Autoencoder for Image Reconstruction – ELEC 475 Lab 1 Report

Noah Serhan

Model Details

The autoencoder model is a 4-layer Multi-Layer Perceptron designed to encode and decode images. The autoencoderMLP4Layer class defines the architecture of the mode. It consists of four fully connected linear layers, including an input layer, a bottleneck layer, and two hidden layers.

- Input Layer:
  - Size: N_input = 784, which corresponds to the flattened MNIST images of size 28x28 pixels.
  - Self.fc1: The first linear layer with N2 neurons (N2 = 392, or half of N_input).
  - Activation: ReLU
  - ReLU helps the model capture complex features and maintain sparsity.
- Bottleneck Layer:
  - Size: N_bottleneck = 8
  - Self.fc2 = The second linear layer with N_bottleneck neurons.
  - Activation: ReLU
  - The bottleneck is where the autoencoder's network learns to represent the input data in a compressed or reduced-dimensional space. A larger bottleneck could capture more details but at the cost of increased dimensionality, and a smaller one could lead to more aggressive compression. The choice is somewhat arbitrary and depends on specific requirements, but in this case, it is 8.
- Decoder Hidden Layer:
  - Self.fc3: The third linear layer with N2 neurons.
  - Activation: ReLU
- Output Layer:
  - Size: N_output = 784, which is the original MNIST image size.
  - Self.fc4: the fourth linear layer with N_output neurons.
  - Activation: Sigmoid
  - Sigmoid is chosen for the final decoder layer because of a few characteristics. The first being its output range, outputting values between 0 and 1 which is comparable to the target values, being pixel values in the MNIST images. Moreover, sigmoid's outputs can be interpreted as probabilities, where 1 is certain and 0 is "off" or false. Finally, it provides a smooth and continuous output, which is useful for image reconstruction given that pixel values should not be either 1 or 0.

Methods

Encoder

The encode method takes MNIST images and compresses them into an 8-dimensional latent space. Hidden layer 1 performs the first stage of feature extraction. Hidden layer 2 is the bottleneck layer that compresses the input into a lower-dimensional latent space. Its size can be configured, in this case it's set to 8. The input tensor X is passed through the encoder layers fc1 and fc2 with ReLU activation functions.

Decoder

The decode method takes the encoded tensor and passes it through the decoder layers fc3 and fc4 with ReLU activation for the former and sigmoid for the latter.

Forward

The forward method chains the encode and decode methods to create the complete autoencoder network.

Training Details

This section goes through all of the necessary details of the training process for the autoencoder model.

Optimization Method

- The optimization method used in this training is “Adam”, which is an adaptive learning rate optimization algorithm with efficient convergence properties that combines the advantages of both the AdaGrad and RMSProp gradient decent methods.
- The optimizer is initialized with a learning rate of 0.001.

Optimization Hyperparameters

- The initial learning rate is set to 0.001. This hyperparameter controls the step size during wight updates and is a crucial factor in determining the training convergence and stability.

Learning Rate Scheduling

- A learning rate scheduler is used in this training to adjust the learning rate during training based on the model’s performance. The specific scheduler employed is ReduceLROnPlateau.
- ReduceLROnPlateau monitors the training loss and dynamically reduces the learning rate when the loss plateaus. It operates in “min” mode, meaning it reduces the learning rate when the loss stops decreasing.

Loss Function

- The Mean Squared Error (MSE) loss function is used when computing the loss, performing backpropagation to update the model’s parameters.

Other Relevant Details

- The training data is loaded using DataLoader and the MNIST dataset, consisting of 60,000 training images of hand drawn single digit numbers, is used for training.
- The device used for training is determined dynamically. If a GPU is available (cuda), then it is prioritized as it is much more optimal for training models; otherwise, it runs on the CPU.
- The model undergoes training for 50 epochs, where each epoch represents a complete iteration through the entire training dataset.
- Training loss values for each epoch are printed, and the training loss curve is saved as a plot.
- The training batch size is set to 2048 for efficient training on available hardware.

Results

Training Loss

The vertical axis of the curve represents the value of the loss function, which is the quantified difference between the predicted output of the model and the ground truth for a given set of input data. The horizontal axis represents the number of training iterations or epochs. The curve shows how the loss changes as the model iteratively updates its parameters to minimize this loss.

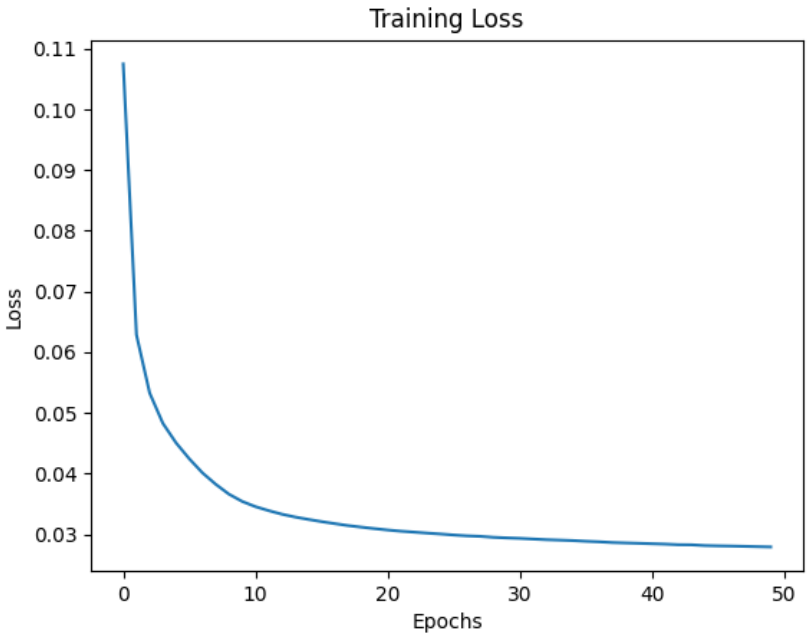

Figure 1: Loss curve plot obtained after training the model for 50 epochs.

At the start of the training, the loss is relatively high (around 0.108). This is because the model's starting weights lead to predictions that are far from the desired targets. As the training progresses through the epochs, the loss decreases, signifying that the model is starting to make more accurate predictions. Towards the last few epochs, the loss curve begins to "flatten out" which is a sign that the loss curve is converging to a stable value. At this point, the model has learned to make very accurate predictions and training it for additional epochs would have a minimal impact on its effectiveness.

Step 4: Image Reconstruction Test

In this test, the model's ability to reconstruct images from the MNIST dataset was evaluated. The model's performance was exactly as expected, as it successfully reconstructed the images with relatively high fidelity. There were certain cases where, due to the relatively low bottleneck size, certain imperfections in the numbers, such as the 0 seen in Figure 2, were smoothed off in the reconstructed image. Otherwise, the model performed swimmingly. The original images and their corresponding counterparts are displayed side by side for visual comparison.

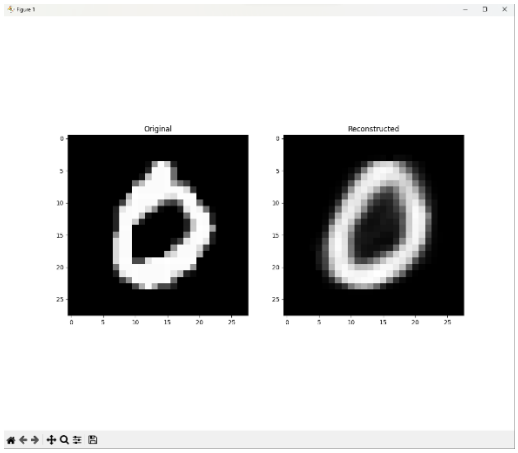

Figure 2: Input/output pair showing slight difference in the shape of the reconstructed number.

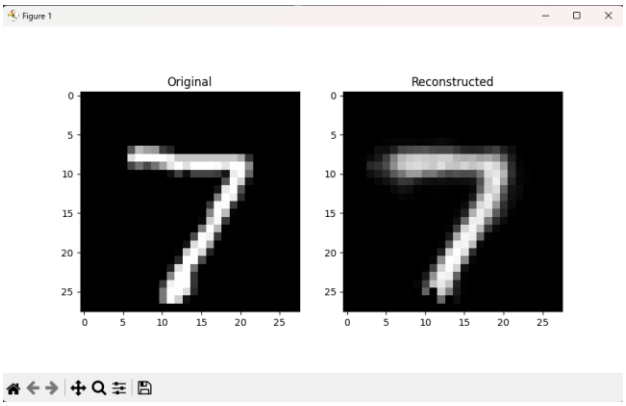

Figure 3: Input/output pair resembling each other with only slight differences.

Step 5: Image Denoising Test

In this test, the model's performance in predicting a number from a noisy image is assessed. The input images are intentionally corrupted with Gaussian noise, scaled down to a factor of 20%, and the model is tasked with removing the noise and reconstructing the images. The model performs well in denoising, as it effectively removes the added noise and reconstructs a clear image that resembles the input image.

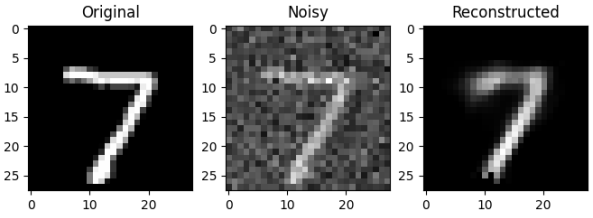

Figure 4: The input image, the input with noise and the reconstructed image of the written number 7.

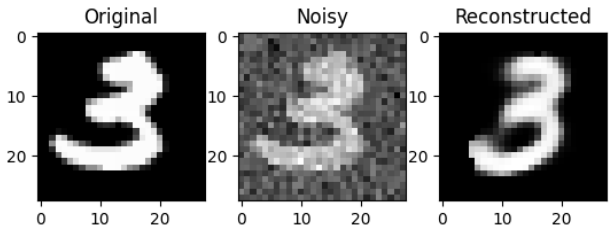

Figure 5: An additional example of a test run in step 5.

Step 6: Image Interpolation Test

The model's ability to interpolate between two images is demonstrated. Two input images are selected, and the model encodes them into a common latent space. Then, the model linearly interpolates between these two latent representations to create a sequence of images that transition seamlessly. The model performed as expected with no significant surprises. There were some cases where the transition would show signs of another MNIST number before reaching the correct result, however, even with these uncommon additional steps, the transition remained seamless and iterative. The test results show the starting image, the eight interpolated images, and the ending image for visual comparison.

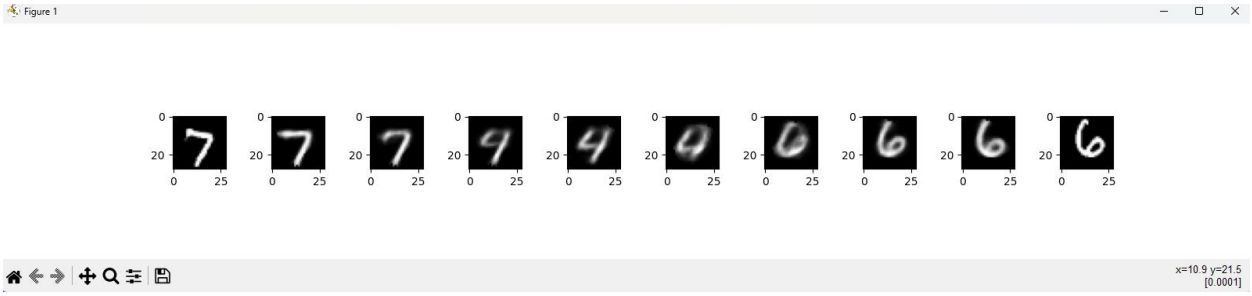

Figure 6: Example of a test run in Step 6.

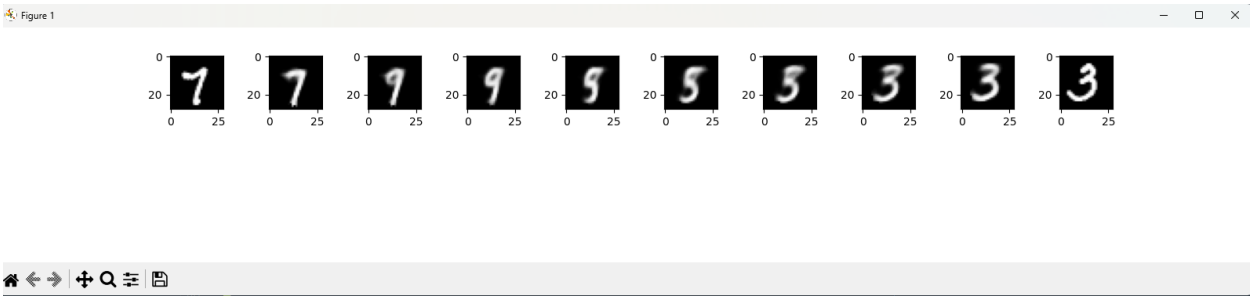

Figure 7: An additional example of a test run in Step 6.