Autoencoder_network_p4

October 22, 2025

Build and demonstrate an autoencoder network using neural layers for data compression on image dataset.

[2]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt

# ------------------------------
# 1. Load Dataset (MNIST)
# ------------------------------
transform = transforms.Compose([transforms.ToTensor()])

train_dataset = torchvision.datasets.MNIST(root='./data', train=True,
                                           download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=128, shuffle=True)

# ------------------------------
# 2. Define Autoencoder
# ------------------------------
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        # Encoder (compression)
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128),
            nn.ReLU(True),
            nn.Linear(128, 64),
            nn.ReLU(True),
            nn.Linear(64, 32)   # Compressed representation
        )
        # Decoder (reconstruction)
        self.decoder = nn.Sequential(
            nn.Linear(32, 64),
            nn.ReLU(True),
```

```
            nn.Linear(64, 128),
            nn.ReLU(True),
            nn.Linear(128, 28 * 28),
            nn.Sigmoid()   # output values between 0-1
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x

# ------------------------------
# 3. Initialize Model
# ------------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = Autoencoder().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# ------------------------------
# 4. Training
# ------------------------------
epochs = 10
for epoch in range(epochs):
    running_loss = 0.0
    for images, _ in train_loader:
        images = images.view(-1, 28*28).to(device)  # Flatten
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, images)  # Compare with original
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss/len(train_loader):.
 ↪4f}")

# ------------------------------
# 5. Demonstrate Compression & Reconstruction
# ------------------------------
# Get some test images
test_dataset = torchvision.datasets.MNIST(root='./data', train=False,
                                          download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=10, shuffle=True)

# Show original and reconstructed
model.eval()
```

```
with torch.no_grad():
    images, _ = next(iter(test_loader))
    images = images.view(-1, 28*28).to(device)
    outputs = model(images)

    # Move to CPU and reshape
    images = images.view(-1, 1, 28, 28).cpu()
    outputs = outputs.view(-1, 1, 28, 28).cpu()

    # Plot
    fig, axes = plt.subplots(2, 10, figsize=(10, 2))
    for i in range(10):
        # Original
        axes[0][i].imshow(images[i].squeeze(), cmap='gray')
        axes[0][i].axis("off")
        # Reconstructed
        axes[1][i].imshow(outputs[i].squeeze(), cmap='gray')
        axes[1][i].axis("off")
    plt.show()
```

```
Epoch 1/10, Loss: 0.0608
Epoch 2/10, Loss: 0.0311
Epoch 3/10, Loss: 0.0248
Epoch 4/10, Loss: 0.0214
Epoch 5/10, Loss: 0.0193
Epoch 6/10, Loss: 0.0176
Epoch 7/10, Loss: 0.0164
Epoch 8/10, Loss: 0.0154
Epoch 9/10, Loss: 0.0147
Epoch 10/10, Loss: 0.0140
```

[ ]: