# ELEC 475 Lab 1

**Name 1:** Matteo Van Der Platt; **Student number:** INSERT

**Name 2:** Jacob Badali; **Student number**: 20290739

## Model Details

This model is a 4 layer MLP autoencoder which takes 60,000 flattened 28x28 MNIST images and feed them through a fully connected network with a bottleneck size of 8.

**Name**: autoencoderMLP4Layer

**Type**: Fully connected autoencoder (MLP) 4 layers

**Input**: Flattened 28x28 image, 784 input features

**Input Tensor Size**: batch\_size, 784

**Encoder + Decoder**:

Layer 1: 784->392 (activation ReLU)

Layer 2 bottleneck: 392->8 (activation ReLU)

Layer 3: 8->392 (activation ReLU)

Layer 4: output: 392->784 (Sigmoid (map to range [0,1]))

**Forward Pass**:

Input X → Encoder (fc1 → ReLU → fc2 → ReLU) → Bottleneck latent vector

Bottleneck vector → Decoder (fc3 → ReLU → fc4 → Sigmoid) → Reconstructed output

## Training Details

## Dataset:

The dataset is 60,000 grayscale MNIST 28x28 images. The default parameters for training are batch size of 256, 30 epochs, 32 bottleneck. These defaults are overwritten with the command line argument

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

Now, the batch size is 2048, # epochs is 50, and bottleneck size is 8. We save the trained weights to the **MLP.8.pth** file and save the loss to **loss.MLP.8.png**.

We have converted the images to tensors using **transforms.ToTensor()** for preprocessing.

## Model

As described above, the model is a 4 layer fully connected autoencoder (MLP). The input is 784 features, and the bottleneck is 8 units (compressed laten space). The decoder reconstructs back to 784 and is converted to 28x28 for viewing.

## Loss, Optimization, Scheduling

The loss function used is Mean Squared Error (MSE). The optimizer used is an Adam optimizer. The learning rate is **1e-3**, and the weight decay is **1e-5**. Adam betas (default): **β₁ = 0.9**, **β₂ = 0.999**. The learning rate scheduler monitors the loss and then reduces learning rate when the loss plateaus (**ReduceLROnPlateau(optimizer, 'min')**).

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1 Output of train.py when you run it

As seen above, the output of train.py explains the model architecture and the training setup.

## Results

The autoencoder was successfully trained on the MNIST dataset using a bottleneck dimension of 8. The system performed almost as expected: it was able to reconstruct the input digits both with clean images and artificially corrupted images.

For clean images, the autoencoder clearly reconstructed the original digits. As seen below, the reconstructed images closely resemble the input, preserving the main structure of each digit. Minor blurring is observed which is typical for an MLP.

A black and white image of a number

AI-generated content may be incorrect.

Figure 1 Clean reconstruction

For noisy images, noise was applied using the functions below adding uniform random noise to the images before processing.

noise\_factor = 0.2 # change this to change the intensity of the noise.

noisy\_img = img + noise\_factor\*torch.rand(img.shape)

The noise was scaled by a factor of 0.2, to ensure that the digits remained visible while providing a noticeable difference. At first we scaled the noise to fully blur the image, which resulted in poor results. Below, you can see the reconstruction of a digit from a noisy image. You can see that the blurring is heavier, and the shape of the digit is distorted somewhat.

A group of black squares with white text

AI-generated content may be incorrect.

Figure 2 Reconstructed image with noise

The interpolation works as a smooth transition between two digits by encoding two images into the bottleneck space and interpolating between them. An example can be seen in the figure below.

A number on a black background

AI-generated content may be incorrect.

Figure 3 Interpolation between ‘7’ and ‘1’ digit

The loss curve shows a steady decrease in MSE during training, indicating that the autoencoder gradually learned to reconstruct the training images. The curve flattened towards the end, suggesting it has converged. As seen below in Figure 4, the loss as the model is trained over epochs reduces beginning around 0.12 and ending around 0.02.

A graph with a line

AI-generated content may be incorrect.

Figure 4 Loss function by epoch

The system successfully reconstructs clean images and is somewhat resilient to moderate noise, though reconstructions become slightly blurrier. The network performs well in bottlenecking despite its small bottleneck and simple MLP architecture.