Module 2 - Session 4: Training and Evaluating Your Classifier

Session 4

Setting Up Training

Everything you need:

- Model (on device)
- Loss function
- Optimizer
- Data loaders

Device and Model Setup

```python
device = torch.device('cuda' if torch.cuda.is_available()
                      else 'cpu')
model = MNISTClassifier().to(device)
```

Both model and data must be on the same device

Loss Function

```python
loss_function = nn.CrossEntropyLoss()
```

Designed for classification tasks

Perfect for choosing a digit from 0-9

Optimizer

```
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Adam: adapts learning rate as it trains

- Larger adjustments early (noisy gradients)
- Smaller corrections later (training stabilizes)

Training Function

```python
def train_one_epoch(model, dataloader, loss_fn, optimizer, device):
    model.train()  # Set to training mode
    running_loss = 0.0
    correct = 0
    total = 0

    for batch_idx, (data, targets) in enumerate(dataloader):
        # Move to device
        data, targets = data.to(device), targets.to(device)

        # Training steps
        optimizer.zero_grad()
        outputs = model(data)
        loss = loss_fn(outputs, targets)
        loss.backward()
        optimizer.step()

        # Track progress
        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()

        if batch_idx % 100 == 0:
            print(f'Loss: {loss.item():.4f}, '
```

Understanding the Training Progress

With 60,000 training images and batch size 64:

- About 938 batches per epoch
- Around 9 progress updates

Watch the numbers:

- Loss: dropping (0.64 → 0.17)
- Accuracy: climbing (81% → 95%)

Evaluation Function

```python
def evaluate(model, dataloader, device):
    model.eval()  # Set to evaluation mode
    correct = 0
    total = 0

    with torch.no_grad():  # Disable gradient tracking
        for data, targets in dataloader:
            data, targets = data.to(device), targets.to(device)
            outputs = model(data)
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

    return 100. * correct / total
```

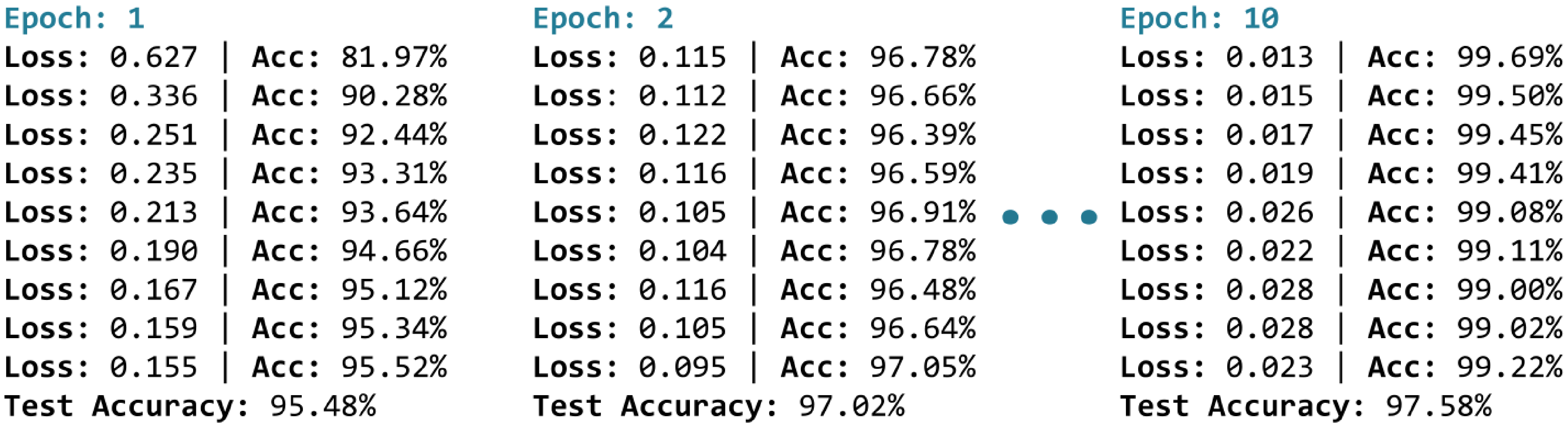
Putting It All Together

```python
num_epochs = 10

for epoch in range(num_epochs):
    print(f'Epoch {epoch+1}/{num_epochs}')

    # Train
    train_one_epoch(model, train_loader, loss_function,
                    optimizer, device)

    # Evaluate
    accuracy = evaluate(model, test_loader, device)
    print(f'Test Accuracy: {accuracy:.2f}%')
    print('-' * 50)
```

10 epochs = 10 full passes through training data

After each epoch: evaluate on test set

What You’ll See



```
Epoch: 1                     Epoch: 2                            Epoch: 10
Loss: 0.627 | Acc: 81.97%    Loss: 0.115 | Acc: 96.78%           Loss: 0.013 | Acc: 99.69%
Loss: 0.336 | Acc: 90.28%    Loss: 0.112 | Acc: 96.66%           Loss: 0.015 | Acc: 99.50%
Loss: 0.251 | Acc: 92.44%    Loss: 0.122 | Acc: 96.39%           Loss: 0.017 | Acc: 99.45%
Loss: 0.235 | Acc: 93.31%    Loss: 0.116 | Acc: 96.59%           Loss: 0.019 | Acc: 99.41%
Loss: 0.213 | Acc: 93.64%    Loss: 0.105 | Acc: 96.91%   • • •   Loss: 0.026 | Acc: 99.08%
Loss: 0.190 | Acc: 94.66%    Loss: 0.104 | Acc: 96.78%           Loss: 0.022 | Acc: 99.11%
Loss: 0.167 | Acc: 95.12%    Loss: 0.116 | Acc: 96.48%           Loss: 0.028 | Acc: 99.00%
Loss: 0.159 | Acc: 95.34%    Loss: 0.105 | Acc: 96.64%           Loss: 0.028 | Acc: 99.02%
Loss: 0.155 | Acc: 95.52%    Loss: 0.095 | Acc: 97.05%           Loss: 0.023 | Acc: 99.22%
Test Accuracy: 95.48%        Test Accuracy: 97.02%               Test Accuracy: 97.58%
```

By epoch 10: Loss: tiny, Accuracy: high (often 95%+)

When accuracy stops improving: Model is done learning, May not need all 10 epochs

Module 2 Summary

You’ve learned:

- PyTorch data pipeline (Dataset, DataLoader, Transforms)
- Building custom models with `nn.Module`
- Loss functions (MSE vs Cross-Entropy)
- How optimizers use gradients to update weights
- Device management (CPU vs GPU)
- Complete image classification pipeline

Lab 1: Building Your First Image Classifier

> “Don’t tell me the moon is shining; show me the glint of light on broken glass.”

CUE: START THE LAB HERE

Assignment 2: EMNIST Letter Detective

> “The eye sees only what the mind is prepared to comprehend.”

CUE: START THE ASSIGNMENT HERE

What’s Next?

In **Module 3: Data Management** we learn:

- How to manage and preprocess data for deep learning
- How to build a robust data pipeline
- How to use data augmentation to improve model performance
- How to use data validation to ensure model performance