hw2_answer

November 9, 2019

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from torch import nn
import torch
from torchvision.datasets import MNIST
import torchvision.transforms as transforms
from sklearn.metrics import accuracy_score
%matplotlib inline
```

For this homework you will be using `pytorch` and `torchvision` library for neural networks and datasets. You can install them with `pip install torch torchvision`.

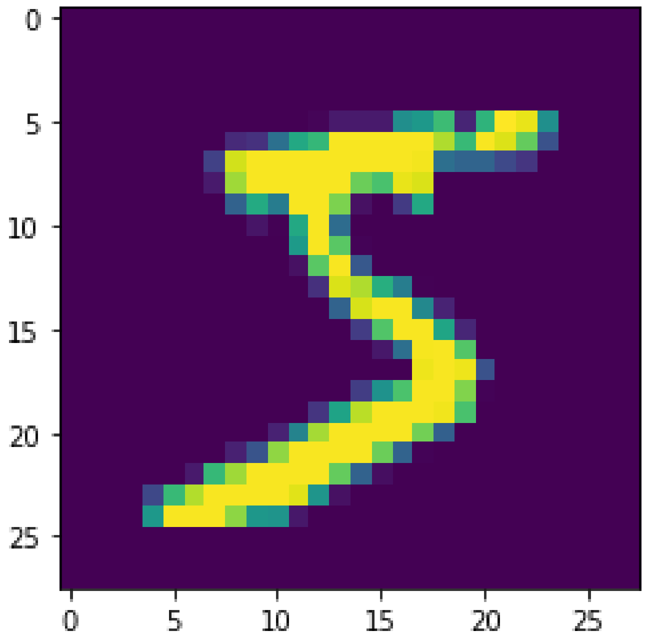
1 Question 1 Principal Component Analysis

This problem will guide you through the principal component analysis. You will be using a classical dataset, the MNIST hand written digit dataset.

[2]:
```
# Load the MNIST dataset
mnist = MNIST('.', download=True)
data = mnist.train_data.numpy()
labels = mnist.train_labels.numpy()
print('shapes:', data.shape, labels.shape)
plt.imshow(data[0])
print('label:', labels[0])
```

```
shapes: (60000, 28, 28) (60000,)
label: 5
```

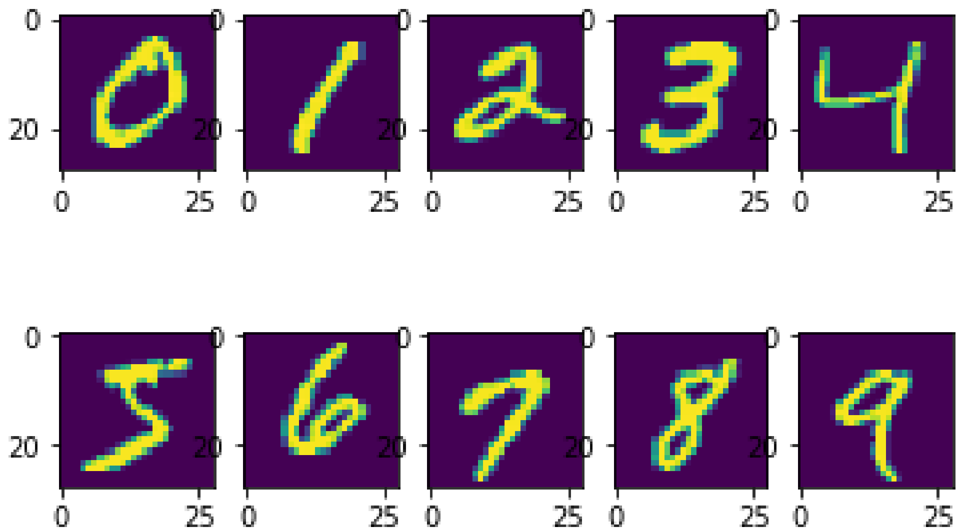
```
/home/fx/.local/lib/python3.6/site-packages/torchvision/datasets/mnist.py:53:
UserWarning: train_data has been renamed data
  warnings.warn("train_data has been renamed data")
/home/fx/.local/lib/python3.6/site-packages/torchvision/datasets/mnist.py:43:
UserWarning: train_labels has been renamed targets
  warnings.warn("train_labels has been renamed targets")
```



1.1 Question 1.1 Familiarize yourself with the data [5pt]

For this task, you will be using the torchvision package that provides the MNIST dataset. For each digit class(0-9), plot 1 image from the class and store those 10 images for each digit class in the array `digit_images`.

[3]:
```
digit_images = np.zeros([10, 28, 28])
### YOUR CODE HERE
for i in range(10):
    img = data[labels==i][0]
    digit_images[i] = img
    plt.subplot(2,5,i+1)
    plt.imshow(img)
### END OF CODE
```



1.2 Question 1.2 PCA

The following questions will guide you through the PCA algorithm.

1.2.1 Question 1.2.1 Centering the data [5pt]

For each image, flatten it to a 1-D vector. To perform PCA on the dataset, we first move the data points so they have 0 mean on each dimension. Store the centered data in variable `data_centered` and the mean of each dimension in variable `data_mean`.

[4]:
```
data_centered = None
data_mean = None
### YOUR CODE HERE
data_1d = data.reshape([data.shape[0], -1])
data_mean = data_1d.mean(0)
data_centered = data_1d - data_mean
### END OF CODE
```

1.2.2 Question 1.2.2 Compute the covariance matrix of the data [5pt]

You need to store the covariance matrix of the data in variable `data_covmat`. You may **not** use numpy.cov

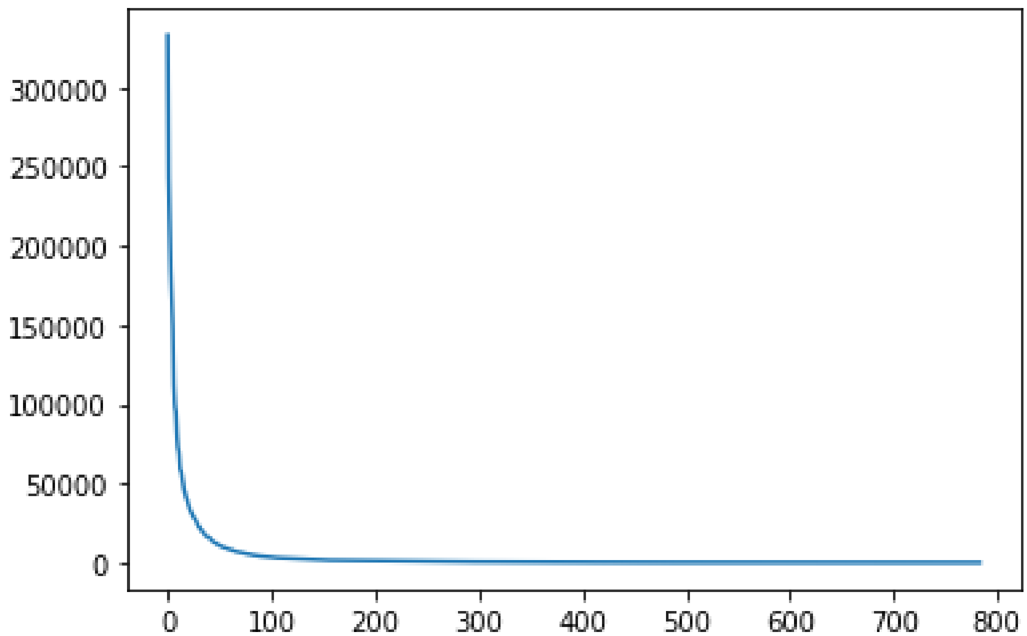
[5]:
```
data_covmat = None
### YOUR CODE HERE
data_covmat = (data_centered.T @ data_centered) /  (data_centered.shape[0]-1)
```

```
### END OF CODE
```

1.2.3 Question 1.2.3 Compute the eigenvalues of the covariance matrix [5pt]

You need to store the eigenvalues of the covariance matrix in variable `covmat_eig`, sorted in descending order. Then you need to plot the eigenvalues with `plt.plot`. You can use any numpy function.

[6]:
```
covmat_eig = None
### YOUR CODE HERE
covmat_eig, covmat_eigvectors = np.linalg.eigh(data_covmat)
covmat_eig = covmat_eig[::-1]
plt.plot(covmat_eig)
covmat_eigvectors = covmat_eigvectors[:, ::-1]
### END OF CODE
```



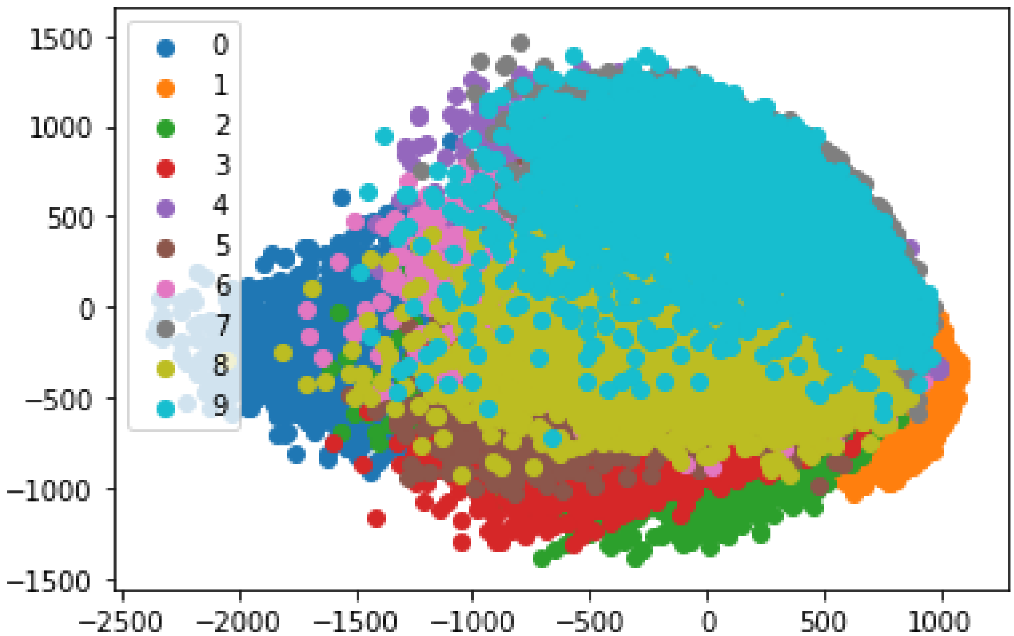
1.2.4 Question 1.2.4 Project data onto the first 2 principal components [5pt]

Now you need to project the centered data on the 2D space formed by the eigenvectors corresponding to the 2 largest eigenvalues. Create a 2D scatter plot where you need to assign a unique color to each digit class.

[7]:
```
### YOUR CODE HERE
projected = data_centered @ covmat_eigvectors[:,:2]
```

```
for i in range(10):
    plt.scatter(projected[labels==i][:,0], projected[labels==i][:, 1],␣
 ↪label=str(i))
plt.legend()
### END OF CODE
```

[7]: <matplotlib.legend.Legend at 0x7fb530a58128>

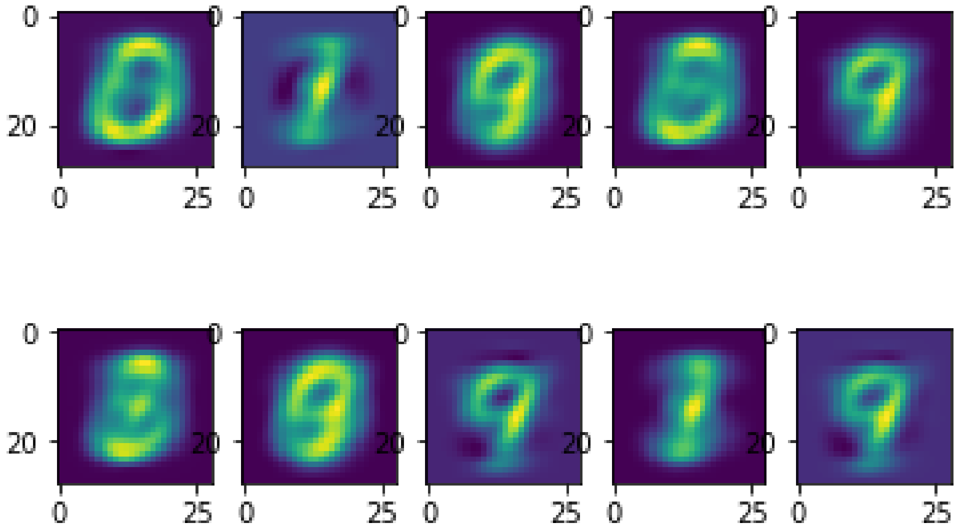


1.2.5 Question 1.2.5 Unproject data back to high dimensions [10pt]

For this question, you need to project the 10 images you plotted in **1.1** on the first 2 principal components, and then unproject the "compressed" 2-D representations back to the original space. Plot the "compressed" digit (the reconstructed digit). Do they look similar to the original images?

[8]:
```
### YOUR CODE HERE
# projected = np.zeros([10, 784])
# projected[:, :2] = (digit_images.reshape([10, -1]) - data_mean) @␣
 ↪covmat_eigvectors[:,:2]
# unprojected = projected @ covmat_eigvectors.T + data_mean
projected = (digit_images.reshape([10, -1]) - data_mean) @ covmat_eigvectors[:,:
 ↪2]
unprojected = projected @ covmat_eigvectors[:, :2].T + data_mean
unprojected = unprojected.reshape([10, 28, 28])
for i in range(10):
    plt.subplot(2, 5, i+1)
```

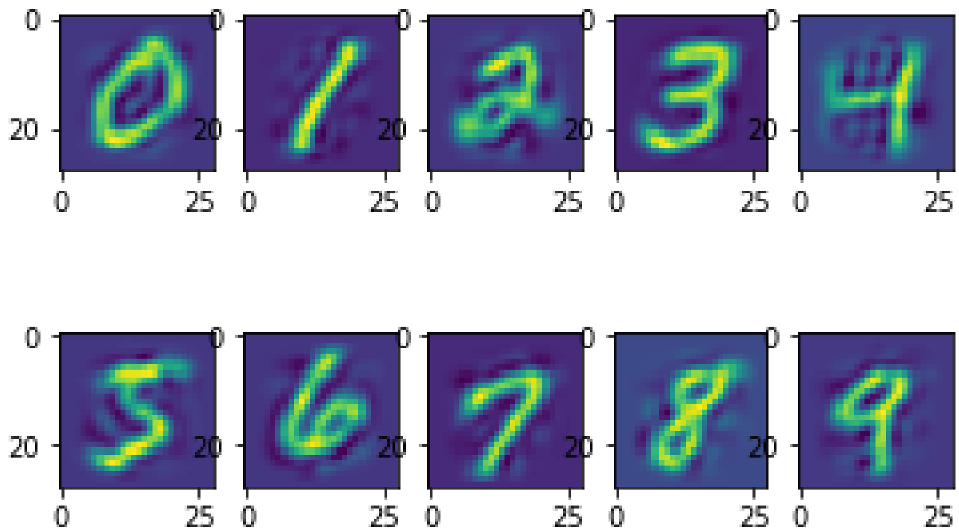
```
    plt.imshow(unprojected[i])
### END OF CODE
```



1.2.6 Question 1.2.6 Choose a better low dimension space. [5pt]

Do the previous problem with more dimensions (e.g. 3, 5, 10, 20, 50, 100). You only need to show results for one of them. Answer the following questinos. How many dimensions are required to represent the digits reasonably well? How are your results related to **question 1.2.3**?

[9]:
```
### YOUR CODE HERE
projected = (digit_images.reshape([10, -1]) - data_mean) @ covmat_eigvectors[:,:
 ↪50]
unprojected = projected @ covmat_eigvectors[:, :50].T + data_mean
unprojected = unprojected.reshape([10, 28, 28])
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(unprojected[i])
### END OF CODE
```



(Your explanation)

Any dimension between 10-100 is acceptable. From 1.2.3, we can see after the first 100 eigenvalues, the remaining eigenvalues are very close to 0, so those components are not important and can be discarded/compressed.

1.3 Question 1.3 Harris Corner and PCA [10pt]

Recall Harris corner detector algorithm: 1. Compute x and y derivatives (I_x, I_y) of an image 2. Compute products of derivatives (I_x^2, I_y^2, I_{xy}) at each pixel 3. Compute matrix M at each pixel, where

$$M(x_0, y_0) = \sum_{x,y} w(x - x_0, y - y_0) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Here, we set weight $w(x, y)$ to be a box filter of size 3×3 (the box is placed centered at (x_0, y_0)).

In this problem, you need to show that Harris Corner detector is really just principal component analysis in the gradient space. Your explanation should answer the following quesions. 1. As we know, PCA is performed on data points. What are the data points in Harris corner detector when we think of it as a PCA? 2. What is the covariance matrix used in Harris corner detector and why it is a covariance matrix? 3. What are the principal components in Harris corner detector? 4. Briefly explain how principal components imply ”cornerness”.

(Your proof here) 1. The data points are the x-direction derivative and y-direction derivatives. 2. The ”covariance matrix” in Harris corner is the matrix M, assuming the data are zero-mean. It is computed as XX^T where each column of X is a pair of I_x, I_y in the neighborhood of a pixel, so it is covariance matrix. (It is actually slightly different from PCA in that the zero-mean is assumed. Correctly identifying this will also be considered as correct for this problem) 3. The principal components are the eigenvectors of M. 4. Since the data is the derivatives, PCA detects 2 dominant directions for local derivatives. When the dominant directions are both large, PCA tells us that there are 2 directions with large derivatives, which implies the pixel is a n corner. This explanation aligns with the Harris Corner algorithm.

2 Question 2 KNN, Softmax Regression

[10]:
```
train_dataset = MNIST(root='.', train=True, transform=transforms.ToTensor,␣
↪download=True)
test_dataset = MNIST('.', train=False, transform=transforms.ToTensor())
train_X = train_dataset.data.numpy()  # training data, uint8 type to reduce␣
↪memory and comparison cost
train_y = train_dataset.targets.numpy()  # training label
test_X = test_dataset.data.numpy() # testing data, uint8 to reduce memory and␣
↪comparison cost
test_y = test_dataset.targets.numpy()  # testing label
```

[11]:
```
train_X = train_X.reshape((train_X.shape[0], -1))
test_X = test_X.reshape((test_X.shape[0], -1))
```

2.1 Question 2.1 K-Nearest Neighbor [10pt]

In this problem you will be implementing the KNN classifier. Fill in the functions in the starter code below. You are are allowed to use `scipy.spatial.KDTree` and `scipy.stats.mode` (in case of a tie, pick any one). Please avoid `sklearn.neighbors.KDTree` as it appears extremely slow. You are **not** allowed to use a library KNN function that directly solves the problem.

If you do not know what a KD-tree is, please read the documentation for `scipy.spatial.KDTree` to understand how you can use it.

[12]:
```
from scipy.spatial import KDTree
from scipy.stats import mode
```

[15]:
```
class KNNClassifier:
    def __init__(self, num_neighbors):
        """
        construct the classifier
        Args:
            num_centers: number of neighbors
        """
        ### YOU CODE HERE
        self.num_neighbors = num_neighbors
        ### END OF CODE

    def fit(self, X, y):
        """
        train KNN classifier
```

```
        Args:
            X: training data, numpy array with shape (Nxk) where N is number of␣
↪data points, k is number of features
            y: training labels, numpy array with shape (N)
        """
        ### YOU CODE HERE
        self.kdtree_ = KDTree(X)
        self.y_ = y
        ### END OF CODE
        return self

    def predict(self, X):
        """
        predict labels
        Args:
            X: testing data, numpy array with shape (Mxk) where M is number of␣
↪data points, k is number of features
        Return:
            y: predicted labels, numpy array with shape (M)
        """
        ### YOU CODE HERE
        dist, ind = self.kdtree_.query(X, k=self.num_neighbors)
        pred = np.array([m[0] for m in mode(self.y_[ind], axis=1).mode])
        return pred
        ### END OF CODE
```

[16]:
```
from sklearn.metrics import accuracy_score
knn = KNNClassifier(3).fit(train_X, train_y)
pred_y = knn.predict(test_X)
print('KNN accuracy:', accuracy_score(test_y, pred_y))
```

```
KNN accuracy: 0.5543
```

2.2 Question 2.2 Softmax Regression

In this problm, you will be implementing the softmax regression(multi-class logistic regression). Here is a brief recap of several important concepts. Suppose the number of features in data points is m,

1. Softmax function S normalize a vector to have sum 1. (it turns any vector into a probability distribution)

$$S(x) = [\frac{e^{x_1}}{\sum_{j=1}^{m} e^{x_j}}, \frac{e^{x_2}}{\sum_{j=1}^{m} e^{x_j}}, ..., \frac{e^{x_m}}{\sum_{j=1}^{m} e^{x_j}}]$$

2. Cross entropy loss J is the multiclass logistic regression loss.

$$J(y', y) = -\sum_{i=1}^{m} y'_i \log y_i$$

where y' is the one-hot ground truth label and y is the predicted label distribution.

3. Softmax regression is the following optimization problem.

$$\min_{W,b} \sum_{(X,y')\in\{\text{training set}\}} J(y', S(Wx+b))$$

4. This objective is optimized with gradient descent. Let

$$L = \sum_{(x,y')\in\{\text{training set}\}} J(y', S(Wx+b))$$

Update W and b with $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial b}$.

2.2.1 Question 2.2.1 Compute the gradients [10pt]

In this question, you need to do the following: 1. Compute the gradient $\frac{\partial J}{\partial y}$. i.e. compute

$$\frac{\partial J}{\partial y_i}$$

Express it in terms of y_i' and y_i. 2. Let $u = Wx+b$, $y_i = S_i(u_j)$ Compute

$$\frac{\partial y_i}{\partial u_j}$$

Express it in terms of y_i, y_j and δ_{ij}, where

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

3. Compute

$$\frac{\partial J}{\partial W_{jk}} \text{ and } \frac{\partial J}{\partial b_j}$$

Express them in terms of y_j, y_j', x_k. Explain your results in a intuitive way. Hint: they should have a very simple form that makes sense intuitively. 4. Compute

$$\frac{\partial J}{\partial W}$$

in the matrix form. It should be a matrix with the same shape as W, and entry jk is $\frac{\partial J}{\partial W_{jk}}$. Similarly, compute

$$\frac{\partial J}{\partial b}$$

(Your proof here) 1.

$$\frac{\partial J}{\partial y_i} = -\frac{y_i'}{y_i}$$

2.

$$\frac{\partial y_i}{\partial u_j} = \begin{cases} S(u_i)(1 - S(u_i)) & i = j \\ -S(u_i)S(u_j) & i \neq j \end{cases} = y_i(\delta_{ij} - y_j)$$

3.

$$u = Wx + b$$
$$u_j = \sum_{k=1}^{m} W_{jk} x_k + b_j$$
$$\frac{\partial u_j}{\partial W_{jk}} = x_k, \frac{\partial u_j}{\partial b_j} = 1$$
$$\frac{\partial J}{\partial W_{jk}} = \sum_i \frac{\partial J}{\partial y_i} \sum_{j'} \frac{\partial y_i}{\partial u_j} \frac{\partial u_{j'}}{\partial W_{jk}} = \sum_i \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial u_j} \frac{\partial u_j}{\partial W_{jk}} = x_k(y_j - y'_j)$$
$$\frac{\partial J}{\partial b_j} = \sum_i \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial u_j} \frac{\partial u_j}{\partial b_j} = y_j - y'_j$$

4.

$$\frac{\partial J}{\partial W} = (y - y')x^T$$
$$\frac{\partial J}{\partial b} = y - y'$$

2.2.2 Question 2.2.2 Stochastic Gradient Descent [10pt]

In gradient descent algorithm, we update W and b with $\partial L/\partial W$ and $\partial L/\partial b$. However, this requires the gradient w.r.t. the whole dataset. Computing such gradient is very slow. Instead, we can update the weights with per-data gradient. This is known as the SGD algorithm, which runs much faster. You need to take the following steps. 1. Implement softmax function S. We need to take special care in this function since e^x tends to overflow easily. However, we observe that $S(x) = S(x - m)$ for any constant vector m. We can stabilize softmax using $S(x) = S(x - \max(x))$. 2. Implement function `J`(loss) and `dJ`(loss gradient). Note: `J` is not required to run the algorithm, but you may want to implement it for debug purposes. 3. Implement the SGD algorithm. 4. Run the algorithm for 20 epochs (each epoch iterates the whole data set once) with learning rate `1e-3` and report accuracy on test set. You may use `sklearn.metrics.accuracy_score`. Hint: accuracy should be $> 90\%$.

[17]:
```
train_dataset = MNIST(root='.', train=True, transform=transforms.ToTensor,␣
 ↪download=True)
test_dataset = MNIST('.', train=False, transform=transforms.ToTensor())
train_X = train_dataset.data.numpy() / 255.  # normalize data to 0-1
train_y = train_dataset.targets.numpy()  # training label
test_X = test_dataset.data.numpy() / 255.  # normalize data to 0-1
test_y = test_dataset.targets.numpy()  # testing label
train_X = train_X.reshape((train_X.shape[0], -1))
test_X = test_X.reshape((test_X.shape[0], -1))
```

[18]:
```
def softmax(x):
    """
    softmax function
    Args:
        x: a 1-d numpy array
```

```python
    Return:
        results of softmax(x)
    """
    ### YOUR CODE HERE
    x -= np.max(x)
    ex = np.exp(x)
    return ex / np.sum(ex)
    ### END OF CODE

def J(W, b, y_true, x):
    """
    Softmax Loss function
    Args:
        W: weights (num_classes x num_features)
        b: bias (num_features)
        y_true: ground truth 1-hot label (num_classes)
        x: input data
    Return:
        J(y', y)
    """
    ### YOUR CODE HERE
    return -np.sum(y_true * softmax(W @ x + b))
    ### END OF CODE

def dJ(W, b, y_true, x):
    """
    Softmax Loss gradient
    Args:
        W: weights (num_classes x num_features)
        b: bias (num_features)
        y_true: ground truth 1-hot label (num_classes)
        x: input data (num_features)
    Return:
        (dW, db): gradient w.r.t. W and b
    """
    ### YOUR CODE HERE
    s = softmax(W @ x + b)
    db = s - y_true

    dw = np.outer(db, x)
    return dw, db
    ### END OF CODE
```

[19]:
```python
from tqdm import tqdm_notebook
```

[20]:
```python
def SGD(f, df, Xs, ys, n_classes=10, lr=1e-3, max_epoch=20):
    """
```

```
    Args:
        f: function to optimize
        df: the gradient of the function
        Xs: input data, numpy array with shape (Nxm) where N is the number of␣
↪data points, m is the number of features
        ys: true label, numpy array with shape (N x num_classes)
        lr: learning rate
        max_epoch: maximum epochs to run SGD
    Return:
        optimal weights and biases
    """
    N, m = Xs.shape
    W = np.random.rand(n_classes, m) - 0.5
    b = np.random.rand(n_classes) - 0.5

    ### YOUR CODE HERE
    for epoch in tqdm_notebook(range(max_epoch)):
        for x, y in zip(Xs, ys):
            dW, db = dJ(W, b, y, x)
            W -= lr * dW
            b -= lr * db
    return W, b
    ### END OF CODE
```

[21]:
```
train_y_onehot = np.zeros((train_y.shape[0], 10))
train_y_onehot[np.arange(len(train_y)), train_y] = 1
W, b = SGD(J, dJ, train_X, train_y_onehot, 10, max_epoch=20)
accuracy_score(test_y, np.argmax(test_X @ W.T + b, axis=1))
```

```
HBox(children=(IntProgress(value=0, max=20), HTML(value='')))
```

[21]: 0.9198

3 Question 3 Convolutional Neural Networks

This question requires you to use the PyTorch framework for neural network training. You will not need GPU to train the networks for this problem.

The following is a code sample for training a simple multi-layer perceptron neural network using PyTorch.

[22]:
```
train_dataset = MNIST(root='.', train=True, transform=transforms.ToTensor(),␣
↪download=True)
test_dataset = MNIST('.', train=False, transform=transforms.ToTensor())
```

[23]:
```
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        """init function builds the required layers"""
        super(MLP, self).__init__()  # This line is always required
        # Hidden layer
        self.layer1 = nn.Linear(input_size, hidden_size)
        # activation
        self.relu = nn.ReLU()
        # output layer
        self.layer2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        """forward function describes how input tensor is transformed to output␣
↪tensor"""
        # flatten the input from (Nx1x28x28) to (Nx784)
        torch.flatten(x, 1)
        x = self.layer1(x)
        x = self.relu(x)
        x = self.layer2(x)
        # Note we do not need softmax layer, since this layer is included in␣
↪the CrossEntropyLoss provided by torch
        return x
```

[24]:
```
model = MLP(784, 1024, 10)
model
```

[24]:
```
MLP(
  (layer1): Linear(in_features=784, out_features=1024, bias=True)
  (relu): ReLU()
  (layer2): Linear(in_features=1024, out_features=10, bias=True)
)
```

[25]:
```
opts = {
    'lr': 5e-4,
    'epochs': 5,
    'batch_size': 64
}
```

[26]:
```
optimizer = torch.optim.Adam(model.parameters(), opts['lr'])  # Adam is a much␣
↪better optimizer compared to SGD
criterion = torch.nn.CrossEntropyLoss()  # loss function
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,␣
↪batch_size=opts['batch_size'], shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,␣
↪batch_size=opts['batch_size'], shuffle=True)
```

[27]:
```
from tqdm import tqdm_notebook
```

[28]:
```
for epoch in range(opts['epochs']):
    train_loss = []
    for i, (data, labels) in tqdm_notebook(enumerate(train_loader),␣
↪total=len(train_loader)):
        # reshape data
        data = data.reshape([-1, 784])
        # pass data through network
        outputs = model(data)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()  # Important! Otherwise the optimizer will␣
↪accumulate gradients from previous runs!
        loss.backward()
        optimizer.step()
        train_loss.append(loss.item())
    test_loss = []
    test_accuracy = []
    for i, (data, labels) in enumerate(test_loader):
        # reshape data
        data = data.reshape([-1, 784])
        # pass data through network
        outputs = model(data)
        _, predicted = torch.max(outputs.data, 1)
        loss = criterion(outputs, labels)
        test_loss.append(loss.item())
        test_accuracy.append((predicted == labels).sum().item() / predicted.
↪size(0))
    print('epoch: {}, train loss: {}, test loss: {}, test accuracy: {}'.
↪format(epoch, np.mean(train_loss), np.mean(test_loss), np.
↪mean(test_accuracy)))
```

```
HBox(children=(IntProgress(value=0, max=938), HTML(value='')))
```

```
epoch: 0, train loss: 0.2908635719109382, test loss: 0.14501928165555, test
accuracy: 0.9573049363057324
```

```
HBox(children=(IntProgress(value=0, max=938), HTML(value='')))
```

```
epoch: 1, train loss: 0.11394707337299835, test loss: 0.09451216615878852, test
accuracy: 0.9721337579617835
```

```
HBox(children=(IntProgress(value=0, max=938), HTML(value='')))
```

```
epoch: 2, train loss: 0.07377839135899664, test loss: 0.07849399797665844, test
accuracy: 0.9757165605095541
```

```
HBox(children=(IntProgress(value=0, max=938), HTML(value='')))
```

```
epoch: 3, train loss: 0.05201037996597509, test loss: 0.06912176294999707, test
accuracy: 0.9784036624203821
```

```
HBox(children=(IntProgress(value=0, max=938), HTML(value='')))
```

```
epoch: 4, train loss: 0.03779585879228548, test loss: 0.06736240834946845, test
accuracy: 0.9775079617834395
```

3.1 Question 3.1 Implementing CNN [15pt]

You need to implement a convolutional neural network for the same task as above. You may find the PyTorch documentation helpful. https://pytorch.org/docs/stable/nn.html

You will need to adjust the network size and training options for better performance. We provide a working network structure. In convolutional layers, (conv MxM, N) means the layer has kernel size M by M and N output channels.

(conv 5x5, 32) -> (relu) -> (maxpool 2x2) -> (conv 5x5, 64) -> (relu) -> (maxpool 2x2) -> (flatten) -> (linear 1024) -> (output 10)

For full score, you need to achieve 99% testing accuracy. Also, plot the hand-written digits that your network got wrong.

[29]:
```
class CNN(nn.Module):
    def __init__(self, input_size, num_classes):
        """
        init convolution and activation layers
        Args:
            input_size: (1,28,28)
            num_classes: 10
        """
        super(CNN, self).__init__()
        ### YOUR CODE HERE
        self.conv1 = nn.Conv2d(1, 32, 5, 1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2)

        self.conv2 = nn.Conv2d(32, 64, 5, 1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2)

        self.dense = nn.Linear(1024, 10)
        ### END OF CODE
```

```
    def forward(self, x):
        """
        forward function describes how input tensor is transformed to output␣
↪tensor
        Args:
            x: (Nx1x28x28) tensor
        """
        ### YOUR CODE HERE
        x = self.pool2(self.relu2(self.conv2(self.pool1(self.relu1(self.
↪conv1(x))))))
        x = torch.flatten(x, 1)
        x = self.dense(x)
        ### END OF CODE
        return x
```

[30]:
```
model = CNN((1, 28, 28), 10)
model
```

[30]:
```
CNN(
  (conv1): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
  (relu1): ReLU()
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
  (relu2): ReLU()
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (dense): Linear(in_features=1024, out_features=10, bias=True)
)
```

[31]:
```
### You may (and should) change these
opts = {
    'lr': 1e-3,
    'epochs': 5,
    'batch_size': 64
}

### if you cannot get 99% with SGD, Adam optimizer can help you
optimizer = torch.optim.Adam(model.parameters(), opts['lr'])
```

[32]:
```
criterion = torch.nn.CrossEntropyLoss()  # loss function
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,␣
↪batch_size=opts['batch_size'], shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,␣
↪batch_size=opts['batch_size'], shuffle=True)
```

[33]:
```
for epoch in range(opts['epochs']):
    train_loss = []
    for i, (data, labels) in tqdm_notebook(enumerate(train_loader),␣
↪total=len(train_loader)):
        # pass data through network
        outputs = model(data)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss.append(loss.item())
    test_loss = []
    test_accuracy = []
    for i, (data, labels) in enumerate(test_loader):
        # pass data through network
        outputs = model(data)
        _, predicted = torch.max(outputs.data, 1)
        loss = criterion(outputs, labels)
        test_loss.append(loss.item())
        test_accuracy.append((predicted == labels).sum().item() / predicted.
↪size(0))
    print('epoch: {}, train loss: {}, test loss: {}, test accuracy: {}'.
↪format(epoch, np.mean(train_loss), np.mean(test_loss), np.
↪mean(test_accuracy)))
```

```
HBox(children=(IntProgress(value=0, max=938), HTML(value='')))
```

```
epoch: 0, train loss: 0.16646374586914012, test loss: 0.06297704938111032, test
accuracy: 0.9796974522292994
```

```
HBox(children=(IntProgress(value=0, max=938), HTML(value='')))
```

```
epoch: 1, train loss: 0.053430334921044584, test loss: 0.0351612860823323, test
accuracy: 0.9878582802547771
```

```
HBox(children=(IntProgress(value=0, max=938), HTML(value='')))
```

```
epoch: 2, train loss: 0.03673469620659502, test loss: 0.03334018622471648, test
accuracy: 0.988953025477707
```

```
HBox(children=(IntProgress(value=0, max=938), HTML(value='')))
```

```
epoch: 3, train loss: 0.02859619347307124, test loss: 0.033397792132606935, test
```

```
accuracy: 0.9892515923566879
```

```
HBox(children=(IntProgress(value=0, max=938), HTML(value='')))
```

```
epoch: 4, train loss: 0.022694049059534507, test loss: 0.02580334813256932, test
accuracy: 0.9912420382165605
```

3.2 Question 3.2 Kernel weights visualization [5pt]

For this question, you need to visualize the kernel weights for your first convolutional layer. Suppose you have 5x5 kernels with 32 output channels. You will plot 32 5x5 images.

hint: You might need to look at PyTorch documentation (or play with the PyTorch model) to figure out how to get the weights.

[49]:
```python
weights = list(model.conv1.parameters())[0].detach().numpy().squeeze()
```

[54]:
```python
for i in range(32):
    plt.subplot(4, 8, i+1)
    plt.imshow(weights[i])
    plt.axis('off')
```