0.13446660097688437
Epoch 4, Batch 100, Loss: 0.11634353306144476
Epoch 4, Batch 200, Loss: 0.11518218833021819
Epoch 4, Batch 300, Loss: 0.10160702900961041
Epoch 4, Batch 400, Loss: 0.11456665815785527
Epoch 4, Batch 500, Loss: 0.1256417452543974
Epoch 4, Batch 600, Loss: 0.11089270087890327
Epoch 4, Batch 700, Loss: 0.11922134518623352
Epoch 4, Batch 800, Loss: 0.1038310061767697 3
Epoch 4, Batch 900, Loss: 0.12258442741818726
Epoch 5, Batch 100, Loss: 0.10172791701741517
Epoch 5, Batch 200, Loss: 0.09544037004932761
Epoch 5, Batch 300, Loss: 0.09939402987249196
Epoch 5, Batch 400, Loss: 0.09724754191935063
Epoch 5, Batch 500, Loss: 0.09840652483515441
Epoch 5, Batch 600, Loss: 0.10456238383427262
Epoch 5, Batch 700, Loss: 0.0940044691041112
Epoch 5, Batch 800, Loss: 0.0917795588960871
Epoch 5, Batch 900, Loss: 0.09578550558770076
Finished Training
```

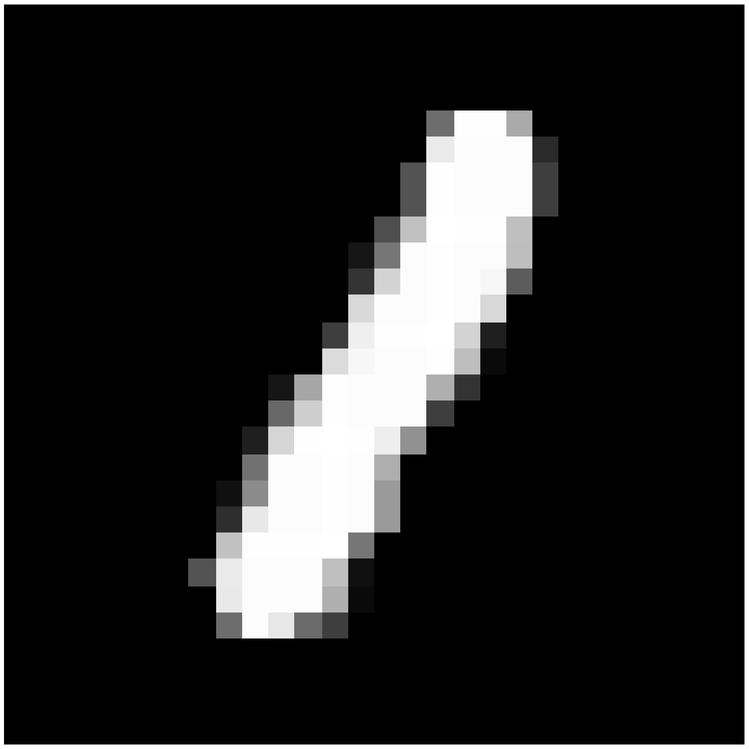
In [9]:
```
# criterion = nn.CrossEntropyLoss()
# optimizer = optim.Adam(model.parameters(), lr=0.001)

# # Training the model
# num_epochs = 5
# for epoch in range(num_epochs):
#     model.train()
#     running_loss = 0.0
#     correct = 0
#     total = 0
#     for i, data in enumerate(train_loader, 0):
#         inputs, labels = data

#         # Forward pass
#         optimizer.zero_grad()
#         outputs = model(inputs)
#         loss = criterion(outputs, labels)

#         # Backward pass and optimization
#         loss.backward()
#         optimizer.step()

#         # Debug: print shape of outputs and labels
#         # print(f"Output shape: {outputs.shape}, Labels shape: {labels.shape}")

#         # Accuracy calculation
#         _, predicted = torch.max(outputs, 1)  # Get the index of the max log-probabili
#         # print(f"Predicted shape: {predicted.shape}")  # Print predicted shape for de

#         # Ensure total and correct are calculated only if shapes match
#         if predicted.shape == labels.shape:
#             total += labels.size(0)
#             correct += (predicted == labels).sum().item()

#         # Print statistics every 100 batches
#         if i % 100 == 99:
#             accuracy = 100 * correct / total  # Calculate accuracy percentage
#             print(f'Epoch {epoch+1}, Batch {i+1}, Loss: {running_loss / 100}, Accuracy
#             running_loss = 0.0
#             correct = 0  # Reset correct for the next 100 batches
#             total = 0    # Reset total for the next 100 batches

# print('Finished Training')
```

In [10]:
```
# Evaluate the model and store prediction
model.eval()
predictions = []
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        predictions.extend(predicted.numpy())
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
acc1 = correct / total
print(f"Accuracy on test set: {correct / total *100}%")
```

```
Accuracy on test set: 94.85666666666667%
```

In [11]:
```
image, label = images[1], labels[1]

image = image.view(28, 28)
image_numpy = image.numpy()

plt.imshow(image_numpy, cmap='gray')
plt.title(f"Predicted Label: {predictions[1]}, Acutal Label: {label.item()}")
plt.axis("off")
plt.show()
```

Predicted Label: 2, Acutal Label: 1



Modify the Model

In [12]:
```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)
        return x

model = MLP()
```

Hypothesis:

I have added one more Dense layer of 128 nodes. This would increase the accuracy, but since the accuracy of the original model is already high, the difference would not be very significant.

Train and evaluate the modified model

In [13]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training the model
num_epochs = 5
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outpus = model(inputs)
        loss = criterion(outpus, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

        if i % 100 == 99:
            print(f'Epoch {epoch+1}, Batch {i+1}, Loss: {running_loss / 100}')
            running_loss = 0.0
            correct = 0
            total = 0

print('Finished Training')
```

```
Epoch 1, Batch 100, Loss: 1.1024345606565475
Epoch 1, Batch 200, Loss: 0.43937609866261484
Epoch 1, Batch 300, Loss: 0.4005265799164772
Epoch 1, Batch 400, Loss: 0.34075415790081026
Epoch 1, Batch 500, Loss: 0.30599621146917344
Epoch 1, Batch 600, Loss: 0.2688695778697729
Epoch 1, Batch 700, Loss: 0.2541291113942862
Epoch 1, Batch 800, Loss: 0.24743629522621632
Epoch 1, Batch 900, Loss: 0.22866396464407443
Epoch 2, Batch 100, Loss: 0.20594951190054417
Epoch 2, Batch 200, Loss: 0.18827336724847554
Epoch 2, Batch 300, Loss: 0.18626722056418657
Epoch 2, Batch 400, Loss: 0.173455670773983
Epoch 2, Batch 500, Loss: 0.1804219925031066
Epoch 2, Batch 600, Loss: 0.16080134995281697
Epoch 2, Batch 700, Loss: 0.1620458686724305
Epoch 2, Batch 800, Loss: 0.1562608550861478
Epoch 2, Batch 900, Loss: 0.1522590560466051
Epoch 3, Batch 100, Loss: 0.13522378703579307
Epoch 3, Batch 200, Loss: 0.1438153250142932
Epoch 3, Batch 300, Loss: 0.12648076373152434
Epoch 3, Batch 400, Loss: 0.13238370528444648
Epoch 3, Batch 500, Loss: 0.13869948080740868
Epoch 3, Batch 600, Loss: 0.11550110856071115
Epoch 3, Batch 700, Loss: 0.12345703564584255
Epoch 3, Batch 800, Loss: 0.11862700291909277
Epoch 3, Batch 900, Loss: 0.1169899763353169
Epoch 4, Batch 100, Loss: 0.09024582475423813
Epoch 4, Batch 200, Loss: 0.1138845357298851
Epoch 4, Batch 300, Loss: 0.09844369042664766
Epoch 4, Batch 400, Loss: 0.10318864565342664
Epoch 4, Batch 500, Loss: 0.09892795329913497
Epoch 4, Batch 600, Loss: 0.10432071981951595
Epoch 4, Batch 700, Loss: 0.10643964521586895
Epoch 4, Batch 800, Loss: 0.10537902262061834
Epoch 4, Batch 900, Loss: 0.09039085002150386
Epoch 5, Batch 100, Loss: 0.09001027457881719
Epoch 5, Batch 200, Loss: 0.09010754416696727
Epoch 5, Batch 300, Loss: 0.08061529095750303
Epoch 5, Batch 400, Loss: 0.09361319766379893
Epoch 5, Batch 500, Loss: 0.08480013421736658
Epoch 5, Batch 600, Loss: 0.09767842716537416
Epoch 5, Batch 700, Loss: 0.08496715379413217
Epoch 5, Batch 800, Loss: 0.08992859991267324
Epoch 5, Batch 900, Loss: 0.08316332587972283
Finished Training
```

In [14]:
```
# Evaluate the model and store prediction
model.eval()
predictions = []
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        predictions.extend(predicted.numpy())
        total += labels.size(0)
        correct += (predicted == labels).sum().item()


acc2 = correct / total
print(f"Accuracy on test set: {correct / total * 100}%")
```

```
Accuracy on test set: 96.32166666666667%
```

In [18]:
```
print(f"Difference of the accuracy between the original model and the modified model: {(
```

```
Difference of the accuracy between the original model and the modified model: 1.465%
```

Result:

The accuracy of the original model was 94.87% and the accuracy of the modified model was 96.3%.

As I predicted earlier, the modified model has slightly better accruacy than the original one. However, the difference gap is not large.

Experiment with different optitions and Test (Step 7)

In [21]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F

class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 10)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = torch.tanh(self.fc1(x))
        x = self.dropout(x)
        x = F.leaky_relu(self.fc2(x))   # LeakyReLU
        x = self.dropout(x)          # Apply dropout
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)
        return x

model = MLP()


criterion = nn.CrossEntropyLoss()

optimizer = optim.RMSprop(model.parameters(), lr=0.001)

# Training the model
num_epochs = 5
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

        if i % 100 == 99:
            print(f'Epoch {epoch+1}, Batch {i+1}, Loss: {running_loss / 100}')
            running_loss = 0.0
            correct = 0
            total = 0

print('Finished Training')
```

```
Epoch 1, Batch 100, Loss: 2.214848326444626
Epoch 1, Batch 200, Loss: 1.2555760842561723
Epoch 1, Batch 300, Loss: 0.8431679540872574
Epoch 1, Batch 400, Loss: 0.7097287169098854
Epoch 1, Batch 500, Loss: 0.6299186795949936
Epoch 1, Batch 600, Loss: 0.5980713966488839
Epoch 1, Batch 700, Loss: 0.5657879862189293
Epoch 1, Batch 800, Loss: 0.564327464401722
Epoch 1, Batch 900, Loss: 0.5020254921913146
Epoch 2, Batch 100, Loss: 0.4863882768154144
Epoch 2, Batch 200, Loss: 0.4778830546140671
Epoch 2, Batch 300, Loss: 0.4540400071442127
Epoch 2, Batch 400, Loss: 0.45590210616588595
Epoch 2, Batch 500, Loss: 0.46058921158313754
Epoch 2, Batch 600, Loss: 0.442907407283783
Epoch 2, Batch 700, Loss: 0.4159662649035454
Epoch 2, Batch 800, Loss: 0.42724900677800176
Epoch 2, Batch 900, Loss: 0.4192894984781742
Epoch 3, Batch 100, Loss: 0.42474769070744517
Epoch 3, Batch 200, Loss: 0.40880165740847585
Epoch 3, Batch 300, Loss: 0.40501497030258177
Epoch 3, Batch 400, Loss: 0.3907017305493355
Epoch 3, Batch 500, Loss: 0.4222481627762318
Epoch 3, Batch 600, Loss: 0.4000792096555233
Epoch 3, Batch 700, Loss: 0.38759288266301156
Epoch 3, Batch 800, Loss: 0.3629374518990517
Epoch 3, Batch 900, Loss: 0.3687103050947189
Epoch 4, Batch 100, Loss: 0.37967546336352825
Epoch 4, Batch 200, Loss: 0.34544234558939935
Epoch 4, Batch 300, Loss: 0.35859316542744635
Epoch 4, Batch 400, Loss: 0.3611032846570015
Epoch 4, Batch 500, Loss: 0.3683587721735239
Epoch 4, Batch 600, Loss: 0.3938875167071819
Epoch 4, Batch 700, Loss: 0.34799066685140134
Epoch 4, Batch 800, Loss: 0.3695702560991049
Epoch 4, Batch 900, Loss: 0.35122118473052977
Epoch 5, Batch 100, Loss: 0.3405734857916832
Epoch 5, Batch 200, Loss: 0.35625638857483866
Epoch 5, Batch 300, Loss: 0.36614094726741314
Epoch 5, Batch 400, Loss: 0.33593999870121477
Epoch 5, Batch 500, Loss: 0.34516993798315526
Epoch 5, Batch 600, Loss: 0.3412584808468819
Epoch 5, Batch 700, Loss: 0.3045620555430651
Epoch 5, Batch 800, Loss: 0.35426315799355507
Epoch 5, Batch 900, Loss: 0.3218463706970215
Finished Training
```

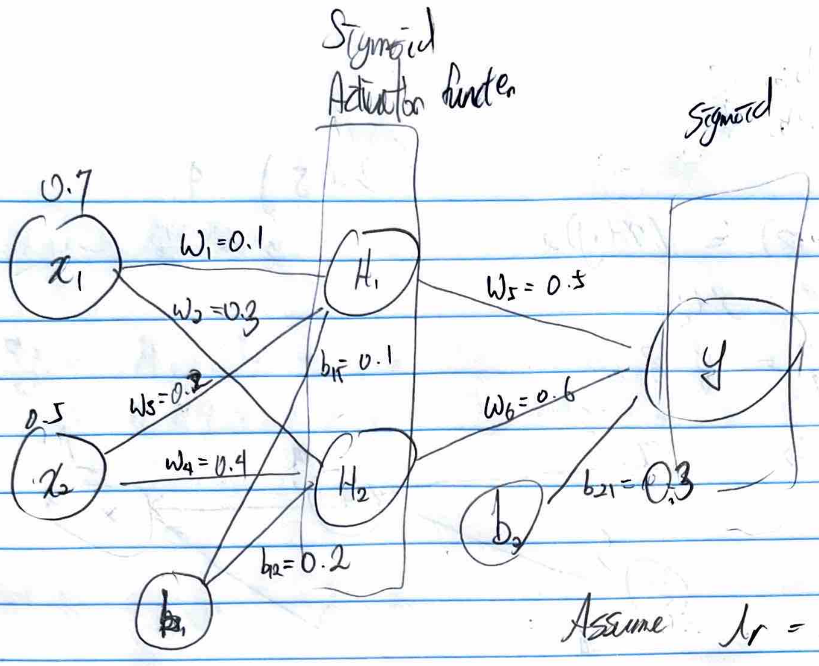
In [22]:

```
# Evaluate the model and store prediction
model.eval()
predictions = []
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        predictions.extend(predicted.numpy())
        total += labels.size(0)
        correct += (predicted == labels).sum().item()


acc3 = correct / total
print(f"Accuracy on test set: {correct / total * 100}%")
```

```
Accuracy on test set: 93.07666666666667%
```

Actually, we can observe that the accuracy has dropped after we modify more. This may because our model is too complex compare to the dataset which may lead it to overfitting.

In [ ]:



Assume $lr = 0.1$

Forward pass

$H_1 = x_1 \cdot W_1 + x_2 \cdot W_3 + b_{11}$

$= 0.7 \cdot 0.1 + 0.5 \cdot 0.2 + 0.1$

$= 0.27$

$H_2 = x_1 \cdot W_2 + x_2 \cdot W_4 + b_{12}$

$= 0.7 \cdot 0.3 + 0.5 \cdot 0.4 + 0.2$

$= 0.21 + 0.2 + 0.2 = 0.61$

Activation function $= \frac{1}{1+e^{-x}}$

Output of $H_1 = \frac{1}{1+e^{-0.27}} = 0.5671$

Output of $H_2 = \frac{1}{1+e^{-0.61}} = 0.6479$

$y =$ output $H_1 \cdot W_5 +$ Output $H_2 \cdot W_6 + b_2$

$= 0.5671 \cdot 0.5 + 0.6479 \cdot 0.6 + 0.3 = 0.9723$

Output $y = \frac{1}{1+e^{-0.9723}} = 0.7286$

$$\frac{\partial Loss}{\partial \hat{y}} = \frac{1}{2}(\hat{y}-y)^2 \quad \frac{1}{2}\cdot 2(\hat{y}-y)$$

Backward pass

Assume true $y = 1$.

$$\frac{\partial Loss}{\partial \hat{y}} = (\hat{y}-y) = 0.7256 - 1 = -0.2744$$

before activation

$$\frac{\partial Loss}{\partial y_6} = \frac{\partial loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial y_6} = -0.2744 \cdot \hat{y}(1-\hat{y})$$
$$= -0.2744 \cdot 0.7256(1-0.7256)$$
$$= -0.0546$$

$$\frac{\partial loss}{\partial W_5} = \frac{\partial Loss}{\partial y_6} \cdot \frac{\partial y_6}{\partial W_5}$$

$y_6 = \text{Output } H_1 \cdot W_5 + \text{Output } H_2 \cdot W_6 + b_2$

$= -0.0546 \cdot 0.5671 = -0.0310$ UPDATE $W_5 = 0.5 - 0.1 \cdot -0.0310$ W_5, $= \boxed{0.5031}$!

$$\frac{\partial Loss}{\partial W_6} = -0.0546 \cdot \text{Output } H_2 = -0.0546 \cdot 0.6479 = -0.0354$$

UPDATED $W_6 = 0.6 - 0.1 \cdot -0.0354 = \boxed{0.6035}$ W_6

$$\frac{\partial Loss}{\partial b_{21}} = \frac{\partial Loss}{\partial y_6} \cdot \frac{\partial y_6}{\partial b_{21}} = -0.0546 \cdot 1 = -0.0546$$

New $b_{21} = 0.3 - 0.1 \cdot -0.0546 = \boxed{0.3055}$ b_{21}

$$\frac{\partial Loss}{\partial \text{output } H_1} = \frac{\partial Loss}{\partial y_6} \cdot \frac{\partial y_6}{\partial \text{Output } H_1} = -0.0546 \cdot W_5 = -0.0546 \cdot 0.5$$
$$= -0.0273$$

$$\frac{\partial Loss}{\partial \text{output } H_2} = -0.0546 \cdot 0.6 = -0.0328$$

$$\frac{\partial Loss}{\partial H_1} = \frac{\partial loss}{\partial \text{output } H_1} \cdot \frac{\partial \text{Output } H_1}{\partial H_1} = -0.0273 \cdot 0.5671(1-0.5671)$$
$$= -0.00669$$

$$\frac{\partial Loss}{\partial H_2} = \frac{\partial loss}{\partial \text{output } H_2} \cdot \frac{\partial \text{Output } H_2}{\partial H_2} = -0.0328 \cdot 0.6479(1-0.6479)$$
$$= -0.00748$$

$$\frac{\partial Loss}{\partial W_1} = \frac{\partial loss}{\partial H_1} \cdot \frac{\partial A_1}{\partial W_1} = -0.0069 \cdot 0.7 = 0.00468.$$

$$\frac{\partial Loss}{\partial W_3} = \frac{\partial Loss}{\partial H_1} \cdot \frac{\partial H_1}{\partial W_3} = -0.0069 \cdot 0.5 = -0.00345$$

$$\frac{\partial Loss}{\partial W_2} = \frac{\partial Loss}{\partial H_2} \cdot \frac{\partial H_2}{\partial W_2} = -0.00748 \cdot 0.7 = -0.00523$$

$$\frac{\partial Loss}{\partial W_4} = \quad // \quad \cdot 0.5 = -0.00374$$

$0.1 - 0.1 \cdot -0.0069$

$\therefore$ UPDATED W

$\Rightarrow W_1 = 0.10047$
$W_2 = 0.30052$
$W_3 = 0.20033$
$W_4 = 0.40037$
$W_5 = 0.5031$
$W_6 = 0.6035$

Updated bias

$\Rightarrow b_{11} = 0.10069$
$b_{12} = 0.20075$
$b_{21} = 0.3055$