22AIE304 Deep Learning Lab Sheet 5

Fifth Semester BTech CSE(AI)

Department of Computer Science and Engineering

Amrita School of Computing

Training a Feedforward Neural Network on the MNIST Dataset Using PyTorch

Exercise 1: Use this sample ipynb file for training and do the following:

- Part 1: Setting Up the Environment
  - Installing Required Libraries
  - Loading the MNIST Dataset
- Part 2: Defining the Feedforward Neural Network
  - Input layer (784 units)
  - 1 or 2 hidden layers (try different configurations like 128, 256, 512 units)
  - Output layer (10 units corresponding to the 10 classes in MNIST)
- Part 3: Training the Model - Use sigmoid activation

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
class FeedForwardNN(nn.Module):
    def __init__(self, input_size=784, output_size=10, hidden_sizes=[128], activation_fn=nn.
        super(FeedForwardNN, self).__init__()
        layers = []
        for hidden_size in hidden_sizes:
            layers.append(nn.Linear(input_size, hidden_size))
            layers.append(activation_fn())
            input_size = hidden_size
        layers.append(nn.Linear(input_size, output_size))
        self.model = nn.Sequential(*layers)
```

```
        def forward(self, x):
            return self.model(x)


    def train_model(model, train_loader, val_loader, epochs, criterion, optimizer, device):
        train_loss, val_loss = [], []
        train_acc, val_acc = [], []

        for epoch in range(epochs):
            model.train()
            epoch_train_loss, correct, total = 0, 0, 0

            for inputs, labels in train_loader:
                inputs, labels = inputs.view(inputs.size(0), -1).to(device), labels.to(device)
                optimizer.zero_grad()
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                loss.backward()
                optimizer.step()

                epoch_train_loss += loss.item() * inputs.size(0)
                _, predicted = torch.max(outputs, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

            train_loss.append(epoch_train_loss / total)
            train_acc.append(correct / total)

            model.eval()
            epoch_val_loss, correct, total = 0, 0, 0

            with torch.no_grad():
                for inputs, labels in val_loader:
                    inputs, labels = inputs.view(inputs.size(0), -1).to(device), labels.to(devic
                    outputs = model(inputs)
                    loss = criterion(outputs, labels)
                    epoch_val_loss += loss.item() * inputs.size(0)
                    _, predicted = torch.max(outputs, 1)
                    total += labels.size(0)
                    correct += (predicted == labels).sum().item()

            val_loss.append(epoch_val_loss / total)
            val_acc.append(correct / total)

        return train_loss, val_loss, train_acc, val_acc


    transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))]
    train_set = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=t
    test_set = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=t

    train_size = int(0.8 * len(train_set))
```

```
val_size = len(train_set) - train_size
train_subset, val_subset = random_split(train_set, [train_size, val_size])

HLayers = [1, 2]
hidden_sizes = [64, 128, 256, 512]
activations = [nn.ReLU, nn.Sigmoid, nn.Tanh, nn.LeakyReLU]
learning_rates = [0.1, 0.01, 0.001]
batch_sizes = [16, 32, 64, 128]
optimizers = {"SGD": optim.SGD, "Adam": optim.Adam, "RMSprop": optim.RMSprop}
epochs = 10
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz to
100%|██████████| 9.91M/9.91M [00:00<00:00, 20.1MB/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz to
100%|██████████| 28.9k/28.9k [00:00<00:00, 619kB/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz to .
100%|██████████| 1.65M/1.65M [00:00<00:00, 5.58MB/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz to .
100%|██████████| 4.54k/4.54k [00:00<00:00, 4.79MB/s]Extracting ./data/MNIST/raw/t10k-lab
```

```
confusion_matrices = []

def compute_confusion_matrix(model, data_loader, device):
    """Compute confusion matrix for the given model and data_loader."""
    model.eval()
```

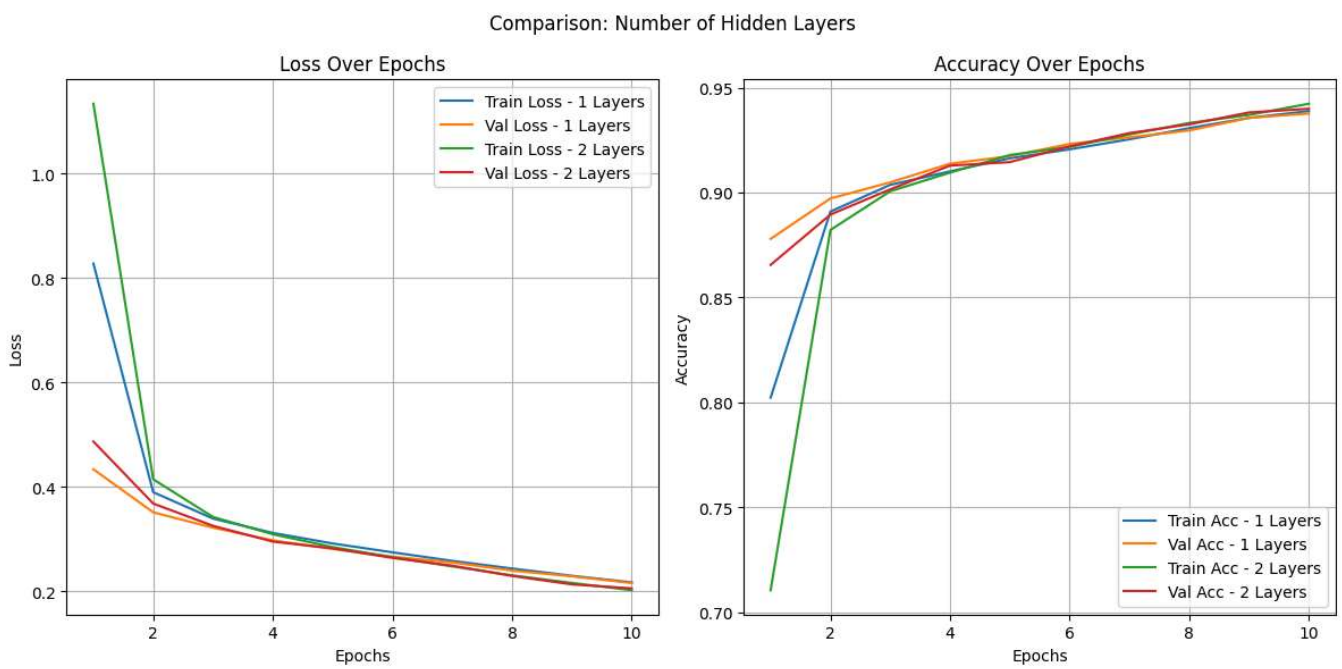
```
    all_preds, all_labels = [], []

    with torch.no_grad():
        for inputs, labels in data_loader:
            inputs, labels = inputs.view(inputs.size(0), -1).to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    return confusion_matrix(all_labels, all_preds)


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))]
train_set = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=t
test_set = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=t

train_size = int(0.8 * len(train_set))
val_size = len(train_set) - train_size
train_subset, val_subset = random_split(train_set, [train_size, val_size])

train_loader = DataLoader(train_subset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_subset, batch_size=64, shuffle=False)
test_loader = DataLoader(test_set, batch_size=64, shuffle=False)
```

Exercise 2: Visualizing the Training Process

- Plot the Training Loss Curve: Modify your code to store the training loss at each epoch and plot a curve that shows how the loss decreases over time. How does the loss change as the model trains? Does the loss converge as the number of epochs increases?
- Plot the Accuracy Curve: Track the training accuracy after each epoch and plot a graph showing how accuracy improves over time (comparing predictions with true labels). How does training accuracy change during training? Does the accuracy saturate after a certain number of epochs?
- Visualize Weight Updates: Visualize the changes in the weights of the hidden layer during training. After each epoch, store the values of the weights and plot them. How do weights evolve throughout training? Are there any patterns in the weight changes?

```
results = []
for hidden_layers in HLayers:
    model = FeedForwardNN(
        hidden_sizes=[128] * hidden_layers,
        activation_fn=nn.ReLU
    ).to(device)
```

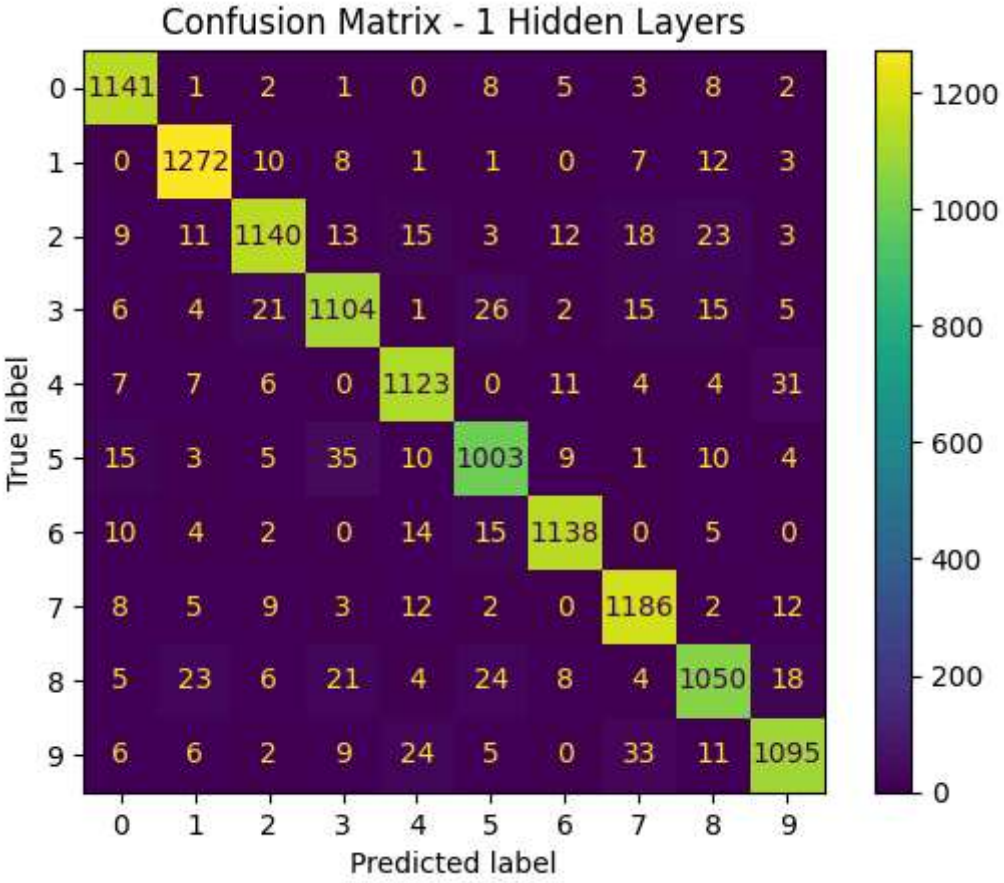
```
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(model.parameters(), lr=0.01)

        train_loss, val_loss, train_acc, val_acc = train_model(
            model, train_loader, val_loader, epochs, criterion, optimizer, device
        )
        results.append((train_loss, val_loss, train_acc, val_acc))

        cm = compute_confusion_matrix(model, val_loader, device)
        confusion_matrices.append({
            "experiment": f"{hidden_layers} Hidden Layers",
            "confusion_matrix": cm
        })

    fig, axs = plt.subplots(1, 2, figsize=(12, 6))
    for i, hidden_layers in enumerate(HLayers):
        axs[0].plot(range(1, epochs + 1), results[i][0], label=f"Train Loss - {hidden_layers} La
        axs[0].plot(range(1, epochs + 1), results[i][1], label=f"Val Loss - {hidden_layers} Laye
        axs[1].plot(range(1, epochs + 1), results[i][2], label=f"Train Acc - {hidden_layers} Lay
        axs[1].plot(range(1, epochs + 1), results[i][3], label=f"Val Acc - {hidden_layers} Layer

    axs[0].set_title("Loss Over Epochs")
    axs[0].set_xlabel("Epochs")
    axs[0].set_ylabel("Loss")
    axs[0].legend()
    axs[0].grid()

    axs[1].set_title("Accuracy Over Epochs")
    axs[1].set_xlabel("Epochs")
    axs[1].set_ylabel("Accuracy")
    axs[1].legend()
    axs[1].grid()

    plt.suptitle("Comparison: Number of Hidden Layers")
    plt.tight_layout()
    plt.show()
```

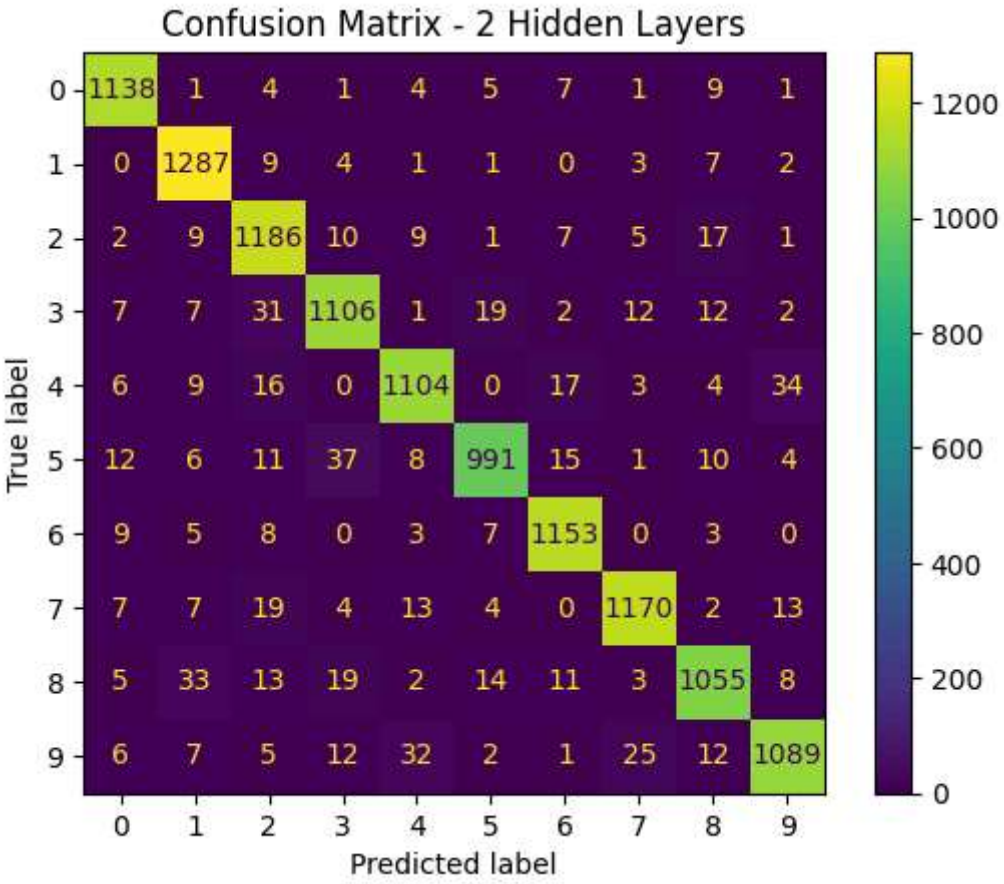

Comparison: Number of Hidden Layers
Loss Over Epochs
Train Loss - 1 Layers
Val Loss - 1 Layers
Train Loss - 2 Layers
Val Loss - 2 Layers
Loss
Epochs
Accuracy Over Epochs
Train Acc - 1 Layers
Val Acc - 1 Layers
Train Acc - 2 Layers
Val Acc - 2 Layers
Accuracy
Epochs

Exercise 3: Visualizing Validation/Testing results

a. Plot confusion Matrix on validation/test set: After the model is trained, plot a confusion matrix to show how well the model classifies each digit (0-9). Which digits does the model classify well, and which ones are more often confused with others
b. Plot validation accuracy and validation error for each epoch. How does the validation accuracy compare to the training accuracy
c. Compare training and validation loss

```
for entry in confusion_matrices:
    experiment_name = entry["experiment"]
    cm = entry["confusion_matrix"]

    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=range(cm.shape[0]))
    disp.plot(cmap="viridis")
    plt.title(f"Confusion Matrix - {experiment_name}")
    plt.show()
```

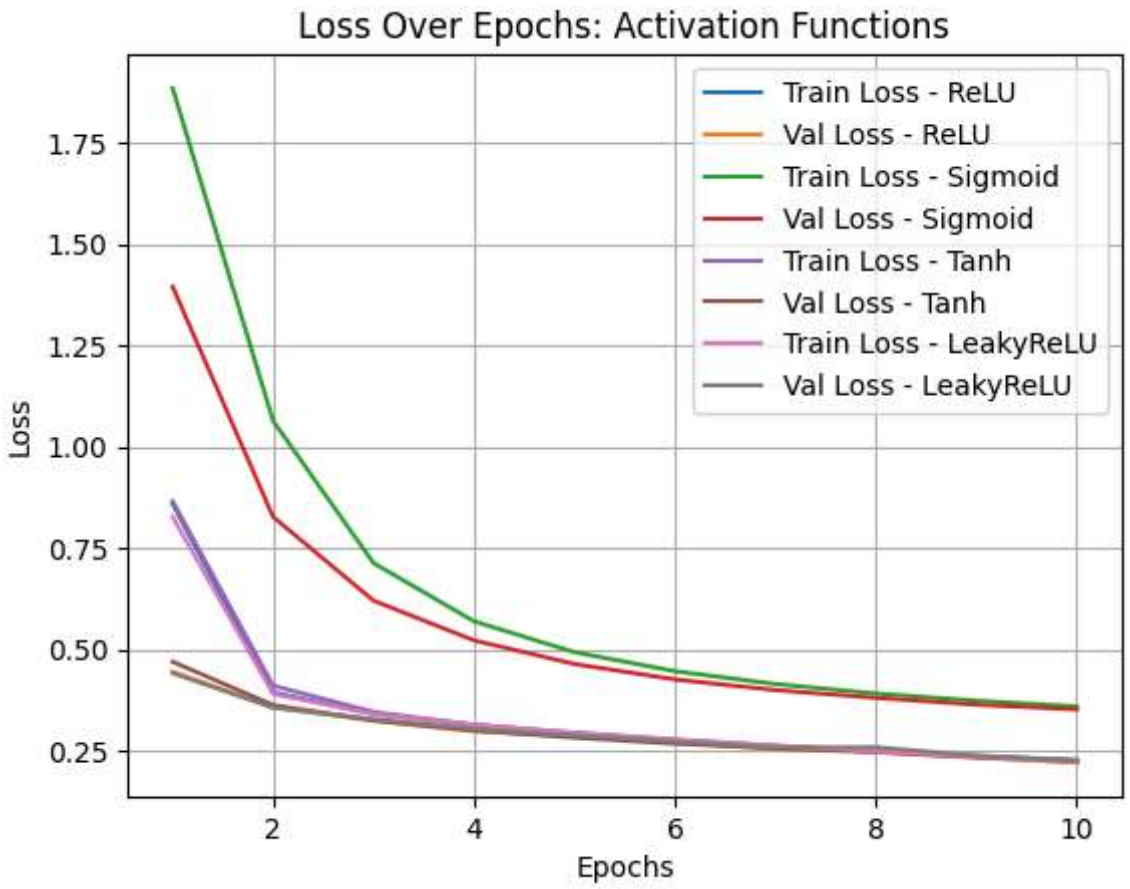

Confusion Matrix - 1 Hidden Layers
True label
Predicted label
0 1141 1 2 1 0 8 5 3 8 2
1 0 1272 10 8 1 1 0 7 12 3
2 9 11 1140 13 15 3 12 18 23 3
3 6 4 21 1104 1 26 2 15 15 5
4 7 7 6 0 1123 0 11 4 4 31
5 15 3 5 35 10 1003 9 1 10 4
6 10 4 2 0 14 15 1138 0 5 0
7 8 5 9 3 12 2 0 1186 2 12
8 5 23 6 21 4 24 8 4 1050 18
9 6 6 2 9 24 5 0 33 11 1095
0 1 2 3 4 5 6 7 8 9
1200
1000
800
600
400
200
0


Confusion Matrix - 2 Hidden Layers
True label
Predicted label
0 1138 1 4 1 4 5 7 1 9 1
1 0 1287 9 4 1 1 0 3 7 2
2 2 9 1186 10 9 1 7 5 17 1
3 7 7 31 1106 1 19 2 12 12 2
4 6 9 16 0 1104 0 17 3 4 34
5 12 6 11 37 8 991 15 1 10 4
6 9 5 8 0 3 7 1153 0 3 0
7 7 7 19 4 13 4 0 1170 2 13
8 5 33 13 19 2 14 11 3 1055 8
9 6 7 5 12 32 2 1 25 12 1089
0 1 2 3 4 5 6 7 8 9
1200
1000
800
600
400
200
0

Exercise 4: Explore different Activation Functions

Replace the activation functions (ReLU, Sigmoid, Tanh, Leaky ReLU) in your network and observe the effect on convergence and accuracy. Plot the performance for different activation functions.

```
for activation_fn in activations:
    model = FeedForwardNN(
        hidden_sizes=[128],
        activation_fn=activation_fn
    ).to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.01)

    train_loss, val_loss, train_acc, val_acc = train_model(
        model, train_loader, val_loader, epochs, criterion, optimizer, device
    )

    plt.plot(range(1, epochs + 1), train_loss, label=f"Train Loss - {activation_fn.__name__}")
    plt.plot(range(1, epochs + 1), val_loss, label=f"Val Loss - {activation_fn.__name__}")

plt.title("Loss Over Epochs: Activation Functions")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()
```

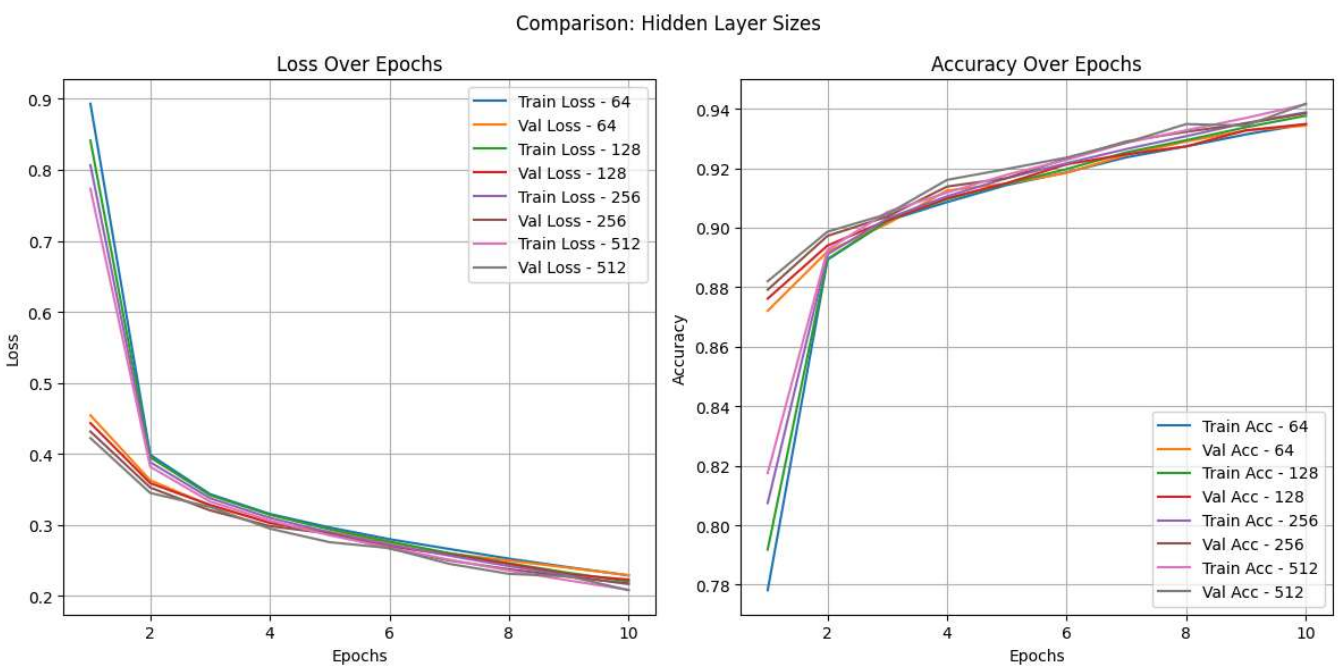


Exercise 5: Experimenting with Hyperparameters

a. Experiment with different learning rates (e.g., 0.1, 0.01, 0.001). Plot the effect of learning rate on training loss and accuracy.
b. Experiment with different batch sizes (e.g., 16, 32, 64, 128). Plot the training time and accuracy for each batch size.
c. Experiment with different numbers of hidden units in the hidden layers (e.g., 64, 128, 256). Plot the validation accuracy as a function of the number of hidden units.
d. Try different optimizers (SGD, Adam, RMSprop) and compare the convergence speed and final performance.

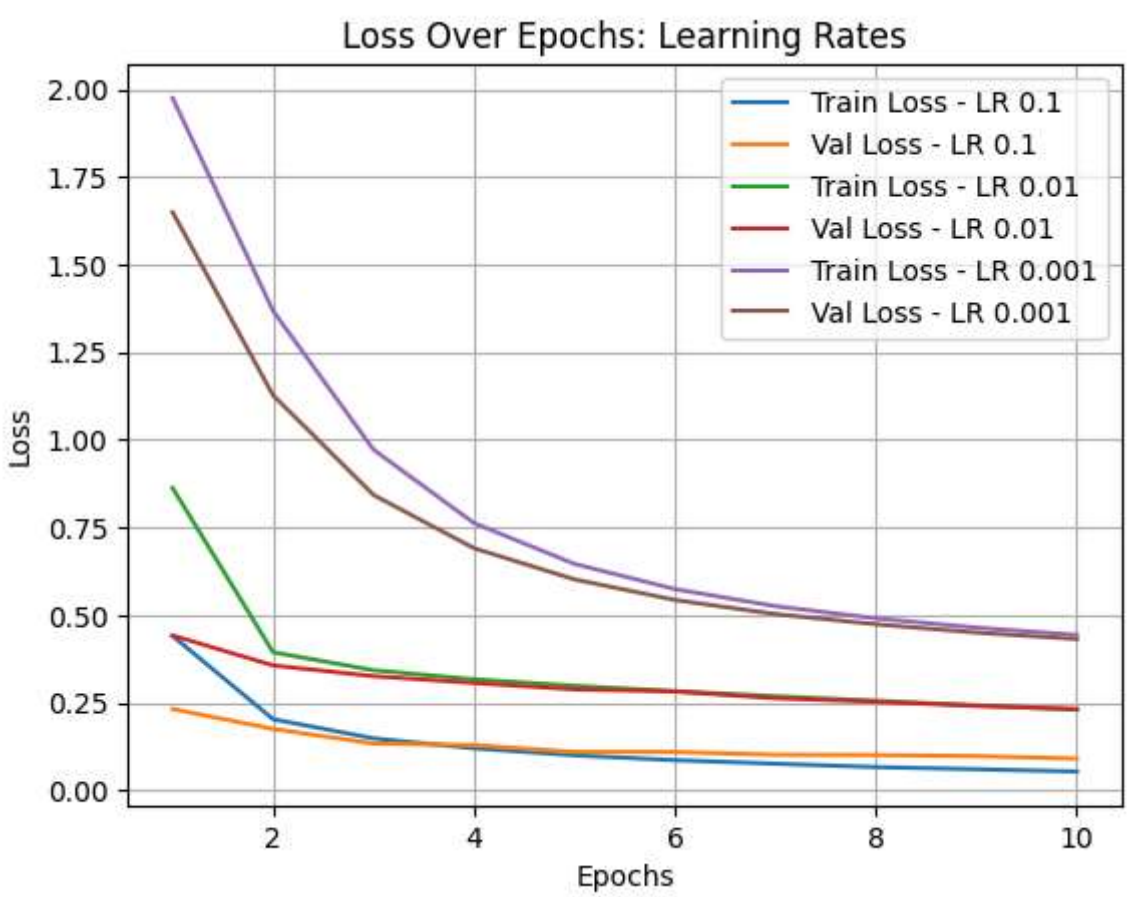
```
results = []
for hidden_size in hidden_sizes:
    model = FeedForwardNN(
        hidden_sizes=[hidden_size] * 1,
        activation_fn=nn.ReLU
    ).to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.01)
```

```
        train_loss, val_loss, train_acc, val_acc = train_model(
            model, train_loader, val_loader, epochs, criterion, optimizer, device
        )
        results.append((train_loss, val_loss, train_acc, val_acc))


    fig, axs = plt.subplots(1, 2, figsize=(12, 6))
    for i, hidden_size in enumerate(hidden_sizes):
        axs[0].plot(range(1, epochs + 1), results[i][0], label=f"Train Loss - {hidden_size}")
        axs[0].plot(range(1, epochs + 1), results[i][1], label=f"Val Loss - {hidden_size}")
        axs[1].plot(range(1, epochs + 1), results[i][2], label=f"Train Acc - {hidden_size}")
        axs[1].plot(range(1, epochs + 1), results[i][3], label=f"Val Acc - {hidden_size}")

    axs[0].set_title("Loss Over Epochs")
    axs[0].set_xlabel("Epochs")
    axs[0].set_ylabel("Loss")
    axs[0].legend()
    axs[0].grid()

    axs[1].set_title("Accuracy Over Epochs")
    axs[1].set_xlabel("Epochs")
    axs[1].set_ylabel("Accuracy")
    axs[1].legend()
    axs[1].grid()

    plt.suptitle("Comparison: Hidden Layer Sizes")
    plt.tight_layout()
    plt.show()
```



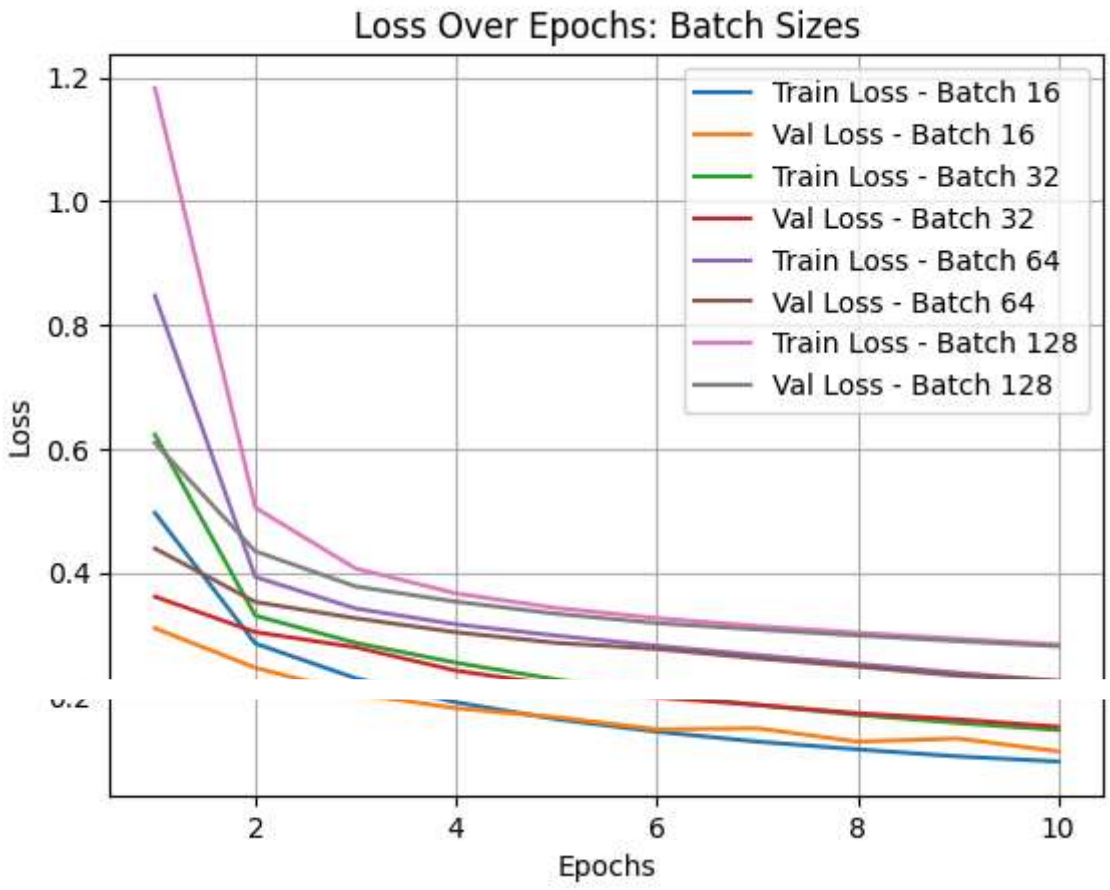

```
for lr in learning_rates:
    model = FeedForwardNN(
        hidden_sizes=[128],
        activation_fn=nn.ReLU
    ).to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=lr)

    train_loss, val_loss, train_acc, val_acc = train_model(
        model, train_loader, val_loader, epochs, criterion, optimizer, device
    )


    plt.plot(range(1, epochs + 1), train_loss, label=f"Train Loss - LR {lr}")
    plt.plot(range(1, epochs + 1), val_loss, label=f"Val Loss - LR {lr}")

plt.title("Loss Over Epochs: Learning Rates")
```

```
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.grid()
    plt.show()
```




```
for batch_size in batch_sizes:
    train_loader = DataLoader(train_subset, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(val_subset, batch_size=batch_size, shuffle=False)

    model = FeedForwardNN(
        hidden_sizes=[128],
        activation_fn=nn.ReLU
    ).to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.01)

    train_loss, val_loss, train_acc, val_acc = train_model(
        model, train_loader, val_loader, epochs, criterion, optimizer, device
    )


    plt.plot(range(1, epochs + 1), train_loss, label=f"Train Loss - Batch {batch_size}"
    plt.plot(range(1, epochs + 1), val_loss, label=f"Val Loss - Batch {batch_size}")
```

```
plt.title("Loss Over Epochs: Batch Sizes")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()
```




```
for name, optimizer_fn in optimizers.items():
    model = FeedForwardNN(
        hidden_sizes=[128],
        activation_fn=nn.ReLU
    ).to(device)
```