Question 1:Develop a neural network using PyTorch for BMI estimation.Given information about the user, such as their gender, height, and weight, your modelshould classify the user into one of the BMI indices. Refer to Kaggle for access to the dataset.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import torch
from torch.utils.data import DataLoader, TensorDataset
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
df = pd.read_csv('bmi.csv')
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})
X = df[['Gender', 'Height', 'Weight']].values
y = df['Index'].values

# Normalize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

print("Training set shape:")
print(f"X_train: {X_train.shape} (Dimensions: {X_train.ndim})")
print(f"y_train: {y_train.shape} (Dimensions: {y_train.ndim})")

print("\nTest set shape:")
print(f"X_test: {X_test.shape} (Dimensions: {X_test.ndim})")
print(f"y_test: {y_test.shape} (Dimensions: {y_test.ndim})")
```

```
Training set shape:
X_train: (350, 3) (Dimensions: 2)
y_train: (350,) (Dimensions: 1)

Test set shape:
X_test: (150, 3) (Dimensions: 2)
y_test: (150,) (Dimensions: 1)
```

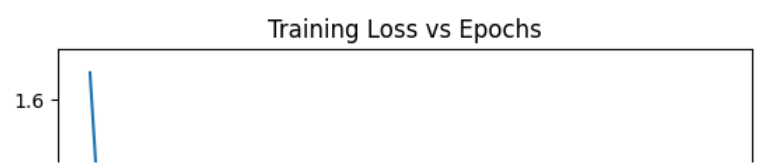
```
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.int64)
y_test_tensor = torch.tensor(y_test, dtype=torch.int64)
```

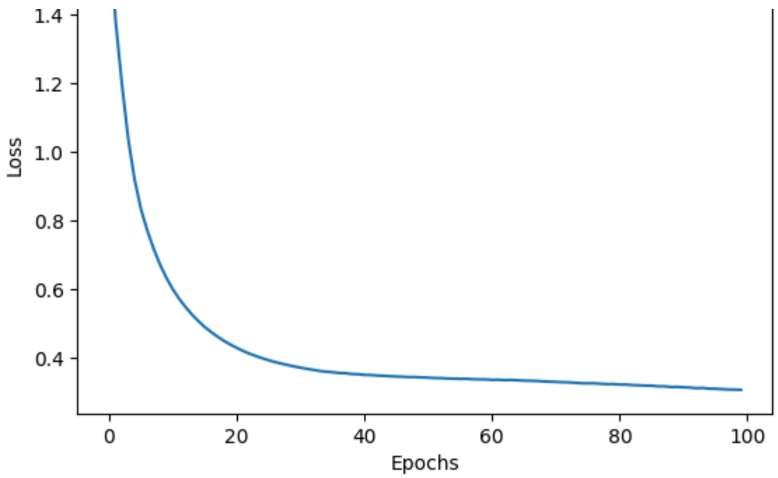
```
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)


torch.manual_seed(0)

# Define the neural network model
model = nn.Sequential(
    nn.Linear(3, 100),
    nn.ReLU(),
    nn.Linear(100, 50),
    nn.ReLU(),
    nn.Linear(50,6)
)


loss_fn = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

# Training parameters
num_epochs = 100
batch_size = 32
losses = []

# Training loop
for epoch in range(num_epochs):
    model.train()
    for i in range(0, len(X_train_tensor), batch_size):

        X_batch = X_train_tensor[i:i + batch_size]
        y_batch = y_train_tensor[i:i + batch_size]
        y_pred = model(X_batch)

        loss = loss_fn(y_pred, y_batch)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    losses.append(loss.item())
    print(f"Epoch {epoch + 1}: loss on final training batch: {loss.item():.4f}")

# Evaluate on the test set
y_test_pred = model(X_test_tensor)
y_test_pred_classes = torch.argmax(y_test_pred, axis=1)
test_loss = loss_fn(y_test_pred, y_test_tensor)
#print("Loss on test set: {:.4f}".format(test_loss.item()))
print("\nTest Set Performance:")
print(classification_report(y_test_tensor.numpy(), y_test_pred_classes.numpy(), zero_division=0))

# Plot the training loss
plt.plot(losses)
plt.title("Training Loss vs Epochs")
```

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

```
Epoch 1: loss on final training batch: 1.6789
Epoch 2: loss on final training batch: 1.3853
Epoch 3: loss on final training batch: 1.1983
Epoch 4: loss on final training batch: 1.0369
Epoch 5: loss on final training batch: 0.9183
Epoch 6: loss on final training batch: 0.8321
Epoch 7: loss on final training batch: 0.7696
Epoch 8: loss on final training batch: 0.7160
Epoch 9: loss on final training batch: 0.6702
Epoch 10: loss on final training batch: 0.6316
Epoch 11: loss on final training batch: 0.5986
Epoch 12: loss on final training batch: 0.5708
Epoch 13: loss on final training batch: 0.5468
Epoch 14: loss on final training batch: 0.5251
Epoch 15: loss on final training batch: 0.5060
Epoch 16: loss on final training batch: 0.4887
Epoch 17: loss on final training batch: 0.4742
Epoch 18: loss on final training batch: 0.4607
Epoch 19: loss on final training batch: 0.4486
Epoch 20: loss on final training batch: 0.4379
Epoch 21: loss on final training batch: 0.4282
Epoch 22: loss on final training batch: 0.4192
Epoch 23: loss on final training batch: 0.4112
Epoch 24: loss on final training batch: 0.4045
Epoch 25: loss on final training batch: 0.3978
Epoch 26: loss on final training batch: 0.3921
Epoch 27: loss on final training batch: 0.3868
Epoch 28: loss on final training batch: 0.3822
Epoch 29: loss on final training batch: 0.3782
Epoch 30: loss on final training batch: 0.3741
Epoch 31: loss on final training batch: 0.3705
Epoch 32: loss on final training batch: 0.3672
Epoch 33: loss on final training batch: 0.3641
Epoch 34: loss on final training batch: 0.3608
Epoch 35: loss on final training batch: 0.3588
Epoch 36: loss on final training batch: 0.3572
Epoch 37: loss on final training batch: 0.3550
Epoch 38: loss on final training batch: 0.3544
Epoch 39: loss on final training batch: 0.3521
Epoch 40: loss on final training batch: 0.3515
Epoch 41: loss on final training batch: 0.3495
Epoch 42: loss on final training batch: 0.3490
Epoch 43: loss on final training batch: 0.3474
Epoch 44: loss on final training batch: 0.3467
Epoch 45: loss on final training batch: 0.3456
Epoch 46: loss on final training batch: 0.3447
Epoch 47: loss on final training batch: 0.3440
Epoch 48: loss on final training batch: 0.3431
Epoch 49: loss on final training batch: 0.3430
Epoch 50: loss on final training batch: 0.3421
Epoch 51: loss on final training batch: 0.3414
Epoch 52: loss on final training batch: 0.3403
Epoch 53: loss on final training batch: 0.3400
Epoch 54: loss on final training batch: 0.3389
Epoch 55: loss on final training batch: 0.3384
Epoch 56: loss on final training batch: 0.3378
Epoch 57: loss on final training batch: 0.3381
Epoch 58: loss on final training batch: 0.3369
Epoch 59: loss on final training batch: 0.3363
Epoch 60: loss on final training batch: 0.3364
Epoch 61: loss on final training batch: 0.3348
```

```
Epoch 61: loss on final training batch: 0.3348
Epoch 62: loss on final training batch: 0.3352
Epoch 63: loss on final training batch: 0.3342
Epoch 64: loss on final training batch: 0.3347
Epoch 65: loss on final training batch: 0.3338
Epoch 66: loss on final training batch: 0.3326
Epoch 67: loss on final training batch: 0.3324
Epoch 68: loss on final training batch: 0.3317
Epoch 69: loss on final training batch: 0.3306
Epoch 70: loss on final training batch: 0.3298
Epoch 71: loss on final training batch: 0.3290
Epoch 72: loss on final training batch: 0.3281
Epoch 73: loss on final training batch: 0.3277
Epoch 74: loss on final training batch: 0.3265
Epoch 75: loss on final training batch: 0.3251
Epoch 76: loss on final training batch: 0.3249
Epoch 77: loss on final training batch: 0.3248
Epoch 78: loss on final training batch: 0.3236
Epoch 79: loss on final training batch: 0.3226
Epoch 80: loss on final training batch: 0.3227
Epoch 81: loss on final training batch: 0.3213
Epoch 82: loss on final training batch: 0.3213
Epoch 83: loss on final training batch: 0.3197
Epoch 84: loss on final training batch: 0.3190
Epoch 85: loss on final training batch: 0.3182
Epoch 86: loss on final training batch: 0.3179
Epoch 87: loss on final training batch: 0.3161
Epoch 88: loss on final training batch: 0.3161
Epoch 89: loss on final training batch: 0.3141
Epoch 90: loss on final training batch: 0.3144
Epoch 91: loss on final training batch: 0.3134
Epoch 92: loss on final training batch: 0.3126
Epoch 93: loss on final training batch: 0.3107
Epoch 94: loss on final training batch: 0.3114
Epoch 95: loss on final training batch: 0.3093
Epoch 96: loss on final training batch: 0.3090
Epoch 97: loss on final training batch: 0.3079
Epoch 98: loss on final training batch: 0.3069
Epoch 99: loss on final training batch: 0.3066
Epoch 100: loss on final training batch: 0.3059

Test Set Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      0.86      0.92         7
           2       0.95      0.86      0.90        21
           3       0.90      0.95      0.93        20
           4       0.90      0.97      0.94        39
           5       0.98      0.97      0.97        59

    accuracy                           0.95       150
   macro avg       0.96      0.93      0.94       150
weighted avg       0.95      0.95      0.95       150
```





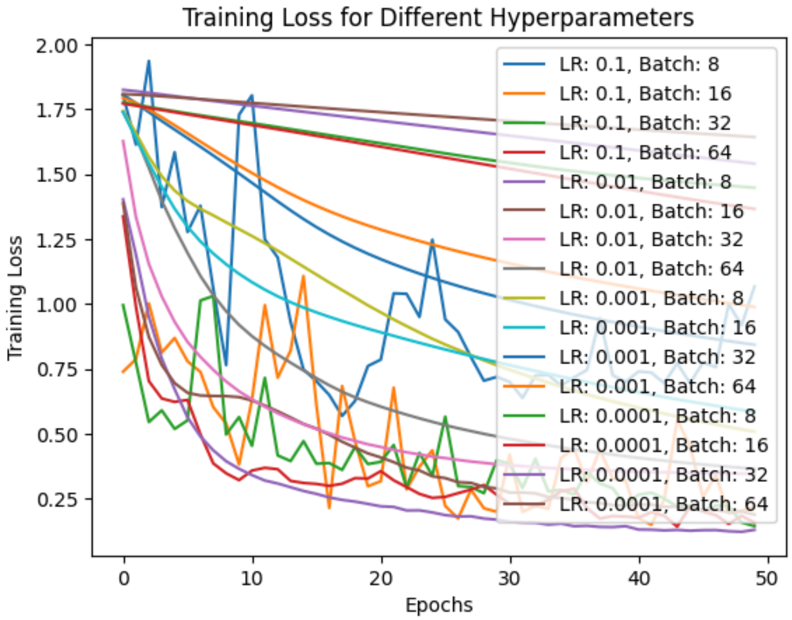
1.4
1.2
1.0
0.8
0.6
0.4
Loss
0
20
40
60
80
100
Epochs

```
# Set seed for reproducibility
torch.manual_seed(0)

# Hyperparameters for fine-tuning
learning_rates = [0.1, 0.01, 0.001, 0.0001]
batch_sizes = [8, 16, 32, 64]
num_epochs = 50
results = {}

def create_model():
    model = nn.Sequential(
        nn.Linear(3, 100),
        nn.ReLU(),
        nn.Linear(100, 50),
        nn.ReLU(),
        nn.Linear(50, 6)
    )
    return model

# Train model
def train_model(model, optimizer, loss_fn, num_epochs, batch_size, X_train_tensor, y_train_tensor):
    losses = []

    for epoch in range(num_epochs):
        for i in range(0, len(X_train_tensor), batch_size):
            X_batch = X_train_tensor[i:i + batch_size]
            y_batch = y_train_tensor[i:i + batch_size]
            y_pred = model(X_batch)

            loss = loss_fn(y_pred, y_batch)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        losses.append(loss.item())

    return losses

# Evaluate the model
def evaluate_model(model, loss_fn, X_test_tensor, y_test_tensor):
        y_test_pred = model(X_test_tensor)
        y_test_pred_classes = torch.argmax(y_test_pred, axis=1)
        test_loss = loss_fn(y_test_pred, y_test_tensor)
        print("Loss on test set: {:.4f}".format(test_loss.item()))

        # Calculate accuracy
        accuracy = accuracy_score(y_test_tensor.numpy(), y_test_pred_classes.numpy())
        print(f"Test Accuracy: {accuracy:.4f}")  # Print test accuracy

        return y_test_pred_classes

for lr in learning_rates:
    for batch_size in batch_sizes:
        print(f"\nTraining with learning rate: {lr}, batch size: {batch_size}")
```

```
        model = create_model()
        loss_fn = nn.CrossEntropyLoss()
        optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9)

        losses = train_model(model, optimizer, loss_fn, num_epochs, batch_size, X_train_tensor, y_train_tensor)
        y_test_pred_classes = evaluate_model(model, loss_fn, X_test_tensor, y_test_tensor)

        results[(lr, batch_size)] = {
            'losses': losses,
            'y_test_pred_classes': y_test_pred_classes
        }

# Plot training losses
for (lr, batch_size), result in results.items():
    plt.plot(result['losses'], label=f"LR: {lr}, Batch: {batch_size}")

plt.title("Training Loss for Different Hyperparameters")
plt.xlabel("Epochs")
plt.ylabel("Training Loss")
plt.legend()
plt.show()
```

```
Training with learning rate: 0.1, batch size: 8
Loss on test set: 0.8963
Test Accuracy: 0.6267

Training with learning rate: 0.1, batch size: 16
Loss on test set: 0.3833
Test Accuracy: 0.8800

Training with learning rate: 0.1, batch size: 32
Loss on test set: 0.2951
Test Accuracy: 0.9333

Training with learning rate: 0.1, batch size: 64
Loss on test set: 0.2692
Test Accuracy: 0.8933

Training with learning rate: 0.01, batch size: 8
Loss on test set: 0.2910
Test Accuracy: 0.9067

Training with learning rate: 0.01, batch size: 16
Loss on test set: 0.2911
Test Accuracy: 0.9067

Training with learning rate: 0.01, batch size: 32
Loss on test set: 0.2915
Test Accuracy: 0.9133

Training with learning rate: 0.01, batch size: 64
Loss on test set: 0.3720
Test Accuracy: 0.8667

Training with learning rate: 0.001, batch size: 8
Loss on test set: 0.4117
Test Accuracy: 0.8467

Training with learning rate: 0.001, batch size: 16
Loss on test set: 0.5729
Test Accuracy: 0.8400

Training with learning rate: 0.001, batch size: 32
Loss on test set: 0.8210
Test Accuracy: 0.7467

Training with learning rate: 0.001, batch size: 64
Loss on test set: 0.9658
Test Accuracy: 0.6200

Training with learning rate: 0.0001, batch size: 8
Loss on test set: 1.1818
Test Accuracy: 0.4000

Training with learning rate: 0.0001, batch size: 16
Loss on test set: 1.3686
Test Accuracy: 0.3933

Training with learning rate: 0.0001, batch size: 32
Loss on test set: 1.5450
Test Accuracy: 0.4067
```

```
Training with learning rate: 0.0001, batch size: 64
Loss on test set: 1.6573
Test Accuracy: 0.4533
```



Question 2 Develop a neural network using PyTorch for Concrete Strength estimation.Given information about the concrete, such as the amount of cement, water,superplasticizer, etc, your model should estimate the strength of the concrete.Refer to Kaggle for access to the dataset.

```
data = pd.read_csv('concrete.csv')
data.head()
```

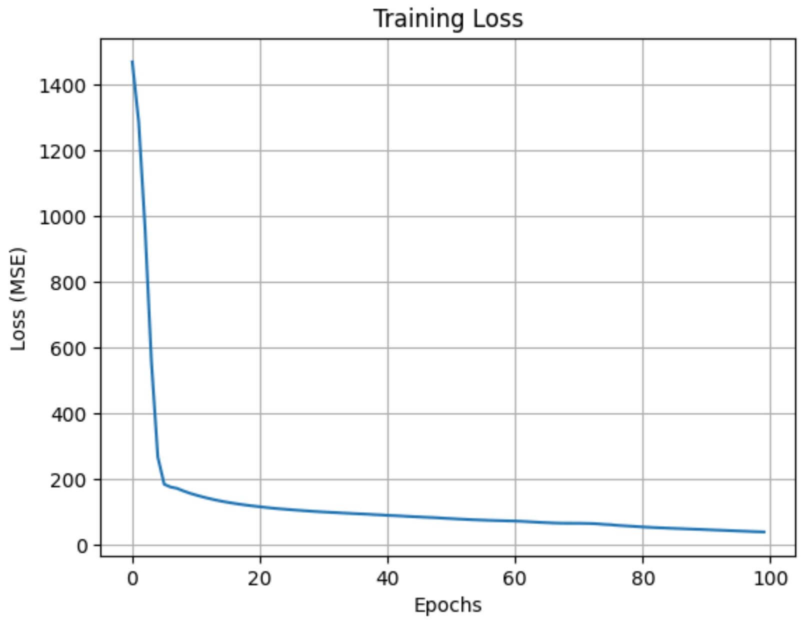
	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30

```
X = data.drop('Strength', axis=1).values
y = data['Strength'].values

scaler = StandardScaler()
X_standardized = scaler.fit_transform(X)

X_tensor = torch.tensor(X_standardized, dtype=torch.float32)
y_tensor = torch.tensor(y, dtype=torch.float32).view(-1, 1)

# Set random seed for reproducibility
torch.manual_seed(0)

# Split the dataset into train (70%) and test (30%) using PyTorch
def split_data(X, y, train_ratio=0.7):
    num_samples = X.size(0)
    num_train = int(num_samples * train_ratio)

    # Shuffle indices
    indices = torch.randperm(num_samples)
    train_indices = indices[:num_train]
    test_indices = indices[num_train:]

    return X[train_indices], y[train_indices], X[test_indices], y[test_indices]

# Split the data
X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor = split_data(X_tensor, y_tensor)

print("Training set shape:")
print(f"X_train: {X_train_tensor.shape} (Dimensions: {X_train_tensor.ndim})")
print(f"y_train: {y_train_tensor.shape} (Dimensions: {y_train_tensor.ndim})")

print("\nTest set shape:")
print(f"X_test: {X_test_tensor.shape} (Dimensions: {X_test_tensor.ndim})")
print(f"y_test: {y_test_tensor.shape} (Dimensions: {y_test_tensor.ndim})")

# Create DataLoader for batching
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
Training set shape:
X_train: torch.Size([721, 8]) (Dimensions: 2)
y_train: torch.Size([721, 1]) (Dimensions: 2)

Test set shape:
X_test: torch.Size([309, 8]) (Dimensions: 2)
y_test: torch.Size([309, 1]) (Dimensions: 2)
```

```
model = nn.Sequential(
    nn.Linear(8, 100),
    nn.ReLU(),
    nn.Linear(100, 50),
    nn.ReLU(),
    nn.Linear(50,1)
)

# loss function and optimizer
loss_fn = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training parameters
num_epochs = 100
batch_size = 32
losses = []

# Training loop
for epoch in range(num_epochs):
    model.train()
    for i in range(0, len(X_train_tensor), batch_size):
        X_batch = X_train_tensor[i:i + batch_size]
        y_batch = y_train_tensor[i:i + batch_size]


        y_pred = model(X_batch)
        loss = loss_fn(y_pred, y_batch)


        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    losses.append(loss.item())
    if (epoch + 1) % 10 == 0:  # Print every 10 epochs
        print(f"Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}")


#Evaluate
y_test_pred = model(X_test_tensor)

# Calculate MSE and MAE
mse = torch.mean((y_test_pred - y_test_tensor) ** 2).item()
mae = torch.mean(torch.abs(y_test_pred - y_test_tensor)).item()

print(f"Mean Squared Error (MSE) on test set: {mse:.4f}")
print(f"Mean Absolute Error (MAE) on test set: {mae:.4f}")

# Plot the training loss
plt.plot(losses)
plt.title("Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss (MSE)")
plt.grid()
plt.show()
```

```
Epoch [10/100], Loss: 156.4801
Epoch [20/100], Loss: 117.3853
Epoch [30/100], Loss: 100.4040
Epoch [40/100], Loss: 90.2560
Epoch [50/100], Loss: 80.0187
Epoch [60/100], Loss: 71.9928
Epoch [70/100], Loss: 64.8122
Epoch [80/100], Loss: 54.9363
Epoch [90/100], Loss: 46.2532
Epoch [100/100], Loss: 38.3753
Mean Squared Error (MSE) on test set: 43.2201
Mean Absolute Error (MAE) on test set: 5.0353
```




```
# Define the neural network model
def create_model():
    model = nn.Sequential(
        nn.Linear(8, 100),  # Input layer with 8 features
        nn.ReLU(),
        nn.Linear(100, 50),
        nn.ReLU(),
        nn.Linear(50, 1)  # Output layer for regression
    )
    return model

# Function to train the model and evaluate performance
def train_and_evaluate(lr, batch_size, num_epochs=100):
    model = create_model()
    loss_fn = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)

    losses = []
```

```
    # Training loop
    for epoch in range(num_epochs):

        for i in range(0, len(X_train_tensor), batch_size):
            X_batch = X_train_tensor[i:i + batch_size]
            y_batch = y_train_tensor[i:i + batch_size]

            # Forward pass
            y_pred = model(X_batch)
            loss = loss_fn(y_pred, y_batch)

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

        losses.append(loss.item())

    # Evaluate on the test set

    y_test_pred = model(X_test_tensor)

    mse = torch.mean((y_test_pred - y_test_tensor) ** 2).item()
    mae = torch.mean(torch.abs(y_test_pred - y_test_tensor)).item()

    return losses, mse, mae

# Hyperparameter grid
learning_rates = [0.1, 0.01, 0.001, 0.0001]
batch_sizes = [8, 16, 32, 64]
results = {}

for lr in learning_rates:
    for batch_size in batch_sizes:
        print(f"Training with Learning Rate: {lr}, Batch Size: {batch_size}")
        losses, mse, mae = train_and_evaluate(lr, batch_size)
        results[(lr, batch_size)] = {
            'losses': losses,
            'mse': mse,
            'mae': mae
        }


# Test Performance
for (lr, batch_size), result in results.items():
    print(f"Learning Rate: {lr}, Batch Size: {batch_size}, MSE: {result['mse']:.4f}, MAE: {result['mae']:.4f}")

# Results
plt.figure(figsize=(12, 8))
for (lr, batch_size), result in results.items():
    plt.plot(result['losses'], label=f"LR: {lr}, Batch: {batch_size}")

plt.title("Training Loss  for Different Hyperparameters")
plt.xlabel("Epochs")
```

```
plt.ylabel("Loss (MSE)")
plt.legend()
plt.grid()
plt.show()
```