COMPUTER SCIENCE DEPARTMENT

Practical Deep Learning

Assignment 2 Report

Authors:
Almog Zemach: 205789001
Raz Monsonego: 313292120

June 2022

Contents

1 Part 1: Warm-up with synthetic data

In this section we write a module called lstm_ae_toy.py in which we implement, train and evaluate an LSTM AE on the synthetic data we created according to the instructions in Part 2 of the assignment.
In order to implement our LSTM AE, we implemented a class that extends *torch.nn.Module*, in which we defined an architecture corresponding to the LSTM AE architecture we discussed in class. Specifically, we define a encoder layer and a decoder layer which are LSTM RNNs, and a linear layer which is responsible for the last transformation after which the output is of the same dimension of the input (since we seek for **reconstruction**).
We could have used a hidden layer size equal to the input size in the decoder (and by thus we would not need the linear layer), but we chose not to, since this would induce a hidden layer size of 1 for the decoder for our data, which is not expressive.
The forward function and the learning loop are quite trivial from the definition of the LSTM AE description and the job description, therefore we do not deliberate on them (see code attached).
We emphasize that for all data-sets used, we enforce normalization (min-max).
In future sections, we use the model we built here and **extend** it with more layers etc. to perform the wanted actions.

1.1 3 Examples of the synthetic data-set

Below are 3 of examples from the synthetic data-set we created according to the assignment instructions.
The numbers in the legend are the indexes of the time-series in the data-set.

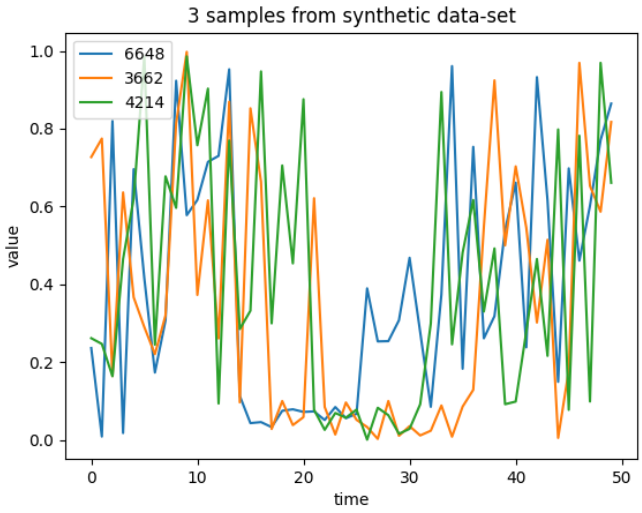


Figure 1.1: 3 samples from the synthetic data-set.

1.2 Reconstructions using our neural-network

The plots below are the plots received by the best model we found when performing a grid search over the following hyper-parameters of the model: hidden state size, mini-batch size, learning rate and gradient clipping.
Attached are two plots of original signals and their reconstructions (annotated 'orig' and 'rec' for 'original' and 'reconstructed'):

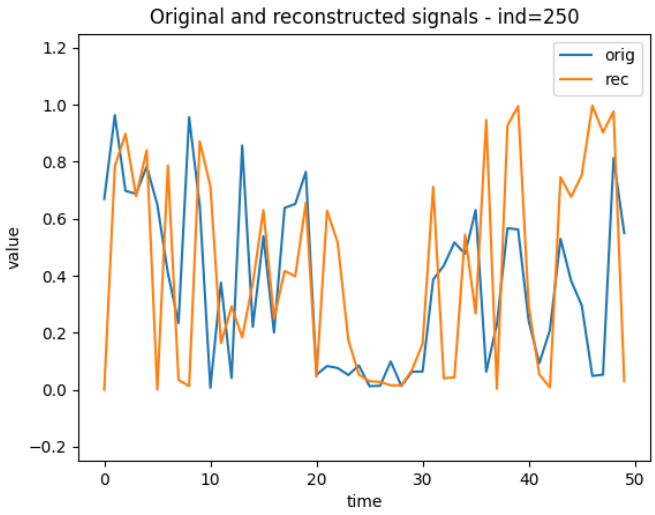


Figure 1.2: Original and reconstructed signals of the time-series in index 250.

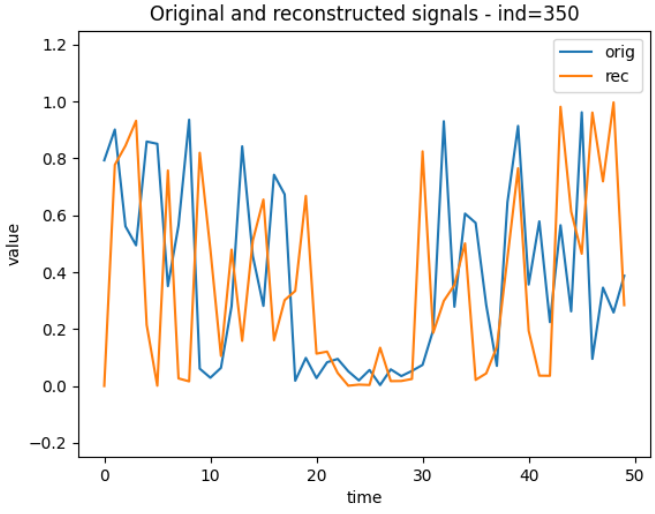


Figure 1.3: Original and reconstructed signals of the time-series in index 350.

2 Part 2: Reconstructing and classifying MNIST images

Same as in Sec. 1 our network construct from two LSTM nodes, one for encoder and one for decoder. We also added a linear layer after the decoder in order to convert the dimensions to the correct values. We normalize the data such that it lies in the range $[0, 1]$, using a transformation (see Code).
Note that the model used here is an extension of the model described in 1.

2.1 Grid-search

First, we conduct a grid search with 42 epochs in order to find the best hyper-parameters:

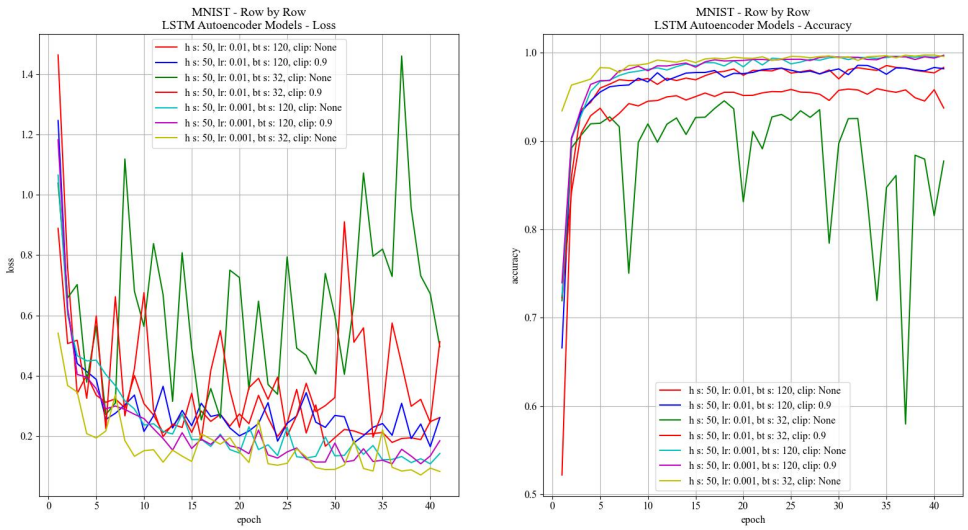
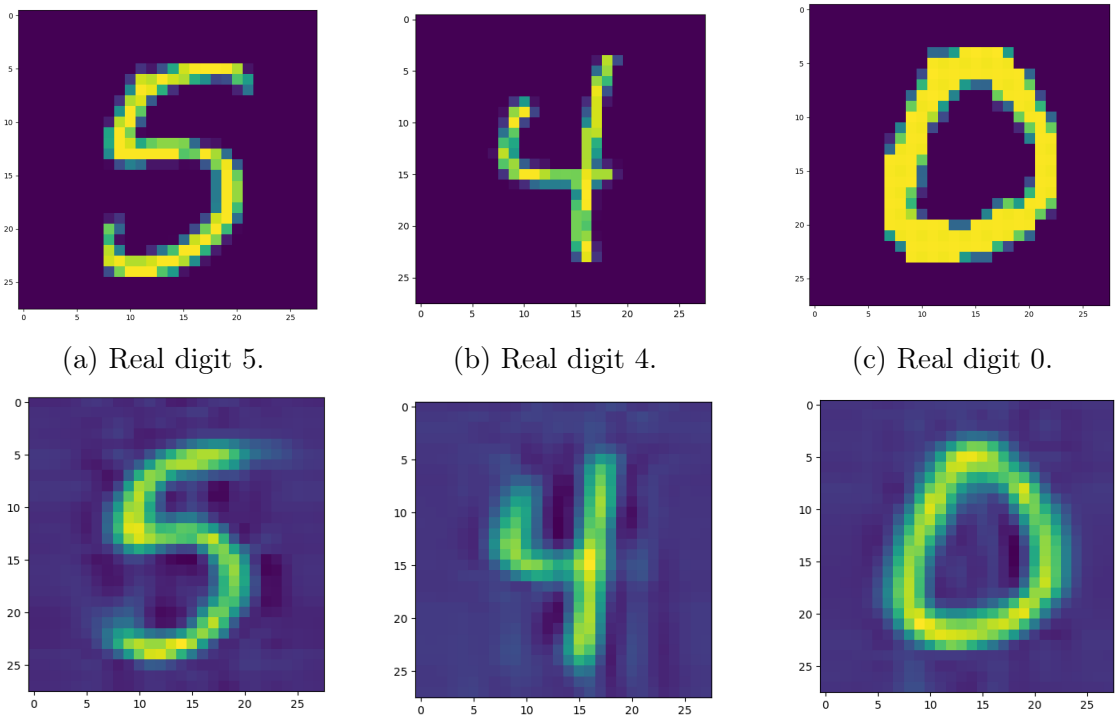


Figure 2.1: MNIST train data - Row by Row. Models were trained upon several hyper-parameters using the following notation: hr - hidden layer size. lr - learning rate. bt_s - batch size. clip - gradient clipping.

By examining fig. 2.1 we decided to choose the purple model which was one of the most accurate models in our grid search.
We trained the network again with the hyper-parameters of the purple model over 100 epochs (see Fig. 2.3).
The reconstruction of the row by row method is shown bellow:



(a) Real digit 5. (b) Real digit 4. (c) Real digit 0.

(d) Reconstructed digit 5. (e) Reconstructed digit 4. (f) Reconstructed digit 0.

Figure 2.2: Three simple digits.

2.2 Adding classification

We add a classifying layer to our network, and add a loss component which corresponds to the cross-entropy loss, so our network will learn how to classify the images of the MNIST data-set.
Namely, each network outputs a vector of size 10 (the amount of classes in the data-base) which corresponds to the probabilities of the input to reside in the corresponding class.
Hyper-parameters labels in Fig. 2.3 and 2.4 are same as in Fig. 2.1.

2.2.1 Row-by-row

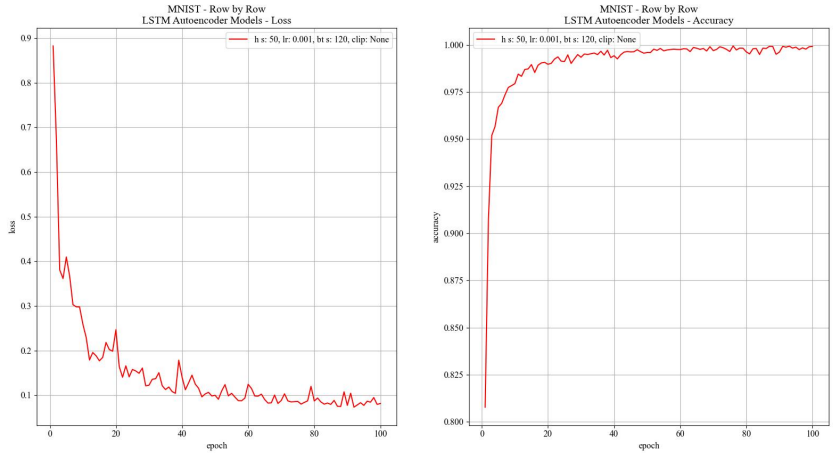


Figure 2.3: MNIST train data - Row by Row. We chose the best model (see Fig. 2.1) but with 100 epochs.

With the model in Fig. 2.3 we achieved an accuracy of 98.883 % on the validation data.

2.2.2 Pixel-by-pixel

We repeat Sec. 2 when the input to the network is inserted pixel-by-pixel (instead of row-by-row). Notice that due to the extremely long training process with the pixel-by-pixel format (9 hours for 5 epochs), we stop the training after 5 epochs, and thus the results are not mind-blowing.

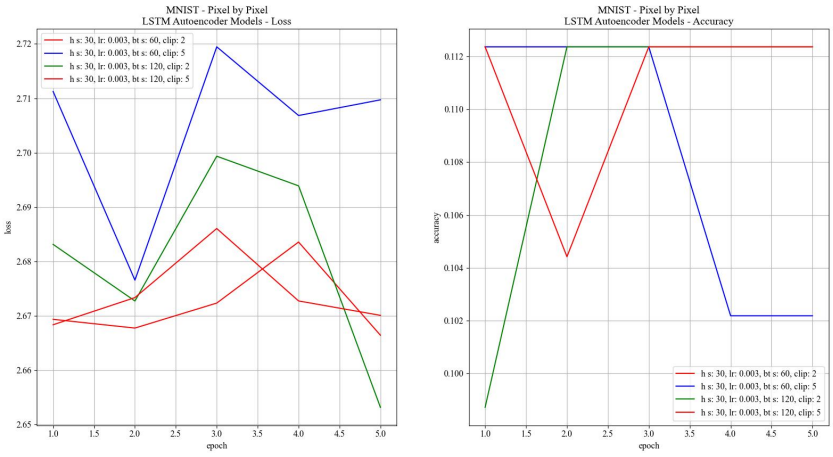


Figure 2.4: MNIST train data - Pixel by Pixel. Iterating over 5 epochs.

The pixel-by-pixel format didn't perform well upon the reconstruction and classification tasks. All of the models we trained did not surpass 11% accuracy.

3 Part 3: Reconstructing and predicting the SP500 index

Regarding data pre-processing for the SP500 stock data-set, we perform the min-max normalization to all input time-series, so that the stocks values reside in $[0, 1]$. Furthermore, we ensure all the time-series we use are of the same length of 1007, and ommit the stocks with missing data-points (NaNs).

3.1 Daily max value of the AMZN and GOOGL stocks

We attach the daily maximum stocks values of the AMZN and GOOGL stocks:

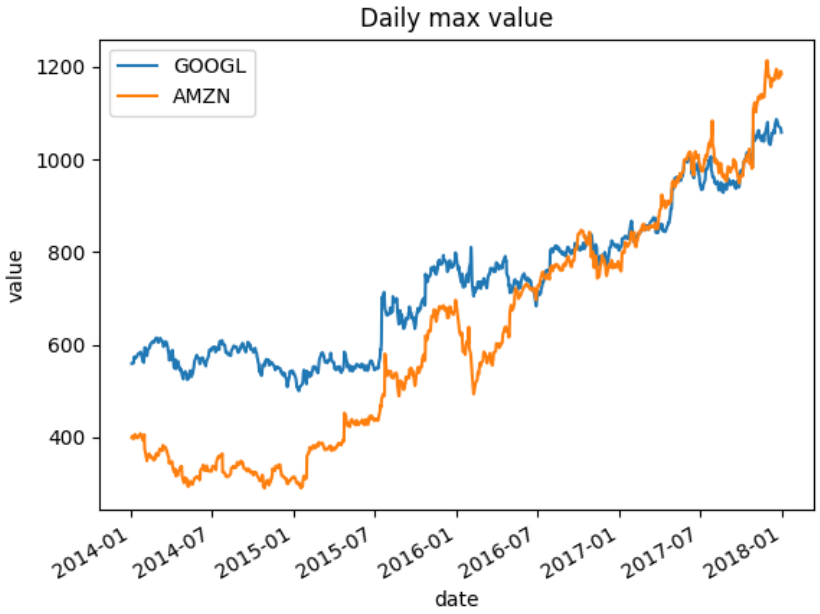


Figure 3.1: Daily maximum stocks values of the AMZN and GOOGL stocks.

3.2 Reconstruction

We now train our auto-encoder (AE) to reconstruct the values of stocks from the SP500 data-set.
We attach 3 plots of original time-series and their reconstructed time-series:

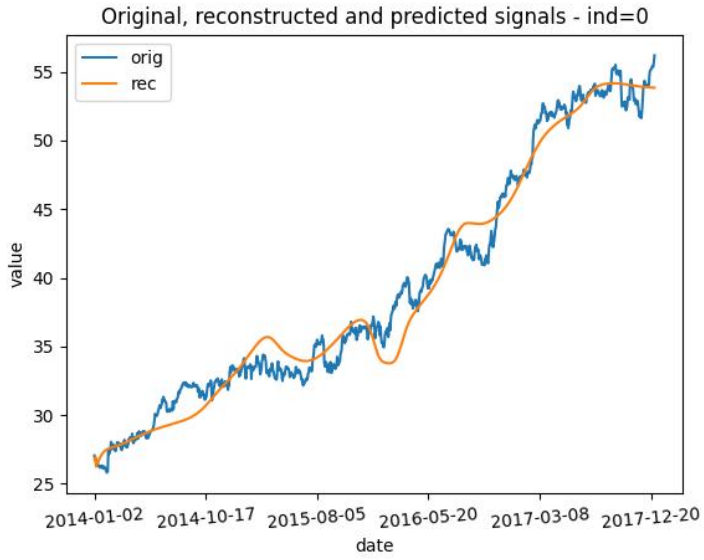


Figure 3.2: Original and reconstructed time series received from our net.

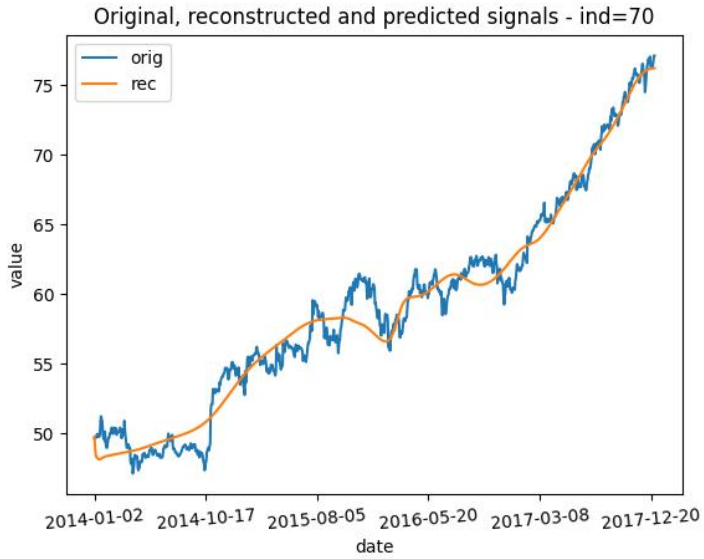


Figure 3.3: Original and reconstructed time series received from our net.

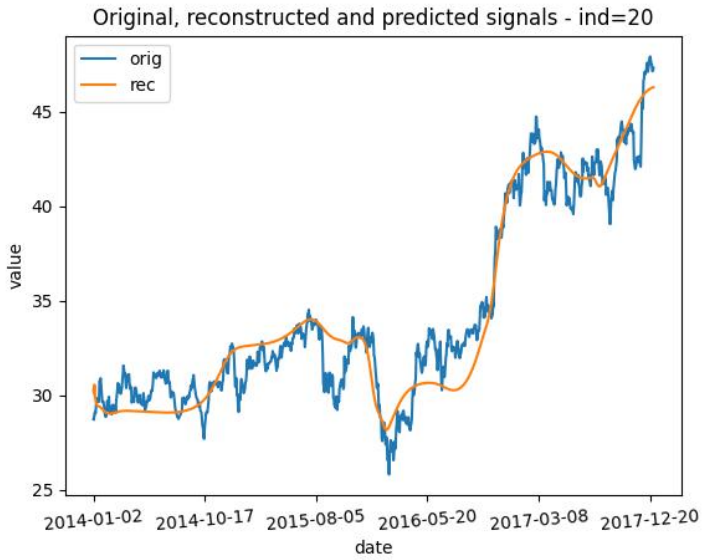


Figure 3.4: Original and reconstructed time series received from our net.

3.3 Prediction

We now wish to train our AE to predict the future values of a time-series.
In order to do so, we generate pairs in the following manner: $(x_t, x_(t+1)$ for every $t \in [1, T-1]$, and in addition to the current loss, we also penalize with $MSE(x_t, \hat{xt})$ so that our network learns to perform prediction as well.

3.3.1 Reconstruction loss & prediction loss

We attach plots of the reconstruction loss vs. time, and of the prediction loss vs. time.

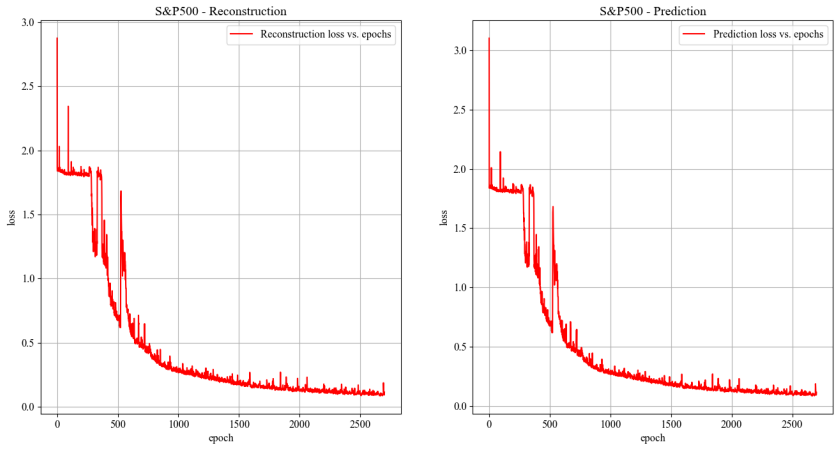


Figure 3.5: Reconstruction and Prediction loss over 2700 epochs.

3.3.2 Multi-step prediction

We implement multi-step prediction in the following manner: Given a time-series $\{x_t\}_{t=1}^{T}$, we feed the network with the input $\{x_t\}_{t=1}^{\frac{T}{2}}$, so that the network outputs $x_{\frac{T}{2}+1}$. In the following manner we compute the next values step-by-step, receiving the overall series $\{x_t\}_{t=1}^{T}$.
We show the multi-step prediction results on the GOOGL time series (prediction starts on 2016-01)

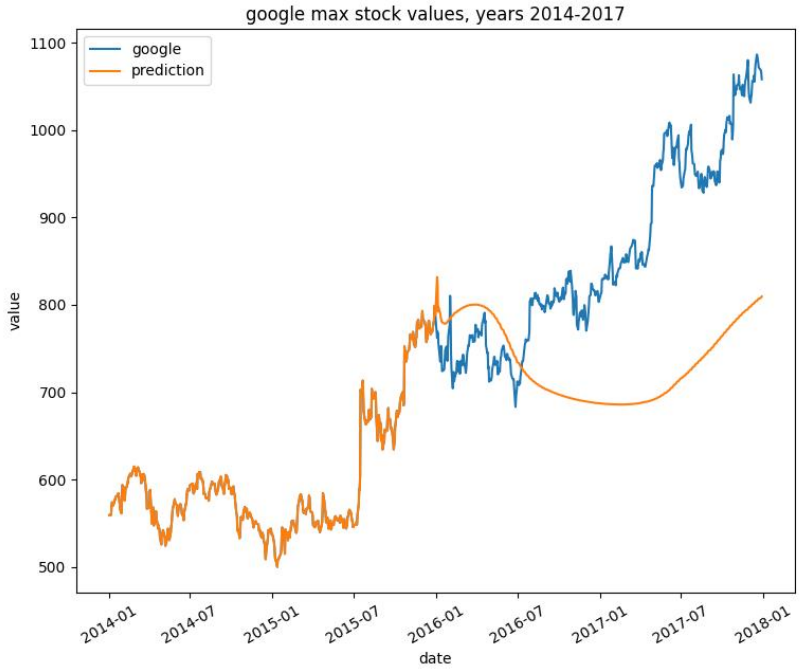


Figure 3.6: Multi-step prediction graph. we inserted the output of the model in each iteration as the input for the next step. Notice that the prediction start from 2016-01 (blue line represent real target value).

4 Conclusions

1. We see that our LSTM AE has performed quite poorly on the synthetic data-set we created (as depicted in the assignment). This is interesting, since the same model was able to learn much better on real-world data-sets such as MNIST and SP500, which indicates that our model is doing something correctly.
   Nevertheless, we see that our network is able to learn that there is a range of smaller magnitude, and is able to reconstruct a time-series with an appropriate lower-magnitude range as the input holds. We think this is the main thing there is to learn for this data-set.
   Our conclusion of this, is that learning random data is a **hard** task, which is not so surprising - since there is not a lot to learn in random data!

2. On a global note, we saw that performing a grid-search for the best hyper-parameters of a model was a **crucial** step in order to create a good AE.
   During our search, we found that even slight changes in the hyper-parameters can sometimes imply dramatic changes in the results, and that it is hard to anticipate the right range of the hyper-parameters for different models. This results in the need for a grid-search for the best hyper-parameters.

3. We indeed managed to surpass the 98% accuracy demand for the accuracy of our classification on MNIST images (got 98.83%). The reconstruction (see Fig. 2.2) also shows good results. This increases the validity of the model.

4. There is a **big** difference between feeding the network an image row-by-row and pixel-by-pixel, as we can see in Fig. 2.4, 2.3.
   This implies (as expected) that it is hard to learn such long time-series as we get when we vectorize the images of MNIST.

5. We see that our model is successful in reconstructing the values of the SP500 data-set, in the sense that the reconstructed signal approximately takes the mean value of the several surrounding values of the time-series at each time-step, which can be thought about as cleaning the noise.

6. When comparing the one-step prediction with the multi-step prediction, we can obviously see that the multi-step prediction is off by quite a margin compared to the one-step prediction.