hw_2

May 11, 2025

[1]:

```python
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim

# PyTorch TensorBoard support
# from torch.utils.tensorboard import SummaryWriter
# import torchvision
# import torchvision.transforms as transforms

from datetime import datetime

import torchvision
import torchvision.transforms as transforms

from torchvision.datasets import FashionMNIST
import matplotlib.pyplot as plt
%matplotlib inline

from torch.utils.data import random_split
from torch.utils.data import DataLoader
import torch.nn.functional as F

from PIL import Image
#import torchvision.transforms as T
```

[2]:

```python
# load the dataset
fmnist_dataset = FashionMNIST(root = 'data/', download=True, train = True,␣
 ↪transform = transforms.ToTensor())
print(fmnist_dataset)
```

```
Dataset FashionMNIST
    Number of datapoints: 60000
    Root location: data/
    Split: Train
    StandardTransform
Transform: ToTensor()
```

0.1 Training and validation data

[6]:

```
train_data, validation_data = random_split(fmnist_dataset, [50000, 10000])
## Print the length of train and validation datasets
print("length of Train Datasets: ", len(train_data))
print("length of Validation Datasets: ", len(validation_data))

batch_size = 128
train_loader = DataLoader(train_data, batch_size, shuffle = True)
val_loader = DataLoader(validation_data, batch_size, shuffle = False)
## MNIST data from pytorch already provides held-out test set!
```

```
length of Train Datasets:  50000
length of Validation Datasets:  10000
```

1 Convolutional Neural Network (CNN)

[8]:

```
class CNN_optim(nn.Module):
    def __init__(self, conv_channels, kernel_size, fc_size):
        super(CNN_optim, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=conv_channels[0],
                kernel_size=kernel_size,
                stride=1,
                padding=kernel_size //2,
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(
                in_channels=conv_channels[0],
                out_channels=conv_channels[1],
                kernel_size=kernel_size,
                stride=1,
                padding=kernel_size // 2,
            ),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )

        self.fully_connected = nn.Sequential(
            nn.Linear(conv_channels[1] * 7 * 7, fc_size),
            nn.ReLU(),
            nn.Linear(fc_size, 10),
        )
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)
        output = self.fully_connected(x)
        return output, x
```

[ ]:

```
test_dataset = FashionMNIST(root='data/', train=False, transform=transforms.
 ↪ToTensor())
test_loader = DataLoader(test_dataset, batch_size=256, shuffle=False)

test_results = []

architectures = [
    {'conv_channels': [16, 32], 'kernel_size': 5, 'fc_size': 128},
    {'conv_channels': [32, 64], 'kernel_size': 3, 'fc_size': 256},
    {'conv_channels': [16, 32, 64], 'kernel_size': 3, 'fc_size': 256},
    {'conv_channels': [8, 16, 32], 'kernel_size': 3, 'fc_size': 128},
    {'conv_channels': [16, 32, 64, 128], 'kernel_size': 3, 'fc_size': 256},
    {'conv_channels': [16, 64, 128], 'kernel_size': 3, 'fc_size': 128},
    {'conv_channels': [32, 64, 128, 256], 'kernel_size': 3, 'fc_size': 512},
]

for arch in architectures:
    print(f"Testing {arch}")
    model = CNN_optim(arch['conv_channels'], arch['kernel_size'],␣
 ↪arch['fc_size'])

    optimizer = optim.Adam(model.parameters(), lr=0.01)
    loss_fn = nn.CrossEntropyLoss()

    model.train()
    for epoch in range(5):
        for xb, yb in train_loader:
            out, _ = model(xb)
            loss = loss_fn(out, yb)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for xb, yb in test_loader:
            out, _ = model(xb)
            preds = out.argmax(1)
            correct += (preds == yb).sum().item()
            total += yb.size(0)
    acc = correct / total
    test_results.append({'config': arch, 'accuracy': acc})
    print(f"Accuracy: {acc:.4f}")
```

```
Testing {'conv_channels': [8, 16], 'kernel_size': 3, 'fc_size': 64}
Accuracy: 0.8838
Testing {'conv_channels': [16, 32], 'kernel_size': 5, 'fc_size': 128}
Accuracy: 0.8893
Testing {'conv_channels': [32, 64], 'kernel_size': 3, 'fc_size': 256}
Accuracy: 0.8988
Testing {'conv_channels': [16, 32, 64], 'kernel_size': 3, 'fc_size': 256}
Accuracy: 0.8931
Testing {'conv_channels': [8, 16, 32], 'kernel_size': 3, 'fc_size': 128}
Accuracy: 0.8934
Testing {'conv_channels': [16, 32, 64, 128], 'kernel_size': 3, 'fc_size': 256}
Accuracy: 0.8919
Testing {'conv_channels': [32, 64, 128], 'kernel_size': 5, 'fc_size': 512}
Accuracy: 0.8793
Testing {'conv_channels': [64, 128], 'kernel_size': 3, 'fc_size': 256}
Accuracy: 0.8819
Testing {'conv_channels': [16, 64, 128], 'kernel_size': 3, 'fc_size': 128}
Accuracy: 0.9012
Testing {'conv_channels': [32, 64, 128, 256], 'kernel_size': 3, 'fc_size': 512}
Accuracy: 0.8843
```

[17]:

```
best = max(test_results, key=lambda x: x['accuracy'])
print("Best architecture:", best['config'])
print("Test accuracy:", best['accuracy'])
```

```
Best architecture: {'conv_channels': [16, 64, 128], 'kernel_size': 3, 'fc_size':
128}
Test accuracy: 0.9012
```