4.1 Defining a Neural Network

```
import torch
import torch.nn as nn

# Define a simple neural network
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        # Define layers
        self.fc1 = nn.Linear(3, 5)  # Input layer to hidden layer
        self.fc2 = nn.Linear(5, 1)  # Hidden layer to output layer

    def forward(self, x):
        # Define forward pass
        x = torch.relu(self.fc1(x))  # Apply ReLU activation
        x = self.fc2(x)
        return x

# Instantiate the network
net = SimpleNN()
print(net)

```

```
SimpleNN(
  (fc1): Linear(in_features=3, out_features=5, bias=True)
  (fc2): Linear(in_features=5, out_features=1, bias=True)
)
```

4.2 Loss Function

```
# Define Mean Squared Error loss function
criterion = nn.MSELoss()
```

4.3 Optimizer

```
import torch.optim as optim

# Define an optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

```

4.4 Preparing Data

```
# Example training data
inputs = torch.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]], dtype=torch.float32)
targets = torch.tensor([[1.0], [2.0], [3.0]], dtype=torch.float32)

# Create a DataLoader
from torch.utils.data import DataLoader, TensorDataset

dataset = TensorDataset(inputs, targets)
data_loader = DataLoader(dataset, batch_size=2, shuffle=True)

```

4.5 Training Loop

```
# Training loop
num_epochs = 100
for epoch in range(num_epochs):
    for batch_inputs, batch_targets in data_loader:
        # Zero the gradient buffers
        optimizer.zero_grad()

        # Forward pass
        outputs = net(batch_inputs)

        # Compute loss
        loss = criterion(outputs, batch_targets)

        # Backward pass
        loss.backward()

        # Update weights
        optimizer.step()

    if epoch % 10 == 0:
```

4.6 Evaluating the Model

```
print("Training complete")
```

```
# Example test data
test_inputs = torch.tensor([[10.0, 11.0, 12.0]], dtype=torch.float32)
test_targets = torch.tensor([[4.0]], dtype=torch.float32)

# Forward pass on test data
with torch.no_grad():
    test_outputs = net(test_inputs)
    test_loss = criterion(test_outputs, test_targets)

print(f"Test Loss: {test_loss.item()}")

```

```
Test Loss: 2.29196302825585e-05
```

Exercises

1. Define a More Complex Network: Create a deeper network with more layers and different activation functions. Experiment with different layer sizes and configurations.
2. Experiment with Different Loss Functions: Try using a different loss function, such as Cross Entropy Loss, and modify the network for a classification task.
3. Optimize with Different Optimizers: Compare the performance of different optimizers (e.g., SGD vs. Adam) on the same task. Adjust learning rates and observe the effects.
4. Add Regularization: Implement L2 regularization by adding a penalty to the loss function. Experiment with dropout layers to prevent overfitting.
5. Data Augmentation: Use data augmentation techniques to artificially increase the size of the training set. Implement these techniques and observe their effects on model performance.