handson

December 20, 2023

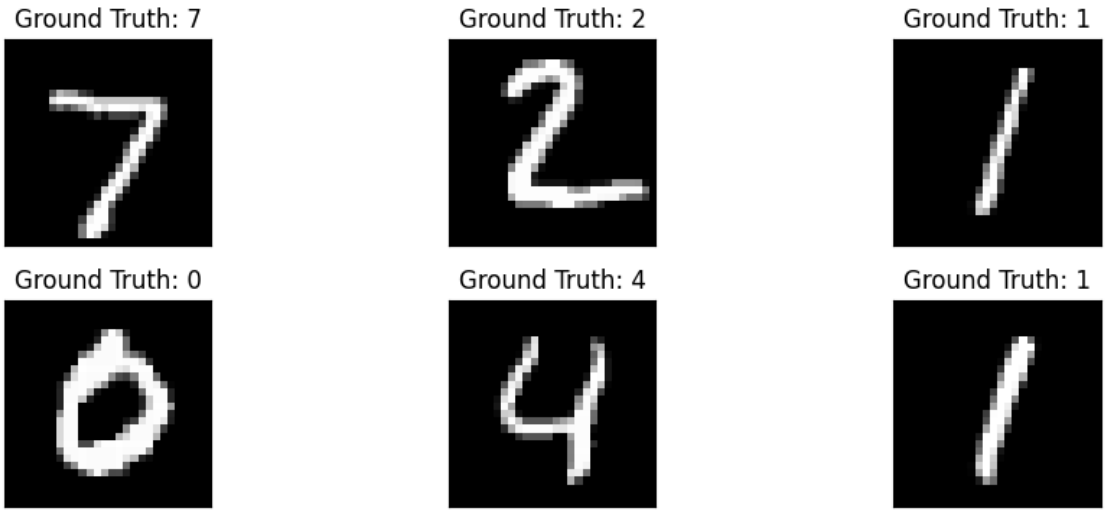
[75]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import os
import pickle
```

[76]:
```
# Transformations applied on each image => converting them to tensor and␣
 ↪normalizing
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.1307,), (0.3081,))])

def load_dataset(train=True):
    filename = './data/mnist_train.pkl' if train else './data/mnist_test.pkl'

    # Check if the dataset is already saved in pickle file
    if os.path.exists(filename):
        print(f"Loading from {filename}")
        with open(filename, 'rb') as f:
            dataset = pickle.load(f)
    else:
        print(f"Downloading and saving to {filename}")
        dataset = datasets.MNIST(root='./data', train=train,␣
 ↪transform=transform, download=True)
        os.makedirs(os.path.dirname(filename), exist_ok=True)
        with open(filename, 'wb') as f:
            pickle.dump(dataset, f)

    return dataset

# Loading datasets
train_dataset = load_dataset(train=True)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)

test_dataset = load_dataset(train=False)
```

```
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

```
Loading from ./data/mnist_train.pkl
Loading from ./data/mnist_test.pkl
```

[77]:
```
examples = enumerate(test_loader)
batch_idx, (example_data, example_targets) = next(examples)

fig = plt.figure(figsize=(10, 4))
for i in range(6):
    plt.subplot(2, 3, i + 1)
    plt.tight_layout()
    plt.imshow(example_data[i][0], cmap='gray', interpolation='none')
    plt.title(f"Ground Truth: {example_targets[i]}")
    plt.xticks([])
    plt.yticks([])
plt.show()
```



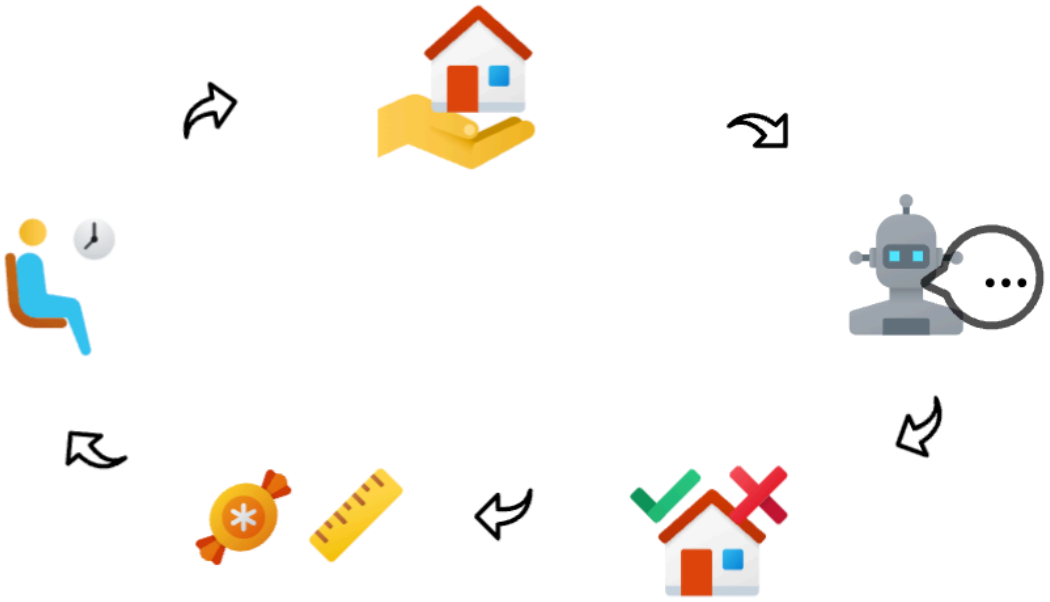
[78]:
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 200) ## Input Image: 28*28; Neuronen: 200
        self.fc2 = nn.Linear(200, 10) ## Neuronen: 200; Output Klassen: 10

    def forward(self, x):
        x = x.view(-1, 28*28) ## Mache aus 28*28 Bild einen 1-dim Vektor
        x = torch.relu(self.fc1(x)) ## Stecke 1-dim Vektor in Neuronenlayer
        x = self.fc2(x) ## Stecke Output von ersten Neuronenlayer in zweiten␣
↪Neuronenlayer
```

```
        return x

model = Net()
criterion = nn.CrossEntropyLoss() ## Kostenfunktion (Y-Ŷ)
optimizer = optim.Adam(model.parameters(), lr=0.001) ## Bieger der Funktion für␣
 ↪bessere Resultate bei Kostenfunktion
```

Bis jetzt ist KI dumm... Was müssen wir also machen?

[79]:
```
from IPython.display import Image
Image(filename='cycle.png')
```

[79]:



[80]:
```
def train(epoch):
    model.train()
    train_loss = 0
    correct = 0
    for data, target in train_loader: # Wir gehen über alle Daten Bild für Bild
        optimizer.zero_grad()
        output = model(data) ## Gebe Bild hinein
        loss = criterion(output, target) ## Y - Ŷ

        train_loss += loss.item()
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()
```

```
        loss.backward()
        optimizer.step() ## Update die Parameter des Models
    train_loss /= len(train_loader.dataset)
    train_accuracy = 100. * correct / len(train_loader.dataset)
    print(f'Train Epoch: {epoch} \tLoss: {train_loss:.6f} \tAccuracy:␣
 ↪{train_accuracy:.2f}%')
    return train_loss, train_accuracy

def test():
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            output = model(data)
            test_loss += criterion(output, target).item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    test_accuracy = 100. * correct / len(test_loader.dataset)
    print(f'\nTest set: Average loss: {test_loss:.4f}, Accuracy: {correct}/
 ↪{len(test_loader.dataset)} ({test_accuracy:.0f}%)\n')
    return test_loss, test_accuracy
```

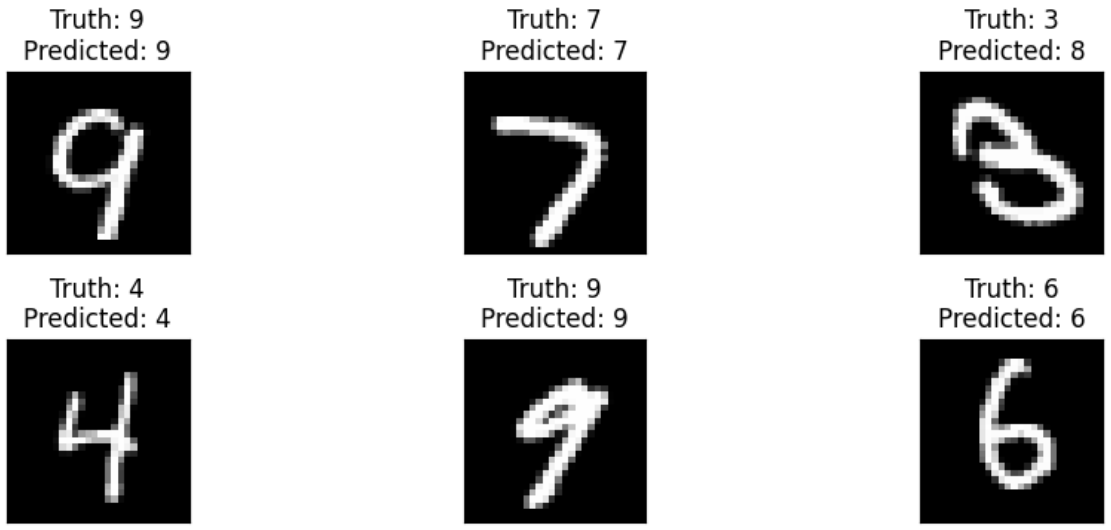
[81]:
```
epochs = 10
train_losses, train_accuracies = [], []
test_losses, test_accuracies = [], []

for epoch in range(1, epochs + 1):
    train_loss, train_accuracy = train(epoch)
    test_loss, test_accuracy = test()
    train_losses.append(train_loss)
    train_accuracies.append(train_accuracy)
    test_losses.append(test_loss)
    test_accuracies.append(test_accuracy)
```

```
Train Epoch: 1  Loss: 0.012488  Accuracy: 93.84%

Test set: Average loss: 0.0068, Accuracy: 9662/10000 (97%)
Train Epoch: 2  Loss: 0.006039  Accuracy: 97.05%

Test set: Average loss: 0.0056, Accuracy: 9739/10000 (97%)
Train Epoch: 3  Loss: 0.004292  Accuracy: 97.79%

Test set: Average loss: 0.0064, Accuracy: 9706/10000 (97%)
Train Epoch: 4  Loss: 0.003504  Accuracy: 98.19%
```

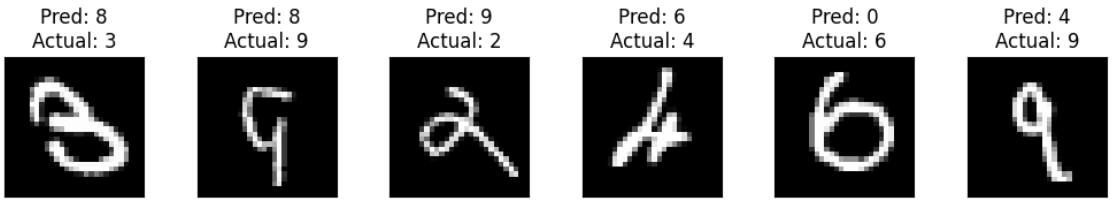
```
Test set: Average loss: 0.0054, Accuracy: 9771/10000 (98%)
Train Epoch: 5  Loss: 0.002924  Accuracy: 98.52%

Test set: Average loss: 0.0063, Accuracy: 9746/10000 (97%)
Train Epoch: 6  Loss: 0.002602  Accuracy: 98.66%

Test set: Average loss: 0.0066, Accuracy: 9739/10000 (97%)
Train Epoch: 7  Loss: 0.002202  Accuracy: 98.89%

Test set: Average loss: 0.0062, Accuracy: 9769/10000 (98%)
Train Epoch: 8  Loss: 0.002078  Accuracy: 98.91%

Test set: Average loss: 0.0067, Accuracy: 9762/10000 (98%)
Train Epoch: 9  Loss: 0.001882  Accuracy: 99.06%

Test set: Average loss: 0.0071, Accuracy: 9755/10000 (98%)
Train Epoch: 10         Loss: 0.001878  Accuracy: 99.10%

Test set: Average loss: 0.0079, Accuracy: 9752/10000 (98%)
```

[82]:
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(range(1, epochs + 1), train_losses, label='Train Loss')
plt.plot(range(1, epochs + 1), test_losses, label='Test Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1,2,2)
plt.plot(range(1, epochs + 1), train_accuracies, label='Train Accuracy')
plt.plot(range(1, epochs + 1), test_accuracies, label='Test Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
plt.legend()

plt.tight_layout()
plt.show()
```

[83]:
```
with torch.no_grad():
    batch_idx, (example_data, example_targets) = next(examples)
    output = model(example_data)
    predicted = output.argmax(dim=1, keepdim=True)

fig = plt.figure(figsize=(10, 4))
for i in range(6):
    plt.subplot(2, 3, i + 1)
    plt.tight_layout()
    plt.imshow(example_data[i][0], cmap='gray', interpolation='none')
    plt.title(f"Truth: {example_targets[i]}\nPredicted: {predicted[i].item()}")
    plt.xticks([])
    plt.yticks([])
plt.show()
```



[84]:
```
def plot_incorrect_predictions(model, loader, num_images=6):
    model.eval()
    incorrect_examples = []

    with torch.no_grad():
        for data, target in loader:
            output = model(data)
            pred = output.argmax(dim=1, keepdim=True)
            idxs_mask = ((pred == target.view_as(pred)) == False).view(-1)
            incorrect_preds = pred[idxs_mask].view(-1)
            actual_labels = target.view_as(pred)[idxs_mask].view(-1)
            data = data[idxs_mask]
            for i in range(data.shape[0]):
                if len(incorrect_examples) < num_images:
                    incorrect_examples.append((data[i], incorrect_preds[i],␣
 ↪actual_labels[i]))
                else:
                    break
            if len(incorrect_examples) == num_images:
                break

    fig = plt.figure(figsize=(10, 4))
    for i, (img, pred, actual) in enumerate(incorrect_examples):
        img = img.view(28, 28)
        plt.subplot(1, num_images, i + 1)
        plt.tight_layout()
        plt.imshow(img, cmap='gray', interpolation='none')
        plt.title(f"Pred: {pred.item()}\nActual: {actual.item()}")
        plt.xticks([])
        plt.yticks([])
    plt.show()

# Usage
plot_incorrect_predictions(model, test_loader)
```



[84]: