Program 1:

```
import torch, torch.nn as nn, torch.optim as optim


X = torch.tensor([[0.,0.], [0.,1.], [1.,0.], [1.,1.]])
y = torch.tensor([[0.], [1.], [1.], [0.]])

model = nn.Sequential(nn.Linear(2, 2), nn.Sigmoid(), nn.Linear(2, 1),
nn.Sigmoid())
opt, criterion = optim.SGD(model.parameters(), lr=0.5), nn.MSELoss()

for epoch in range(10000):
    opt.zero_grad(); loss = criterion(model(X), y); loss.backward();
opt.step()
if epoch % 1000 == 0: print(f"Epoch {epoch} Loss: {loss.item():.4f}")

print("Predictions:\n", model(X).detach())
```

Program 2:

```
import torch, torch.nn as nn, torch.optim as optim

data = [(torch.randn(32, 1, 28, 28), torch.randint(0, 62, (32,))) for _ in
range(10)]

class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(1, 32, 3), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3), nn.ReLU(), nn.MaxPool2d(2),
            nn.Flatten(), nn.Linear(64*5*5, 128), nn.ReLU(), nn.Linear(128,
62))
    def forward(self, x): return self.net(x)

model = CNN()
opt, criterion = optim.Adam(model.parameters()), nn.CrossEntropyLoss()


for epoch in range(5):
    for x, y in data:
        opt.zero_grad(); loss = criterion(model(x), y); loss.backward();
opt.step()
    print(f"Epoch {epoch+1} Loss: {loss.item():.4f}")
```

Program 3:

```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
from torchvision import transforms
from PIL import Image

img = Image.open("D:/vs/ml_gpu_env/xray.png").convert('L').resize((64, 64))
x = transforms.ToTensor()(img).unsqueeze(0)

model = nn.Sequential(
    nn.Conv2d(1, 16, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
    nn.Conv2d(16, 8, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
    nn.Conv2d(8, 16, 3, padding=1), nn.ReLU(), nn.Upsample(scale_factor=2),
    nn.Conv2d(16, 1, 3, padding=1), nn.Sigmoid(), nn.Upsample(scale_factor=2)
)
opt = torch.optim.Adam(model.parameters())

print("Training...")
for i in range(10):
    loss = ((x - model(x))**2).mean()
    opt.zero_grad(); loss.backward(); opt.step()
    print(f"Epoch {i+1}, Loss: {loss.item():.4f}")

plt.subplot(1,2,1); plt.imshow(x[0,0].detach(), cmap='gray');
plt.title("Original")
plt.subplot(1,2,2); plt.imshow(model(x)[0,0].detach(), cmap='gray');
plt.title("Reconstructed")
plt.show()
```

Program 4:

```
import torch
import torch.nn as nn
from torchvision import transforms
from PIL import Image

img = Image.open("D:/vs/ml_gpu_env/ter.png").convert('RGB')
x = transforms.Compose([transforms.Resize((224,224)),
transforms.ToTensor()])(img).unsqueeze(0)
y = torch.tensor([0])

model = nn.Sequential(
    nn.Conv2d(3, 16, 3), nn.ReLU(), nn.MaxPool2d(2),
    nn.Conv2d(16, 32, 3), nn.ReLU(), nn.MaxPool2d(2),
    nn.Flatten(), nn.Linear(32*54*54, 4)
)
opt = torch.optim.Adam(model.parameters())
loss_fn = nn.CrossEntropyLoss()

print("Training...")
for i in range(5):
    loss = loss_fn(model(x), y)
    opt.zero_grad(); loss.backward(); opt.step()
    print(f"Epoch {i+1}, Loss: {loss.item():.4f}")

print("Prediction Class:", torch.argmax(model(x)).item())
```

Program 5:

```
import torch, torch.nn as nn, torch.optim as optim
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('D:/vs/ml_gpu_env/creditcard.csv')
data = torch.tensor(df.drop('Class', axis=1).values, dtype=torch.float32)

model = nn.Sequential(nn.Linear(30, 10), nn.ReLU(), nn.Linear(10, 30))
opt, criterion = optim.Adam(model.parameters()), nn.MSELoss()

for epoch in range(10):
    opt.zero_grad(); loss = criterion(model(data), data); loss.backward();
opt.step()
    print(f"Epoch {epoch+1} Loss: {loss.item():.4f}")

with torch.no_grad():
    errors = torch.mean((data - model(data))**2, dim=1).numpy()
    threshold = torch.quantile(torch.tensor(errors), 0.95).item()

    plt.figure(figsize=(10,5))
    plt.plot(errors, label='Error')
    plt.axhline(threshold, color='r', linestyle='--', label='Threshold')
    plt.title(f"Fraud Detection (Threshold: {threshold:.4f})")
    plt.legend(); plt.show()
```

Program 6:

```
import torch, torch.nn as nn, torch.optim as optim

vocab = {'good': 1, 'bad': 2, 'happy': 3, 'sad': 4}
data = torch.tensor([[1], [2], [3], [4]])
labels = torch.tensor([1, 0, 1, 0])

class RNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.emb = nn.Embedding(5, 10)
        self.rnn = nn.LSTM(10, 20, batch_first=True)
        self.fc = nn.Linear(20, 2)
    def forward(self, x): return self.fc(self.rnn(self.emb(x))[1][0][-1])

model = RNN(); opt = optim.Adam(model.parameters(), lr=0.1)

for i in range(50):
    loss = nn.CrossEntropyLoss()(model(data), labels)
    opt.zero_grad(); loss.backward(); opt.step()
    if i % 10 == 0: print(f"Loss: {loss.item():.4f}")

print("Pred (Happy):", torch.argmax(model(torch.tensor([[3]]))).item())
```