StuDocu.com

Ex9 - Ex9 on classification

Automatic Image Analysis (Technische Universität Berlin)

StuDocu wird von keiner Universität gesponsert oder unterstützt.
Heruntergeladen durch QING HUANG (qhuang.y@gmail.com)

Ex9

June 23, 2021

1 Task 1: Classification

- define a Neural Network
- def

[1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np
from utils import NoisyFashionMNIST
from torch.optim.lr_scheduler import StepLR

%matplotlib inline
def show(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1,2,0)), interpolation='nearest')
```

1.1 Dataset

[2]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

transform=transforms.Compose([
        transforms.ToTensor()])

train_dataset = datasets.FashionMNIST("./data", train = True, download=True,␣
 ↪transform=transform)
test_dataset = datasets.FashionMNIST("./data", train = False, download=True,␣
 ↪transform=transform)


idx_to_class = {v: k for k, v in train_dataset.class_to_idx.items()}
```


This document is available free of charge on StuDocu.com

Heruntergeladen durch QING HUANG (qhuang.y@gmail.com)

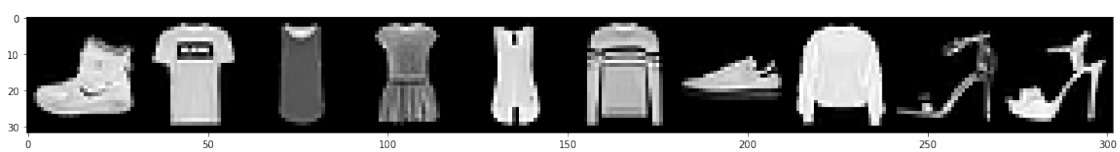
[3]:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5, 1)
        self.batchN1 = nn.BatchNorm2d(num_features=6)
        self.conv2 = nn.Conv2d(6, 12, 5, 1)
        self.batchN2 = nn.BatchNorm2d(num_features=12)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.25)
        self.fc1 = nn.Linear(1200, 60)
        self.fc2 = nn.Linear(60, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

[4]:

```
x = [train_dataset[i][0] for i in range(10)]
labels = [idx_to_class[train_dataset[i][1]] for i in range(10)]
print(labels)

plt.figure(figsize=(20,10))
show(torchvision.utils.make_grid(x, nrow=10))
plt.show()
```

```
['Ankle boot', 'T-shirt/top', 'T-shirt/top', 'Dress', 'T-shirt/top', 'Pullover',
'Sneaker', 'Pullover', 'Sandal', 'Sandal']
```



[5]:

```
def train(model, device, train_loader, optimizer, epoch):
    model.train()
```


Heruntergeladen durch QING HUANG (qhuang.y@gmail.com)

```
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()), end='\r')
```

[6]:
```
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  #␣
↪sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the␣
↪max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.
↪format(
    test_loss, correct, len(test_loader.dataset),
    100. * correct / len(test_loader.dataset)), end='\r')
```

[ ]:
```
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64)
test_loader = torch.utils.data.DataLoader(test_dataset,  batch_size=64)

model = Net().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)

scheduler = StepLR(optimizer, step_size=5, gamma=0.1)
for epoch in range(1,25 + 1):
    train(model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)
    scheduler.step()
```

```
Train Epoch: 1 [59520/60000 (99%)]      Loss: 0.783967
```


This document is available free of charge on StuDocu.com
Heruntergeladen durch QING HUANG (qhuang.y@gmail.com)

```
Test set: Average loss: 0.5035, Accuracy: 8153/10000 (82%)
Train Epoch: 2 [59520/60000 (99%)]      Loss: 0.559346
Test set: Average loss: 0.4278, Accuracy: 8480/10000 (85%)
Train Epoch: 3 [59520/60000 (99%)]      Loss: 0.510979
Test set: Average loss: 0.3744, Accuracy: 8579/10000 (86%)
Train Epoch: 4 [59520/60000 (99%)]      Loss: 0.442671
Test set: Average loss: 0.3485, Accuracy: 8723/10000 (87%)
Train Epoch: 5 [59520/60000 (99%)]      Loss: 0.413945
Test set: Average loss: 0.3374, Accuracy: 8744/10000 (87%)
Train Epoch: 6 [59520/60000 (99%)]      Loss: 0.379380
Test set: Average loss: 0.3124, Accuracy: 8858/10000 (89%)
Train Epoch: 7 [59520/60000 (99%)]      Loss: 0.350224
Test set: Average loss: 0.3072, Accuracy: 8868/10000 (89%)
Train Epoch: 8 [59520/60000 (99%)]      Loss: 0.361336
Test set: Average loss: 0.3057, Accuracy: 8865/10000 (89%)
Train Epoch: 9 [59520/60000 (99%)]      Loss: 0.304667
Test set: Average loss: 0.3028, Accuracy: 8881/10000 (89%)
Train Epoch: 10 [59520/60000 (99%)]     Loss: 0.374543
Test set: Average loss: 0.3013, Accuracy: 8882/10000 (89%)
Train Epoch: 11 [59520/60000 (99%)]     Loss: 0.326367
Test set: Average loss: 0.3001, Accuracy: 8893/10000 (89%)
Train Epoch: 12 [59520/60000 (99%)]     Loss: 0.386963
Test set: Average loss: 0.2998, Accuracy: 8901/10000 (89%)
Train Epoch: 13 [59520/60000 (99%)]     Loss: 0.298528
Test set: Average loss: 0.2996, Accuracy: 8900/10000 (89%)
Train Epoch: 14 [59520/60000 (99%)]     Loss: 0.356862
Test set: Average loss: 0.2993, Accuracy: 8898/10000 (89%)
Train Epoch: 15 [59520/60000 (99%)]     Loss: 0.274130
Test set: Average loss: 0.2988, Accuracy: 8898/10000 (89%)
Train Epoch: 16 [59520/60000 (99%)]     Loss: 0.336278
Test set: Average loss: 0.2988, Accuracy: 8902/10000 (89%)
Train Epoch: 17 [27520/60000 (46%)]     Loss: 0.395879
```

2 Task 2:

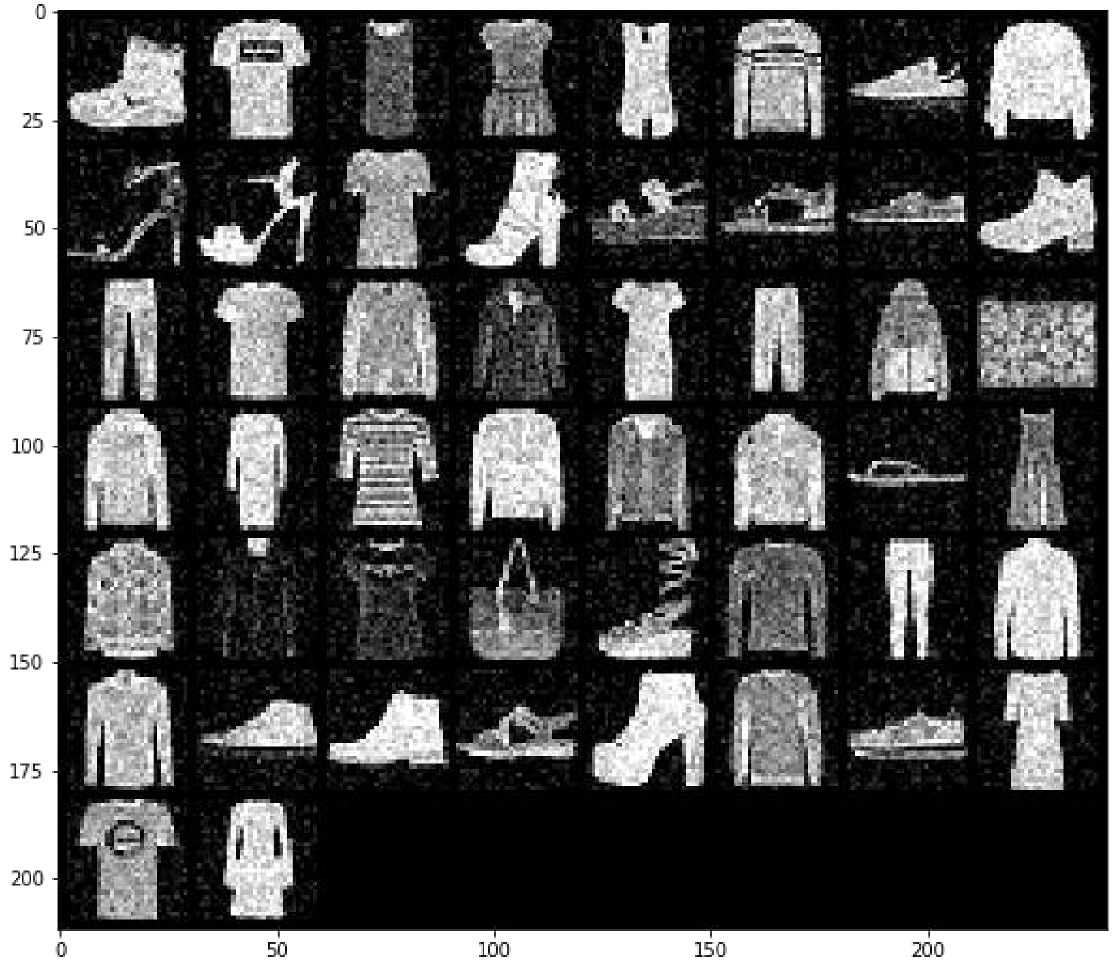
[4]:
```python
train_dataset = NoisyFashionMNIST("./data", True)
test_dataset = NoisyFashionMNIST("./data", False)
```

[5]:
```python
x = [train_dataset[i][0] for i in range(50)]
y = [train_dataset[i][1] for i in range(50)]

plt.figure(figsize=(10,10))
show(torchvision.utils.make_grid(x))
plt.show()

plt.figure(figsize=(10,10))
show(torchvision.utils.make_grid(y))
```

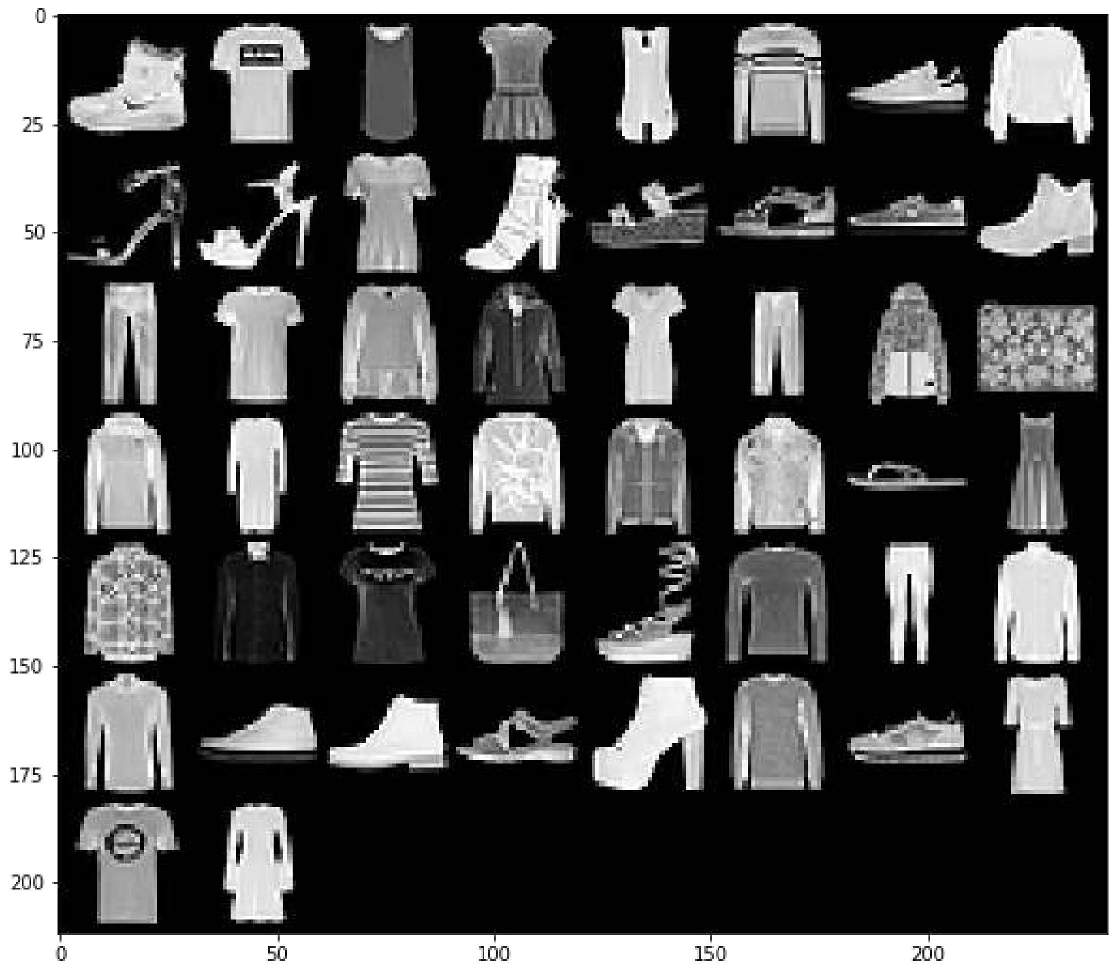

Heruntergeladen durch QING HUANG (qhuang.y@gmail.com)

```
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



This document is available free of charge on StuDocu.com
Heruntergeladen durch QING HUANG (qhuang.y@gmail.com)




```
[ ]:
```

```
[ ]:
```


Heruntergeladen durch QING HUANG (qhuang.y@gmail.com)