ELEC 475 Lab 1

MLP Autoencoder

Nathan Duncan | 20283699

# 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 has sufficient details to fully and exactly recreate the model.**

The implemented MLP autoencoder model uses the same structure as discussed in lecture. Our model inherits from torch.nn.model, a pytorch base class for neural networks. The inputs are MNIST 28x28 pixel grayscale images of handwritten digits. The input is flattened to a 1x784 vector and passed to the first fully connected layer of the four-layer MLP autoencoder and truncates the data to 1x392 vector. The Second fully connected layer takes the 1x392 vector and further compresses it to a 1x8 bottleneck vector. The decoder is the exact mirror of the encoder with two MLP layers scaling the bottleneck from 1x8, up to 1x392, and back to the full 1x784 vector.

A diagram of a number of numbers

AI-generated content may be incorrect.

Figure : Graphical representation of the MPL autoencoder architecture.

# 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 all other relevant details**

We used the provided training setup in “train.py” to train our model discussed in the previous section with a 1x8 bottleneck vector. We use the Adam optimizer from torch.optim with learning rate 1e-3, and weight decay 1e-5 to guide optimization. A scheduler is also used to reduce the learning rate value once the loss plateaus. The loss function itself is a mean squared error function from torch.nn.MSELoss with mean reduction of the output. One modification we did make was doubling the training data loader’s batch size from 2048 to 4096 to maximize the use of computer resources and enabled shuffling of the data. The model was trained for 50 epochs.

command

# Results

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

The training loss per epoch shown in Figure 2 shows the loss decreased exponentially during the first ~20. At this point it begins nearing a plateau and the schedular likely begins reducing the learning rate for finer steps in tuning the model weights. The loss decreases marginally over the next 30 epochs to a final loss of about 0.02. A graph with a line

AI-generated content may be incorrect.

Figure : Loss vs training epochs of the MNIST dataset autoencoder.

Although we did not create any quantitative post-training evaluation methods of our own, qualitatively reconstructions of the autoencoder match many features of the input (i.e. graphics we visually determine to be a certain number, generally produce a reconstruction with is also visually recognizable as the same number). The biggest shortcoming of our model was in denoising images, which often produced illegible output images. Although though a 1x8 is already a great compression from the original 1x784 input, one option to improve denoising is to edit the size of the bottleneck reducing it further or possibly extending it slightly. Additionally we could experiment with tuning of the training hyperparameters and select other optimizers for improved performance.