Practical Deep Learning - HW2

LSTM AutoEncoders

Yael Azulay

1 Background

Nothing to submit.

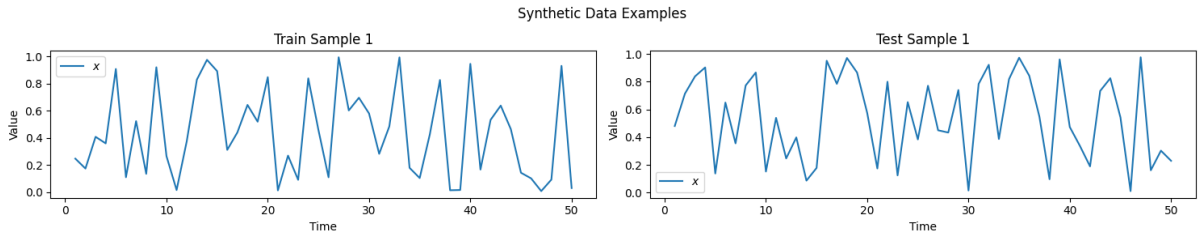
2 Data

Nothing to submit.

3 Specific Tasks

3.1 Warm-up with the synthetic data

1. Two examples from the synthetic dataset (one training one test). Graphs of signal value vs. time:

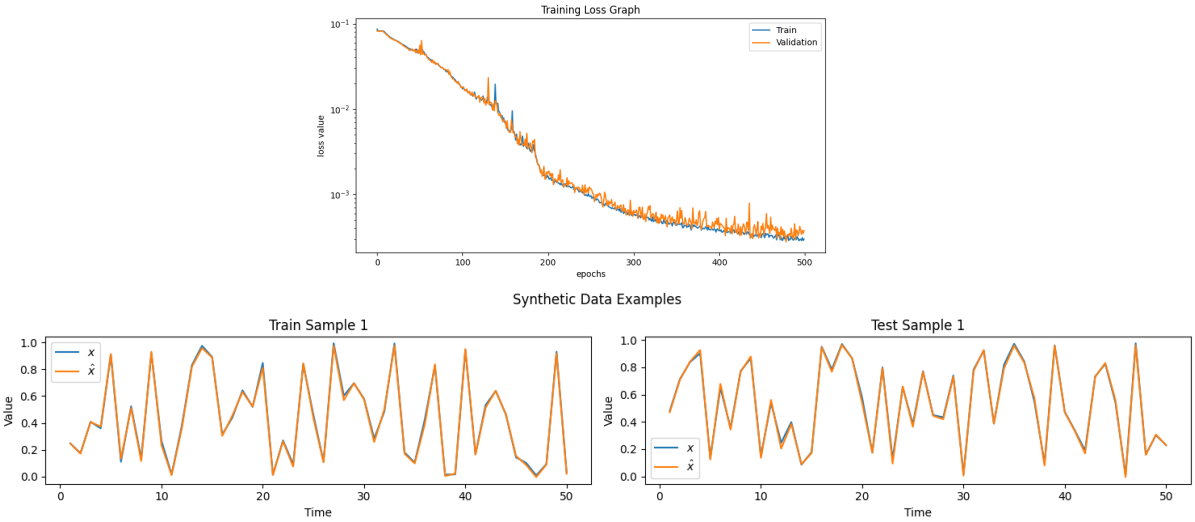


   We can see (or at least get a sense) that the segments are indeed normalized and centered.

2. In this question we were required to reconstruct the input sequences using the LSTM AE neural network. Our loss function for the reconstruction task is:

$$l_{MSE} = \|\hat{x_t} - x_t\|_2^2 \quad \text{where} \quad \hat{x_t} = \sigma(U(\hat{h_t} = Decoder(z = Encoder(x_t))))$$

   Training graph and visual results on two examples, one from the training set and one from the test set:

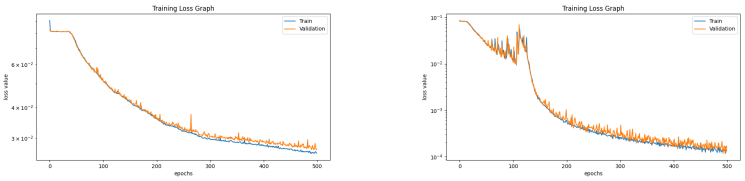


We can see that in our reconstruction test there is no problem of over-fitting - the model succeeds with the same level of accuracy on sequences not seen in training. The loss function error is 0.0003.
The hyper-parameters I chose using cross-validation:

- Hidden state size = 64 ∈ [32,64,128,256]

  The ambition is to achieve a small hidden layer dimension, both for the computational cost and to avoid over-fitting. For a smaller hidden state size (a), the graph has converged to a larger error (≈ 0.02) and its reconstruction ability is limited. For larger hidden state size (b), we got almost identical results (≈ 0.0001) and without over-fitting, so we can also take large hidden state size but this will affect the cost of calculation.

  hs=32 hs=128

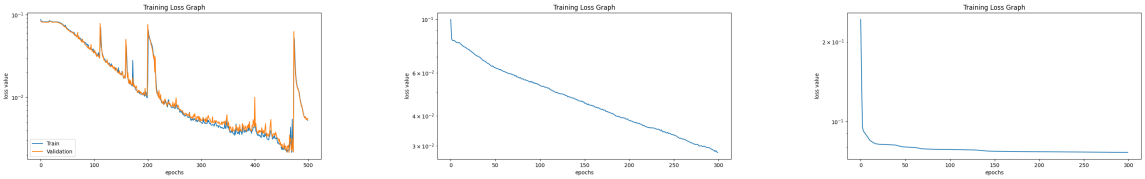


  (a) (b)

- Learning rate = 0.005 ∈ [0.05, 0.01, 0.005, 0.001, 0.0005]

  A larger learning rate resulted in jumps in the convergence graph (a) and a smaller learning rate decreased the convergence factor (b) and even got stuck when it was too small (c).

lr=0.01 lr=0.001 lr=0.0005



(a) (b) (c)

- Batch size = 32 ∈ [32, 64, 128]

  Too big a batch size resulted in a slightly slower convergence and too small a batch size resulted in larger jumps in the convergence graph (both on the training set and on the validation set). Impact is quite similar to the selection of the LR hyper-parameter, but more minor.

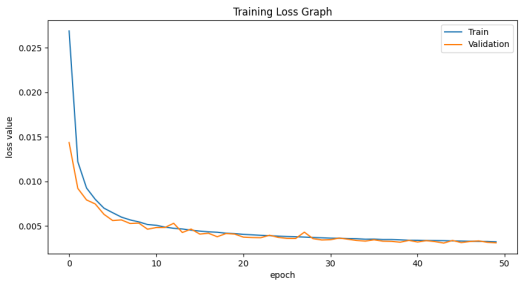
- Gradient Clipping = without

  It can be seen that the training convergence graph does not suffer from the problem of exploding gradients. Since there is a requirement to select hyper parameters using cross validation also for the gradient block values using the gradient clipping technique, I tried the method - in situations where the graph looked normal, no improvement was observed (naturally), and for situations where the graph seemed problematic (jumps in convergence or crashes), the method did not help. Most of the improvements were due to the correct choice of the number of neurons in the hidden layer and the appropriate step size.

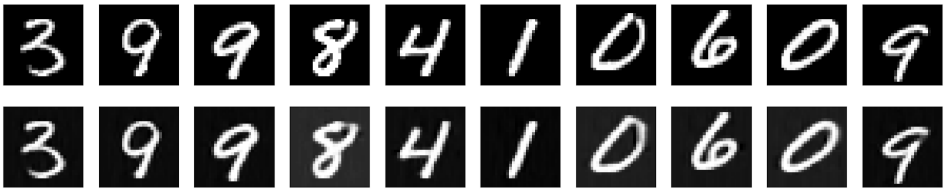
3.2 Reconstructing and classifying MNIST images

In this section, my computational resources were limited so I put most of my effort into training 28-length sequences with a size of 28 (row by row). I also tried to train on 784-length sequences (pixel by pixel), the training phase took a few days, yielded less successful results and it was difficult to select the hyperparameters using the validation set. Therefore in this report I will focus and present the results on the short sequences.

1. In this question we were required to reconstruct the MNIST sequence-images using the LSTM AutoEncoder we defined. Because the Y-axis value ranges are small in this question, the graph is not displayed on a log scale. Training graph (a) and visual results on 10 examples from the test set (b):



(a)



(b)

The hyper-parameters I chose using cross-validation:

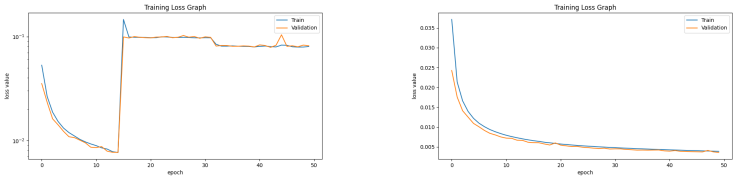
- Hidden state size = 64 $\in$ [32,64,128,256]

  The ambition is to achieve a small hidden layer dimension, both for the computational cost and to avoid over-fitting. I saw that even a network with a hidden layer of 64 neurons provides a loss function error on the order of 10^{-2}. A hidden layer in a larger dimension resulted in a faster convergence but not necessarily a more accurate result.

- Learning rate = 0.01 $\in$ [0.05, 0.01, 0.005, 0.001]

  For a larger learning rate value, some of the gradient steps kept us away from the minimum and the training failed to converge. Example of too big-step in the problematic direction in (a). For a too small-learning rate value, the convergence was rather slow and did not achieve greater accuracy or success than the selected learning rate. It can be seen that for a small step but not too small the convergence graph for the validation set is smoother (b), which makes sense because the size of the change at each step is smaller.

lr=0.05 lr=0.001

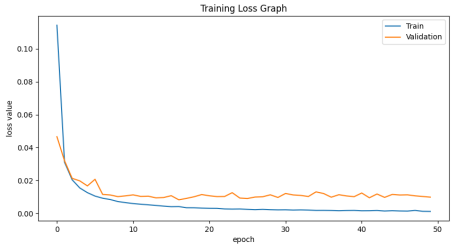


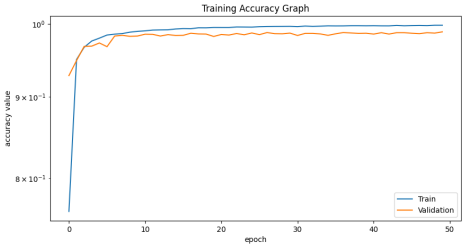
(a) (b)

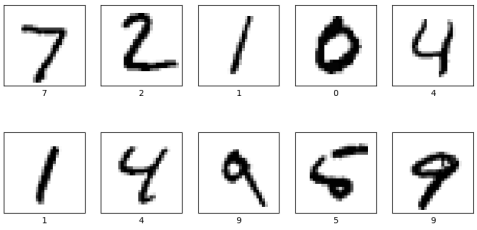
- Batch size = 64 $\in$ [32, 64, 128]

  For this question, the batch size hyper-parameter did not affect on the success or the convergence and I obtained quite similar results for the different trials.

2. Classifying MNIST images - I added to the neural network a layer of Dense, which extracts a 10-dimension-vector, so that each entry in that vector expresses the probability that the input sequence is a manuscript of this index digit. The cross-entropy loss function checks the reliability of the probability vector and the accuracy value checks whether we returned the highest probability to the correct digit. Training graphs (loss and accuracy) and visual results on 10 examples from the test set:







It can be seen that for this task, there is a small gap between the results on the training set and the validation set, to reduce the gap I added a Dropout component whose role is to make it harder for the model to memorize the input and prevent over-fitting. The results have improved and the existing gap is quite negligible, the accuracy on the validation set is greater than 99%.

Train Loss: 0.0009 Validation Loss: 0.008

Train Accuracy: 99.88% Validation Accuracy: 99.04%

I selected the following hyper-parameters in a similar way to the previous sections. In this question the results were similar for different parameters, the effect of the parameters was smaller.

- Hidden state size = 64 $\in$ [32,64,128,256]
- Learning rate = 0.005 $\in$ [0.05, 0.01, 0.005, 0.001]
- Batch size = 64 $\in$ [32, 64, 128]
- Dropout

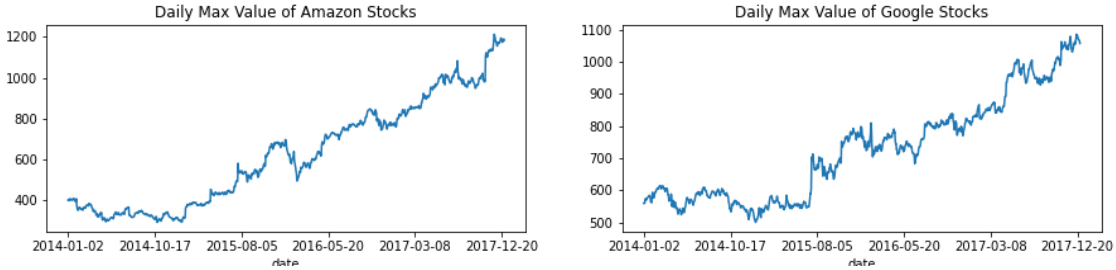
3.3 Reconstructing and predicting the SP500 index

In this collection there are 479 sequences with length of 1007 time points. The ratio between the length of the sequence and the number of sequences does not provide good training. Here, too, the length of the sequences made it difficult for the time and complexity of the training and in addition due to the small amount of the sequences, especially after splitting into training, validation and testing sets, the network quickly reached over-fitting for the training set.

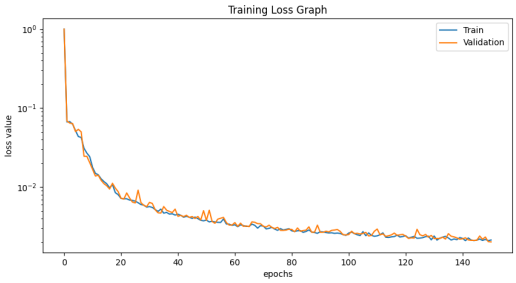
Because of this, I defined a function that builds a M-sized set of N-sized sequences (user-select parameters), the function randomly selects a M-times sequence from the original collection and takes from it N time points from a random start index (I asked the lecturer and he confirmed the cut method). For the training, test,

and validation sets, the method prepares a set from a separate set of 1007 original sequences. When the original groups are foreign.

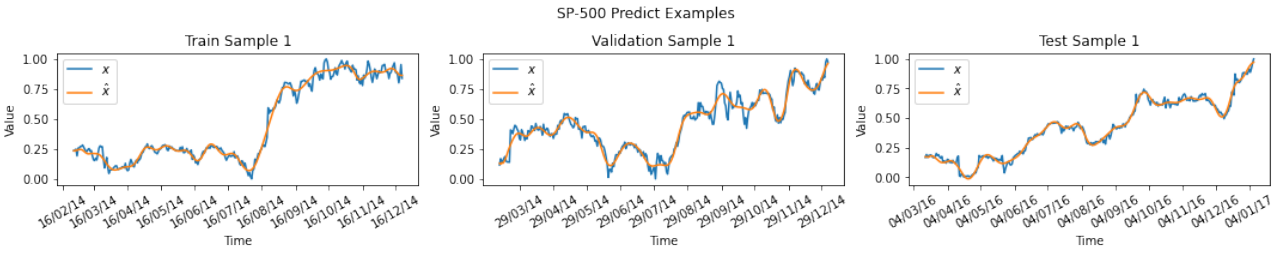
1. Graphs of the daily max value for the stocks AMZN and GOOGL:



2. Reconstructing the stocks' prices - as described in the introduction to this section, I have prepared a new data-set with a size of 3,000 sequences with length of 300 time points (M=3,000 and N=300). Each sequence is normalized and centered. Using cross-validation I reached the following results. The training graph (a) and three reconstruction results (b):



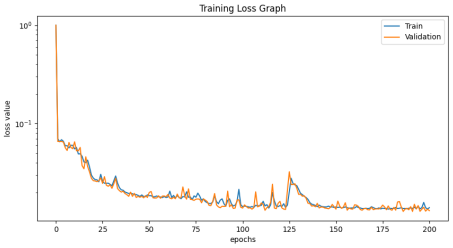
(a)

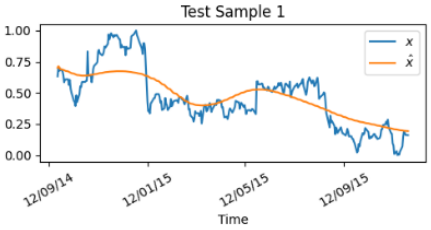


(b)

The hyper-parameters I chose using cross-validation:

- Hidden state size = 128 $\in$ [64,128,256,512]
- Learning rate = 0.005 $\in$ [0.05, 0.01, 0.005, 0.001]
  For a larger step I got a convergence to a larger error, for example (lr=0.01):





- Batch size = 64 ∈ [32, 64, 128]

  For this question, the batch size hyper-parameter did not affect on the success or the convergence and I obtained quite similar results for the different trials.

3. Predicting the stocks' prices - in the modified neural network the encoder remains the same and another decoder has been added whose function is to extract the next time point for each t.

   Model input:

$$x_t \quad \text{s.t.} \quad t \in [1, T-1]$$

   Model output:

$$[\hat{x_t}, \hat{x_{t+1}}] \quad \text{s.t.} \quad t \in [1, T-1]$$

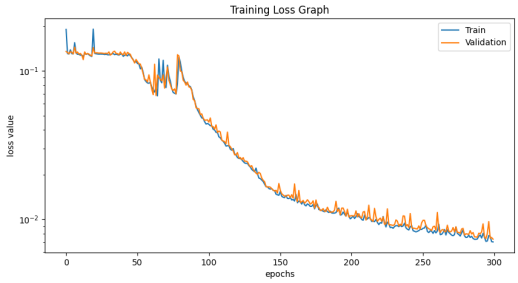
$$\hat{x_t} = \sigma(U_1(\hat{h_t} = Decoder1(z = Encoder(x_t))))$$

$$\hat{x_{t+1}} = \sigma(U_2(\hat{h_t} = Decoder2(z = Encoder(x_t))))$$

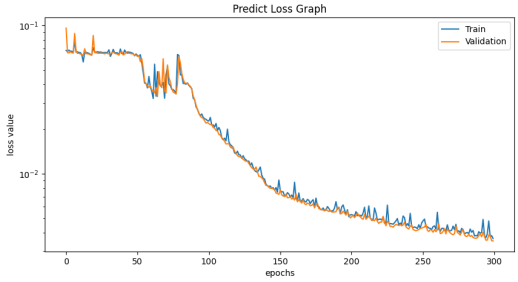
   Model loss:

$$l_{MSE} = \left[\|\hat{x_t} - x_t\|_2^2, \|\hat{x_{t+1}} - x_{t+1}\|_2^2 \right]$$

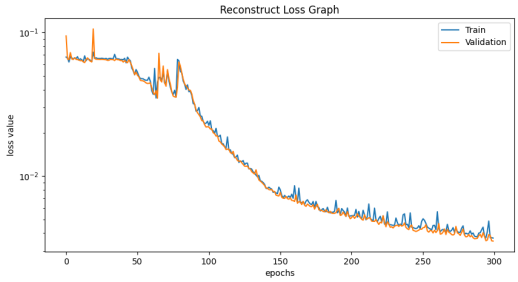
   I assume that the requirement is to attach graphs of the loss vs. epochs. Graphs of training loss vs. epochs (a), prediction loss vs. epochs (b) and reconstruction loss vs. epochs (c):



(a) full loss



(b) prediction loss



(c) reconstruction loss

It can be seen that the two tasks were similarly minimized and therefore no regularization or adjustment of parameters was required.

Similar to the previous sections, I checked different values for the hidden layer size and the learning rate values and saw how they affect the success of the model.