Chelsea Jaculina
DATA 255
Assignment 1 - Building a Feed Forward Network using Pytorch
September 15, 2025

This report provides an overview of the implementation and results for DATA 255 assignment #1, which involved utilizing PyTorch to create a basic feedforward neural network from scratch. Using a subset of the MNIST dataset, the assignment was based on a binary classification task. The focus was categorizing grayscale pictures of the handwritten numbers 5 and 6, which were associated with the labels 0 and 1, respectively. Each image was provided in CSV format as a row of 784 pixel values (28x28 resolution), with the label shown in the first column. A total of 11,339 training examples from the mnist_train_binary.csv dataset and 1,850 test examples from the mnist_test_binary.csv dataset were used.

Part 1. Display the Sample Images

The first task involved loading and visualizing samples from the MNIST dataset provided in CSV format. I used pandas to read the data and matplotlib.pyplot for rendering the grayscale images. There were 785 values in each row of the CSV, with the first value being a label of either 0 or 1. The next 784 values were pixel values. A grayscale colormap displayed a 3x3 grid of randomly selected samples after each image was resized to a 28x28 array. To verify that the dataset was loaded and reshaped correctly, as shown in Figure 1, this step was crucial.

1. Display the Sample Images

Load MNIST CSV files

[175]
0s

```python
def load_mnist_csv(filename, num_samples=None):
    data = [] # list to store label and pixels as tuples
    with open(filename, 'r') as f: # open csv file on read
        reader = csv.reader(f)
        next(reader)  # skip header
        for i, row in enumerate(reader):
            if num_samples and i >= num_samples: # if a sample limit is provide, stop loop through each row in file
                break
            row = list(map(int, row)) # convert all string values to integers in row
            label = row[0] # first row is the label (0 or 1)
            pixels = row[1:] # remaining 784 values are the pixel values for image
            data.append((label, pixels)) # store the label and pixel as a tuple in list
    return data
```

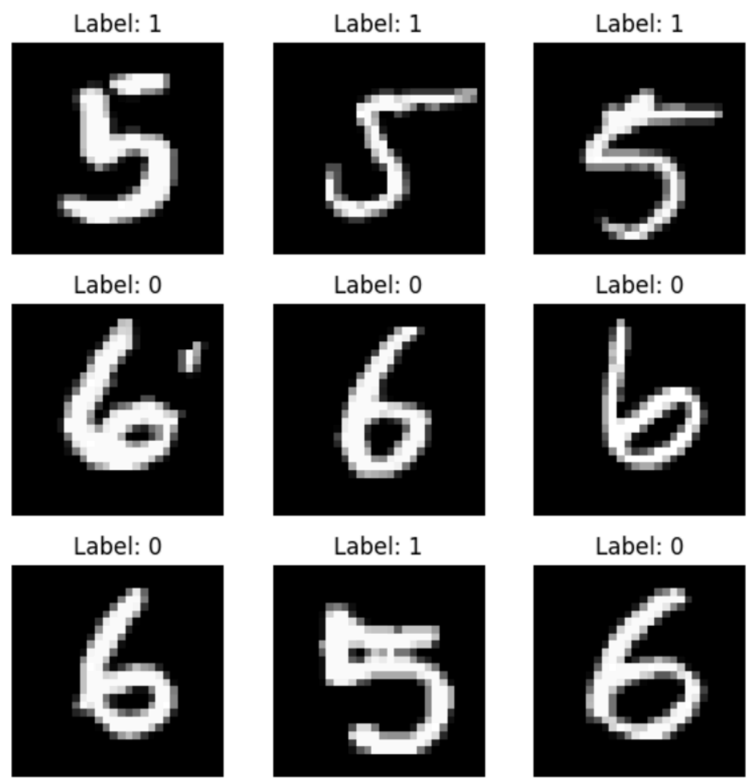
[176]
1s

```python
# load training data from csv file
train_samples = load_mnist_csv("/content/drive/MyDrive/MSDA 2024-2026/04 Fall 2025/DATA 255 - Deep Learning/mnist_train_binary.csv")

# load test data from csv file
test_samples  = load_mnist_csv("/content/drive/MyDrive/MSDA 2024-2026/04 Fall 2025/DATA 255 - Deep Learning/mnist_test_binary.csv")

print(f"Loaded {len(train_samples)} training samples successfully")
print(f"Loaded {len(test_samples)} test samples successfully")

```

```
Loaded 11339 training samples successfully
Loaded 1850 test samples successfully
```

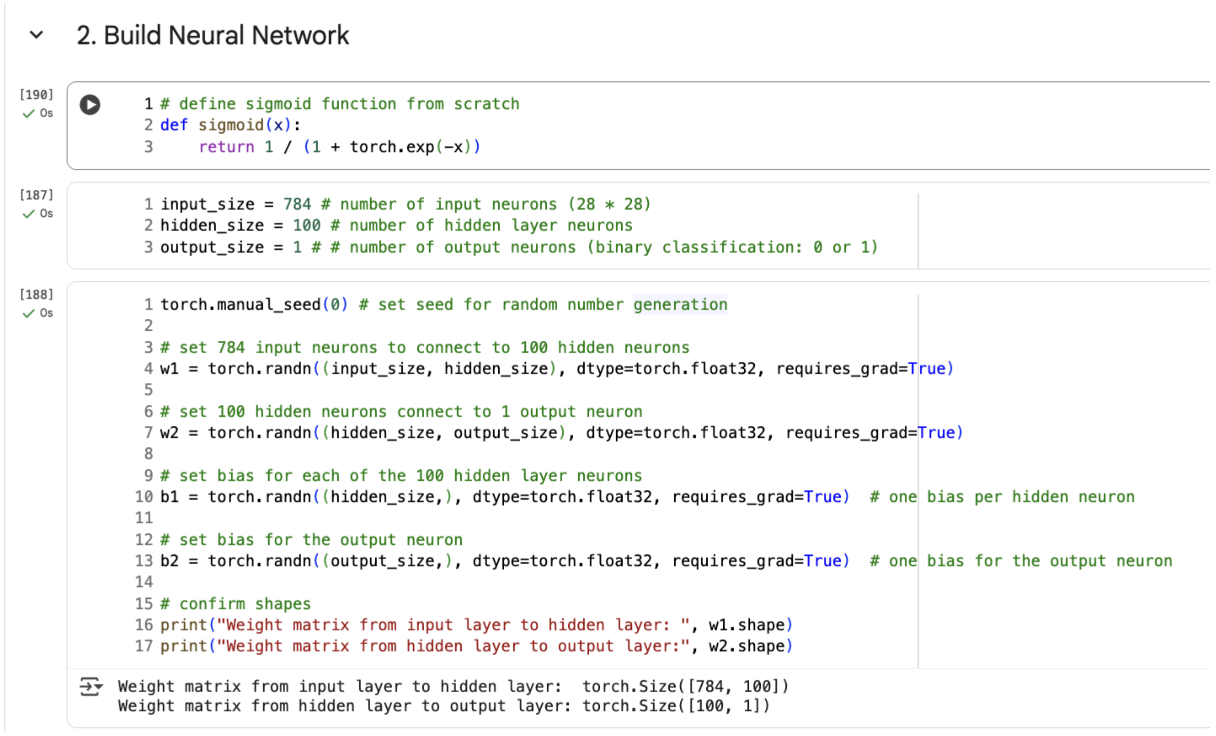
Load the dataset from the csv files and display the sample images as shown in Fig.1.

```python
plt.figure(figsize=(6, 6))
for i, (label, pixels) in enumerate(train_samples[:9]):
    image = torch.tensor(pixels, dtype=torch.float32).reshape(28, 28)
    plt.subplot(3, 3, i + 1)
    plt.imshow(image, cmap="gray")
    plt.title(f"Label: {label}")
    plt.axis("off")
plt.tight_layout()
plt.show()
```



Part 2. Build Neural Network

The primary part of the assignment involved constructing a neural network utilizing PyTorch, without relying on any external libraries like NumPy or TensorFlow. The design complies with the specifications closely: an input layer consisting of 784 neurons, a hidden layer with 100 neurons, and one output neuron. I set the weight matrices W1 and W2 by hand with small random values to disrupt symmetry and applied torch.manual_seed(0) for consistent results. The sigmoid activation function was employed in both the hidden and output layers, which I personally implemented through the formula sigma(x) ={1}/{1 + exp(-x)}. The forward propagation involved multiplying the input by W1 with a bias, applying the sigmoid activation, then passing the output through W2 with a bias followed by another sigmoid activation to achieve the final result.



2. Build Neural Network

[190] 0s

```
# define sigmoid function from scratch
def sigmoid(x):
    return 1 / (1 + torch.exp(-x))
```

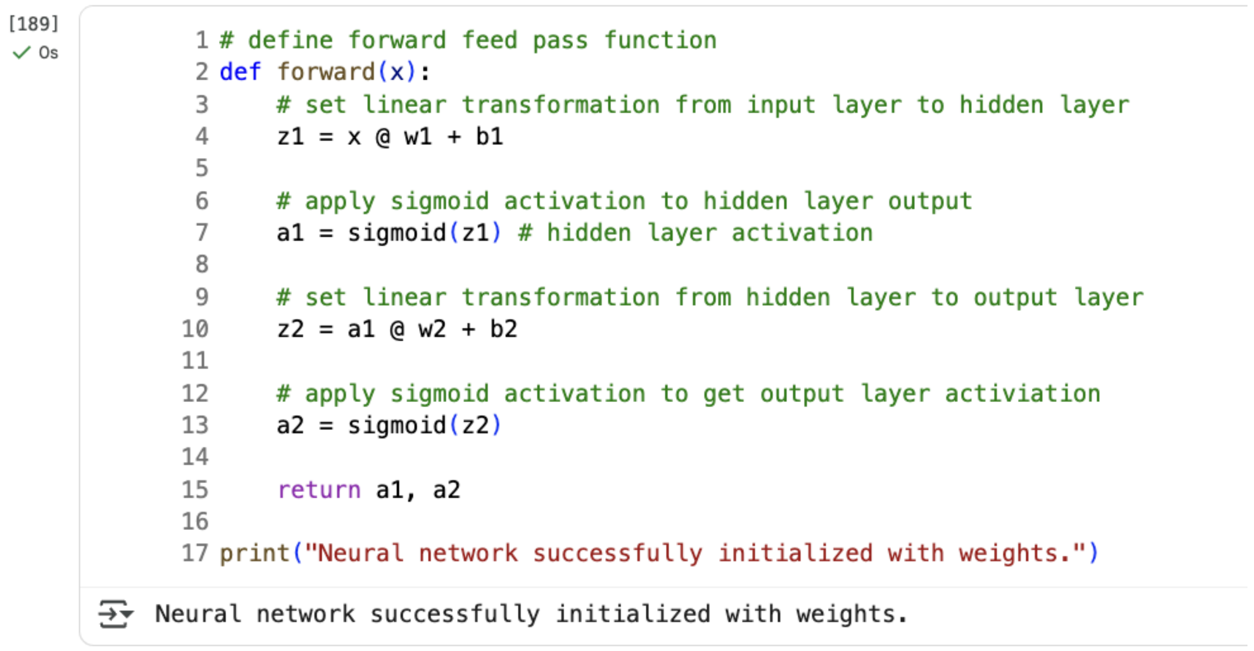
[187] 0s

```
input_size = 784 # number of input neurons (28 * 28)
hidden_size = 100 # number of hidden layer neurons
output_size = 1 # # number of output neurons (binary classification: 0 or 1)
```

[188] 0s

```
torch.manual_seed(0) # set seed for random number generation

# set 784 input neurons to connect to 100 hidden neurons
w1 = torch.randn((input_size, hidden_size), dtype=torch.float32, requires_grad=True)

# set 100 hidden neurons connect to 1 output neuron
w2 = torch.randn((hidden_size, output_size), dtype=torch.float32, requires_grad=True)

# set bias for each of the 100 hidden layer neurons
b1 = torch.randn((hidden_size,), dtype=torch.float32, requires_grad=True)  # one bias per hidden neuron

# set bias for the output neuron
b2 = torch.randn((output_size,), dtype=torch.float32, requires_grad=True)  # one bias for the output neuron

# confirm shapes
print("Weight matrix from input layer to hidden layer: ", w1.shape)
print("Weight matrix from hidden layer to output layer:", w2.shape)
```

```
Weight matrix from input layer to hidden layer:  torch.Size([784, 100])
Weight matrix from hidden layer to output layer: torch.Size([100, 1])
```



```
[189]
✓ 0s
```

```python
# define forward feed pass function
def forward(x):
    # set linear transformation from input layer to hidden layer
    z1 = x @ w1 + b1

    # apply sigmoid activation to hidden layer output
    a1 = sigmoid(z1) # hidden layer activation

    # set linear transformation from hidden layer to output layer
    z2 = a1 @ w2 + b2

    # apply sigmoid activation to get output layer activiation
    a2 = sigmoid(z2)

    return a1, a2

print("Neural network successfully initialized with weights.")
```

```
Neural network successfully initialized with weights.
```

Part 3. Calculate $a^{[2]}$

In part 3, I calculated the output layer activations $a^{[2]}$ for the initial 64 samples from the training dataset. Before the forward pass, pixel values were scaled from the range [0, 255] to [0, 1] to enhance numerical stability and replicate standard preprocessing methods. The normalized samples were subsequently processed by the manually constructed network utilizing the forward propagation method. The result was a tensor of dimensions (64 × 1) featuring activation values ranging from 0 to 1, indicating the likelihood that a particular sample is associated with class 1 (digit 6). The printed output values were nearly 0.5, which is the expected outcome for a randomly initialized model before training. This verified that the model structure and forward propagation process were operating properly, despite the absence of learning or optimization.

3. Calculate a^[2]

[182]
0s

```python
batch_pixels = [] # list for pixels
batch_labels = [] # list for labels

# use the first 64 samples from training data set
for label, pixels in train_samples[:64]:
    # convert the list of 784 pixel values and normalize them to [0, 1] to train faster and for more stability
    batch_pixels.append(torch.tensor(pixels, dtype=torch.float32) / 255.0)

    # convert the label (0 or 1) to a shape of 1
    batch_labels.append(torch.tensor([label], dtype=torch.float32))

x = torch.stack(batch_pixels)
y = torch.stack(batch_labels)

print(f"x shape: {x.shape}")
print(f"y shape: {y.shape}")
```

```
x shape: torch.Size([64, 784])
y shape: torch.Size([64, 1])
```

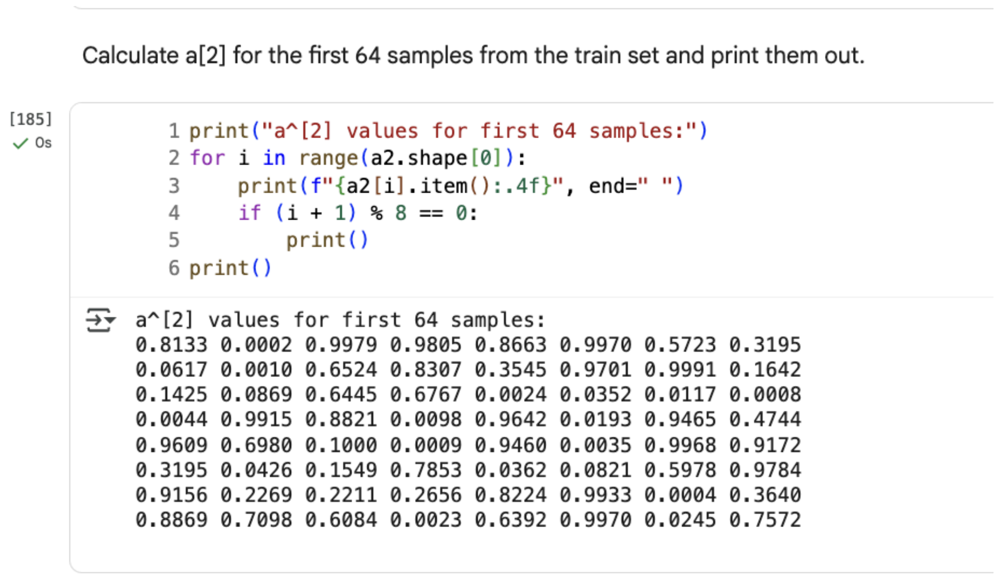
[183]
0s

```python
# perform a forward pass to compute activation for hidden layer (a1) and output layer (a2)
a1, a2 = forward(x)
```

[184]
0s

```python
# x: 64 images, each flattened to 784 pixel values (28x28)
print(f"Input batch x shape: {x.shape}")  # [64, 784]

# w1: weights connecting 784 input pixels to 100 hidden units
print(f"W1 shape: {w1.shape}")  # [784, 100]

# a1: hidden layer output after sigmoid, one row per image
print(f"Hidden layer a1 shape: {a1.shape}")  # [64, 100]

# w2: weights from 100 hidden units to 1 output unit
print(f"W2 shape: {w2.shape}")  # [100, 1]

# a2: final output — one sigmoid value per image
print(f"Output layer a2 shape: {a2.shape}")  # [64, 1]
```

```
Input batch x shape: torch.Size([64, 784])
W1 shape: torch.Size([784, 100])
Hidden layer a1 shape: torch.Size([64, 100])
W2 shape: torch.Size([100, 1])
Output layer a2 shape: torch.Size([64, 1])
```



Calculate a[2] for the first 64 samples from the train set and print them out.

[185] ✓ 0s

```python
print("a^[2] values for first 64 samples:")
for i in range(a2.shape[0]):
    print(f"{a2[i].item():.4f}", end=" ")
    if (i + 1) % 8 == 0:
        print()
print()
```

```
a^[2] values for first 64 samples:
0.8133 0.0002 0.9979 0.9805 0.8663 0.9970 0.5723 0.3195
0.0617 0.0010 0.6524 0.8307 0.3545 0.9701 0.9991 0.1642
0.1425 0.0869 0.6445 0.6767 0.0024 0.0352 0.0117 0.0008
0.0044 0.9915 0.8821 0.0098 0.9642 0.0193 0.9465 0.4744
0.9609 0.6980 0.1000 0.0009 0.9460 0.0035 0.9968 0.9172
0.3195 0.0426 0.1549 0.7853 0.0362 0.0821 0.5978 0.9784
0.9156 0.2269 0.2211 0.2656 0.8224 0.9933 0.0004 0.3640
0.8869 0.7098 0.6084 0.0023 0.6392 0.9970 0.0245 0.7572
```

Learning Takeaways

This assignment provided me the opportunity to learn the core principles of a neural network specifically for forward propagation, manual weight configuration, and the use of activation functions like sigmoid.

Future Work

For future work, including a loss function with backpropagation would allow the model to adjust its weights and learn the correlation between the input images and digit labels. Adding these elements would enable training and improving predictive accuracy. Overall, this assignment provided a strong basis for grasping deep learning from fundamental concepts and equipped me to tackle more architectures and learning algorithms.