ELEC 475 Lab 1: MLP Autoencoder

Nicholas Krsikapa

20102039

# Model Details

The autoencoder model used for this lab consists of 4 fully connected layers, the first two belong encoder while the others are used in the decoder. When creating an instance of this model, three optional parameters can be passed **N\_inputs**, **N\_bottlenecks,** and **N\_outputs** which all have default values of 784, 8, 784 respectively. These parameters will affect the size of the data at each layer in the model, including the bottleneck size. The following table outlines details of the 4 layers in the model

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Type | Number of Inputs | Number of Outputs |
| 1 (Encoder) | Linear | N\_inputs | (N\_inputs / 2) |
| 2 (Encoder) | Linear | (N\_inputs / 2) | N\_bottlenecks |
| 3 (Decoder) | Linear | N\_bottlenecks | (N\_inputs / 2) |
| 4 (Decoder) | Linear | (N\_inputs / 2) | N\_inputs |

In total, the model has four functions, \_\_init\_\_, encoder, decoder, and forward. The \_\_init\_\_ function declares all the layers above with the appropriate sizes. The encoder takes in a tensor and puts it through the following steps:

1. Call Layer 1
2. Apply relu activation function
3. Call Layer 2
4. Apply relu activation function
5. Return the result of step 4

The decoder also takes in a tensor and applies these steps:

1. Call Layer 3
2. Apply relu activation function
3. Call Layer 4
4. Apply sigmoid activation function
5. Return the result of step 4

Please note that the relu and sigmoid activation functions used above comes from the module *torch.nn.functional* and *torch.* Lastly, the forward function is used to call both the encoder and decoder function one after another and return the results.

# Training Details

The training module made for this lab iterates through a dataset several times, putting each image through the model, comparing the loss, update the weights of the model, and has the ability to save the model as well as save the plot of the loss over each epoch. The table below outlines the parameters passed to the training function, a brief description, and what was used in this labs training:

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter Name | Description | Used for Model | Notes |
| n\_epochs | Number of epochs, the amount of times the model will be put through the training data | 50 | Results are based on 50 epochs |
| optimizer | Holds an optimizer function | Adam | * torch.optim * Learning Rate = 1e-4 * Weight\_decay = 1e-5 |
| model | The model being trained | autoencoderMLP4Layer |  |
| loss\_fn | The loss function | MSELoss | * torch.nn |
| train\_loader | a Dataloader object of training data to put through the model | MNIST training set | Imported from torchvision |
| scheduler | Scheduler function | ReduceLROnPlateau | * torch.optim.lr\_scheduler * Learning Rate = 1e-4 * Weight\_decay = 1e-5 |
| device | “cpu” or “cuda” depending on what piece of hardware training will be done on | Cpu |  |
| save\_file | (*optional*) name to save model by | MLP.8.pth |  |
| plot\_file | (*optional*) Name for image for the loss plot | Loss.MLP.8.png |  |

# Results

**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.**

Using the training function as outlined above resulted in a system that seemed to work well most of the time. After running the training multiple times and testing different learning rates, optimizers, and schedulers used the smallest reproduceable loss was 0.023. The loss curve plot of the final training function looked as what was expected from the example in lectures, please see below.

A graph with a line

Description automatically generated

Over the first 5 epochs a sharp slope reducing the loss from about 0.11 to 0.045. From there the curve drastically plateaued, and eventually from the 15th – 50th epoch the slope was almost linear with a slight decline.

When passing validation images through the model, the reconstructed output was usually close to the original that the same number could be identified but did not always have the same unique features. The model occasionally struggled by producing an image of an 8 when the original was any other number. In the denoising test, it was a surprise to see how accurate the reconstructed image looked to the original when given the noisy image.