Watson Studio democratizes machine learning and deep learning to accelerate infusion of AI in your business to drive innovation. Watson Studio provides a suite of tools and a collaborative environment for data scientists, developers and domain experts.

(http://cocl.us/pytorch_link_top)

In [1]:

```
#by Christopher Harrison
```

Objective

- How to classify handwritten digits using Neural Network.

Table of Contents

In this lab, you will use a single layer neural network to classify handwritten digits from the MNIST database.

Estimated Time Needed: **25 min**

Preparation

We'll need the following libraries

In [2]:

```
# Import the libraries we need for this lab

# Using the following line code to install the torchvision library
# !conda install -y torchvision

import torch
import torch.nn as nn
import torchvision.transforms as transforms
import torchvision.datasets as dsets
import torch.nn.functional as F
import matplotlib.pylab as plt
import numpy as np
```

Use the following helper functions for plotting the loss:

In [3]:

```
# Define a function to plot accuracy and loss

def plot_accuracy_loss(training_results):
    plt.subplot(2, 1, 1)
    plt.plot(training_results['training_loss'], 'r')
    plt.ylabel('loss')
    plt.title('training loss iterations')
    plt.subplot(2, 1, 2)
    plt.plot(training_results['validation_accuracy'])
    plt.ylabel('accuracy')
    plt.xlabel('epochs')
    plt.show()
```

Use the following function for printing the model parameters:

In [4]:

```
# Define a function to plot model parameters

def print_model_parameters(model):
    count = 0
    for ele in model.state_dict():
        count += 1
        if count % 2 != 0:
            print ("The following are the parameters for the layer ", count // 2 +
1)
        if ele.find("bias") != -1:
            print("The size of bias: ", model.state_dict()[ele].size())
        else:
            print("The size of weights: ", model.state_dict()[ele].size())
```

Define the neural network module or class:

In [5]:

```
# Define a function to display data

def show_data(data_sample):
    plt.imshow(data_sample.numpy().reshape(28, 28), cmap='gray')
    plt.show()
```

Neural Network Module and Training Function

Define the neural network module or class:

In [6]:

```
# Define a Neural Network class

class Net(nn.Module):

    # Constructor
    def __init__(self, D_in, H, D_out):
        super(Net, self).__init__()
        self.linear1 = nn.Linear(D_in, H)
        self.linear2 = nn.Linear(H, D_out)

    # Prediction
    def forward(self, x):
        x = torch.sigmoid(self.linear1(x))
        x = self.linear2(x)
        return x
```

Define a function to train the model. In this case, the function returns a Python dictionary to store the training loss and accuracy on the validation data.

In [7]:

```
# Define a training function to train the model

def train(model, criterion, train_loader, validation_loader, optimizer, epochs=100
):
    i = 0
    useful_stuff = {'training_loss': [],'validation_accuracy': []}
    for epoch in range(epochs):
        for i, (x, y) in enumerate(train_loader):
            optimizer.zero_grad()
            z = model(x.view(-1, 28 * 28))
            loss = criterion(z, y)
            loss.backward()
            optimizer.step()
             #loss for every iteration
            useful_stuff['training_loss'].append(loss.data.item())
        correct = 0
        for x, y in validation_loader:
            #validation
            z = model(x.view(-1, 28 * 28))
            _, label = torch.max(z, 1)
            correct += (label == y).sum().item()
        accuracy = 100 * (correct / len(validation_dataset))
        useful_stuff['validation_accuracy'].append(accuracy)
    return useful_stuff
```

Make Some Data

Load the training dataset by setting the parameters `train` to `True` and convert it to a tensor by placing a transform object in the argument `transform`.

In [8]:

```
# Create training dataset

train_dataset = dsets.MNIST(root='./data', train=True, download=True, transform=tra
nsforms.ToTensor())
```

Load the testing dataset by setting the parameters `train` to `False` and convert it to a tensor by placing a transform object in the argument `transform`:

In [9]:

```
# Create validating dataset

validation_dataset = dsets.MNIST(root='./data', train=False, download=True, transfo
rm=transforms.ToTensor())
```

Create the criterion function:

In [10]:

```
# Create criterion function

criterion = nn.CrossEntropyLoss()
```

Create the training-data loader and the validation-data loader objects:

In [11]:

```
# Create data loader for both train dataset and valdiate dataset

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=2000,
shuffle=True)
validation_loader = torch.utils.data.DataLoader(dataset=validation_dataset, batch_s
ize=5000, shuffle=False)
```

Define the Neural Network, Optimizer, and Train the Model

Create the model with 100 neurons:

In [12]:

```
# Create the model with 100 neurons

input_dim = 28 * 28
hidden_dim = 100
output_dim = 10

model = Net(input_dim, hidden_dim, output_dim)
```

Print the model parameters:

In [13]:

```
# Print the parameters for model

print_model_parameters(model)
```

```
The following are the parameters for the layer  1
The size of weights:  torch.Size([100, 784])
The size of bias:  torch.Size([100])
The following are the parameters for the layer  2
The size of weights:  torch.Size([10, 100])
The size of bias:  torch.Size([10])
```

Define the optimizer object with a learning rate of 0.01:

In [14]:

```
# Set the learning rate and the optimizer

learning_rate = 0.01
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

Train the model by using 100 epochs **(this process takes time)**:

In [15]:

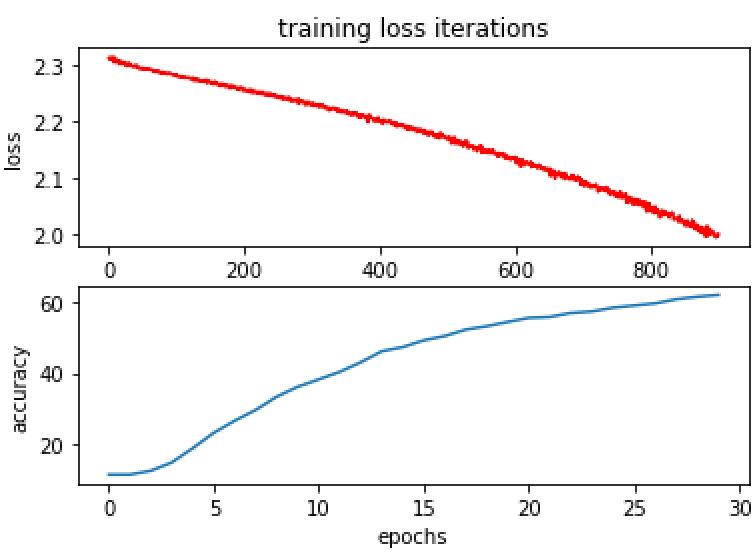
```
# Train the model

training_results = train(model, criterion, train_loader, validation_loader, optimiz
er, epochs=30)
```

Analyze Results

Plot the training total loss or cost for every iteration and plot the training accuracy for every epoch:

In [16]:

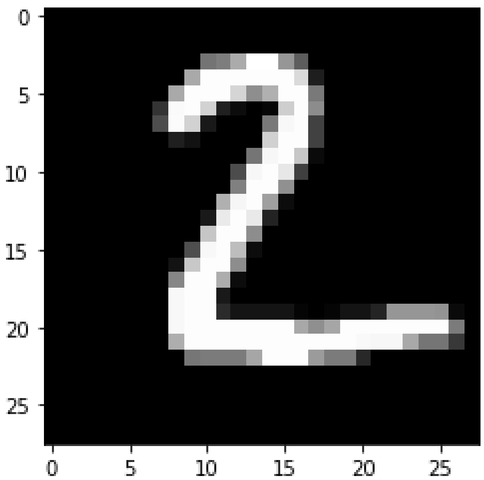
```
# Plot the accuracy and loss

plot_accuracy_loss(training_results)
```

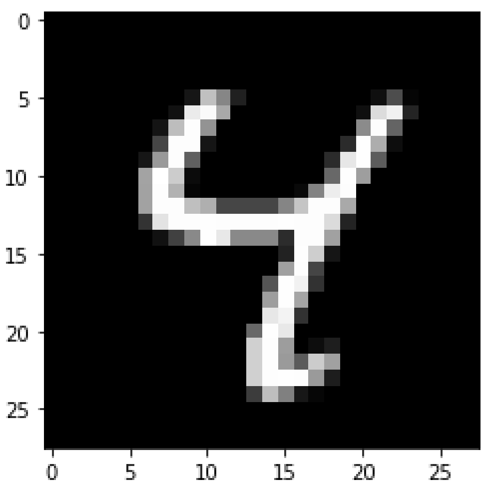


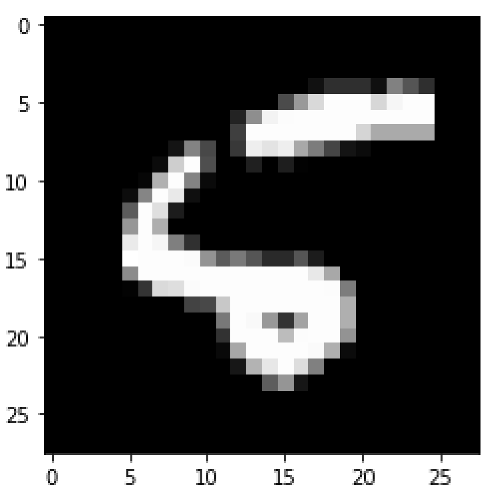
Plot the first five misclassified samples:

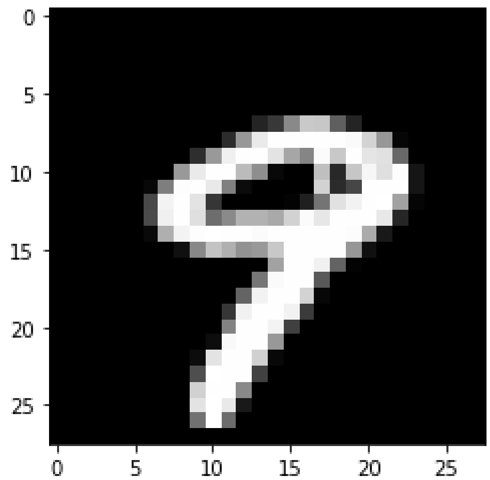
In [17]:

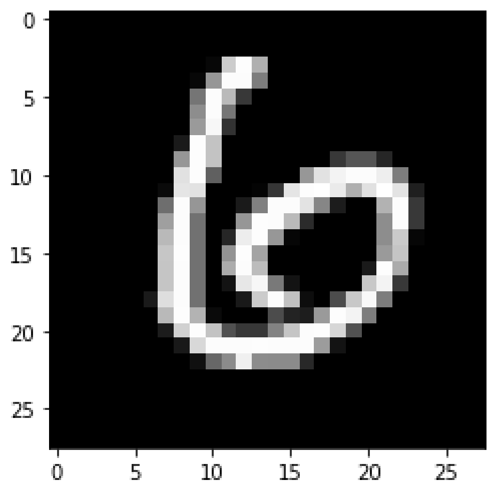
```
# Plot the first five misclassified samples

count = 0
for x, y in validation_dataset:
    z = model(x.reshape(-1, 28 * 28))
    _,yhat = torch.max(z, 1)
    if yhat != y:
        show_data(x)
        count += 1
    if count >= 5:
        break
```


0
5
10
15
20
25
0 5 10 15 20 25


0
5
10
15
20
25
0 5 10 15 20 25


0
5
10
15
20
25
0 5 10 15 20 25



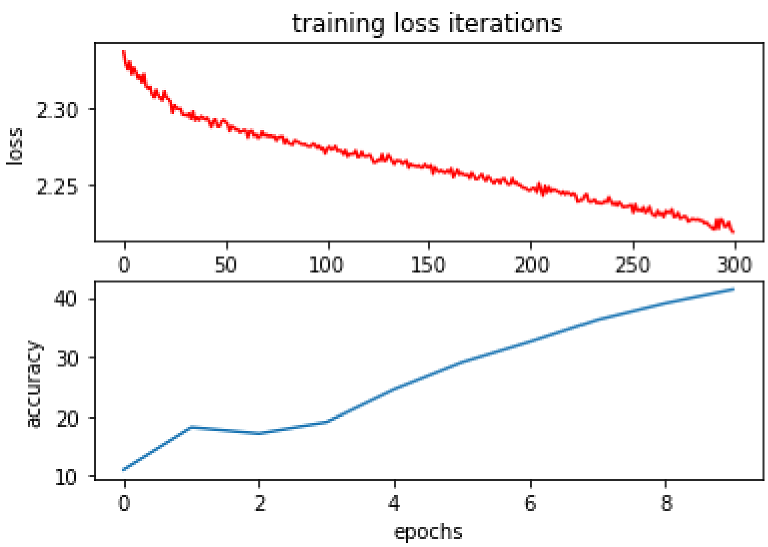


Practice

Use `nn.Sequential` to build exactly the same model as you just built. Use the function train to train the model and use the function `plot_accuracy_loss` to see the metrics. Also, try different epoch numbers.

In [18]:

```
# Practice: Use nn.Sequential to build the same model. Use plot_accuracy_loss to pr
int out the accuarcy and loss

# Type your code here
input_dim = 28 * 28
hidden_dim = 100
output_dim = 10

model = torch.nn.Sequential(
    torch.nn.Linear(input_dim, hidden_dim),
    torch.nn.Sigmoid(),
    torch.nn.Linear(hidden_dim, output_dim),
)
learning_rate = 0.01
optimizer = torch.optim.SGD(model.parameters(), lr = learning_rate)
training_results = train(model, criterion, train_loader, validation_loader, optimiz
er, epochs = 10)
plot_accuracy_loss(training_results)
```



About the Authors:

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/) has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Other contributors: Michelle Carey (https://www.linkedin.com/in/michelleccarey/), Mavis Zhou (www.linkedin.com/in/jiahui-mavis-zhou-a4537814a)

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-23	2.0	Shubham	Migrated Lab to Markdown and added to course repo in GitLab