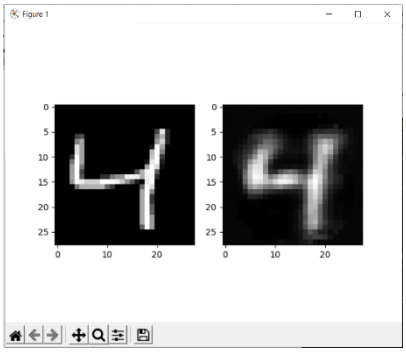
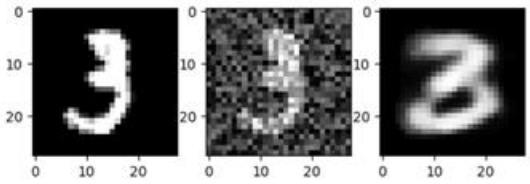




ELEC 475

Lab 1 – MLP Autoencoder

Lab 1 – MLP Autoencoder

Michael Greenspan

Contents

1. PyCharm Installation

If you haven't done so already, install PyCharm on your system. Once installed, you can install other packages that you'll need through the PyCharm terminal, which is accessed by selecting the "Terminal" tab at the bottom of the page (2nd tab from the right). In the terminal, enter the following commands:

```
pip install numpy
pip install torch
pip install matplotlib
pip install torch-summary
```

You can test to make sure that your installation works by opening and executing the **`checkInstall.py`** module distributed with this lab.

Anytime you need a missing package to execute a program, you can load it from the PyCharm terminal as above, using pip.

2. Visualizing the MNIST Dataset

Write a program that loads the MNIST dataset, prompts the user to input an integer index between 0 and 59999, and displays the indexed image and its label.

The syntax to read MNIST is:

```
train_transform = transforms.Compose([transforms.ToTensor()])
train_set = MNIST('./data/mnist', train=True, download=True,
                  transform=train_transform)
```

This will download the training partition of the dataset to your local directory **./data/mnist**. Once instantiated, you can access the images and through the **`train_set.data`** field, and the corresponding image labels through the **`train_set.targets`** field.

The image at index **idx** can be displayed using the syntax:

```
plt.imshow(train_set.data[idx], cmap='gray')
plt.show()
```

3. Implement and Train an MLP Autoencoder

Implement a 4 layer MLP autoencoder, by completing the **__init__()** and **forward()** methods in the **model.py** module. Train your model by invoking the **train.py** module with the following command line arguments in the PyCharm terminal:

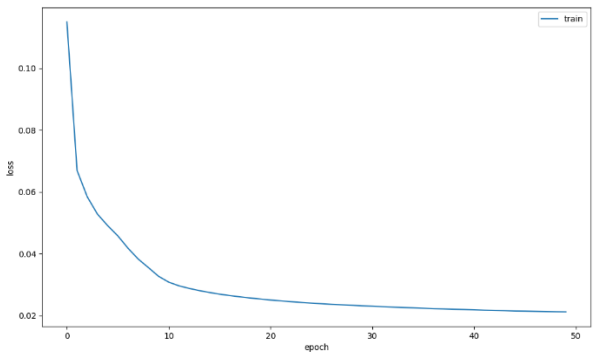
```
python train.py -z 8 -e 50 -b 2048 -s MLP.8.pth -p loss.MLP.8.png
```

Hint 1: The call to **torchsummary** should return something like this:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1               [-1, 1, 392]         307,720
            Linear-2                 [-1, 1, 8]           3,144
            Linear-3               [-1, 1, 392]           3,528
            Linear-4               [-1, 1, 784]         308,112
================================================================
Total params: 622,504
Trainable params: 622,504
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 2.37
Estimated Total Size (MB): 2.39
----------------------------------------------------------------
```

Hint 2: After each of the two fully connected layers in the encoder, and after the first fully connected layer in the decoder, apply a ReLU activation. After the second fully connected layer in the decoder, apply a sigmoid activation.

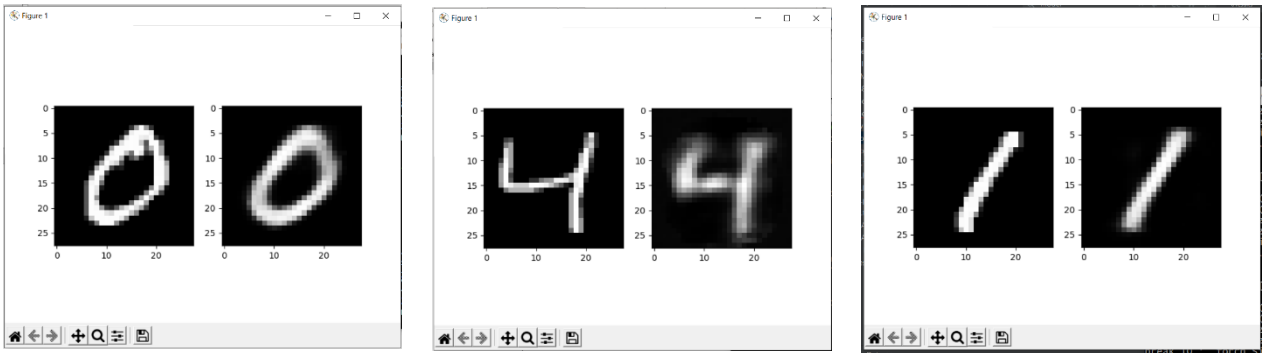
Hint 3: Use the **loss.MLP.8.png** plot to help you determine whether your system is training properly. A well behaved training session should yield a loss curve that looks something like this:



Hint 4: Training on a cpu device for 50 epochs should take ~10 mins. Training on a gpu device will be faster.

4. Test Your Autoencoder

Modify your visualization program from Step 2, to show the reconstruction results from the autoencoder. First, instatiate a version of your model, with the **`MLP.8.pth`** network weights that you generated during training. Then, pass each indexed image as input to the model, and display both the input and the output images side-by-side. Some example input/output pairs should look like this:



Hint 5: Before you pass an image as an argument to the model, make sure that it is a 1x784 dimensional tensor of type **`torch.float32`**. Also make sure that its values are normalized to fall between 0 and 1.

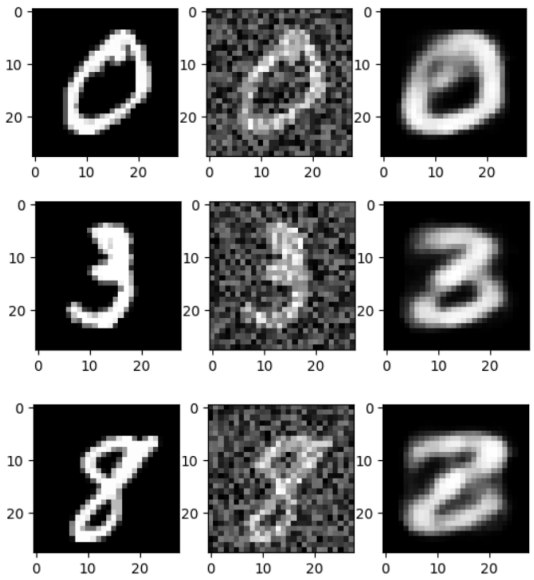
Hint 6: Before invoking your model (i.e. calling a forward inference), make sure that it is in **`eval`** mode, and that you disable gradient calculations.

Hint 7: Use **`matplotlib`** to render the images side-by-side, as follows:

```
f = plt.figure()
f.add_subplot(1,2,1)
plt.imshow(img, cmap='gray')
f.add_subplot(1,2,2)
plt.imshow(output, cmap='gray')
plt.show()
```

5. Image Denoising

Autoencoders can be used to remove noise from an image. Test your autoencoder's ability to remove image noise, by repeating the test in Step 4 with noise added to each image. The output should look something like this:



Hint 8: Uniform noise can be generated using the pytorch **`torch.rand()`** method.

6. Bottleneck Interpolation

Add two new methods to your model class, called **encode** and **decode**. The **encode** method takes the same input as the existing **forward** method (i.e. the flattened image tensor), and returns the bottleneck tensor. The **decode** method takes as input the bottleneck tensor, and returns the reconstructed image. These two new methods are really just the first and second half of the existing **forward** method, which could then be rewritten in the following way:

```
def forward(self, X):
    return self.decode(self.encode(X))
```

Next, create a module that passes two images through the **encode** method, returning their two bottleneck tensors. Then, linearly interpolate through these two tensors for n steps, creating a set of new bottleneck tensors. Pass each of these new tensors through the **decode** method, and plot the results. Some examples of linear interpolations of 8 steps between two images are as follows:

7. Submission

The submission for this prelab should include a brief report, and your complete source code.

The report should include a title (e.g. minimally ELEC 475 Lab 1), your name and student number. If you are working in a team of two, then include the information for both partners in the report (but only make one submission in OnQ). The report should also contain the following sections:

1. Model Details: This section should contain all details of the network, sufficient to recreate the model. This could be in textual form, diagrammatic, or other. The main requirement is that the description is sufficient to recreate the model.

2. Training Details: This section should include all details of the training, sufficient to exactly reproduce the training. This should include the optimization method used (e.g. SGD, Adam, etc.) , optimization hyperparameters (e.g. initial learning rate, momentum, etc.), any learning rate scheduling used, and any other relevant details.

3. Results: A brief description of how well the system worked. Was it as expected, or were there some difficulties and surprizes? Include the loss curve plot in this section, and specifically comment on its behaviour.

Both the report and the complete working source directory should be compressed into a single **`<zzz>.zip`** `file`, where filename **`<zzz>`** is replaced with your student number. If you are in a team of 2, then concatenate both student numbers, separated by an underscore. The zipped directory should include your trained parameter file (e.g. **`MLP.8.pth`**).

Your code should train (as described in Step 3) by entering the following command on a PyCharm terminal:

```
python train.py -z 8 -e 50 -b 2048 -s MLP.8.pth -p loss.MLP.8.png
```

Your code should then execute by entering the following command on the PyCharm terminal:

```
python lab1.py -l MLP.8.pth
```

This should display your equivalent outputs for the figures in Steps 4, 5 and 6.

The marking rubric is as follows:

Item	mark
Step 3 Training	1
Step 4 Output	1
Step 5 Output	1
Step 6 Output	1
Report	2
Correct submission format	1
Total:	**7**