Lab 3 Report:

MNIST Classification with FCN

Name:

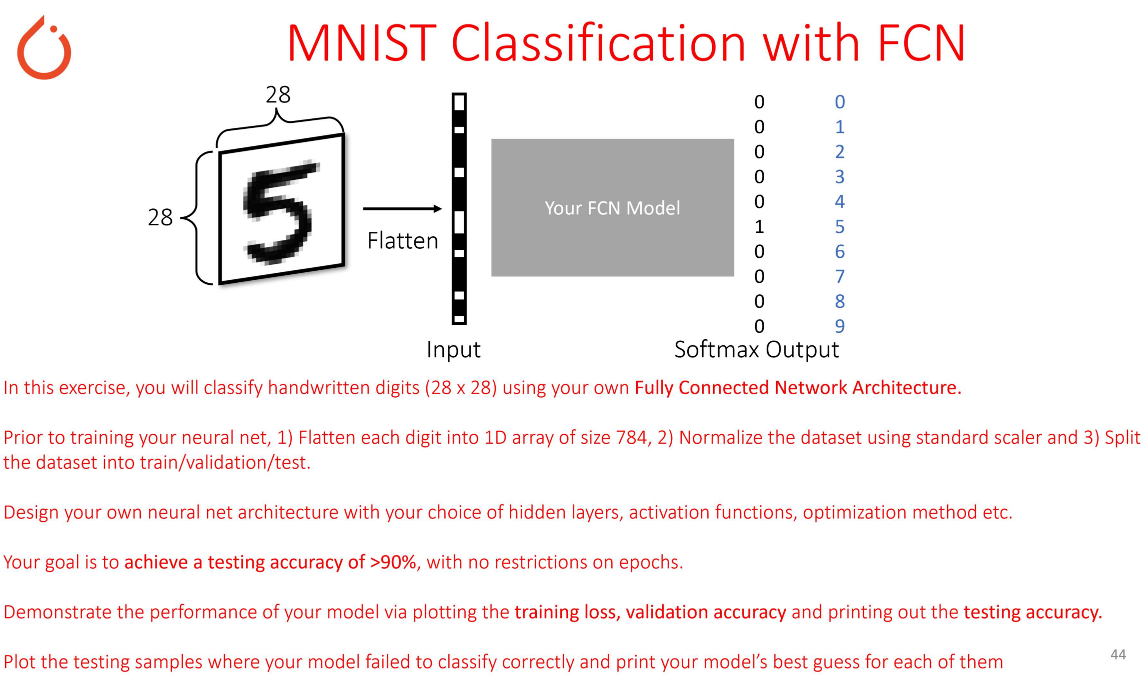
In [ ]:
```python
# Import necessary packages

%matplotlib inline

import matplotlib.pyplot as plt

import torch
import torchvision
import numpy as np
```

```
c:\Users\ebaca\anaconda3\envs\Phys417\Lib\site-packages\torchvision\io\image.py:13: UserWarning: Failed to load image Python extension: '[WinError 127] The specified procedure could not be found'If you don't plan on using image functionality from `torchvision.io`, you can ignore this warning. Otherwise, there might be something wrong with your environment. Did you have `libjpeg` or `libpng` installed before building `torchvision` from source?
  warn(
```

In [ ]:
```python
from IPython.display import Image # For displaying images in colab jupyter cell
```

In [ ]:
```python
Image('lab3_exercise.PNG', width = 1000)
```

Out[ ]:



Prepare Data

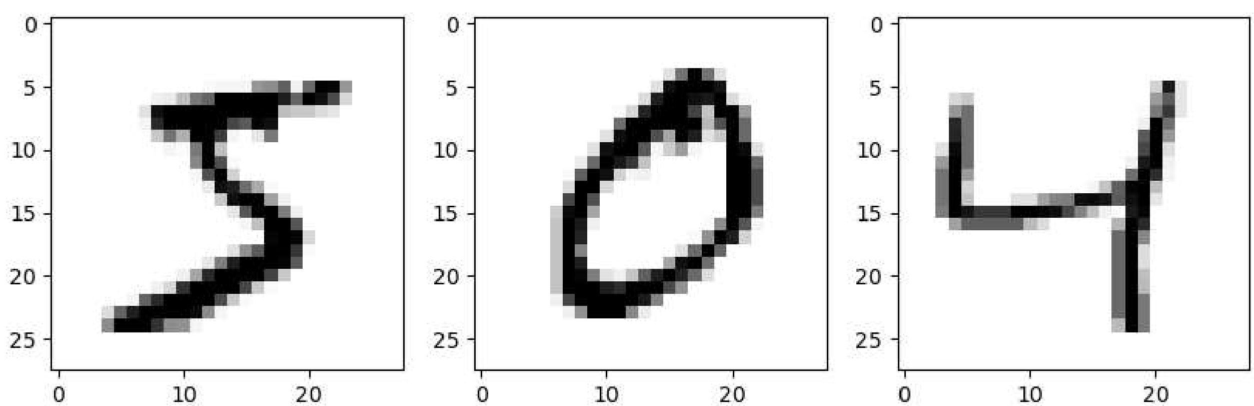
In [ ]:
```python
# Load MNIST Dataset in Numpy

# 1000 training samples where each sample feature is a greyscale image with shape (28, 28)
# 1000 training targets where each target is an integer indicating the true digit
train_feats = np.load('mnist_train_features.npy')
train_targs = np.load('mnist_train_targets.npy')

# 100 testing samples + targets
```

```
test_feats = np.load('mnist_test_features.npy')
test_targs = np.load('mnist_test_targets.npy')

# Print the dimensions of training sample features/targets
print(train_feats.shape, train_targs.shape)
# Print the dimensions of testing sample features/targets
print(test_feats.shape, test_targs.shape)
```

```
(1000, 28, 28) (1000,)
(100, 28, 28) (100,)
```

In [ ]:
```
# Let's visualize some training samples

plt.figure(figsize = (10, 10))

plt.subplot(1,3,1)
plt.imshow(train_feats[0], cmap = 'Greys')

plt.subplot(1,3,2)
plt.imshow(train_feats[1], cmap = 'Greys')

plt.subplot(1,3,3)
plt.imshow(train_feats[2], cmap = 'Greys')
```

Out[ ]: <matplotlib.image.AxesImage at 0x26101af3d10>



In [ ]:
```
train_feats.shape[0], train_feats.shape[1], train_feats.shape[2]
```

Out[ ]: (1000, 28, 28)

In [ ]:
```
# Reshape features via flattening the images
# This refers to reshape each sample from a 2d array to a 1d array (i.e. each sample should be 1x784)
# hint: np.reshape() function could be useful here

train_feats = train_feats.reshape((train_feats.shape[0], train_feats.shape[1]*train_feats.shape[2]))
test_feats = test_feats.reshape((test_feats.shape[0], test_feats.shape[1]*test_feats.shape[2]))

print(train_feats.shape, test_feats.shape)
```

```
(1000, 784) (100, 784)
```

In [ ]:
```
# Scale the dataset according to standard scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

train_feats = scaler.fit_transform(train_feats)
test_feats = scaler.fit_transform(test_feats)
```

In [ ]:
```
# Split training dataset into Train (90%), Validation (10%)
# features = x, targets = y in lab 3 example

validate_feats = train_feats[:int(len(test_feats))]
validate_targs = train_targs[:int(len(test_feats))]
```

```
train_feats = train_feats[int(len(test_feats)):]
train_targs = train_targs[int(len(test_feats)):]
```

Define Model

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
```

In [ ]:
```
# torch.nn.softmax() gives probabilities of outputs

class mnistClassification(torch.nn.Module):

    def __init__(self, input_dim, output_dim, hidden1_dim, hidden2_dim): # Feel free to add hidden_dim as para
        super(mnistClassification, self).__init__()

        # Connecting each layer:
        self.layer1 = torch.nn.Linear(input_dim, hidden1_dim) # input to 1st hidden layer
        self.layer2 = torch.nn.Linear(hidden1_dim, hidden2_dim) # 1st hidden layer to 2nd hidden layer
        self.layer3 = torch.nn.Linear(hidden2_dim, output_dim) # 2nd hidden layer to final output
        self.dropout = torch.nn.Dropout(p = 0.25)

    def forward(self, x):

        # Applying activation function
        x = torch.nn.functional.relu(self.layer1(x))          # activating layer 1 output
        x = self.dropout(x)
        x = torch.nn.functional.relu(self.layer2(x))       # activating layer 2 output
        output = self.layer3(x)

        return output
```

Define Hyperparameters

In [ ]:
```
# Initialize our neural network model with input and output dimensions
model = mnistClassification(input_dim = 784,
                            output_dim = 10,
                            hidden1_dim = 190,
                            hidden2_dim = 190)

# Define the learning rate and epoch
learning_rate = 0.00011
epochs = 1200

# accuracies will change between executions due to drop out regularization
# h1 & h2 = 20, lr = 0.00011, eps = 1200 --> 74% accuracy
# h1 & h2 = 20, lr = 0.0011, eps = 1200 --> 81% accuracy
# h1 & h2 = 30, lr = 0.00011, eps = 1200 --> 83% accuracy
# h1 & h2 = 60, lr = 0.00011, eps = 1200 --> 83% accuracy
# h1 & h2 = 80, lr = 0.00011, eps = 1200 --> 86% accuracy
# h1 & h2 = 190, lr = 0.000011, eps = 1200 --> 87% accuracy
# h1 & h2 = 190, lr = 0.00011, eps = 1200 --> 87% accuracy
# h1 & h2 = 100, lr = 0.00011, eps = 1200 --> 87% accuracy

# Define loss function and optimizer
loss_func = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)

# Run this line if you have PyTorch GPU version
# if torch.cuda.is_available():
#     model.cuda()

model
```

Out[ ]:
```
mnistClassification(
  (layer1): Linear(in_features=784, out_features=190, bias=True)
  (layer2): Linear(in_features=190, out_features=190, bias=True)
  (layer3): Linear(in_features=190, out_features=10, bias=True)
  (dropout): Dropout(p=0.25, inplace=False)
)
```

Identify Tracked Values

In [ ]:
```python
# Placeholders for training loss and validation accuracy during training
# Training loss should be tracked for each iteration (1 iteration -> single forward pass to the network)
# Validation accuracy should be evaluated every 'Epoch' (1 epoch -> full training dataset)
# If using batch gradient, 1 iteration = 1 epoch

losses = np.zeros((epochs,))
validation_accs = np.zeros((epochs,))
```

Train Model

In [ ]:
```python
import tqdm

# Convert the training, validation, testing dataset (NumPy arrays) into torch tensors

inputs = torch.from_numpy(train_feats).float()
train_targets_tensor = torch.from_numpy(train_targs) # Convert to 64-bit integer
train_targets_one_hot = torch.nn.functional.one_hot(train_targets_tensor) # convert to one hot labels

validation_inputs = torch.from_numpy(validate_feats).float()
validation_targets = torch.from_numpy(validate_targs).long()

testing_inputs = torch.from_numpy(test_feats).float()
testing_targets = torch.from_numpy(test_targs).long()

# Training Loop ---------------------------------------------------------------------------------

for epoch in tqdm.trange(epochs):

    optimizer.zero_grad()
    outputs = model(inputs)
    loss = loss_func(outputs, train_targets_one_hot.float())
    losses[epoch] = loss.item()
    loss.backward()
    optimizer.step()

    # Compute Validation Accuracy ------------------------------------------------------------------

    with torch.no_grad():

        # Pass the validation feature data to the network
        validation_outputs = model(validation_inputs)

        # converting labels back to regular digits
        correct = (torch.argmax(validation_outputs, dim=1) ==
                   validation_targets).type(torch.FloatTensor)

        validation_accs[epoch] = correct.mean()
```

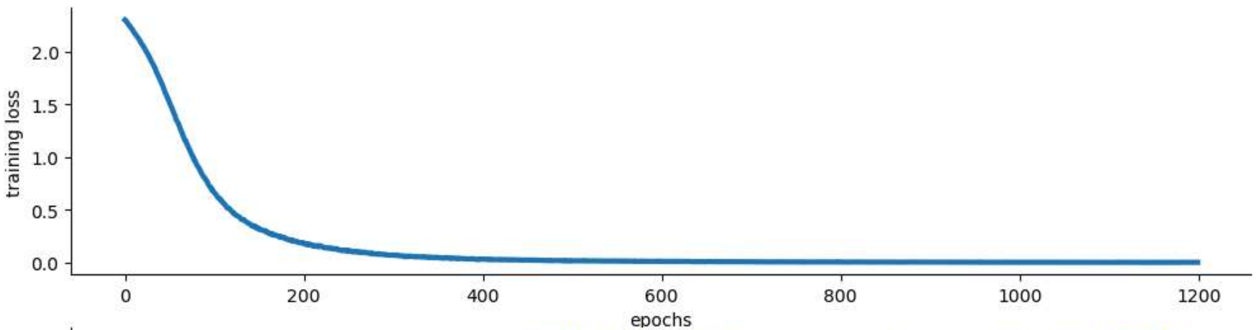
```
  0%|          | 0/1200 [00:00<?, ?it/s]100%|██████████| 1200/1200 [00:23<00:00, 51.47it/s]
```

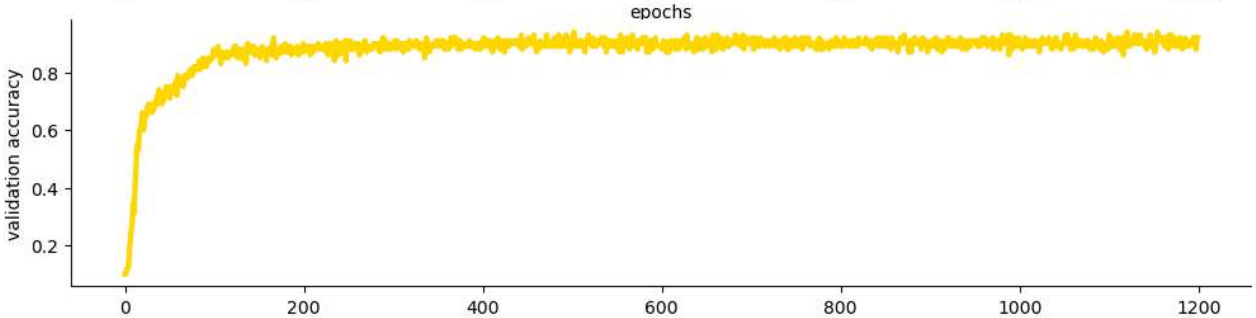
Visualize and Evaluate Model

In [ ]:
```python
# Import seaborn for prettier plots

import seaborn as sns
```

In [ ]:
```python
# Visualize training loss
```

```
plt.figure(figsize = (12, 6))

# Visualize training loss with respect to iterations (1 iteration -> single batch)
plt.subplot(2, 1, 1)
plt.plot(losses, linewidth = 3)
plt.ylabel("training loss")
plt.xlabel("epochs")
sns.despine()

# Visualize validation accuracy with respect to epochs
plt.subplot(2, 1, 2)
plt.plot(validation_accs, linewidth = 3, color = 'gold')
plt.ylabel("validation accuracy")
sns.despine()
```





In [ ]:
```
# Compute the testing accuracy

with torch.no_grad():

    # Pass the testing feature data (30 samples) to the network to produce model predictions
    y_pred_test = model(testing_inputs)

    # Use the same technique as above to commpute the testing classification accuracy
    correct = (torch.argmax(y_pred_test, dim=1) == testing_targets).type(torch.FloatTensor)

    print("Testing Accuracy: " + str(correct.mean().numpy()*100) + '%')
```

```
Testing Accuracy: 91.00000262260437%
```

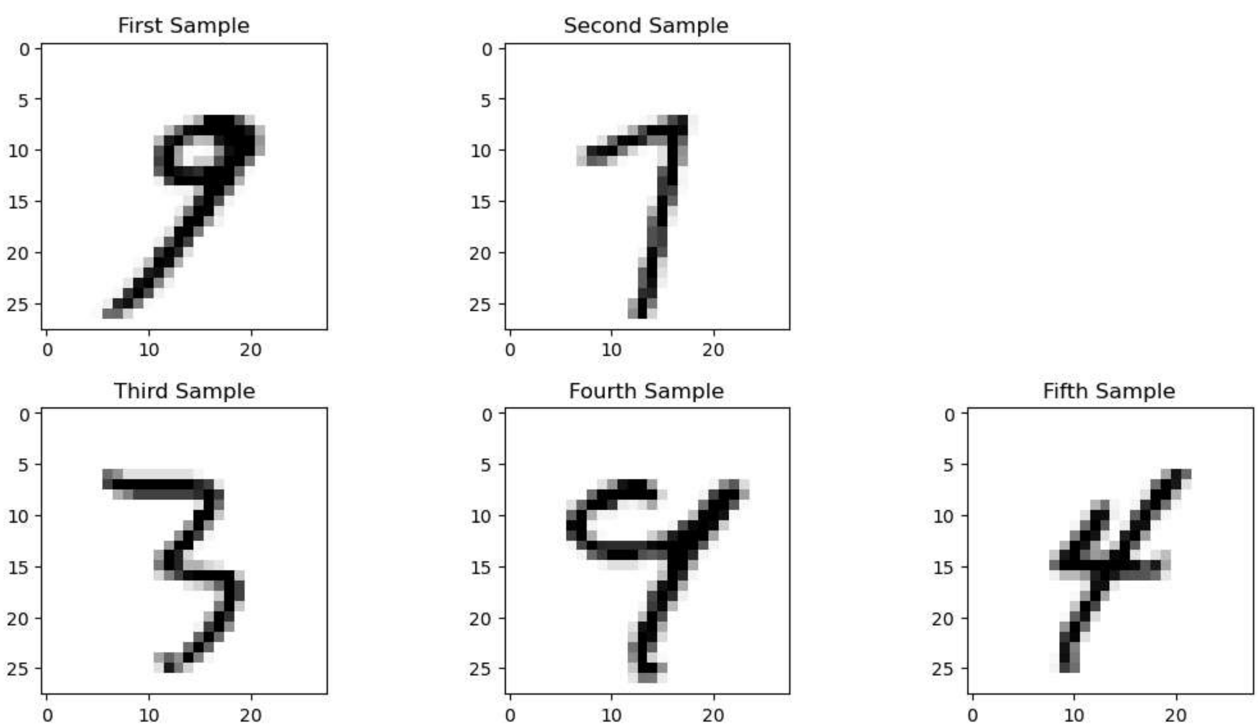
In [ ]:
```
# Plot 5 incorrectly classified testing samples and print the model predictions for each of them
# You can use np.reshape() to convert flattened 1D array back to 2D array
#np.load('mnist_train_features.npy')

train_feats_incorrect = np.load('mnist_train_features.npy')
print(train_feats_incorrect.shape)

counter = 0
incorrect_index = []
inc_predictions = []
for n in torch.argmax(y_pred_test, dim=1):
    if n != testing_targets[counter]:
        incorrect_index.append(counter)
        inc_predictions.append(n.item())
    counter = counter + 1
print(incorrect_index, inc_predictions)
```

```
(1000, 28, 28)
[33, 42, 44, 61, 65, 66, 73, 80, 87] [5, 9, 5, 2, 9, 2, 7, 9, 5]
```

In [ ]:

```python
plt.figure(figsize = (10, 10))

plt.subplot(2,3,1)
plt.title("First Sample")
plt.imshow(train_feats_incorrect[33], cmap = 'Greys')
print(f"First incorrect: {inc_predictions[0]}")

plt.subplot(2,3,2)
plt.title("Second Sample")
plt.imshow(train_feats_incorrect[42], cmap = 'Greys')
print(f"Second incorrect: {inc_predictions[1]}")

plt.subplot(1,3,1)
plt.title("Third Sample")
plt.imshow(train_feats_incorrect[44], cmap = 'Greys')
print(f"Third incorrect: {inc_predictions[2]}")

plt.subplot(1,3,2)
plt.title("Fourth Sample")
plt.imshow(train_feats_incorrect[54], cmap = 'Greys')
print(f"Fourth incorrect: {inc_predictions[3]}")

plt.subplot(1,3,3)
plt.title("Fifth Sample")
plt.imshow(train_feats_incorrect[61], cmap = 'Greys')
print(f"Fifth incorrect: {inc_predictions[4]}")


plt.tight_layout(pad=0, h_pad=10, w_pad=6)
```

```
First incorrect: 5
Second incorrect: 9
Third incorrect: 5
Fourth incorrect: 2
Fifth incorrect: 9
```



In [ ]: