Problem 1

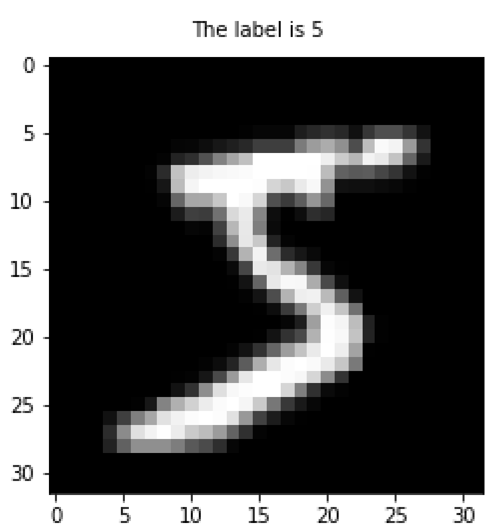
In [1]:
```
import numpy as np
from datetime import datetime

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

from torchvision import datasets, transforms

%matplotlib inline
import matplotlib.pyplot as plt
```

In [2]:
```
# define transforms
transforms = transforms.Compose([transforms.Resize((32, 32)),
                                 transforms.ToTensor()])

# download and create datasets
train_dataset = datasets.MNIST(root='mnist_data',
                               train=True,
                               transform=transforms,
                               download=True)

valid_dataset = datasets.MNIST(root='mnist_data',
                               train=False,
                               transform=transforms)
```

1.1.1

In [3]:
```
plt.imshow(train_dataset[0][0][0], cmap='gray')
plt.text(10, -2, 'The label is ' + str("5"))
```

Out[3]:
```
Text(10, -2, 'The label is 5')
```



In [4]:
```
# hyper parameters
RANDOM_SEED = 42
LEARNING_RATE = 0.001
BATCH_SIZE = 32
N_EPOCHS = 15
```

```
IMG_SIZE = 32
N_CLASSES = 10
```

1.1.2

In [5]:
```
# define the data loaders
train_loader = DataLoader(dataset=train_dataset,
                          batch_size=BATCH_SIZE,
                          shuffle=True)

valid_loader = DataLoader(dataset=valid_dataset,
                          batch_size=BATCH_SIZE,
                          shuffle=True)
```

1.1.3

In [6]:
```
def train(train_loader, model, criterion, optimizer):
    '''
    Train one epoch.
    '''

    model.train()
    running_loss = 0

    for X, y_true in train_loader:

        optimizer.zero_grad()


        # Forward pass
        y_hat, _ = model(X)
        loss = criterion(y_hat, y_true)
        running_loss += loss.item() * X.size(0)

        # Backward pass
        loss.backward()
        optimizer.step()

    epoch_loss = running_loss / len(train_loader.dataset)
    return model, optimizer, epoch_loss
```

1.1.4

In [7]:
```
def validate(valid_loader, model, criterion):
    '''
    Function for the validation step of the training loop.
    Returns the model and the loss on the test set.
    '''

    model.eval()
    running_loss = 0

    for X, y_true in valid_loader:

        # Forward pass and record loss
        y_hat, _ = model(X)
        loss = criterion(y_hat, y_true)

        running_loss += loss.item() * X.size(0)
```

```python
    epoch_loss = running_loss / len(valid_loader.dataset)

    return model, epoch_loss
```

In [8]:

```python
def training_loop(model, criterion, optimizer, train_loader, valid_loader, epochs, print
    '''
    Function defining the entire training loop
    '''

    # set objects for storing metrics
    best_loss = 1e10
    train_losses = []
    valid_losses = []
    train_accs = []
    valid_accs = []

    # Train model
    for epoch in range(0, epochs):

        # training
        model, optimizer, train_loss = train(train_loader, model, criterion, optimizer)
        train_losses.append(train_loss)

        # validation
        with torch.no_grad():
            model, valid_loss = validate(valid_loader, model, criterion)
            valid_losses.append(valid_loss)

        if epoch % print_every == (print_every - 1):

            train_acc = get_accuracy(model, train_loader,)
            train_accs.append(train_acc)
            valid_acc = get_accuracy(model, valid_loader)
            valid_accs.append(valid_acc)

            print(f'{datetime.now().time().replace(microsecond=0)} '
                  f'Epoch: {epoch}\t'
                  f'Train loss: {train_loss:.4f}\t'
                  f'Valid loss: {valid_loss:.4f}\t'
                  f'Train accuracy: {100 * train_acc:.2f}\t'
                  f'Valid accuracy: {100 * valid_acc:.2f}')

    performance = {
        'train_losses':train_losses,
        'valid_losses': valid_losses,
        'train_acc': train_accs,
        'valid_acc':valid_accs
    }

    return model, optimizer, performance
```

1.1.5

In [9]:

```python
def get_accuracy(model, data_loader):
    '''
    Function for computing the accuracy of the predictions over the entire data_loader
    '''

    correct_pred = 0
    n = 0

    with torch.no_grad():
```

```
        model.eval()
        for X, y_true in data_loader:

            y_hat, _ = model(X)
            _, predicted_labels = torch.max(y_hat.data, 1)

            n += y_true.size(0)
            correct_pred += (predicted_labels == y_true).sum()

    return correct_pred.float() / n



def plot_performance(performance):
    '''
    Function for plotting training and validation losses
    '''

    # temporarily change the style of the plots to seaborn
    plt.style.use('seaborn')

    fig, ax = plt.subplots(1, 2, figsize = (16, 4.5))
    for key, value in performance.items():
        if 'loss' in key:
            ax[0].plot(value, label=key)
        else:
            ax[1].plot(value, label=key)
    ax[0].set(title="Loss Over Epochs",
            xlabel='Epoch',
            ylabel='Loss')
    ax[1].set(title="Accuracy Over Epochs",
            xlabel='Epoch',
            ylabel='Loss')
    ax[0].legend()
    ax[1].legend()
    plt.show()

    # change the plot style to default
    plt.style.use('default')
```

1.2.1

In [10]:

```
class LeNet5(nn.Module):

    def __init__(self, n_classes):
        super(LeNet5, self).__init__()

        self.convolution_layer = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, stride=1),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=2, stride=2, padding=0),
            nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stride=1),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=2, stride=2, padding=0),
            nn.Conv2d(in_channels=16, out_channels=120, kernel_size=5, stride=1),
            nn.Tanh()
        )

        self.linear_layer = nn.Sequential(
            nn.Linear(in_features=120, out_features=84),
            nn.Tanh(),
            nn.Linear(in_features=84, out_features=n_classes)
        )
```

```
    def forward(self, x):
        x = self.convolution_layer(x)
        x = torch.flatten(x, 1)
        x = self.linear_layer(x)
        #logits = self.classifier(x)
        logits = x
        probs = F.softmax(logits, dim=1)
        return logits, probs
```

1.2.2

In [11]:
```
class MLP(nn.Module):

    def __init__(self, layers):
        super(MLP, self).__init__()



        self.all_layers = []
        self.all_layers.append(nn.Flatten())
        self.all_layers.append(nn.Linear(layers[0], layers[1]))
        for i in range(len(layers) - 2):
            self.all_layers.append(nn.Tanh())
            self.all_layers.append(nn.Linear(layers[i+1], layers[i+2]))

        self.layers = nn.Sequential(*self.all_layers)


    def forward(self, x):
        x = self.layers(x)
        logits = x
        probs = F.softmax(logits, dim=1)
        return logits, probs
```

1.3.1

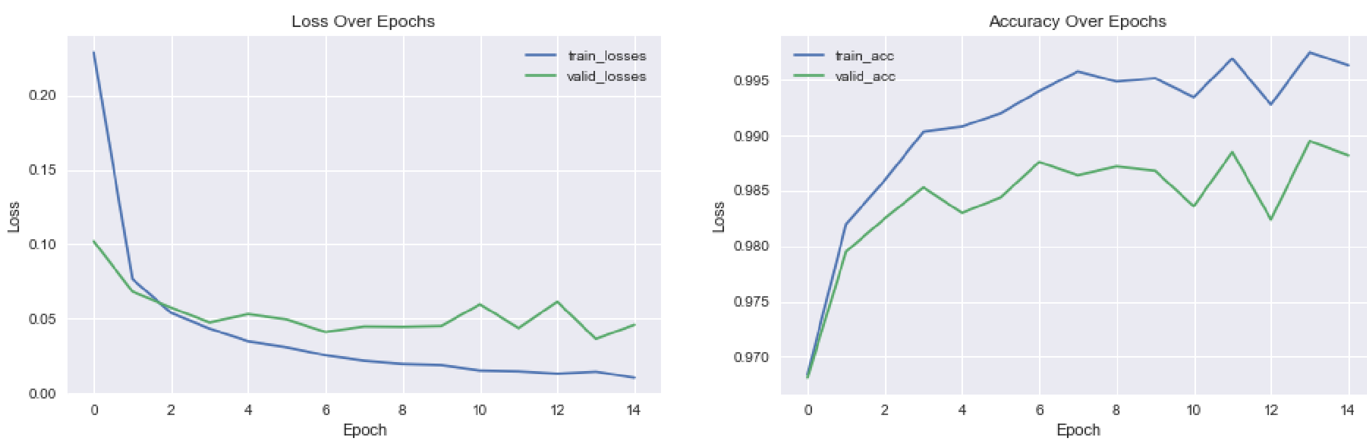
In [12]:
```
torch.manual_seed(RANDOM_SEED)

model = LeNet5(N_CLASSES)
print(model)
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
criterion = nn.CrossEntropyLoss()
```

```
LeNet5(
  (convolution_layer): Sequential(
    (0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): Tanh()
    (2): AvgPool2d(kernel_size=2, stride=2, padding=0)
    (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (4): Tanh()
    (5): AvgPool2d(kernel_size=2, stride=2, padding=0)
    (6): Conv2d(16, 120, kernel_size=(5, 5), stride=(1, 1))
    (7): Tanh()
  )
  (linear_layer): Sequential(
    (0): Linear(in_features=120, out_features=84, bias=True)
    (1): Tanh()
    (2): Linear(in_features=84, out_features=10, bias=True)
  )
)
```

In [13]:
```
model, optimizer, performance_1 = training_loop(model, criterion, optimizer, train_loade
#plot_performance(performance_1)
```

```
18:12:51 Epoch: 0       Train loss: 0.2290      Valid loss: 0.1020      Train accuracy:
96.84   Valid accuracy: 96.81
18:13:22 Epoch: 1       Train loss: 0.0766      Valid loss: 0.0681      Train accuracy:
98.20   Valid accuracy: 97.95
18:13:53 Epoch: 2       Train loss: 0.0538      Valid loss: 0.0573      Train accuracy:
98.60   Valid accuracy: 98.25
18:14:25 Epoch: 3       Train loss: 0.0432      Valid loss: 0.0473      Train accuracy:
99.03   Valid accuracy: 98.53
18:14:56 Epoch: 4       Train loss: 0.0346      Valid loss: 0.0529      Train accuracy:
99.08   Valid accuracy: 98.30
18:15:27 Epoch: 5       Train loss: 0.0306      Valid loss: 0.0493      Train accuracy:
99.20   Valid accuracy: 98.44
18:15:58 Epoch: 6       Train loss: 0.0253      Valid loss: 0.0409      Train accuracy:
99.40   Valid accuracy: 98.76
18:16:29 Epoch: 7       Train loss: 0.0216      Valid loss: 0.0446      Train accuracy:
99.58   Valid accuracy: 98.64
18:17:00 Epoch: 8       Train loss: 0.0194      Valid loss: 0.0444      Train accuracy:
99.49   Valid accuracy: 98.72
18:17:31 Epoch: 9       Train loss: 0.0187      Valid loss: 0.0450      Train accuracy:
99.52   Valid accuracy: 98.68
18:18:02 Epoch: 10      Train loss: 0.0150      Valid loss: 0.0595      Train accuracy:
99.35   Valid accuracy: 98.36
18:18:33 Epoch: 11      Train loss: 0.0145      Valid loss: 0.0435      Train accuracy:
99.69   Valid accuracy: 98.85
18:19:04 Epoch: 12      Train loss: 0.0129      Valid loss: 0.0613      Train accuracy:
99.28   Valid accuracy: 98.24
18:19:35 Epoch: 13      Train loss: 0.0142      Valid loss: 0.0363      Train accuracy:
99.75   Valid accuracy: 98.95
18:20:06 Epoch: 14      Train loss: 0.0101      Valid loss: 0.0458      Train accuracy:
99.63   Valid accuracy: 98.82
```

In [14]:
```
plot_performance(performance_1)
```



1.3.2

In [15]:
```
torch.manual_seed(RANDOM_SEED)

layers = [1024, 256, 64, 16, N_CLASSES]
model = MLP(layers)
print(model)
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
criterion = nn.CrossEntropyLoss()
```

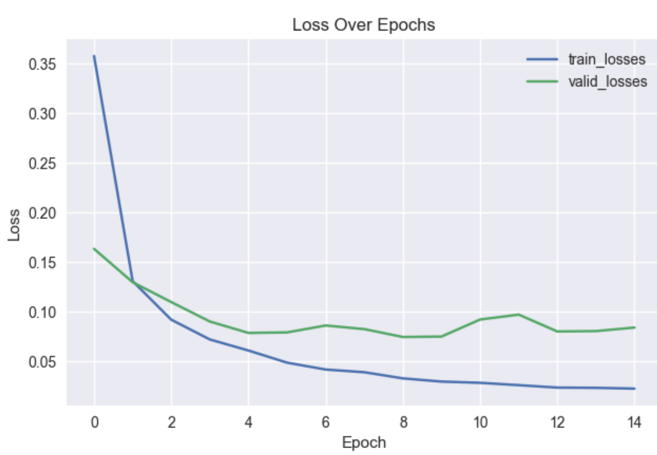
```
MLP(
  (layers): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
```

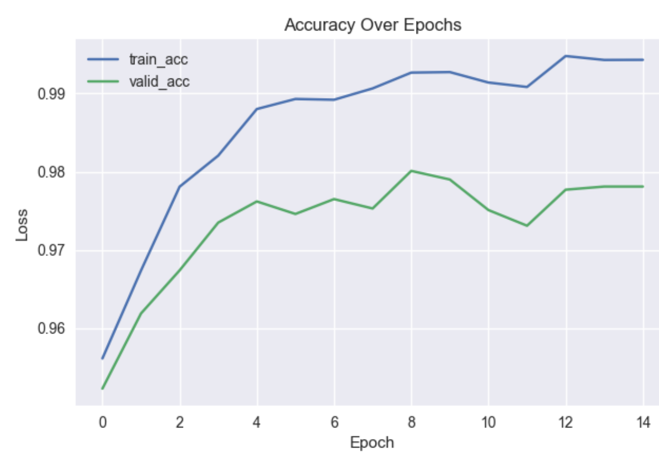
```
    (1): Linear(in_features=1024, out_features=256, bias=True)
    (2): Tanh()
    (3): Linear(in_features=256, out_features=64, bias=True)
    (4): Tanh()
    (5): Linear(in_features=64, out_features=16, bias=True)
    (6): Tanh()
    (7): Linear(in_features=16, out_features=10, bias=True)
  )
)
```

In [16]:
```
model, optimizer, performance_2 = training_loop(model, criterion, optimizer, train_loade
```

```
18:20:27 Epoch: 0	Train loss: 0.3575	Valid loss: 0.1636	Train accuracy:
95.61	Valid accuracy: 95.23
18:20:48 Epoch: 1	Train loss: 0.1311	Valid loss: 0.1300	Train accuracy:
96.74	Valid accuracy: 96.19
18:21:09 Epoch: 2	Train loss: 0.0923	Valid loss: 0.1100	Train accuracy:
97.81	Valid accuracy: 96.74
18:21:30 Epoch: 3	Train loss: 0.0724	Valid loss: 0.0905	Train accuracy:
98.21	Valid accuracy: 97.35
18:21:51 Epoch: 4	Train loss: 0.0613	Valid loss: 0.0790	Train accuracy:
98.80	Valid accuracy: 97.62
18:22:12 Epoch: 5	Train loss: 0.0491	Valid loss: 0.0795	Train accuracy:
98.93	Valid accuracy: 97.46
18:22:33 Epoch: 6	Train loss: 0.0421	Valid loss: 0.0864	Train accuracy:
98.92	Valid accuracy: 97.65
18:22:54 Epoch: 7	Train loss: 0.0394	Valid loss: 0.0828	Train accuracy:
99.06	Valid accuracy: 97.53
18:23:15 Epoch: 8	Train loss: 0.0332	Valid loss: 0.0749	Train accuracy:
99.26	Valid accuracy: 98.01
18:23:36 Epoch: 9	Train loss: 0.0300	Valid loss: 0.0753	Train accuracy:
99.27	Valid accuracy: 97.90
18:23:56 Epoch: 10	Train loss: 0.0288	Valid loss: 0.0924	Train accuracy:
99.14	Valid accuracy: 97.51
18:24:17 Epoch: 11	Train loss: 0.0264	Valid loss: 0.0974	Train accuracy:
99.08	Valid accuracy: 97.31
18:24:38 Epoch: 12	Train loss: 0.0240	Valid loss: 0.0804	Train accuracy:
99.48	Valid accuracy: 97.77
18:24:59 Epoch: 13	Train loss: 0.0237	Valid loss: 0.0808	Train accuracy:
99.43	Valid accuracy: 97.81
18:25:20 Epoch: 14	Train loss: 0.0229	Valid loss: 0.0844	Train accuracy:
99.43	Valid accuracy: 97.81
```

In [17]:
```
plot_performance(performance_2)
```





In [18]:
```
#performance
```

1.4.1

The total number of trainable parameters of LeNet is 61750.

Layer	# Filters / Neurons	Filter Size	Stride	Size of Feature Map	Acitvation Function	Bias Terms	Trainable Parameter Formula	Trainable Parameters
Input	-	-	-	32 x 32 x 1	-	-	-	-
Conv 1	6	5 * 5	1	28 x 28 x 6	tanh	1	((5 5 1) + 1) * 6	156
Avg. Pooling 1		2 * 2	2	14 x 14 x 6	-	1	6 * 2	12
Conv 2	16	5 * 5	1	10 x 10 x 16	tanh	1	((5 5 6) + 1) * 16	2416
Avg. Pooling 2		2 * 2	2	5 x 5 x 16	-	1	16 * 2	32
Conv 3	120	5 * 5	1	120	tanh	1	((5 5 16) + 1) * 120	48120
Fully Connected 1	-	-	-	84	tanh	1	(120 + 1) * 84	10164
Fully Connected 2	-	-	-	10	Softmax	1	(84 + 1) * 10	850

1.4.2

The total number of trainable parameters of MLP is 280058.

Layer	Input Size	Output Size	Acitvation Function	Bias Terms	Trainable Parameter Formula	Trainable Parameters
Input Layer	-	1024	-	-	-	-
Hidden Layer 1	1024	256	tanh	1	(1024 + 1) * 256	262400
Hidden Layer 2	256	64	tanh	1	(256 + 1) * 64	16448
Hidden Layer 3	64	16	tanh	1	(64 + 1) * 16	1040
Output Layer	16	10	tanh	1	(16 + 1) * 10	170

1.4.3

The model that has better performance in terms of prediction accuracy on the test data is LeNet and this could be because LeNet is a Convolutional Neural Network and has layers that are not fully connected comapred to Multi Layer Perceptron.

Statement of Collaboration

I, Andy Quoc Anh Dinh Tran, did this assignment by myself.