CyArt

Project Structure

```
pytorch_foundation/
|— 01_intro/
|   └── intro_tensor.py
|— 02_tensors/
|   └── tensor_ops.py
|— 03_autograd/
|   └── autograd_demo.py
|— 04_nn_basics/
|   └── mlp_toy.py
|— 05_datasets_dataloaders/
|   └── mnist_dataloader.py
|— 06_training_loops/
|   └── training_loop.py
|— 07_cnns/
|   └── cnn_mnist.py
|— 08_rnns/
|   └── rnn_text_generation.py
|— 09_advanced/
|    ├── transfer_learning.py
|   └── transformer_demo.py
|— 10_deployment/
|   └── save_load_torchscript.py
|— README.md
```

Example

01_intro/intro_tensor.py

```
import torch

# Create a tensor
x = torch.tensor([1, 2, 3])
print("Tensor:", x)

# Check attributes
print("Shape:", x.shape)
print("Dtype:", x.dtype)
print("Device:", x.device)
```

------- Expected Output Screenshot --------

```
Tensor: tensor([1, 2, 3])
Shape: torch.Size([3])
Dtype: torch.int64
Device: cpu
```

03_autograd/autograd_demo.py

```
import torch

x = torch.tensor(2.0, requires_grad=True)
y = x ** 2 + 3 * x + 1

y.backward()  # dy/dx
print("Gradient at x=2:", x.grad)
```

----------- Expected Output Screenshot -------------

```
Gradient at x=2: tensor(7.)
```

04_nn_basics/mlp_toy.py

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

# Simple dataset
X = torch.randn(100, 2)
y = (X[:, 0] + X[:, 1] > 0).float().unsqueeze(1)

# Define model
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(2, 16),
            nn.ReLU(),
            nn.Linear(16, 1),
            nn.Sigmoid()
        )
    def forward(self, x): return self.net(x)

model = MLP()
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Training loop
for epoch in range(10):
    optimizer.zero_grad()
    outputs = model(X)
    loss = criterion(outputs, y)
    loss.backward()
    optimizer.step()
    print(f"Epoch {epoch+1}: Loss = {loss.item():.4f}")
```

---------- Expected Output Screenshot ------------

```
Epoch 1: Loss = 0.6823
Epoch 2: Loss = 0.6471
...
Epoch 10: Loss = 0.3892
```

07_cnns/cnn_mnist.py

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

# MNIST dataset
train_data = datasets.MNIST(root="./data", train=True, transform=transforms.ToTensor(), download=True)
train_loader = torch.utils.data.DataLoader(train_data, batch_size=64, shuffle=True)

# CNN Model
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(1, 32, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.fc = nn.Sequential(
            nn.Linear(32 * 14 * 14, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )
    def forward(self, x):
        x = self.conv(x)
        x = x.view(x.size(0), -1)
        return self.fc(x)

model = CNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# One epoch training
for batch, (images, labels) in enumerate(train_loader):
    optimizer.zero_grad()
    outputs = model(images)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    if batch % 100 == 0:
        print(f"Batch {batch}, Loss = {loss.item():.4f}")
```

--------- Expected Output Screenshot ------------------

```
Batch 0, Loss = 2.3019
Batch 100, Loss = 0.7212
Batch 200, Loss = 0.3528
...
```

```
Tensor: tensor([1, 2, 3]
Shape: torch.Size([3]
Dtype: torch.int64
Device: cpu
```