Paralleling Neural Network with PyTorch

- a simpe network with synthetic dataset
- build input pipeline, model, loss function and optimizer

NN with Synthetic Data

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

```
# 1. Create synthetic data
X = torch.randn(100, 10)  # 100 samples, 10 features each
y = torch.randint(0, 2, (100,))  # 100 labels (binary classification: 0 or 1)
```

```
# 2. Define a simple neural network
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(10, 32)  # input layer (10) -> hidden layer (32)
        self.fc2 = nn.Linear(32, 2)   # hidden layer (32) -> output layer (2 classes)

    def forward(self, x):
        x = self.fc1(x)
        x = torch.relu(x) # define activation function
        x = self.fc2(x)
        return x

model = SimpleNN()
```

```
# 3. Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

```
# 4. Training loop
for epoch in range(20):  # train for 20 epochs
    outputs = model(X)
    loss = criterion(outputs, y)

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

    if (epoch + 1) % 5 == 0:
        print(f'Epoch [{epoch+1}/20], Loss: {loss.item():.4f}')
```

```
Epoch [5/20], Loss: 0.6076
Epoch [10/20], Loss: 0.5595
Epoch [15/20], Loss: 0.5218
Epoch [20/20], Loss: 0.4761
```

NN with large dataset

📚 FashionMNIST dataset size:

Training set: 60,000 images

Test set: 10,000 images

Each image is:

Size 28×28 pixels (grayscale, 1 channel)

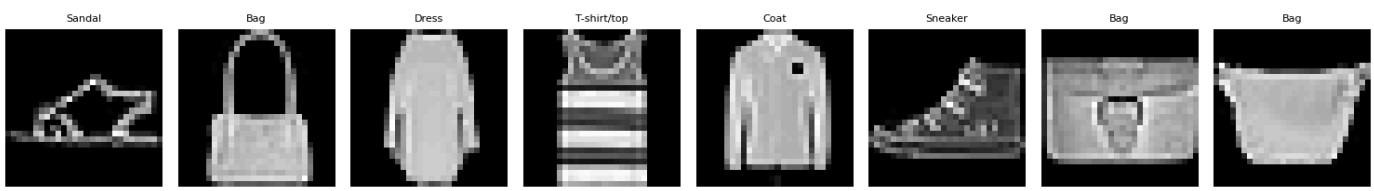
Belongs to 1 of 10 classes

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
```

```
# 1. Prepare dataset and dataloader
transform = transforms.ToTensor()

train_dataset = datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

```
100%|██████████| 26.4M/26.4M [00:01<00:00, 18.1MB/s]
100%|██████████| 29.5k/29.5k [00:00<00:00, 302kB/s]
100%|██████████| 4.42M/4.42M [00:00<00:00, 5.56MB/s]
100%|██████████| 5.15k/5.15k [00:00<00:00, 18.1MB/s]
```

```
# Visualize the first 8 images from FashionMNIST
import matplotlib.pyplot as plt

classes = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
           'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Get one batch
images, labels = next(iter(train_loader))

# Plot the first 8 images
fig, axes = plt.subplots(1, 8, figsize=(15, 2))
for idx in range(8):
    img = images[idx].squeeze()  # remove channel dimension (1,28,28) → (28,28)
    label = labels[idx].item()
    axes[idx].imshow(img, cmap='gray')
    axes[idx].set_title(classes[label], fontsize=8)
    axes[idx].axis('off')
plt.tight_layout()
plt.show()
```



```
# 2. Define a simple neural network
class FashionNN(nn.Module):
    def __init__(self):
        super(FashionNN, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)  # 10 classes

    def forward(self, x):
        x = x.view(-1, 28*28)  # Flatten the image
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

model = FashionNN()
```

```
# 3. Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
# 4. Training loop
for epoch in range(5):
    running_loss = 0.0
    for images, labels in train_loader:
        outputs = model(images)
        loss = criterion(outputs, labels)

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

        running_loss += loss.item()  # add batch loss to total

    avg_loss = running_loss / len(train_loader)  # average loss per batch
    print(f'Epoch [{epoch+1}/5], Average Loss: {avg_loss:.4f}')
```

```
Epoch [1/5], Average Loss: 0.5566
Epoch [2/5], Average Loss: 0.3883
Epoch [3/5], Average Loss: 0.3466
Epoch [4/5], Average Loss: 0.3234
Epoch [5/5], Average Loss: 0.3072
```

```
# 5. Save the model
torch.save(model.state_dict(), 'fashion_mnist_model.pth')
print("Model saved!")
```

```
Model saved!
```

```
# 6. Load the model
model2 = FashionNN()
model2.load_state_dict(torch.load('fashion_mnist_model.pth'))
model2.eval()
print("Model reloaded and ready!")
```

```
Model reloaded and ready!
```

Your Work

MNIST is the simplest dataset in torchvision.datasets. Please replace FashionMNIST with MNIST for above codes and play around it with your parameters. Submit the notebook in PDF to BS by 5/11 11:59 pm.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
```

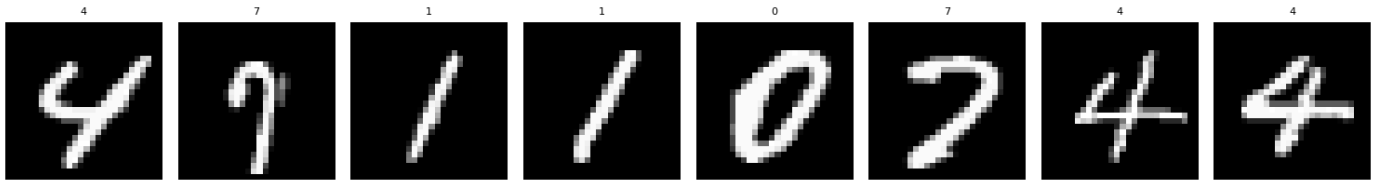
```
# 1. Prepare dataset and dataloader
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False, num_workers=2)
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 50.2MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 1.62MB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 14.7MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 5.91MB/s]
```

```
# Visualize the first 8 images
classes = [str(i) for i in range(10)]
```

```
# Get one batch
images, labels = next(iter(train_loader))
```

```
# Plot the first 8 images
fig, axes = plt.subplots(1, 8, figsize=(15, 2))
for idx in range(8):
    img = images[idx].squeeze()
    label = labels[idx].item()
    axes[idx].imshow(img, cmap='gray')
    axes[idx].set_title(classes[label], fontsize=8)
    axes[idx].axis('off')
plt.tight_layout()
plt.show()
```



```
# 2. Define a neural network with dropout for regularization
class MNIST_NN(nn.Module):
    def __init__(self):
        super(MNIST_NN, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = torch.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x

model = MNIST_NN()
```

```
# 3. Loss and optimizer with weight decay (L2 regularization)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)
```

```
# 4. Training loop with validation
for epoch in range(10):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in train_loader:
        outputs = model(images)
        loss = criterion(outputs, labels)

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

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    train_loss = running_loss / len(train_loader)
    train_acc = 100 * correct / total

    # Validation
    model.eval()
    val_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for images, labels in test_loader:
            outputs = model(images)
            val_loss += criterion(outputs, labels).item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    val_loss /= len(test_loader)
    val_acc = 100 * correct / total

    print(f'Epoch [{epoch+1}/10], Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%, Val Loss: {val_loss:.
```

```
Epoch [1/10], Train Loss: 0.3476, Train Acc: 89.54%, Val Loss: 0.1371, Val Acc: 95.69%
Epoch [2/10], Train Loss: 0.1693, Train Acc: 94.95%, Val Loss: 0.1051, Val Acc: 96.86%
```

```
Epoch [3/10], Train Loss: 0.1318, Train Acc: 96.07%, Val Loss: 0.0965, Val Acc: 97.00%
Epoch [4/10], Train Loss: 0.1139, Train Acc: 96.54%, Val Loss: 0.0837, Val Acc: 97.46%
Epoch [5/10], Train Loss: 0.0997, Train Acc: 96.94%, Val Loss: 0.0821, Val Acc: 97.57%
Epoch [6/10], Train Loss: 0.0915, Train Acc: 97.17%, Val Loss: 0.0863, Val Acc: 97.61%
Epoch [7/10], Train Loss: 0.0863, Train Acc: 97.29%, Val Loss: 0.0805, Val Acc: 97.69%
Epoch [8/10], Train Loss: 0.0800, Train Acc: 97.48%, Val Loss: 0.0806, Val Acc: 97.61%
Epoch [9/10], Train Loss: 0.0742, Train Acc: 97.61%, Val Loss: 0.0788, Val Acc: 97.84%
Epoch [10/10], Train Loss: 0.0725, Train Acc: 97.73%, Val Loss: 0.0747, Val Acc: 97.82%
```

```
# 5. Save the model
torch.save({
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
}, 'mnist_model.pth')
print("Model saved!")
```

```
Model saved!
```

```
# 6. Load the model
model2 = MNIST_NN()
checkpoint = torch.load('mnist_model.pth')
model2.load_state_dict(checkpoint['model_state_dict'])
model2.eval()
print("Model reloaded and ready!")
```

```
Model reloaded and ready!
```