1_lenet

April 1, 2025

Deadline: January 22, 2025 (Wednesday) 23:00

1 Exercise 1. Convolutional neural networks. LeNet-5.

In this exercise, you will train a very simple convolutional neural network used for image classification tasks.

You may find it useful to look at this tutorial: * Neural Networks

[26]:
```
skip_training = True  # Set this flag to True before validation and submission
```

[2]:
```
# During evaluation, this cell sets skip_training to True
# skip_training = True

import tools, warnings
warnings.showwarning = tools.customwarn
```

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import torch
import torchvision
import torchvision.transforms as transforms

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import tools
import tests
```

[4]:
```
# When running on your own computer, you can specify the data directory by:
# data_dir = tools.select_data_dir('/your/local/data/directory')
data_dir = tools.select_data_dir()
```

```
The data directory is /coursedata
```

[5]:
```
# Select the device for training (use GPU if you have one)
#device = torch.device('cuda:0')
device = torch.device('cpu')
```

[6]:
```
if skip_training:
    # The models are always evaluated on CPU
    device = torch.device("cpu")
```

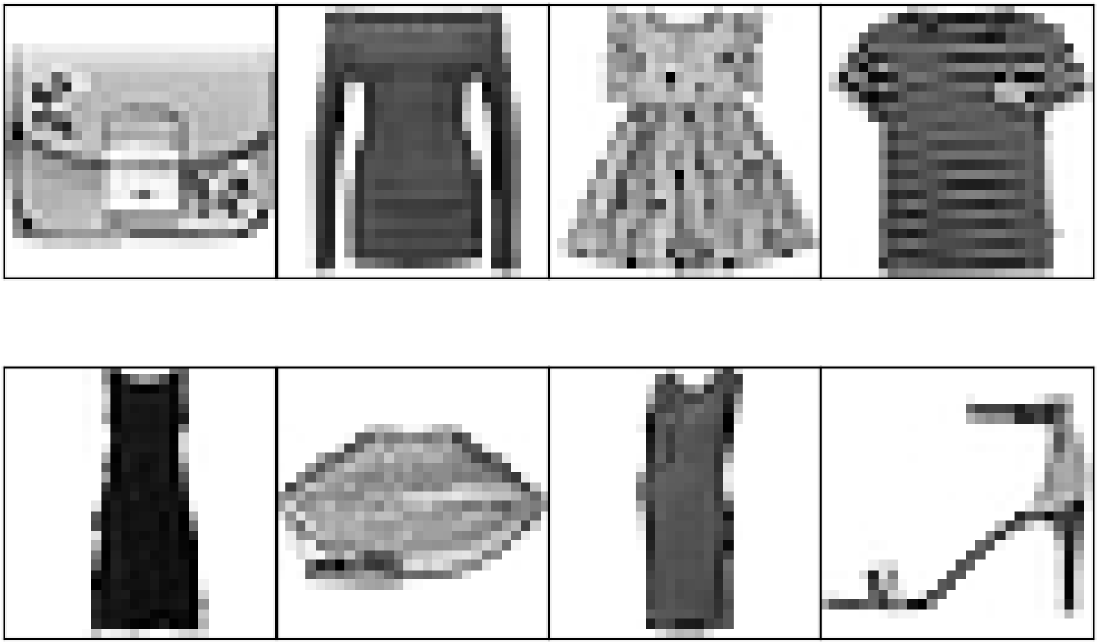
1.1 FashionMNIST dataset

Let us use the FashionMNIST dataset. It consists of 60,000 training images of 10 classes: 'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'.

[7]:
```
transform = transforms.Compose([
    transforms.ToTensor(),  # Transform to tensor
    transforms.Normalize((0.5,), (0.5,))  # Scale images to [-1, 1]
])

trainset = torchvision.datasets.FashionMNIST(root=data_dir, train=True,␣
 ↪download=True, transform=transform)
testset = torchvision.datasets.FashionMNIST(root=data_dir, train=False,␣
 ↪download=True, transform=transform)

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

trainloader = torch.utils.data.DataLoader(trainset, batch_size=32, shuffle=True)
testloader = torch.utils.data.DataLoader(testset, batch_size=5, shuffle=False)
```

Let us visualize the data.

[8]:
```
images, labels = next(iter(trainloader))
tests.plot_images(images[:8], n_rows=2)
```



2 1. Simple convolutional network

In the first exercise, your task is to create a convolutional neural network with the architecture inspired by the classical LeNet-5 (LeCun et al., 1998).

The architecture of the convolutional network that you need to create: * 2d convolutional layer with: * one input channel * 6 output channels * kernel size 5 (no padding) * followed by ReLU * Max-pooling layer with kernel size 2 and stride 2 * 2d convolutional layer with: * 16 output channels * kernel size 5 (no padding) * followed by ReLU * Max-pooling layer with kernel size 2 and stride 2 * A fully-connected layer with: * 120 outputs * followed by ReLU * A fully-connected layer with: * 84 outputs * followed by ReLU * A fully-connected layer with 10 outputs and without nonlinearity.

[9]:
```
class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5, self).__init__()
        # YOUR CODE HERE
        # Convolutional layers
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)  # Input: 1 channel, Output: 6 channels
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)  # Max pooling 1

        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)  # Input: 6 channels, Output: 16 channels
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)  # Max pooling 2
```

```
        # Fully connected layers
        self.fc1 = nn.Linear(16 * 4 * 4, 120)  # Input size: 16 * 4 * 4, Output␣
↪size: 120
        self.fc2 = nn.Linear(120, 84)  # Input size: 120, Output size: 84
        self.fc3 = nn.Linear(84, 10)  # Input size: 84, Output size: 10


    def forward(self, x):
        """
        Args:
          x of shape (batch_size, 1, 28, 28): Input images.

        Returns:
          y of shape (batch_size, 10): Outputs of the network.
        """
        # YOUR CODE HERE
        # Convolutional layers with ReLU and MaxPooling
        x = F.relu(self.conv1(x))  # Conv1 + ReLU
        x = self.pool1(x)  # Pool1
        x = F.relu(self.conv2(x))  # Conv2 + ReLU
        x = self.pool2(x)  # Pool2

        # Flatten the output for fully connected layers
        x = torch.flatten(x, 1)  # Flatten to (batch_size, 16 * 4 * 4)

        # Fully connected layers
        x = F.relu(self.fc1(x))  # Fully Connected 1 + ReLU
        x = F.relu(self.fc2(x))  # Fully Connected 2 + ReLU
        x = self.fc3(x)  # Fully Connected 3 (No activation for logits)
        return x
```

[10]:
```
def test_LeNet5_shapes():
    net = LeNet5()

    # Feed a batch of images from the training data to test the network
    with torch.no_grad():
        images, labels = next(iter(trainloader))
        print('Shape of the input tensor:', images.shape)

        y = net(images)
        assert y.shape == torch.Size([trainloader.batch_size, 10]), "Bad shape␣
↪of y: y.shape={}".format(y.shape)

    print('Success')

test_LeNet5_shapes()
```

```
Shape of the input tensor: torch.Size([32, 1, 28, 28])
Success
```

[11]:
```
def test_LeNet5():
    net = LeNet5()

    # get gradients for parameters in forward path
    net.zero_grad()
    x = torch.randn(1, 1, 28, 28)
    outputs = net(x)
    outputs[0,0].backward()

    parameter_shapes = sorted(tuple(p.shape) for p in net.parameters() if p.
 ↪grad is not None)
    print(parameter_shapes)
    expected = [(6,), (6, 1, 5, 5), (10,), (10, 84), (16,), (16, 6, 5, 5),␣
 ↪(84,), (84, 120), (120,), (120, 256)]
    assert parameter_shapes == expected, "Wrong number of training parameters."

    print('Success')

test_LeNet5()
```

```
[(6,), (6, 1, 5, 5), (10,), (10, 84), (16,), (16, 6, 5, 5), (84,), (84, 120),
(120,), (120, 256)]
Success
```

3 Train the network

[12]:
```
# This function computes the accuracy on the test dataset
def compute_accuracy(net, testloader):
    net.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in testloader:
            images, labels = images.to(device), labels.to(device)
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return correct / total
```

3.0.1 Training loop

Your task is to implement the training loop. The recommended hyperparameters: * Stochastic Gradient Descent (SGD) optimizer with learning rate 0.001 and momentum 0.9. * Cross-entropy loss. Note that we did not use softmax nonlinearity in the final layer of our network. Therefore, we need to use a loss function with log_softmax implemented, such as nn.CrossEntropyLoss. * Number of epochs: 10. Please use mini-batches produces by `trainloader` defined above.

We recommend you to use function `compute_accuracy()` defined above to track the accuracy during training. The test accuracy should be above 0.87.

[13]:
```
# Create network
net = LeNet5()
```

[21]:
```
# Implement the training loop in this cell
if not skip_training:
    # YOUR CODE HERE
    # Hyperparameters
    learning_rate = 0.001
    momentum = 0.9
    num_epochs = 10

    # Loss function and optimizer
    criterion = nn.CrossEntropyLoss()  # Cross-entropy loss for classification
    optimizer = optim.SGD(net.parameters(), lr=learning_rate,␣
 ↪momentum=momentum)  # SGD optimizer

    # Move the model to the appropriate device
    net.to(device)

    # Training loop
    for epoch in range(num_epochs):
        net.train()  # Set the network to training mode
        running_loss = 0.0

        for inputs, labels in trainloader:
            # Move data to the correct device
            inputs, labels = inputs.to(device), labels.to(device)

            # Zero the parameter gradients
            optimizer.zero_grad()

            # Forward pass
            outputs = net(inputs)

            # Compute the loss
            loss = criterion(outputs, labels)

            # Backward pass
            loss.backward()

            # Update the parameters
```

```
            optimizer.step()

            # Accumulate the loss
            running_loss += loss.item()

        # Compute accuracy after each epoch
        accuracy = compute_accuracy(net, testloader)

        # Print statistics for the epoch
        print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {running_loss /␣
 ↪len(trainloader):.4f}, Accuracy: {accuracy:.4f}")

else:
    print("Training skipped.")
```

```
Epoch 1/10, Loss: 0.3123, Accuracy: 0.8789
Epoch 2/10, Loss: 0.3024, Accuracy: 0.8778
Epoch 3/10, Loss: 0.2953, Accuracy: 0.8759
Epoch 4/10, Loss: 0.2876, Accuracy: 0.8799
Epoch 5/10, Loss: 0.2808, Accuracy: 0.8816
Epoch 6/10, Loss: 0.2746, Accuracy: 0.8777
Epoch 7/10, Loss: 0.2686, Accuracy: 0.8871
Epoch 8/10, Loss: 0.2621, Accuracy: 0.8792
Epoch 9/10, Loss: 0.2585, Accuracy: 0.8849
Epoch 10/10, Loss: 0.2539, Accuracy: 0.8877
```

[22]:
```
# Save the model to disk (the pth-files will be submitted automatically␣
 ↪together with your notebook)
# Set confirm=False if you do not want to be asked for confirmation before␣
 ↪saving.
if not skip_training:
    tools.save_model(net, '1_lenet5.pth', confirm=True)
```

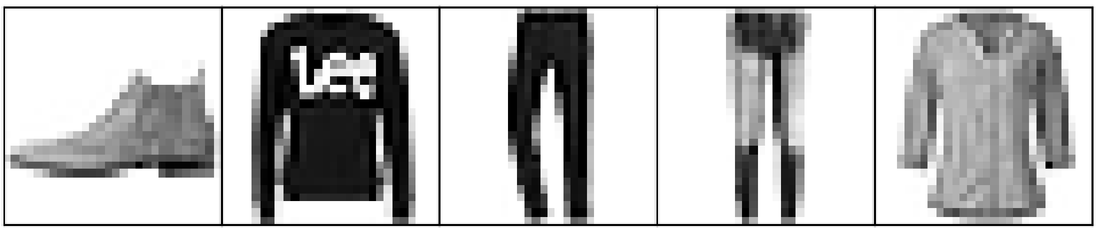
```
Do you want to save the model (type yes to confirm)?  yes

Model saved to 1_lenet5.pth.
```

[23]:
```
if skip_training:
    net = LeNet5()
    tools.load_model(net, '1_lenet5.pth', device)
```

[24]:
```
# Display random images from the test set, the ground truth labels and the␣
 ↪network's predictions
net.eval()
with torch.no_grad():
    images, labels = next(iter(testloader))
    tests.plot_images(images[:5], n_rows=1)
```

```
    # Compute predictions
    images = images.to(device)
    y = net(images)

print('Ground truth labels: ', ' '.join('%10s' % classes[labels[j]] for j in range(5)))
print('Predictions:         ', ' '.join('%10s' % classes[j] for j in y.argmax(dim=1)))
```

```
Ground truth labels:  Ankle boot   Pullover    Trouser    Trouser      Shirt
Predictions:          Ankle boot   Pullover    Trouser    Trouser      Shirt
```



[25]:
```
# Compute the accuracy on the test set
accuracy = compute_accuracy(net, testloader)
print('Accuracy of the network on the test images: %.3f' % accuracy)
assert accuracy > 0.85, "Poor accuracy {:.3f}".format(accuracy)
print('Success')
```

```
Accuracy of the network on the test images: 0.888
Success
```