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)
```

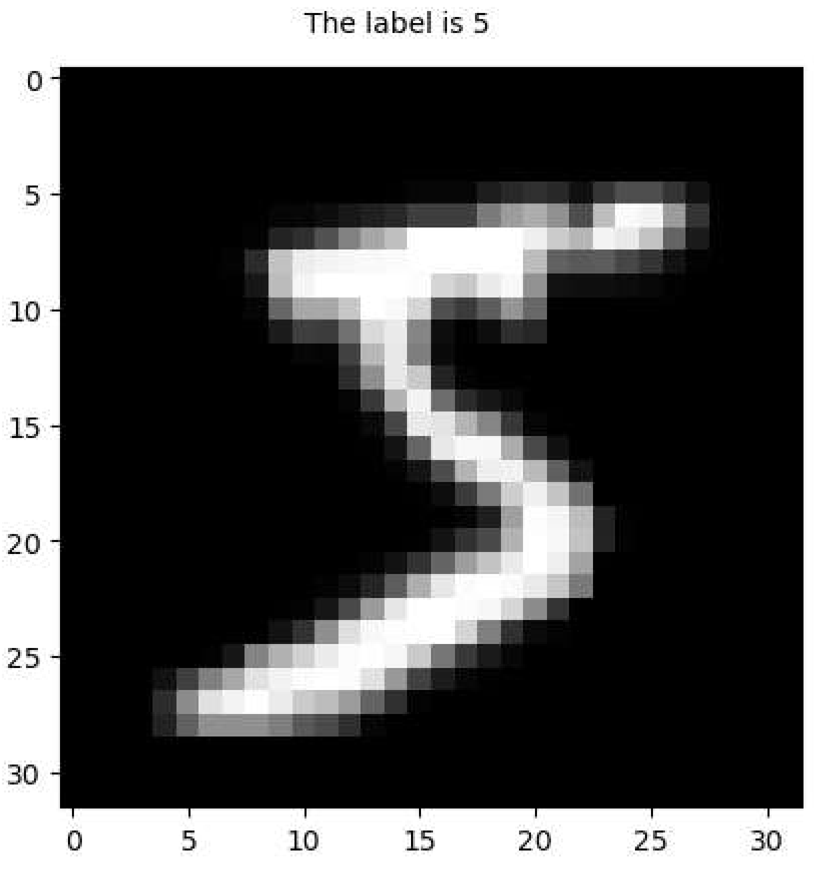
In [3]:
```
train_dataset[0][0].shape
```

Out[3]:
```
torch.Size([1, 32, 32])
```

1.1.1

In [4]:
```
plt.imshow(train_dataset[0][0].squeeze(), cmap='gray')
plt.text(10, -2, 'The label is ' + str(train_dataset[0][1]))
```

Out[4]:
```
Text(10, -2, 'The label is 5')
```



In [5]:
```
train_dataset[0][0].shape
```

Out[5]:
```
torch.Size([1, 32, 32])
```

In [6]:
```
# 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 [7]:
```
# 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 [8]:
```
def train(train_loader, model, criterion, optimizer):

    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)

        loss.backward()
        optimizer.step()


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

1.1.4

In [9]:
```
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:
        y_hat = model(X)
        loss = criterion(y_hat,y_true)
        running_loss += loss.item() * X.size(0)



    epoch_loss = running_loss / len(valid_loader.dataset)

    return model, epoch_loss
```

In [10]:
```
def training_loop(model, criterion, optimizer, train_loader, valid_loader, epochs, print_every=1):
    '''
    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 [11]:
```
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:

            predicted_labels = torch.argmax(model(X), dim=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-v0_8')

    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 [12]:
```
class LeNet5(nn.Module):

    def __init__(self, n_classes):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.activation = nn.Tanh()
        self.avg_pool = nn.AvgPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
```

```
        self.fc1 = nn.Linear(120, 84)
        self.fc2 = nn.Linear(84, n_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.activation(x)
        x = self.avg_pool(x)
        x = self.conv2(x)
        x = self.activation(x)
        x = self.avg_pool(x)
        x = self.conv3(x)
        x = self.activation(x)
        x = x.reshape(x.shape[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        probs = F.softmax(x, dim=1)
        return probs
```

1.2.2

In [13]:
```
class MLP(nn.Module):

    def __init__(self, layers):
        super(MLP, self).__init__()
        self.fc = []

        for i in range(len(layers)-1):
            self.fc.append(nn.Linear(layers[i], layers[i+1]))
            if i !=len(layers)-2:
                self.fc.append(nn.Tanh())
            self.all_layers = nn.Sequential(*self.fc)

    def forward(self, x):
        x =  x.reshape(x.shape[0], -1)
        x = self.all_layers(x)
        probs = F.softmax(x, dim=1)
        return probs
```

1.3.1

In [23]:
```
torch.manual_seed(RANDOM_SEED)

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

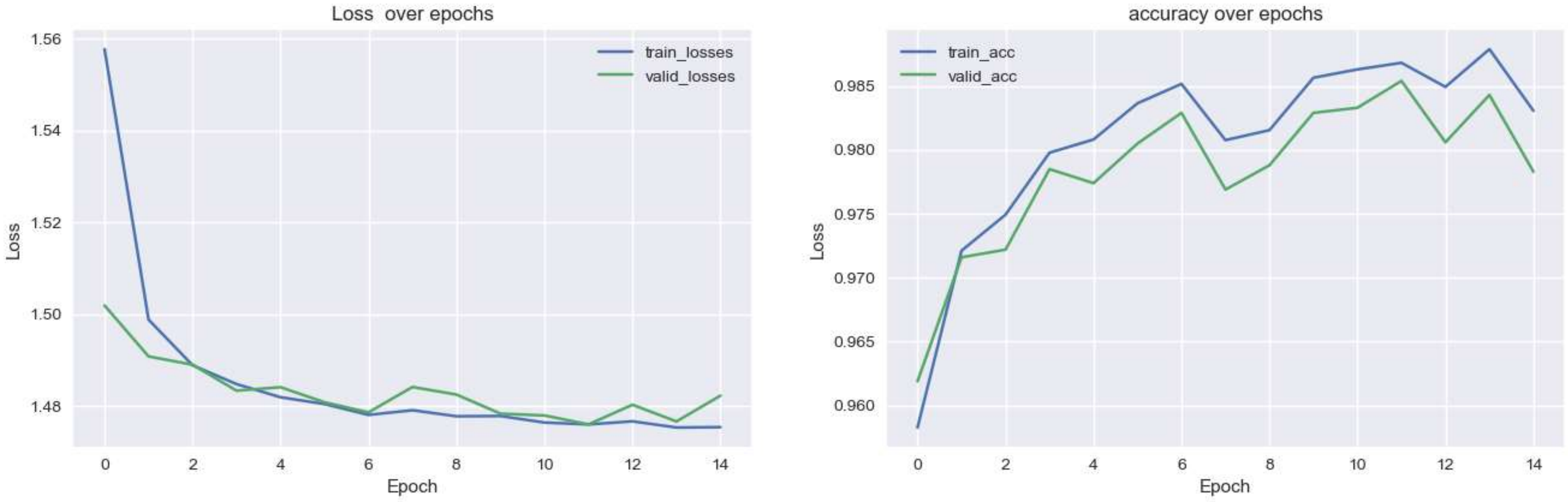
In [24]:
```
model
```

Out[24]:
```
LeNet5(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (activation): Tanh()
  (avg_pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (conv3): Conv2d(16, 120, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=120, out_features=84, bias=True)
  (fc2): Linear(in_features=84, out_features=10, bias=True)
)
```

In [25]:
```
model, optimizer, performance_1 = training_loop(model, criterion, optimizer, train_loader, valid_loader, N_EPOCHS)
```

```
23:12:59 Epoch: 0	Train loss: 1.5576	Valid loss: 1.5019	Train accuracy: 95.83	Valid accuracy: 96.19
23:13:32 Epoch: 1	Train loss: 1.4988	Valid loss: 1.4908	Train accuracy: 97.21	Valid accuracy: 97.16
23:14:07 Epoch: 2	Train loss: 1.4889	Valid loss: 1.4890	Train accuracy: 97.49	Valid accuracy: 97.22
23:14:42 Epoch: 3	Train loss: 1.4848	Valid loss: 1.4834	Train accuracy: 97.98	Valid accuracy: 97.85
23:15:18 Epoch: 4	Train loss: 1.4820	Valid loss: 1.4841	Train accuracy: 98.08	Valid accuracy: 97.74
23:15:52 Epoch: 5	Train loss: 1.4805	Valid loss: 1.4809	Train accuracy: 98.37	Valid accuracy: 98.05
23:16:26 Epoch: 6	Train loss: 1.4781	Valid loss: 1.4787	Train accuracy: 98.52	Valid accuracy: 98.29
23:17:01 Epoch: 7	Train loss: 1.4791	Valid loss: 1.4842	Train accuracy: 98.08	Valid accuracy: 97.69
23:17:35 Epoch: 8	Train loss: 1.4778	Valid loss: 1.4825	Train accuracy: 98.15	Valid accuracy: 97.88
23:18:09 Epoch: 9	Train loss: 1.4779	Valid loss: 1.4784	Train accuracy: 98.57	Valid accuracy: 98.29
23:18:43 Epoch: 10	Train loss: 1.4765	Valid loss: 1.4780	Train accuracy: 98.63	Valid accuracy: 98.33
23:19:18 Epoch: 11	Train loss: 1.4760	Valid loss: 1.4761	Train accuracy: 98.68	Valid accuracy: 98.54
23:19:52 Epoch: 12	Train loss: 1.4767	Valid loss: 1.4803	Train accuracy: 98.49	Valid accuracy: 98.06
23:20:28 Epoch: 13	Train loss: 1.4754	Valid loss: 1.4767	Train accuracy: 98.79	Valid accuracy: 98.43
23:21:09 Epoch: 14	Train loss: 1.4755	Valid loss: 1.4823	Train accuracy: 98.31	Valid accuracy: 97.83
```

In [31]:
```
plot_performance(performance_1)
```



1.3.2

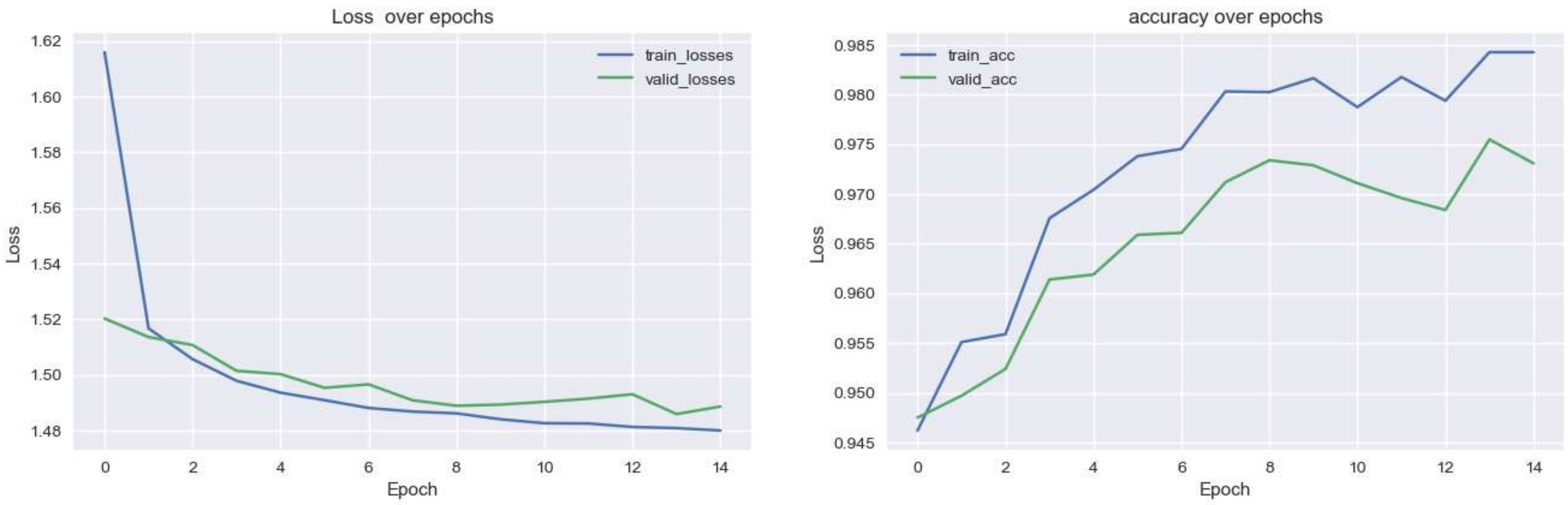
In [17]:
```
torch.manual_seed(RANDOM_SEED)
layers = [1024, 256, 64, 16, N_CLASSES]
model = MLP(layers)
print(model)
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
criterion = nn.CrossEntropyLoss()
```

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

In [18]:
```
model, optimizer, performance_2 = training_loop(model, criterion, optimizer, train_loader, valid_loader, N_EPOCHS)
```

```
22:35:08 Epoch: 0	Train loss: 1.6159	Valid loss: 1.5202	Train accuracy: 94.62	Valid accuracy: 94.75
22:35:50 Epoch: 1	Train loss: 1.5167	Valid loss: 1.5136	Train accuracy: 95.51	Valid accuracy: 94.97
22:36:28 Epoch: 2	Train loss: 1.5056	Valid loss: 1.5107	Train accuracy: 95.59	Valid accuracy: 95.24
22:36:58 Epoch: 3	Train loss: 1.4978	Valid loss: 1.5014	Train accuracy: 96.76	Valid accuracy: 96.14
22:37:33 Epoch: 4	Train loss: 1.4936	Valid loss: 1.5002	Train accuracy: 97.04	Valid accuracy: 96.19
22:38:15 Epoch: 5	Train loss: 1.4909	Valid loss: 1.4953	Train accuracy: 97.38	Valid accuracy: 96.59
22:38:46 Epoch: 6	Train loss: 1.4881	Valid loss: 1.4966	Train accuracy: 97.46	Valid accuracy: 96.61
22:39:13 Epoch: 7	Train loss: 1.4868	Valid loss: 1.4909	Train accuracy: 98.03	Valid accuracy: 97.12
22:39:39 Epoch: 8	Train loss: 1.4862	Valid loss: 1.4889	Train accuracy: 98.03	Valid accuracy: 97.34
22:40:06 Epoch: 9	Train loss: 1.4840	Valid loss: 1.4893	Train accuracy: 98.17	Valid accuracy: 97.29
22:40:32 Epoch: 10	Train loss: 1.4826	Valid loss: 1.4903	Train accuracy: 97.88	Valid accuracy: 97.11
22:40:59 Epoch: 11	Train loss: 1.4825	Valid loss: 1.4915	Train accuracy: 98.18	Valid accuracy: 96.96
22:41:26 Epoch: 12	Train loss: 1.4813	Valid loss: 1.4930	Train accuracy: 97.94	Valid accuracy: 96.84
22:41:52 Epoch: 13	Train loss: 1.4809	Valid loss: 1.4859	Train accuracy: 98.43	Valid accuracy: 97.55
22:42:20 Epoch: 14	Train loss: 1.4800	Valid loss: 1.4886	Train accuracy: 98.43	Valid accuracy: 97.31
```

In [19]:
```
plot_performance(performance_2)
```



1. What is the number of trainable parameters of LeNet?

LeNet5((conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))

-> (5×5×1+1)×6

(activation): Tanh()

(avg_pool): AvgPool2d(kernel_size=2, stride=2, padding=0)

(conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
-> (5×5×6+1)×16
(conv3): Conv2d(16, 120, kernel_size=(5, 5), stride=(1, 1))
-> (5×5×16+1)×120
(fc1): Linear(in_features=120, out_features=84, bias=True)
-> (120+1)×84
(fc2): Linear(in_features=84, out_features=10, bias=True)
-> (84+1)×10
)

In [30]:
```
(5*5*1+1)*6+(5*5*6+1)*16+(5*5*16+1)*120+(120+1)*84+(84+1)*10
```

Out[30]:
```
61706
```

2. What is the number of trainable parameters of MLP?

MLP(
(all_layers): Sequential(
(0): Linear(in_features=1024, out_features=256, bias=True)
-> (1024+1)*256*
(1): Tanh()
(2): Linear(in_features=256, out_features=64, bias=True)
-> *(256+1)*64 (3): Tanh()
(4): Linear(in_features=64, out_features=16, bias=True)
-> (64+1)*16*
(5): Tanh()
(6): Linear(in_features=16, out_features=10, bias=True)
-> *(16+1)*10
)
)

In [40]:
```
layers = [1024, 256, 64, 16, N_CLASSES]
number_of_weights = 0
for i in range(1,len(layers)):
    number_of_weights = number_of_weights+(layers[i-1]+1)*layers[i]
number_of_weights
```

Out[40]:
```
280058
```

Which model has better performance in terms of prediction accuracy on the test data? Give a reason why this model works better than the other

LeNet5 has better performance in terms of prediction accuracy on the test data.
The best performance of LeNet5:
Epoch: 13 Train loss: 1.4754 Valid loss: 1.4767 Train accuracy: 98.79 Valid accuracy: 98.43
The best performance of MLP:
Epoch: 13 Train loss: 1.4809 Valid loss: 1.4859 Train accuracy: 98.43 Valid accuracy: 97.55

Both the train accuracy and valid accuracy of LeNet are better than MLP, and the difference between Train and valid accuracy is smaller in LeNet5.

Statement of Collaboration
I did homework by myself.