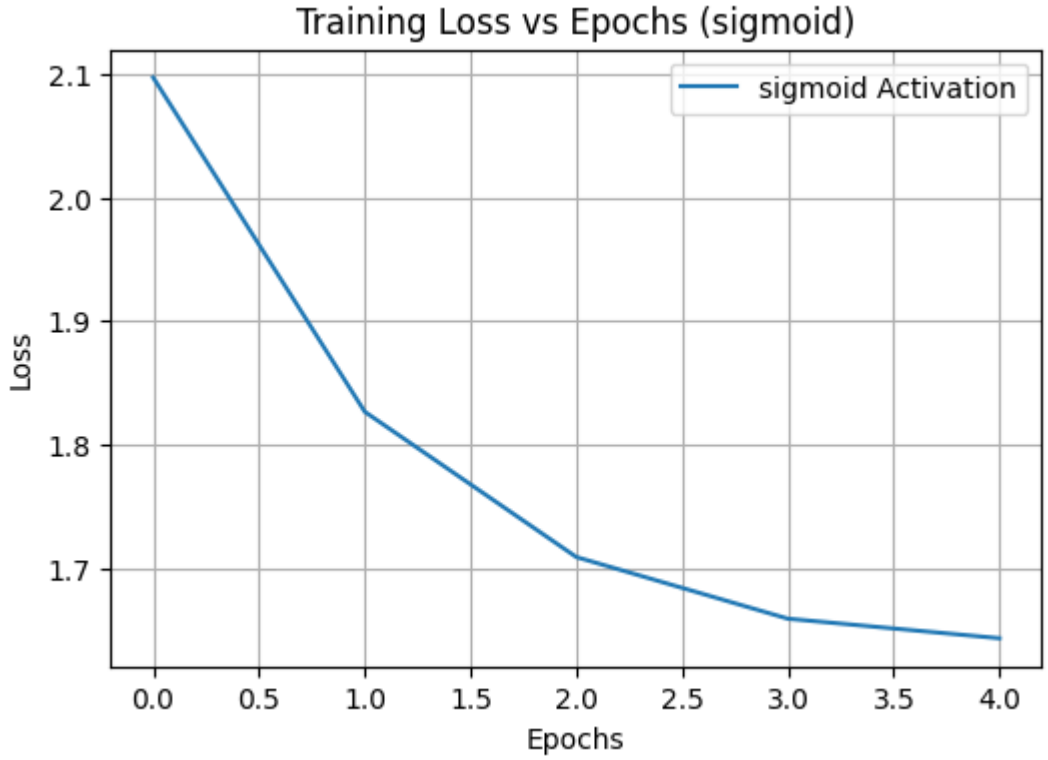
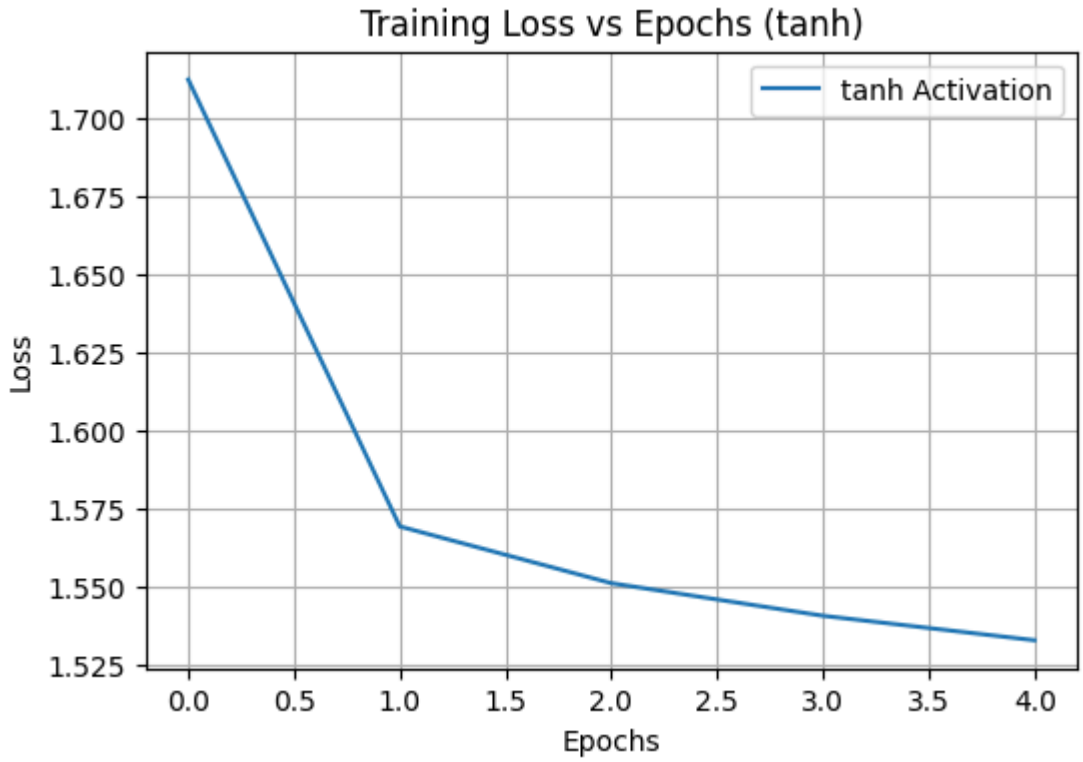
In [ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# MNIST dataset
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5
train_dataset = datasets.MNIST(root='./data', train=True, download=True, transfo
test_dataset = datasets.MNIST(root='./data', train=False, download=True, transfo

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

# Neural network class
class MNIST_NN(nn.Module):
    def __init__(self, activation='relu'):
        super(MNIST_NN, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 10)
        self.activation = activation

    def forward(self, x):
        x = x.view(-1, 28*28)
        if self.activation == 'sigmoid':
            x = torch.sigmoid(self.fc1(x))
        elif self.activation == 'tanh':
            x = torch.tanh(self.fc1(x))
        elif self.activation == 'relu':
            x = F.relu(self.fc1(x))
        elif self.activation == 'leaky_relu':
            x = F.leaky_relu(self.fc1(x))
        else:  # softmax hidden still ReLU
            x = F.relu(self.fc1(x))
        x = F.softmax(self.fc2(x), dim=1)
        return x

# Training function
def train_model(activation, epochs=5):
    model = MNIST_NN(activation)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.1)
    train_losses = []

    for epoch in range(epochs):
        epoch_loss = 0
        for images, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
        train_losses.append(epoch_loss/len(train_loader))
        print(f'Epoch {epoch+1}/{epochs} - Loss: {train_losses[-1]:.4f}')
```

```
    # Compute test accuracy
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    print(f'Test Accuracy with {activation} activation: {accuracy:.2f}%\n')

    return model, train_losses, accuracy

# Train and plot
activations = ['sigmoid','tanh','relu','leaky_relu','softmax']
activation_accuracies = {}

for act in activations:
    print(f'Training with {act} activation...')
    _, loss, acc = train_model(act, epochs=5)
    activation_accuracies[act] = acc
    plt.figure(figsize=(6,4))
    plt.plot(loss, label=f'{act} Activation')
    plt.title(f'Training Loss vs Epochs ({act})')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.show()

# Compare activations
print("Activation Function Comparison (Test Accuracy):")
for act, acc in activation_accuracies.items():
    print(f'{act}: {acc:.2f}%')
```

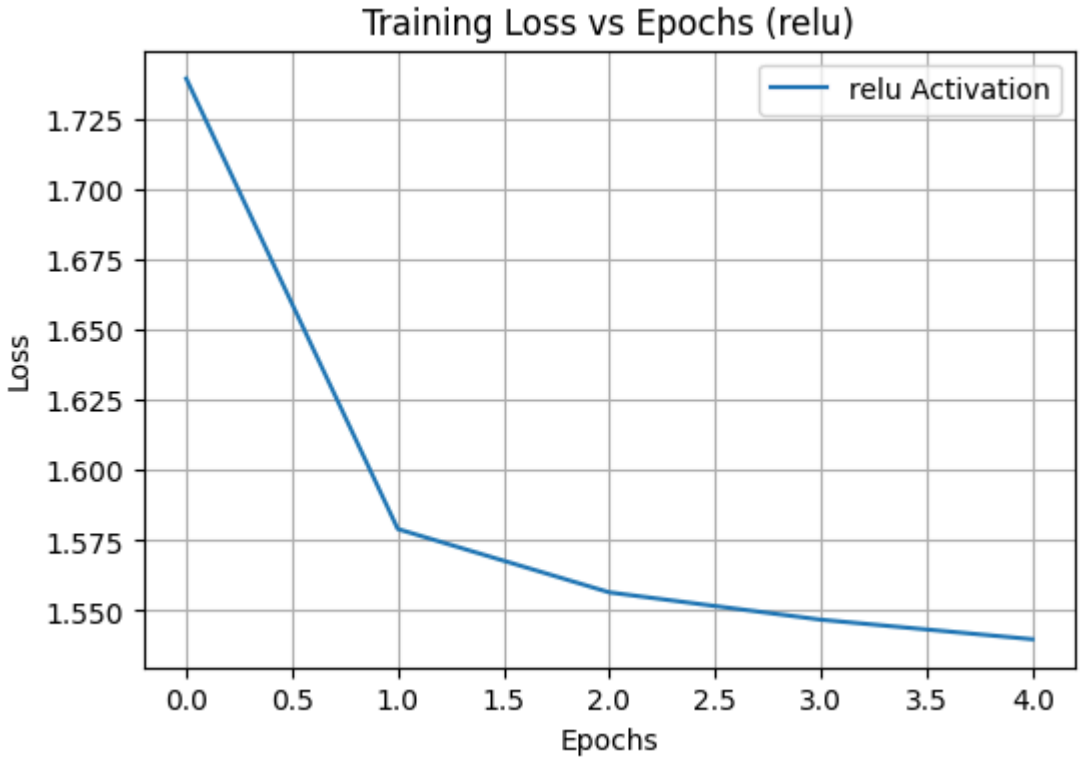
```
Training with sigmoid activation...
Epoch 1/5 - Loss: 2.0970
Epoch 2/5 - Loss: 1.8267
Epoch 3/5 - Loss: 1.7092
Epoch 4/5 - Loss: 1.6594
Epoch 5/5 - Loss: 1.6435
Test Accuracy with sigmoid activation: 84.59%
```




```
Training with tanh activation...
Epoch 1/5 - Loss: 1.7122
Epoch 2/5 - Loss: 1.5693
Epoch 3/5 - Loss: 1.5512
Epoch 4/5 - Loss: 1.5408
Epoch 5/5 - Loss: 1.5330
Test Accuracy with tanh activation: 93.79%
```



```
Training with relu activation...
Epoch 1/5 - Loss: 1.7393
Epoch 2/5 - Loss: 1.5791
Epoch 3/5 - Loss: 1.5565
Epoch 4/5 - Loss: 1.5468
Epoch 5/5 - Loss: 1.5399
Test Accuracy with relu activation: 92.56%
```




```
Training with leaky_relu activation...
Epoch 1/5 - Loss: 1.7315
Epoch 2/5 - Loss: 1.5727
```

In [ ]: