Chem 277B Spring 2024 Tutorial 6

Outline

- Principal Component Analysis (PCA) in sklearn
- Dropout and L2 regularization in PyTorch
- Dataset and DataLoader in PyTorch

MNIST Dataset

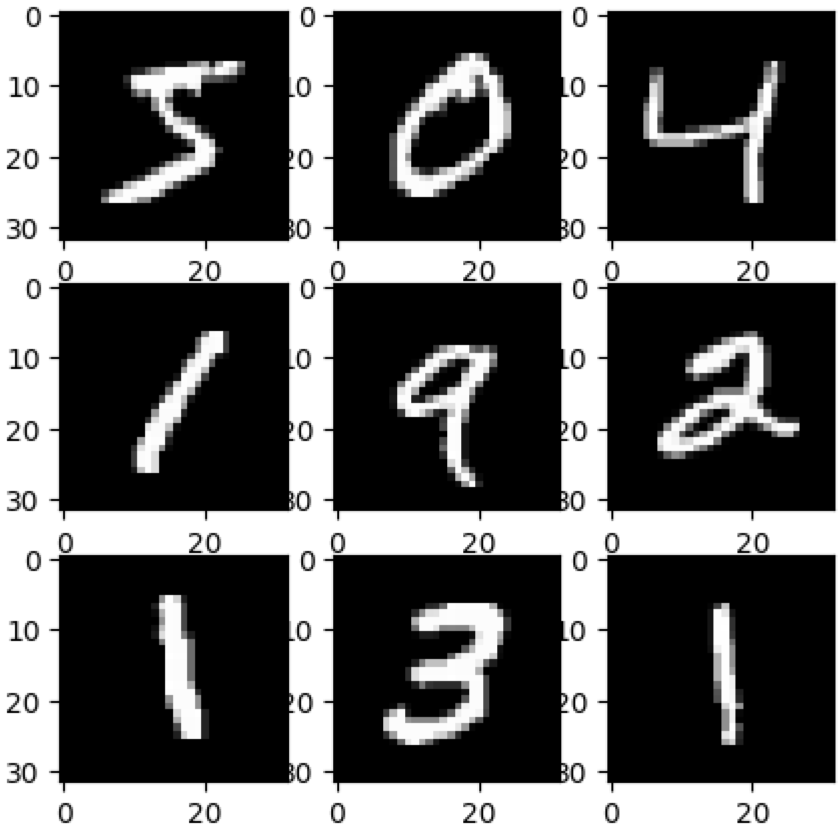
In [ ]:

```
import pickle
import torch

def load_dataset(path):
    with open(path, 'rb') as f:
        train_data, test_data = pickle.load(f)

    X_train = torch.tensor(train_data[0], dtype=torch.float)
    y_train = torch.tensor(train_data[1], dtype=torch.long)
    X_test = torch.tensor(test_data[0], dtype=torch.float)
    y_test = torch.tensor(test_data[1], dtype=torch.long)
    return X_train, y_train, X_test, y_test

X_train, y_train, X_test, y_test = load_dataset("mnist.pkl")
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: torch.Size([60000, 32, 32])
X_test shape: torch.Size([10000, 32, 32])
y_train shape: torch.Size([60000])
y_test shape: torch.Size([10000])
```

In [ ]:

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(3, 3, figsize=(5, 5))
for i, ax in enumerate(axes.flatten()):
    ax.imshow(X_train[i], cmap='gray')
```



Principal Component Analysis (PCA)

In [ ]:
```
# Flatten the inputs & normalization
X_train = X_train.reshape(X_train.shape[0], -1) / torch.max(X_train)
X_test = X_test.reshape(X_test.shape[0], -1) / torch.max(X_test)
print(X_train.shape)
```

```
torch.Size([60000, 1024])
```

In [ ]:
```
from sklearn.decomposition import PCA

# keeping specific number of features
pca = PCA(n_components=256)
# fit
pca.fit(X_train)
# transform
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print(X_train_pca.shape, X_test_pca.shape)
```

```
(60000, 256) (10000, 256)
```

In [ ]:
```
# keeping amount of variance
pca = PCA(n_components=0.99)
# fit
pca.fit(X_train)
# transform
X_train_pca = pca.transform(X_train)
```

```
X_test_pca = pca.transform(X_test)
print(X_train_pca.shape, X_test_pca.shape)
```

```
(60000, 331) (10000, 331)
```

In [ ]:

```
X_train_pca = torch.tensor(X_train_pca, dtype=torch.float)
X_test_pca = torch.tensor(X_test_pca, dtype=torch.float)
```

In [ ]:

```
print(X_train_pca.shape, X_test_pca.shape)
```

```
torch.Size([60000, 331]) torch.Size([10000, 331])
```

In [ ]:

```
sum(pca.explained_variance_ratio_)
```

Out[ ]:

```
0.990012942492929
```

Dropout & L2

Use `nn.Dropout` layer:

During training, randomly zeroes some of the elements of the input tensor with probability p.

In [ ]:

```
import torch.nn as nn


class NetDropout(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(331, 100),
            nn.Dropout(p=0.1),
            nn.Sigmoid(),
            nn.Linear(100, 10),
            nn.Dropout(p=0.1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.layers(x)
```

Set `weight_decay` to use L2 regularization.

$$\text{Loss\_L2} = \text{Loss} + \lambda \sum \theta_i^2$$

In [ ]:

```
model = NetDropout()
optimizer = torch.optim.Adam(model.parameters(), 1e-3, weight_decay=1e-5)
```

Dataset & DataLoader

In [ ]:

```
from torch.utils.data import Dataset, DataLoader


class MnistDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
```

In [ ]:

```
train_data = MnistDataset(X_train_pca, y_train)
test_data = MnistDataset(X_test_pca, y_test)
train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
test_loader = DataLoader(test_data, batch_size=128, shuffle=True)
```

In [ ]:

```
for X_batch, y_batch in train_loader:
    print(X_batch.shape, y_batch.shape)
    break
```

```
torch.Size([128, 331]) torch.Size([128])
```

Trainer

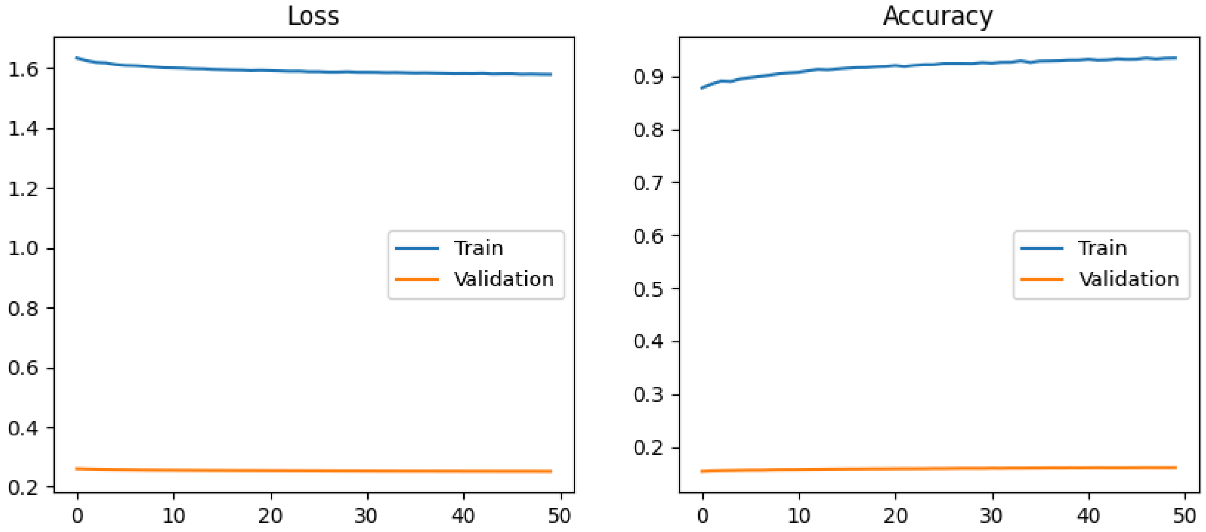
In [ ]:

```
import numpy as np
from tqdm import tqdm

class Trainer:

    def __init__(self, model, opt_method, learning_rate, batch_size, epoch,
        self.model = model

        if opt_method == "adam":
            self.optimizer = torch.optim.Adam(model.parameters(), learning_r
        else:
            raise NotImplementedError("This optimization is not supported")

        self.epoch = epoch
        self.batch_size = batch_size

    def train(self, train_data, val_data, early_stop=True, verbose=True, dra
        train_loader = DataLoader(train_data, batch_size=self.batch_size, sh

        train_loss_list, train_acc_list = [], []
        val_loss_list, val_acc_list = [], []
        weights = self.model.state_dict()
        lowest_val_loss = np.inf
        loss_func = nn.CrossEntropyLoss()
        for n in tqdm(range(self.epoch), leave=False):
            # enable train mode
            self.model.train()
            epoch_loss, epoch_acc = 0.0, 0.0
            for X_batch, y_batch in train_loader:
                # batch_importance is the ratio of batch_size
                batch_importance = y_batch.shape[0]/len(train_data)
                y_pred = self.model(X_batch)
                batch_loss = loss_func(y_pred, y_batch)

                self.optimizer.zero_grad()
                batch_loss.backward()
                self.optimizer.step()

                epoch_loss += batch_loss.detach().cpu().item() * batch_impor
                batch_acc = torch.sum(torch.argmax(y_pred, axis=1) == y_batc
                epoch_acc += batch_acc.detach().cpu().item() * batch_importa
            train_loss_list.append(epoch_loss)
            train_acc_list.append(epoch_acc)
            val_loss, val_acc = self.evaluate(val_data)
            val_loss_list.append(val_loss)
            val_acc_list.append(val_acc)

            if early_stop:
                if val_loss < lowest_val_loss:
                    lowest_val_loss = val_loss
                    weights = self.model.state_dict()

        if draw_curve:
            x_axis = np.arange(self.epoch)
            fig, axes = plt.subplots(1, 2, figsize=(10, 4))
            axes[0].plot(x_axis, train_loss_list, label="Train")
            axes[0].plot(x_axis, val_loss_list, label="Validation")
            axes[0].set_title("Loss")
            axes[0].legend()
            axes[1].plot(x_axis, train_acc_list, label='Train')
            axes[1].plot(x_axis, val_acc_list, label='Validation')
            axes[1].set_title("Accuracy")
            axes[1].legend()

        if early_stop:
            self.model.load_state_dict(weights)

        return {
            "train_loss_list": train_loss_list,
            "train_acc_list": train_acc_list,
            "val_loss_list": val_loss_list,
            "val_acc_list": val_acc_list,
        }

    def evaluate(self, data, print_acc=False):
        # enable evaluation mode
        self.model.eval()
        loader = DataLoader(data, batch_size=self.batch_size, shuffle=True)
        loss_func = nn.CrossEntropyLoss()
        acc, loss = 0.0, 0.0
        for X_batch, y_batch in loader:
            with torch.no_grad():
                batch_importance = y_batch.shape[0]/len(train_data)
                y_pred = self.model(X_batch)
                batch_loss = loss_func(y_pred, y_batch)
                batch_acc = torch.sum(torch.argmax(y_pred, axis=1) == y_batc
                acc += batch_acc.detach().cpu().item() * batch_importance
                loss += batch_loss.detach().cpu().item() * batch_importance
        if print_acc:
            print(f"Accuracy: {acc:.3f}")
        return loss, acc
```

In [ ]:

```
trainer = Trainer(model, "adam", 1e-3, 128, 50, 1e-5)
trainer.train(train_data,test_data)
```

```
Out[ ]:  {'train_loss_list': [1.6345654050827023,
           1.6256413174947117,
           1.619539224243165,
           1.6178119778315228,
           1.6122155399958304,
           1.6093794220606485,
           1.608482451057434,
           1.6063486288070667,
           1.6038572382609038,
           1.6014241783142082,
           1.6007745227177927,
           1.599812180264789,
           1.597846146583558,
           1.5972427301406853,
           1.5956069282531724,
           1.5947243247985832,
           1.5936766189575198,
           1.593202598508199,
           1.5918138434728,
           1.5926415494918846,
           1.5917330230712887,
           1.5907306164423625,
           1.5894648267745985,
           1.5896575952529912,
           1.5875960771560667,
           1.58758466536204,
           1.5862048535664877,
           1.586286747423808,
           1.5872448308308928,
           1.585802490043641,
           1.58577879892985,
           1.5855446199417098,
           1.5844957898457848,
           1.5848428560892744,
           1.5839182889938355,
           1.5830990733464563,
           1.5834927261988339,
           1.5828149834314986,
           1.5822036504745487,
           1.5814984322230017,
           1.58175454451243,
           1.5815143378575645,
           1.5822904492060332,
           1.5805196723302206,
           1.5810784157435107,
           1.5812023804346726,
           1.5794532040913887,
           1.5800079744338982,
           1.5792736788431794,
           1.5789225581487012],
          'train_acc_list': [0.8782166666984605,
           0.8854500000000044,
           0.8912833333333375,
           0.89041666669846,
           0.8952833333651263,
           0.8975166666666704,
  0.9000000000000036,
  0.9019833333015477,
  0.9049166666348809,
  0.9064166666984592,
  0.9075166666666697,
  0.9104333333651254,
  0.912983333301547,
  0.9121833333015469,
  0.9136833333015468,
  0.9153333333651251,
  0.9166333333333356,
  0.91691666663488,
  0.9181666666348799,
  0.9188333333333355,
  0.9206000000000022,
  0.9186333333333355,
  0.920966666698458,
  0.9221666666666686,
  0.922249999968213,
  0.924400000031791,
  0.9245166666984577,
  0.9244666666666684,
  0.9242666666666685,
  0.9261833333651242,
  0.9251666666984576,
  0.9269000000000017,
  0.9269833333651242,
  0.9297333333333346,
  0.9265166666666683,
  0.9292500000000012,
  0.9295500000000015,
  0.9300333333333348,
  0.9310166666666677,
  0.9311499999682122,
  0.9327333333333343,
  0.9309333333333345,
  0.9315333333333344,
  0.9333333333651236,
  0.9324833333333344,
  0.9327666666666676,
  0.935000000000008,
  0.9331666666348787,
  0.934700000000001,
  0.9350666666666676],
 'val_loss_list': [0.2588511471748352,
  0.25773222007751473,
  0.25691719643274935,
  0.2563522413253785,
  0.2558027157147726,
  0.25547989222208656,
  0.2551308627128602,
  0.2547252587318421,
  0.25442750892639165,
  0.25422911443710333,
  0.2540464429219565,
  0.2537937870661418,
 0.2536431848526002,
 0.2534067885080974,
 0.25320728416442884,
 0.25312399593989054,
 0.25289309854507447,
 0.2528072364171346,
 0.25273120349248246,
 0.25260152072906494,
 0.25241151371002196,
 0.252415691057841,
 0.2522326436996459,
 0.25216817359924315,
 0.2520019748369852,
 0.25197507321039825,
 0.2518450228055318,
 0.25169793780644745,
 0.2516503425916036,
 0.25158187856674197,
 0.251466447766622,
 0.2513963534673056,
 0.25126730022430416,
 0.2512035391171774,
 0.2511685480117798,
 0.2510992826461793,
 0.251028265062968,
 0.25103135042190555,
 0.250923332309723,
 0.2508750089009603,
 0.25078297662734994,
 0.25076765464146933,
 0.2507223012606303,
 0.2506392167091371,
 0.2505964373906454,
 0.25054782549540205,
 0.25052587922414143,
 0.2505193968772888,
 0.25040930995941174,
 0.25038921445210777],
'val_acc_list': [0.15325,
 0.15410000000000004,
 0.1544833333333334,
 0.15483333333333338,
 0.15510000000000002,
 0.1554666666666667,
 0.15543333333333334,
 0.15590000000000004,
 0.1562666666666667,
 0.15631666666666674,
 0.15641666666666665,
 0.15664999999999996,
 0.15681666666666666,
 0.15701666666666658,
 0.15715000000000007,
 0.1573333333333333,
 0.15738333333333332,
 0.15756666666666666,
  0.15773333333333336,
  0.1577666666666667,
  0.15793333333333331,
  0.1580333333333333,
  0.1580833333333333,
  0.1581833333333333,
  0.1584833333333333,
  0.15844999999999995,
  0.1587166666666666,
  0.15891666666666665,
  0.1589166666666666,
  0.15886666666666663,
  0.1592333333333333,
  0.15909999999999996,
  0.1593999999999999,
  0.15936666666666663,
  0.15944999999999995,
  0.15958333333333322,
  0.15973333333333328,
  0.15969999999999995,
  0.15981666666666666,
  0.15978333333333328,
  0.15981666666666663,
  0.16001666666666658,
  0.15984999999999996,
  0.1598166666666666,
  0.1599166666666666,
  0.1599833333333333,
  0.1601666666666666,
  0.16008333333333322,
  0.1601333333333333,
  0.16018333333333332]}
```



In [ ]: