Neural Networks and Deep Learning
Homework 2: Unsupervised Deep Learning

Giulio Zani

February, 2021

Contents

1 Introduction

In this homework we were asked to explore unsupervised deep learning models. I have choosen to work with the MNIST dataset, as in homework 1. Yet this time, during training, I have ignored the labels of the dataset.

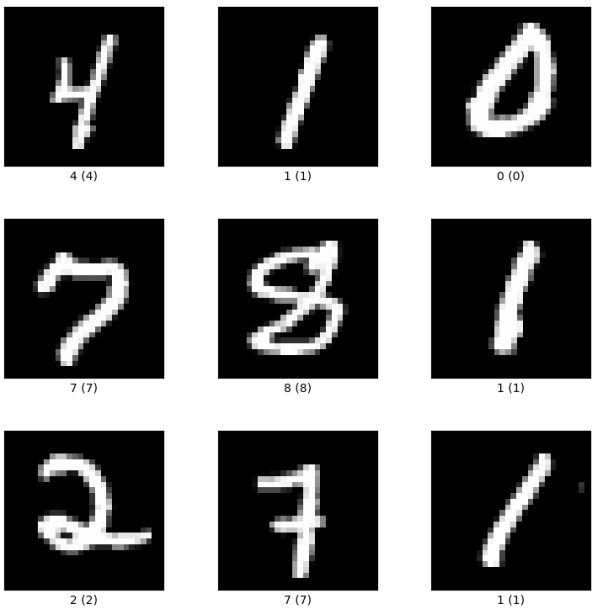

Figure 1: Some examples of MNIST images

Besides training and parameters fine-tuning, we were asked to perform these tasks:

- Implement and test denoising autoencoders
- Fine-tune the (convolutional) autoencoder using a supervised classification task
- explore the latent space structure (e.g., PCA, t-SNE) and generate new samples from latent codes

Unfortunately due to time contraints I did not perform the second subtask: fine-tuning witha a supervised task.

2 Method

When I first learned abouit variational autoencoders (VAEs) I was charmed by them and thus decded to use them for this project. I started out from an example I have found on the internet and then improved it to finally get to the convolutional variational autoencoder I have used for this task.
This is the network structure blueprint:

```
ConvVAE(
  (conv): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
  (fc21): Linear(in_features=21632, out_features=20, bias=True)
  (fc22): Linear(in_features=21632, out_features=20, bias=True)
  (fc3): Linear(in_features=20, out_features=21632, bias=True)
  (t_conv): ConvTranspose2d(32, 1, kernel_size=(3, 3), stride=(1, 1))
)
```

In the above blueprint conv, fc21 and fc22 are part of the encoder structure, whereas fc3 and t_conv (transpose convolution) are part of the decoder. I have used **ReLU** activation function and **Adam** optimizer. Training was performed for 30 epochs.

2.1 Optimizing Hyperparameters with Genetic Algorithm

I have used the same genetic optimization tool I created for homework 1 to optimize hyperparameters in this task. As done previously to compute hyperparameters' fitness I ran the model for one epoch and computed the test loss. I ran it with a population size of 30 and for 12 generations. The hyperparameters and their ranges are the following:

$$\text{batch size} \in \{10, ..., 128\}$$

$$\text{lr} \in [1e-8, 1.0]$$

$$\text{gamma} \in [0, 1.0]$$

$$\text{weight decay} \in [1e-7, 1e-4]$$

The evolved hyperparameters are:

$$\text{batch size} = 49$$

$$\text{lr} = 0.428$$

$$\text{gamma} = 1.0$$

$$\text{weight decay} = 3.2318e-05$$

With more time and computational resources it would have been interesting to explore network structure optimization.

2.2 Training and network analysis

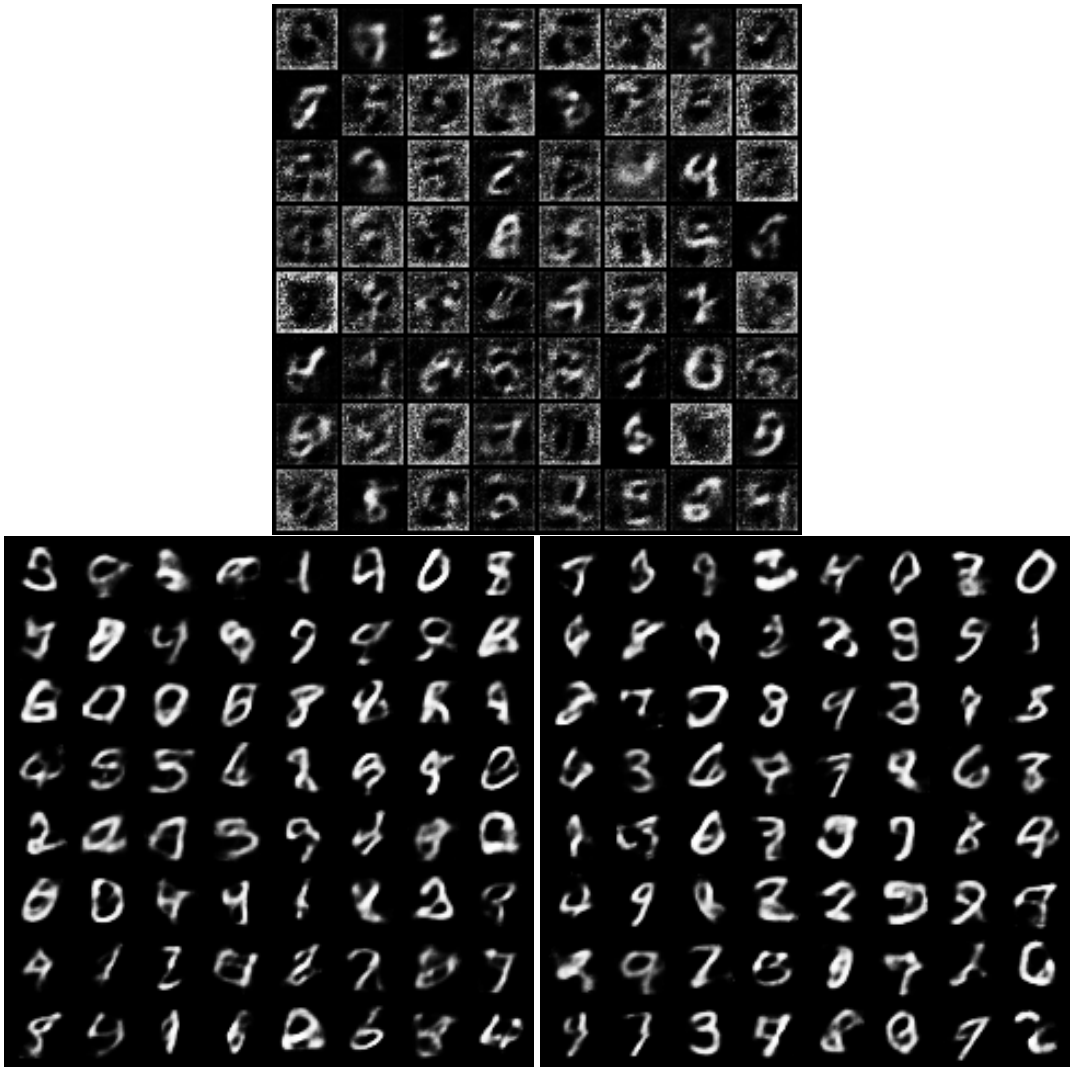

Figure 2: Sampling after 1 (upper most) 15 (left) and 30 (right) epochs.

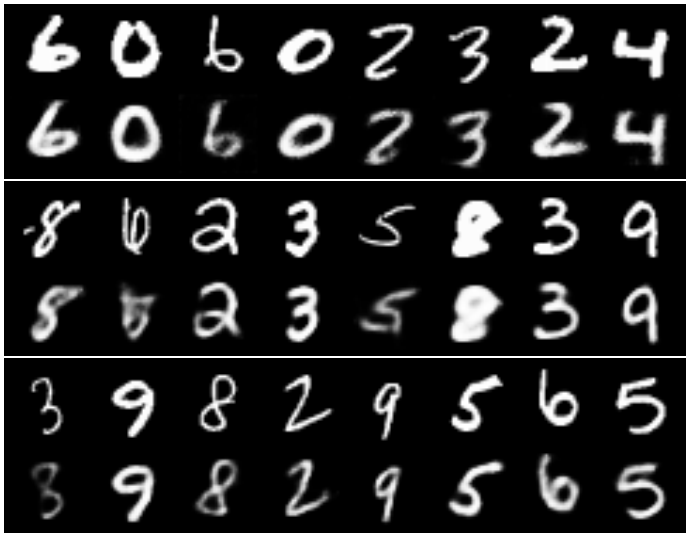

Figure 3: Reconstruction after 1, 15 and 30 epochs

2.3 Denoising

I have tried adding some noise to original MNIST images and checked the quality of the images output from the network:

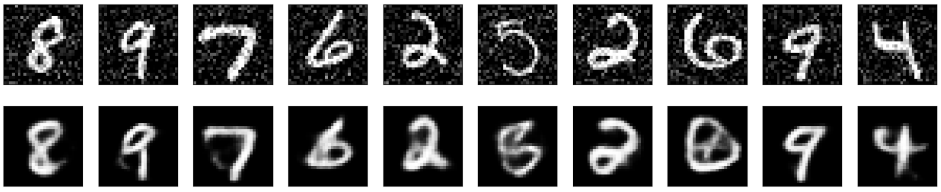

Figure 4: Random noise added to MNIST images (above) and its VAE output (below)

Results were interesting but far from perfect, for example a five was turned into an eight.

2.4 PCA of latent structure

I did the following: for each of the 20000 images I've choosen to pick for this task, I encoded it, then reparametrised and finally performed a 2-dimension PCA of the resulting matrix. I have used the PCA implementaion of the sklearn package.

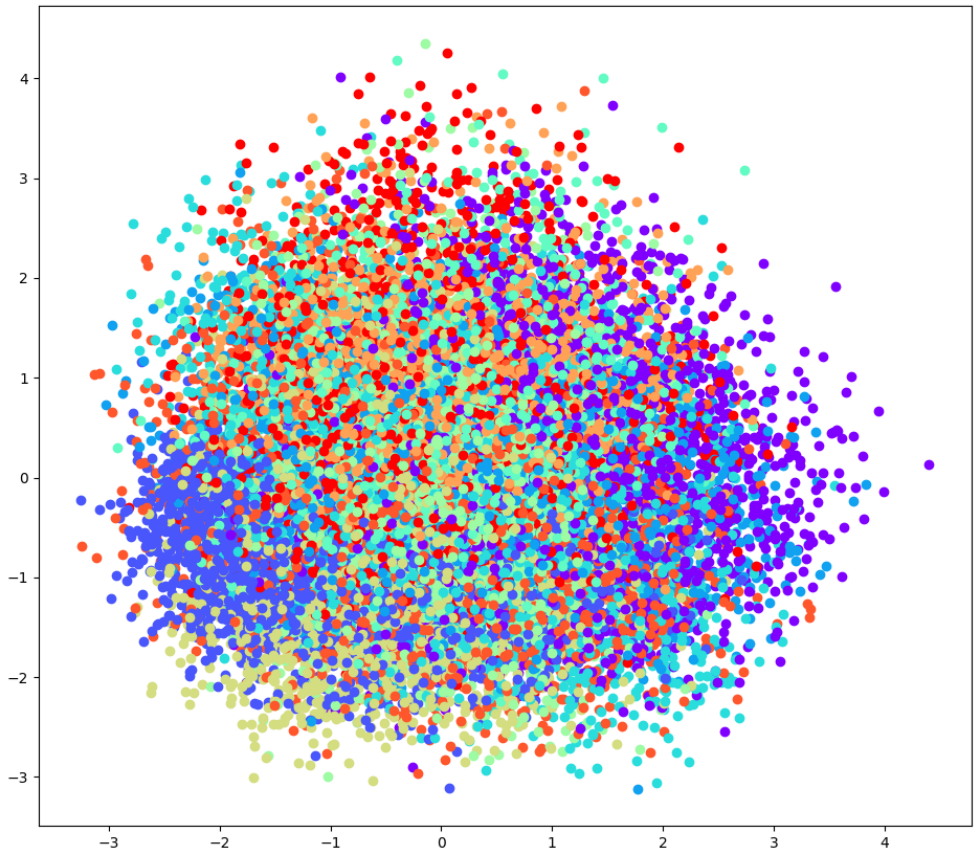

Figure 5: PCA of the network's latent space, sampled using 20000 images. Different colors represent the different labels (0 to 9)

As one can see in Figure**??** some clusters are visible, but they're very fuzzy and they overlap with each others. There appear to be colors, i.e., digits which don't have clusters. Perhaps this is due to suboptimal learning or due to too low dimensionality pca. I have also tried performin a 3 dimensional PCA with a similar procedure, using 2000 images (see Figure**??**).

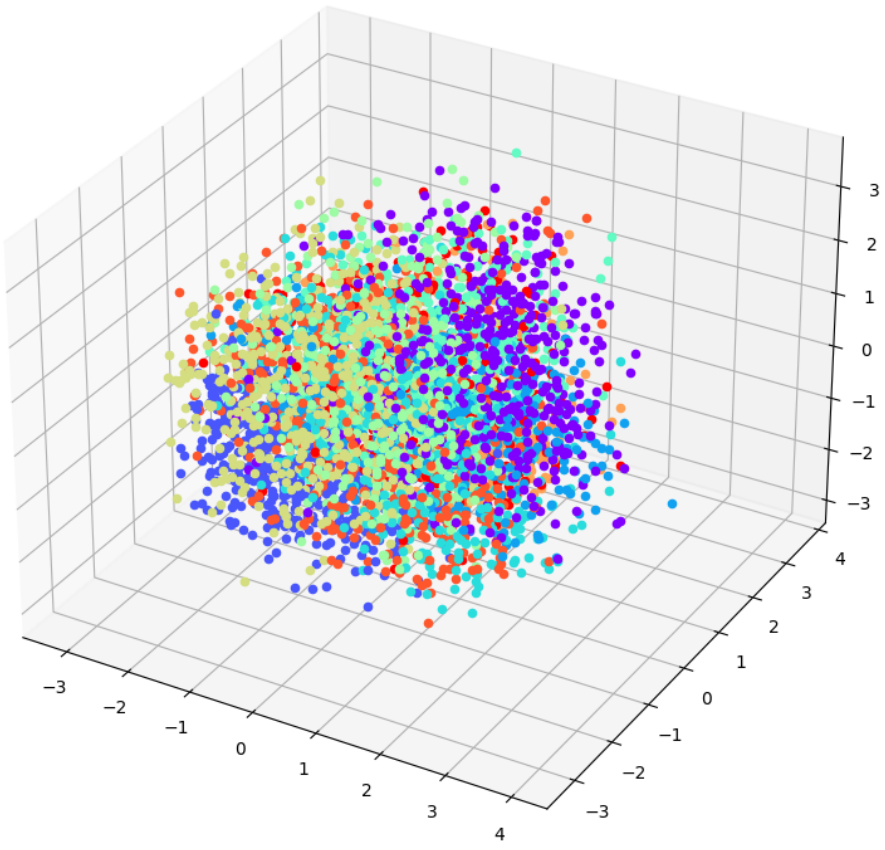

Figure 6: 3 dimentional PCA of the network's latent space, sampled using 2000 images. Different colors represent the different labels (0 to 9)

```
python -m deep_learning_hw2
```

3 Result

The test loss of the trained model is 100.9932