Unsupervised Learning
Autoencoder for MNIST dataset

Introduction

This report summarizes the development of an autoencoder with the purpose of detecting anomalies in a sample dataset. The latter is the MNIST collection of handwritten digits, which consists of grey scale images 28x28 pixels. The goal is to build and train an autoencoder that is able to detect the digits 4 and 8 as anomalies, whereas all the other digits have to be considered normal. The library used to create the autoencoder is **pytorch**, framework for building deep learning models.

Data Setup

The original structure of the dataset consists of 2 subsets of the data:

- Train data: 55000 rows of images and their corresponding labels
- Test data: 10000 rows of images and their corresponding labels

Every row of the dataset contains a 28x28 representation of an image, whereas the label of an image is a 10 binary items list, which is filled with zeros except for the same index as the value of the digit, where a 1 is placed (e.g: if an image represents a 2, the label will be [0, 0, 1, 0, 0, 0, 0, 0, 0, 0].

These two sets of data have been furtherly split into 3 sub-groups, which are train/test 4, having all the images representing the digit 4, train/test 8 having all the images representing the digit 8, and lastly train/test normal which contain all the other digits images excluded 4 and 8. After having performed this division of data, the sizes of each split are the following:

	Normal	4	8
Train	44301	5307	5389
Test	8044	982	974

Except for this split into the 3 groups of data, there was no need to ulteriorly pre-process the data.

For the next chapters, the following pipe-line has to be considered:

1. The training of the model is done with the Train Normal data
2. The validation and parameter's tuning are computed on Test Normal and Test 8 (development set)
3. The final evaluation / test is done using the Test 4 data (evaluation dataset)

Algorithm Setup

Since it is desired to have a similar reconstruction of the original image, I use a topology with a symmetric structure of the neural network's layers, since it is common practice to do so.

The training of the autoencoder model requires the following parameters, which need to be optimized based on this specific task:

1. Topology, or the overall structure of the layers. As aforementioned, for this neural network I use a symmetric structure of the layers.
2. Batch size: the size of a single batch that the data will be split into during the training process.

3. Epochs: the number of epochs that will be used for training the model
4. Learning rate: the learning rate of the optimization function used to train the model

There is also another parameter called the weight decay of the optimizer function, but I preferred to focus more on the other 4 listed parameters since they have the most impact on the model and, once tuned well, already yielded good results.

A possible stopping criterion for the training process could be either the reconstruction loss reaching a plateau / too little improvement in training between an epoch and another, or also a maximum number of epochs. I am using the latter because of computational and technical limitations of my hardware. Ideally, if I didn't have to wait a big amount of time for each training process (depending on the model structure it could take up to 40 minutes, if not more), I probably would've gone for the first approach.

The anomaly score chosen for this problem, to quantify how much an image has to be considered anomalous or not, was the autoencoder's reconstruction loss of an image. Once all the images of the Test Normal and Test 8 had been reconstructed, the MSE (Mean Squared Error) of the respective image sets have been computed and a threshold has to be chosen.

The latter has been calculated by maximizing the difference between True Positive Rate and the False Positive Rate, since it is wanted to maximize the number of anomalous images found in the 8 data (anomalous dataset) and minimize the number of anomalous images mistakenly classified as such in the Normal data.

I tried also to choose a threshold that maximizes the value of the f1-score, but it's not a good choice since the two datasets Test Normal and Test Eight, on which the threshold is decided, are highly unbalanced (respectively 8044 and 974).

Experiments and Results

I performed 6 major sets of 3 experiments each, which are the following:

1. Layers number
2. Number of nodes for every layer
3. Number of nodes in the bottleneck
4. Learning rate
5. Batch size
6. Number of epochs

For the first 3 sets of experiments, I used by default 128 as batch size, 0.0001 as learning rate and 100 epochs each.

Layers number

By increasing the number of layers in the topology, the TPR kept on increasing and FPR kept on decreasing. After seeing such a progress and improvement I wanted to add other 2 layers, but with 9 total layers it was significantly slower than with respect to 7, which already yielded good results

Number of layers	3 layers	5 layers	7 layers
TPR	0.9189	0.9497	0.9825
FPR	0.2115	0.1532	0.1246

Number of nodes for every layer

After fixing the number of layers, I experimented with both bigger and smaller values for each hidden layer than in the topology proposed. Below a table showing the results of these experiments:

Topology	[784, 256, 128, 10, 128, 256, 784]	[784, 400, 200, 10, 200, 400, 784]	[784, 512, 256, 10, 256, 512, 784]
TPR	0.9538	0.96	0.9559
FPR	0.1204	0.1067	0.0981

Having less nodes was the worst outcome out of the three obtained, whereas having that same number of nodes or having more yielded very similar result. Out of the two I went with the latest since it had a lesser FPR, but still maintaining a very high TPR.

Number of nodes in the bottleneck layer

With the presented topology I performed another series of experiments modifying the size of the bottleneck layer by trying as values 10, 20, 30. Below you can find a table summarizing the results, which basically told me to go on having as optimal size of the bottleneck layer 10, having outperformed the other two.

Nodes in bottleneck	10	20	30
TPR	0.963	0.9127	0.8912
FPR	0.0955	0.1272	0.2141

Learning rate

Having found an optimal topology, I tried to change the learning rate, maintaining still the 100 epochs, and it turns out that the one that I was using this whole time was still the best out of 0.001, 0.0001 and 0.00001.

Learning rate	0.001	0.0001	0.00001
TPR	0.9435	0.962	0.9487
FPR	0.1253	0.0998	0.152

Batch size

I tried as batch sizes the following ones: 64, 128 and 256. If I picked a lower number, it would take very long, whereas with a higher value it would significantly decrease in terms of performances. The comparison of the three is the following, out of which the I went with is 64

Batch size	64	128	256
TPR	0.9435	0.96	0.9487
FPR	00929	0.1149	0.152

Epochs

The last parameter I tried to tune was the number of epochs used for the training process. By increasing the number of epochs, I noticed that the MSE of both Test Normal and Test 8 kept on decreasing, but the percentage of anomalies spotted usually decreased, therefore I chose to prioritize the anomalies detected rather than the MSE of the images.

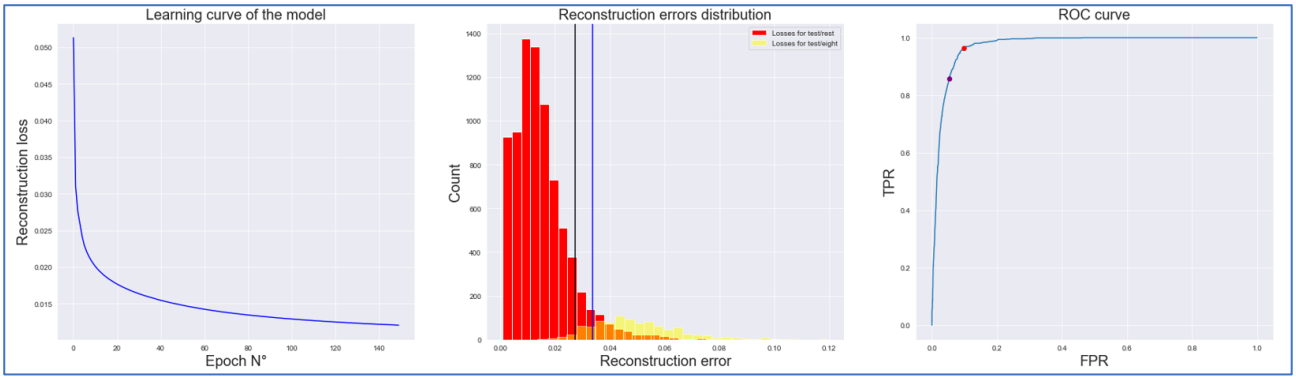
Epochs	100	150	200
TPR	0.9682	0.9733	0.9559
FPR	0.1206	0.1192	0.1094

Results

Using these found parameters, the final validation results are the following:

- MSE For Train/Normal: 0.011896
- MSE For Test/Normal: 0.014910
- MSE For Test/8: 0.049692
- Optimal threshold TH maximizing TPR – FPR: 0.027
- Anomalies correctly spotted in Test/8 using TH: 96.41 %
- Anomalies mistakenly spotted in Test/Normal using TH: 9.76 %
- Best f1 score: 0.75
- Optimal threshold F1TH maximizing the f1-score: 0.034
- Anomalies correctly spotted in Test/8 using F1TH: 85.83 %
- Anomalies mistakenly spotted in Test/Normal using F1TH: 5.23%
- AUC-ROC: 0.9726

Below is reported the plot summarizing the most important data. Even though I don't use the f1 optimization threshold, I still decided to insert it in the overall plot just for visualization purposes.



On the left plot can be seen the learning curve of the model through the 150 epochs. In the middle plot there are the losses plotted for both Test Normal (red) and for Test 8 (yellow). The vertical white line indicates the threshold that has been found by maximizing the difference between TPR and FPR, whereas the blue line is the hypothetical threshold that I would use if I were to consider the maximization of the f1 score.

On the last plot staying on the right, I plotted the ROC curve for all the different thresholds and I marked in red the point that represents my threshold maximizing TPR – FPR, whereas the pink point is the one maximizing the f1 score.

Since we're trying to solve the problem of detecting the anomalies, I chose as reference the white threshold because it gives more importance to the TPR while still keeping low the FPR, whereas the blue threshold, the one maximizing the f1 score, keeps very low the FPR at the expense of overlooking many anomalies.

Evaluation

The final evaluation, after having found the optimal model for the anomaly classification task, has given back the following results:

- MSE for Test/Normal: 0.0149
- MSE for Test/4: 0.0418
- Anomalies correctly spotted in Test/4 using max(TPR-FPR) threshold: 78%
- Anomalies correctly spotted in Test/4 using max(f1-score) threshold: 59%

In conclusion, I can say that the model found after all these experiments performs quite well in both spotting the anomalies (between test/8 and test/4 the average percentage of anomalies spotted is of 87.2%) and recognizing normal data as such. In this case, as previously mentioned, I won't be using the f1 score as metric for deciding the best threshold since it has proven to be less accurate than the original threshold calculation.

Most probably there exists a better model that would be better trained and that it would distinct better normal and anomalous data, but considering the time available and the computational power of my device, I still managed to find a reasonable solution to the problem.

Below is reported an image showcasing the reconstruction capabilities of the model, which reconstructs well the normal image (left) and reconstructs poorly the anomalous images (middle and right) by misclassifying them often as one of the normal ones (since the model was trained on them).