In [1]:

```
from mnist import MNIST
from torch import nn
import torch
from torchvision import datasets, transforms
import numpy

if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')

print('Using PyTorch version:', torch.__version__, ' Device:', device)
```

```
Using PyTorch version: 1.4.0  Device: cpu
```

MNIST Data

In [2]:

```
transform = transforms.Compose([
                            transforms.ToTensor(),
                            transforms.Normalize((0.5,), (0.5,)),
                        ])

train_set = datasets.MNIST('./Dataset-3', download=True, train=True, transform=trans
val_set = datasets.MNIST('./Dataset-3', download=True, train=False, transform=transf
train_loader = torch.utils.data.DataLoader(train_set, batch_size=32, shuffle=True)
val_loader = torch.utils.data.DataLoader(val_set, batch_size=32, shuffle=True)
```

1. Multilayer Perceptron

In [3]:

```python
input_size = 784
hidden_sizes = [128, 64]
output_size = 10

model = nn.Sequential(nn.Linear(input_size, hidden_sizes[0]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[0], hidden_sizes[1]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[1], output_size))

optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
criterion = nn.CrossEntropyLoss()

print(model)
```

```
Sequential(
  (0): Linear(in_features=784, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=64, bias=True)
  (3): ReLU()
  (4): Linear(in_features=64, out_features=10, bias=True)
)
```

In [4]:

```python
def train(epoch, log_interval=200):
    # Set model to training mode
    model.train()

    # Loop over each batch from the training set
    for batch_idx, (data, target) in enumerate(train_loader):

        data = data.view(data.shape[0], -1)
        # Copy data to GPU if needed
        data = data.to(device)
        target = target.to(device)

        # Zero gradient buffers
        optimizer.zero_grad()

        # Pass data through the network
        output = model(data)

        # Calculate loss
        loss = criterion(output, target)

        # Backpropagate
        loss.backward()

        # Update weights
        optimizer.step()

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

In [5]:

```
def validate(loss_vector, accuracy_vector):
    pred_arr = []
    model.eval()
    val_loss, correct = 0, 0
    for data, target in val_loader:
        data = data.view(data.shape[0], -1)
        data = data.to(device)
        target = target.to(device)
        output = model(data)
        val_loss += criterion(output, target).data.item()
        pred = output.data.max(1)[1] # get the index of the max log-probability
        for i in numpy.array(pred):
            pred_arr.append(int(i))
        correct += pred.eq(target.data).cpu().sum()

    val_loss /= len(val_loader)
    loss_vector.append(val_loss)

    accuracy = 100. * correct.to(torch.float32) / len(val_loader.dataset)
    accuracy_vector.append(accuracy)

    print('\nValidation set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.form
        val_loss, correct, len(val_loader.dataset), accuracy))

    return pred_arr
```

In [6]:

```
%%time
epochs = 10

lossv, accv = [], []
for epoch in range(1, epochs + 1):
    train(epoch)
    output = validate(lossv, accv)
```

```
Train Epoch: 1 [0/60000 (0%)]	Loss: 2.309464
Train Epoch: 1 [6400/60000 (11%)]	Loss: 0.907003
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.376824
Train Epoch: 1 [19200/60000 (32%)]	Loss: 0.276851
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.345412
Train Epoch: 1 [32000/60000 (53%)]	Loss: 0.304346
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.207404
Train Epoch: 1 [44800/60000 (75%)]	Loss: 0.338337
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.520478
Train Epoch: 1 [57600/60000 (96%)]	Loss: 0.228887

Validation set: Average loss: 0.2728, Accuracy: 9211/10000 (92%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.250406
Train Epoch: 2 [6400/60000 (11%)]	Loss: 0.558774
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.287441
Train Epoch: 2 [19200/60000 (32%)]	Loss: 0.353965
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.112871
Train Epoch: 2 [32000/60000 (53%)]	Loss: 0.270819
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.485234
Train Epoch: 2 [44800/60000 (75%)]	Loss: 0.133416
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.115772
Train Epoch: 2 [57600/60000 (96%)]	Loss: 0.180388

Validation set: Average loss: 0.2171, Accuracy: 9343/10000 (93%)

Train Epoch: 3 [0/60000 (0%)]	Loss: 0.403647
Train Epoch: 3 [6400/60000 (11%)]	Loss: 0.063474
Train Epoch: 3 [12800/60000 (21%)]	Loss: 0.304317
Train Epoch: 3 [19200/60000 (32%)]	Loss: 0.082626
Train Epoch: 3 [25600/60000 (43%)]	Loss: 0.156674
Train Epoch: 3 [32000/60000 (53%)]	Loss: 0.280566
Train Epoch: 3 [38400/60000 (64%)]	Loss: 0.082290
Train Epoch: 3 [44800/60000 (75%)]	Loss: 0.044991
Train Epoch: 3 [51200/60000 (85%)]	Loss: 0.423108
Train Epoch: 3 [57600/60000 (96%)]	Loss: 0.108776

Validation set: Average loss: 0.1633, Accuracy: 9511/10000 (95%)

Train Epoch: 4 [0/60000 (0%)]	Loss: 0.026563
Train Epoch: 4 [6400/60000 (11%)]	Loss: 0.201686
Train Epoch: 4 [12800/60000 (21%)]	Loss: 0.101088
Train Epoch: 4 [19200/60000 (32%)]	Loss: 0.119784
Train Epoch: 4 [25600/60000 (43%)]	Loss: 0.160938
Train Epoch: 4 [32000/60000 (53%)]	Loss: 0.250833
Train Epoch: 4 [38400/60000 (64%)]	Loss: 0.149968
Train Epoch: 4 [44800/60000 (75%)]	Loss: 0.036563
Train Epoch: 4 [51200/60000 (85%)]	Loss: 0.207013
Train Epoch: 4 [57600/60000 (96%)]	Loss: 0.153458

Validation set: Average loss: 0.1290, Accuracy: 9596/10000 (96%)
```

```
Train Epoch: 5 [0/60000 (0%)]	Loss: 0.145281
Train Epoch: 5 [6400/60000 (11%)]	Loss: 0.117787
Train Epoch: 5 [12800/60000 (21%)]	Loss: 0.091086
Train Epoch: 5 [19200/60000 (32%)]	Loss: 0.278759
Train Epoch: 5 [25600/60000 (43%)]	Loss: 0.102493
Train Epoch: 5 [32000/60000 (53%)]	Loss: 0.115219
Train Epoch: 5 [38400/60000 (64%)]	Loss: 0.096556
Train Epoch: 5 [44800/60000 (75%)]	Loss: 0.070058
Train Epoch: 5 [51200/60000 (85%)]	Loss: 0.094554
Train Epoch: 5 [57600/60000 (96%)]	Loss: 0.047594

Validation set: Average loss: 0.1228, Accuracy: 9620/10000 (96%)

Train Epoch: 6 [0/60000 (0%)]	Loss: 0.062618
Train Epoch: 6 [6400/60000 (11%)]	Loss: 0.082642
Train Epoch: 6 [12800/60000 (21%)]	Loss: 0.101368
Train Epoch: 6 [19200/60000 (32%)]	Loss: 0.072434
Train Epoch: 6 [25600/60000 (43%)]	Loss: 0.076895
Train Epoch: 6 [32000/60000 (53%)]	Loss: 0.063236
Train Epoch: 6 [38400/60000 (64%)]	Loss: 0.181962
Train Epoch: 6 [44800/60000 (75%)]	Loss: 0.325390
Train Epoch: 6 [51200/60000 (85%)]	Loss: 0.040160
Train Epoch: 6 [57600/60000 (96%)]	Loss: 0.197853

Validation set: Average loss: 0.1244, Accuracy: 9606/10000 (96%)

Train Epoch: 7 [0/60000 (0%)]	Loss: 0.054344
Train Epoch: 7 [6400/60000 (11%)]	Loss: 0.006310
Train Epoch: 7 [12800/60000 (21%)]	Loss: 0.206730
Train Epoch: 7 [19200/60000 (32%)]	Loss: 0.179749
Train Epoch: 7 [25600/60000 (43%)]	Loss: 0.174143
Train Epoch: 7 [32000/60000 (53%)]	Loss: 0.046840
Train Epoch: 7 [38400/60000 (64%)]	Loss: 0.045654
Train Epoch: 7 [44800/60000 (75%)]	Loss: 0.030797
Train Epoch: 7 [51200/60000 (85%)]	Loss: 0.064237
Train Epoch: 7 [57600/60000 (96%)]	Loss: 0.028015

Validation set: Average loss: 0.0988, Accuracy: 9697/10000 (97%)

Train Epoch: 8 [0/60000 (0%)]	Loss: 0.044861
Train Epoch: 8 [6400/60000 (11%)]	Loss: 0.122501
Train Epoch: 8 [12800/60000 (21%)]	Loss: 0.073936
Train Epoch: 8 [19200/60000 (32%)]	Loss: 0.010186
Train Epoch: 8 [25600/60000 (43%)]	Loss: 0.186015
Train Epoch: 8 [32000/60000 (53%)]	Loss: 0.136657
Train Epoch: 8 [38400/60000 (64%)]	Loss: 0.092304
Train Epoch: 8 [44800/60000 (75%)]	Loss: 0.088638
Train Epoch: 8 [51200/60000 (85%)]	Loss: 0.100790
Train Epoch: 8 [57600/60000 (96%)]	Loss: 0.061675

Validation set: Average loss: 0.0932, Accuracy: 9691/10000 (97%)

Train Epoch: 9 [0/60000 (0%)]	Loss: 0.144328
Train Epoch: 9 [6400/60000 (11%)]	Loss: 0.006296
Train Epoch: 9 [12800/60000 (21%)]	Loss: 0.073050
Train Epoch: 9 [19200/60000 (32%)]	Loss: 0.162591
Train Epoch: 9 [25600/60000 (43%)]	Loss: 0.005200
Train Epoch: 9 [32000/60000 (53%)]	Loss: 0.077137
Train Epoch: 9 [38400/60000 (64%)]	Loss: 0.091312
Train Epoch: 9 [44800/60000 (75%)]	Loss: 0.061812
Train Epoch: 9 [51200/60000 (85%)]      Loss: 0.043815
Train Epoch: 9 [57600/60000 (96%)]      Loss: 0.035309

Validation set: Average loss: 0.1006, Accuracy: 9687/10000 (97%)

Train Epoch: 10 [0/60000 (0%)]  Loss: 0.111077
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.202822
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.093402
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.059557
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.042134
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.011308
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.117342
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.122974
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.009525
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.094501

Validation set: Average loss: 0.0835, Accuracy: 9734/10000 (97%)

CPU times: user 5min 28s, sys: 2.76 s, total: 5min 31s
Wall time: 1min 57s
```

In [7]:

```
# print(output)
```

2. Convolutional Neural Network

In [8]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR
```

In [9]:

```
transform = transforms.Compose([
                            transforms.ToTensor(),
                            transforms.Normalize((0.5,), (0.5,)),
                        ])

train_set = datasets.MNIST('./Dataset-3', download=True, train=True, transform=trans
val_set = datasets.MNIST('./Dataset-3', download=True, train=False, transform=transf
train_loader = torch.utils.data.DataLoader(train_set, batch_size=32, shuffle=True)
val_loader = torch.utils.data.DataLoader(val_set, batch_size=32, shuffle=True)
```

In [10]:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 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
```

In [11]:

```
def train(epoch, log_interval=200):
    # Set model to training mode
    model.train()

    # Loop over each batch from the training set
    for batch_idx, (data, target) in enumerate(train_loader):

#         data = data.view(data.shape[0], -1)
        # Copy data to GPU if needed
        data = data.to(device)
        target = target.to(device)

        # Zero gradient buffers
        optimizer.zero_grad()

        # Pass data through the network
        output = model(data)

        # Calculate loss
        loss = F.nll_loss(output, target)

        # Backpropagate
        loss.backward()

        # Update weights
        optimizer.step()

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

In [12]:

```
def validate(loss_vector, accuracy_vector):
    pred_arr = []
    model.eval()
    val_loss, correct = 0, 0
    with torch.no_grad():
        for data, target in val_loader:
    #         data = data.view(data.shape[0], -1)
            data = data.to(device)
            target = target.to(device)
            output = model(data)
            val_loss += F.nll_loss(output, target, reduction='sum').data.item()
            pred = output.data.max(1)[1] # get the index of the max log-probability
            for i in numpy.array(pred):
                pred_arr.append(int(i))
            correct += pred.eq(target.data).cpu().sum()

    val_loss /= len(val_loader)
    loss_vector.append(val_loss)

    accuracy = 100. * correct.to(torch.float32) / len(val_loader.dataset)
    accuracy_vector.append(accuracy)

    print('\nValidation set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.form
        val_loss, correct, len(val_loader.dataset), accuracy))

    return pred_arr
```

In [13]:

```
model = Net().to(device)
optimizer = optim.Adadelta(model.parameters(), lr=0.01)
scheduler = StepLR(optimizer, step_size=1, gamma=0.5)
```

In [14]:

```
%%time
epochs = 10

lossv, accv = [], []
for epoch in range(1, epochs + 1):
    train(epoch)
    output = validate(lossv, accv)
    scheduler.step()
```

```
Train Epoch: 9 [44800/60000 (75%)]      Loss: 0.210749
Train Epoch: 9 [51200/60000 (85%)]      Loss: 0.343448
Train Epoch: 9 [57600/60000 (96%)]      Loss: 0.174268

Validation set: Average loss: 6.1696, Accuracy: 9440/10000 (94%)

Train Epoch: 10 [0/60000 (0%)]  Loss: 0.131489
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.312594
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.088218
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.870516
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.364307
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.085118
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.342535
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.098029
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.260434
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.382975

Validation set: Average loss: 6.1661, Accuracy: 9441/10000 (94%)

CPU times: user 1h 1min 15s, sys: 2min 20s, total: 1h 3min 36s
```

In [15]:

```
# print(output)
```

Support Vector Machine

In [16]:

```
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.metrics import confusion_matrix
```

In [17]:

```
transform = transforms.Compose([
                            transforms.ToTensor(),
                            transforms.Normalize((0.5,), (0.5,)),
                        ])

train_set = datasets.MNIST('./Dataset-3', download=True, train=True, transform=trans
val_set = datasets.MNIST('./Dataset-3', download=True, train=False, transform=transf
train_loader = torch.utils.data.DataLoader(train_set)
val_loader = torch.utils.data.DataLoader(val_set)
```

In [18]:

```
x_train = []
y_train = []

x_val = []
y_val = []
```

In [19]:

```
for x,y in train_loader:
    img = numpy.array(numpy.reshape(x, (28, 28))).flatten()
    x_train.append(img)
    y_train.append(int(y))

for x,y in val_loader:
    img = numpy.array(numpy.reshape(x, (28, 28))).flatten()
    x_val.append(img)
    y_val.append(int(y))
```

In [20]:

```
x_train = numpy.array(x_train)
y_train = numpy.array(y_train)

x_val = numpy.array(x_val)
y_val = numpy.array(y_val)
```

In [21]:

```
# linear model

model_linear = SVC(C= 0.85, kernel='poly')
model_linear.fit(x_train, y_train)

# predict
y_pred = model_linear.predict(x_val)
```

In [22]:

```
# print(y_pred)
```

In [23]:

```
# accuracy
print("accuracy:", metrics.accuracy_score(y_true=y_val, y_pred=y_pred), "\n")

# confusion matrix
print(metrics.confusion_matrix(y_true=y_val, y_pred=y_pred))
```

```
accuracy: 0.9827

[[ 973    0    2    0    0    2    0    1    2    0]
 [    0 1131    1    1    0    1    0    1    0    0]
 [    5    1 1014    0    1    0    1    6    4    0]
 [    0    0    2  994    0    2    0    4    5    3]
 [    0    0    4    0  967    0    2    0    0    9]
 [    3    0    0    7    1  874    3    0    2    2]
 [    5    2    0    0    2    5  942    0    2    0]
 [    1    7    8    3    2    0    0 1000    0    7]
 [    4    0    2    3    3    2    2    3  951    4]
 [    2    2    0    5   10    3    0    5    1  981]]
```

In [ ]:

```

```