sina2

May 25, 2024

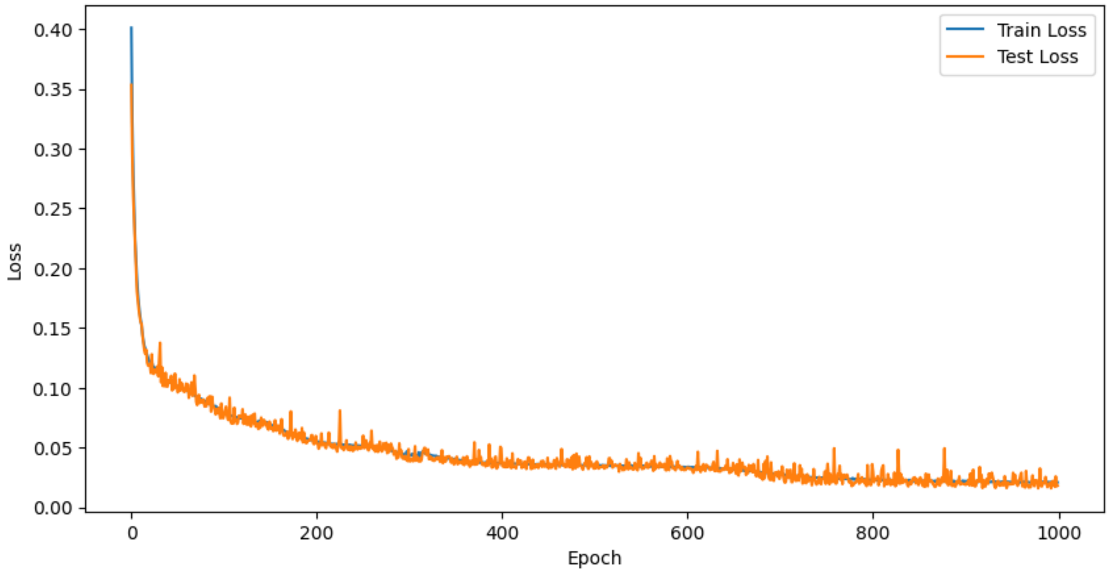
[2]:

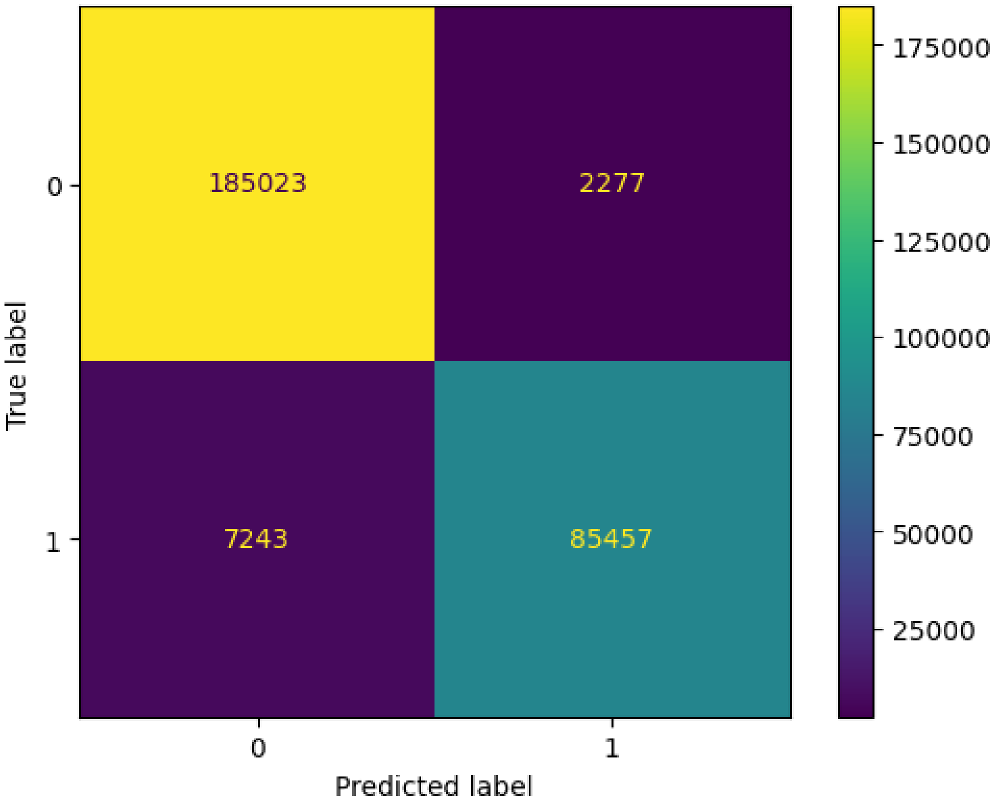
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np

# Check if GPU is available and set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Define the neural network model using nn.Sequential
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(25, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 14)
        )

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

# Custom dataset class
class SineWaveDataset(Dataset):
    def __init__(self, num_samples):
        x = torch.rand(num_samples) * 2 * torch.pi
        self.inputs = torch.stack([x + i * 0.1 for i in range(25)], dim=1)
        self.outputs = torch.stack([torch.sin((i+1) * x) for i in range(14)],␣
 ↪dim=1)

    def __len__(self):
        return len(self.inputs)
```

```python
    def __getitem__(self, idx):
        return self.inputs[idx], self.outputs[idx]

def train_and_evaluate(model, criterion, optimizer, train_loader, test_loader, num_epochs):
    train_losses = []
    test_losses = []
    for epoch in range(num_epochs):
        model.train()
        total_train_loss = 0
        for inputs, outputs in train_loader:
            inputs, outputs = inputs.to(device), outputs.to(device)
            optimizer.zero_grad()
            predictions = model(inputs)
            loss = criterion(predictions, outputs)
            loss.backward()
            optimizer.step()
            total_train_loss += loss.item()

        avg_train_loss = total_train_loss / len(train_loader)
        train_losses.append(avg_train_loss)

        # Evaluate on test set
        avg_test_loss, _, _ = evaluate(model, criterion, test_loader)
        test_losses.append(avg_test_loss)

        if (epoch + 1) % 100 == 0:
            print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {avg_train_loss:.4f}, Test Loss: {avg_test_loss:.4f}')

    return train_losses, test_losses

def evaluate(model, criterion, data_loader):
    model.eval()
    total_loss = 0
    all_outputs = []
    all_predictions = []
    with torch.no_grad():
        for inputs, outputs in data_loader:
            inputs, outputs = inputs.to(device), outputs.to(device)
            predictions = model(inputs)
            loss = criterion(predictions, outputs)
            total_loss += loss.item()
            all_outputs.append(outputs.cpu())
            all_predictions.append(predictions.cpu())

    avg_loss = total_loss / len(data_loader)
    all_outputs = torch.cat(all_outputs)
    all_predictions = torch.cat(all_predictions)
    return avg_loss, all_outputs, all_predictions

def plot_losses(train_losses, test_losses):
    plt.figure(figsize=(10, 5))
    plt.plot(train_losses, label='Train Loss')
    plt.plot(test_losses, label='Test Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

def plot_confusion_matrix(outputs, predictions, num_classes=14):
    outputs = (outputs > 0.5).int()
    predictions = (predictions > 0.5).int()
    cm = confusion_matrix(outputs.view(-1), predictions.view(-1), labels=[0, 1])
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
    disp.plot()
    plt.show()

# Parameters
num_samples = int(1e5)
num_epochs = 1000
batch_size = 32

# Initialaze the dataset and dataloader
train_dataset = SineWaveDataset(num_samples)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# Separate dataset for validation/testing
test_dataset = SineWaveDataset(int(num_samples * 0.2))
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Initialize the model, loss function, and optimizer
model = Net().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Train the model and track losses
train_losses, test_losses = train_and_evaluate(model, criterion, optimizer,␣
 ↪train_loader, test_loader, num_epochs)

print(f'Final Test Loss: {test_losses[-1]:.4f}')

# Plot training and test losses
plot_losses(train_losses, test_losses)
```

```
# Evaluate the model on the test dataset to get final outputs and predictions
_, test_outputs, test_predictions = evaluate(model, criterion, test_loader)

# Plot confusion matrix
plot_confusion_matrix(test_outputs, test_predictions)

# Save the model
torch.save(model.state_dict(), 'model.pth')
print('Model saved as model.pth')
```

```
Epoch [100/1000], Train Loss: 0.0789, Test Loss: 0.0744
Epoch [200/1000], Train Loss: 0.0559, Test Loss: 0.0597
Epoch [300/1000], Train Loss: 0.0435, Test Loss: 0.0487
Epoch [400/1000], Train Loss: 0.0371, Test Loss: 0.0495
Epoch [500/1000], Train Loss: 0.0349, Test Loss: 0.0323
Epoch [600/1000], Train Loss: 0.0336, Test Loss: 0.0332
Epoch [700/1000], Train Loss: 0.0273, Test Loss: 0.0237
Epoch [800/1000], Train Loss: 0.0235, Test Loss: 0.0361
Epoch [900/1000], Train Loss: 0.0218, Test Loss: 0.0193
Epoch [1000/1000], Train Loss: 0.0208, Test Loss: 0.0181
Final Test Loss: 0.0181
```





```
Model saved as model.pth
```