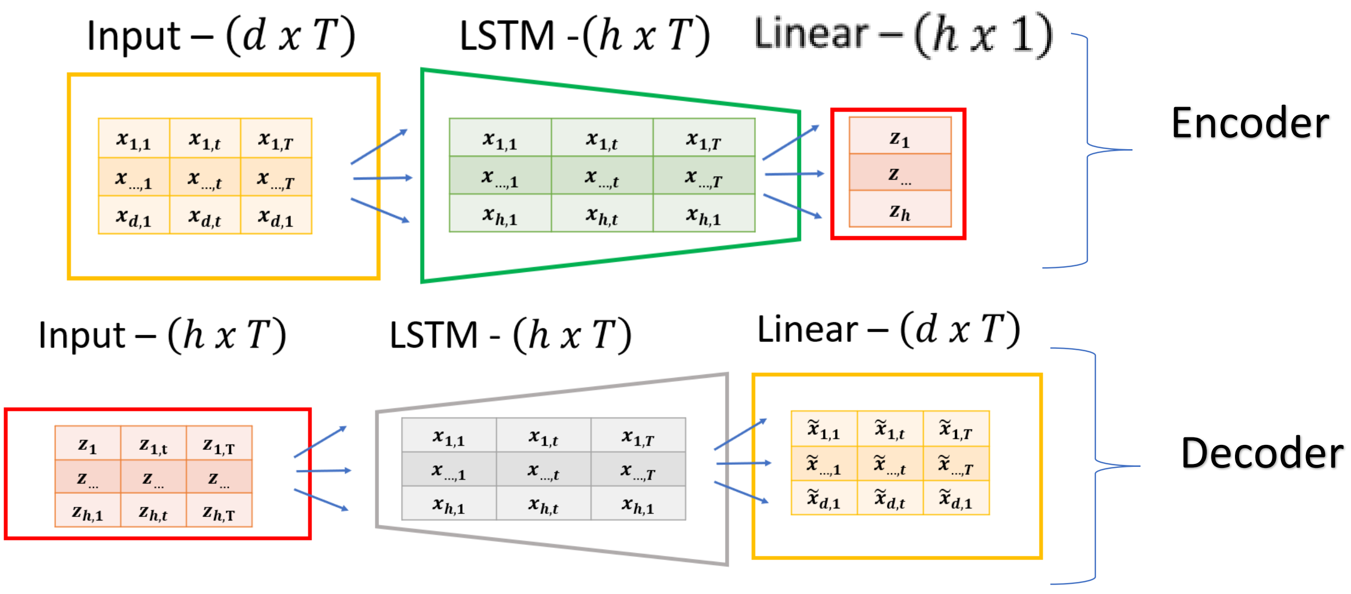
HW 2, Deep Learning - report

Liran Nochumsohn – 204693410

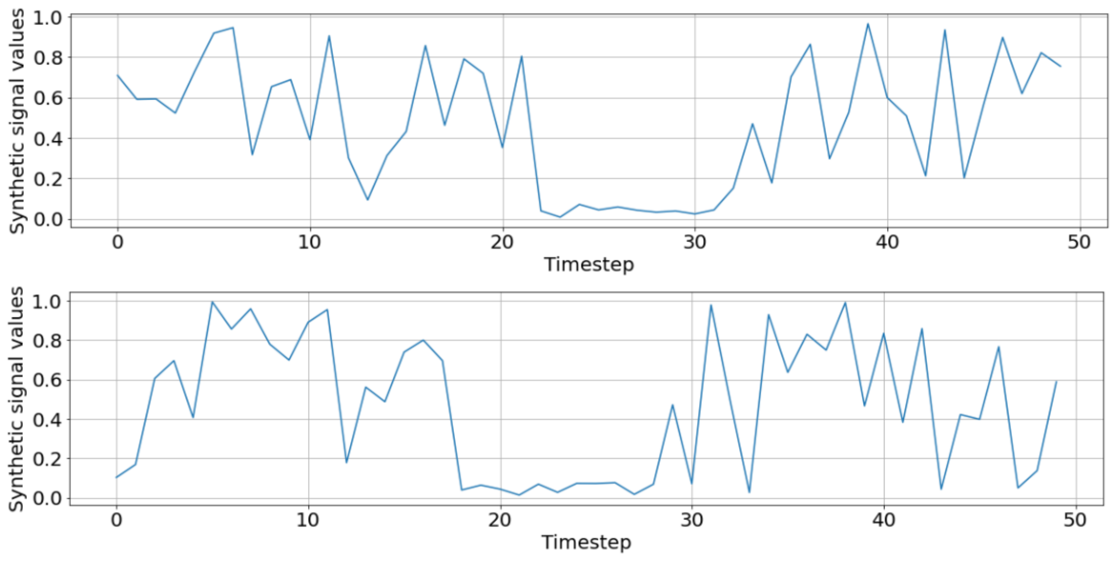
Liel Leman – 315929372

3.1.1

In this section we implemented an LSTM based Auto-Encoder. The AE is comprised of a single LSTM layer followed by a fully-connected layer connected the hidden output of the last timestep referred to as the encoder and another LSTM layer followed by a linear layer referred to as the decoder. The following image describes a simplified version of the architecture used in this project. h - hidden size, d - input size (dimension) and T-timesteps.


Input – $(d\ x\ T)$
LSTM -$(h\ x\ T)$
Linear – $(h\ x\ 1)$
Encoder
$x_{1,1}$ $x_{1,t}$ $x_{1,T}$ $x_{...,1}$ $x_{...,t}$ $x_{...,T}$ $x_{d,1}$ $x_{d,t}$ $x_{d,1}$
$x_{1,1}$ $x_{1,t}$ $x_{1,T}$ $x_{...,1}$ $x_{...,t}$ $x_{...,T}$ $x_{h,1}$ $x_{h,t}$ $x_{h,1}$
z_1 $z_{...}$ z_h
Input – $(h\ x\ T)$
LSTM - $(h\ x\ T)$
Linear – $(d\ x\ T)$
Decoder
z_1 $z_{1,t}$ $z_{1,T}$ $z_{...}$ $z_{...}$ $z_{...}$ $z_{h,1}$ $z_{h,t}$ $z_{h,T}$
$x_{1,1}$ $x_{1,t}$ $x_{1,T}$ $x_{...,1}$ $x_{...,t}$ $x_{...,T}$ $x_{h,1}$ $x_{h,t}$ $x_{h,1}$
$\tilde{x}_{1,1}$ $\tilde{x}_{1,t}$ $\tilde{x}_{1,T}$ $\tilde{x}_{...,1}$ $\tilde{x}_{...,t}$ $\tilde{x}_{...,T}$ $\tilde{x}_{d,1}$ $\tilde{x}_{d,t}$ $\tilde{x}_{d,1}$

Examples of the generated synthetic data:



3.1.2

The given Table presents the hyper-parameters and their corresponding validation loss. The best performing parameter combination is marked with the blue rectangle encircle and is used for reconstructing the images in the next section.

The parameter variables:

Optimizer: [adam, SGD]

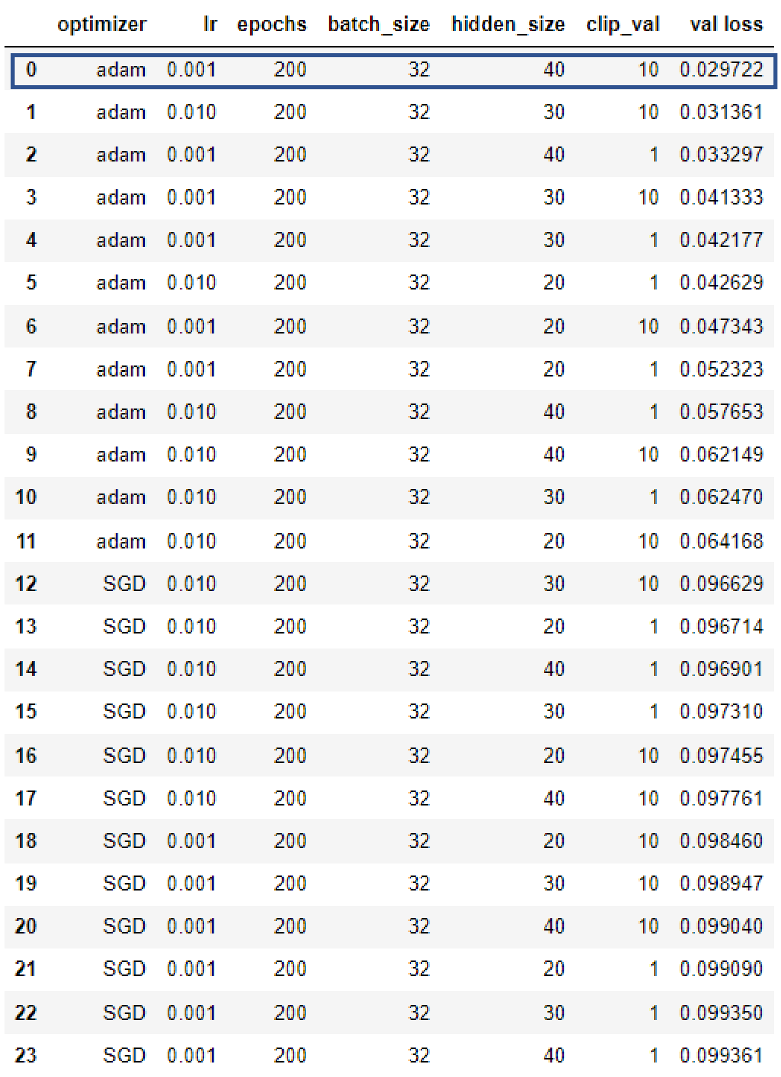
Learning rate: [0.01,0.001]

Epochs: 200

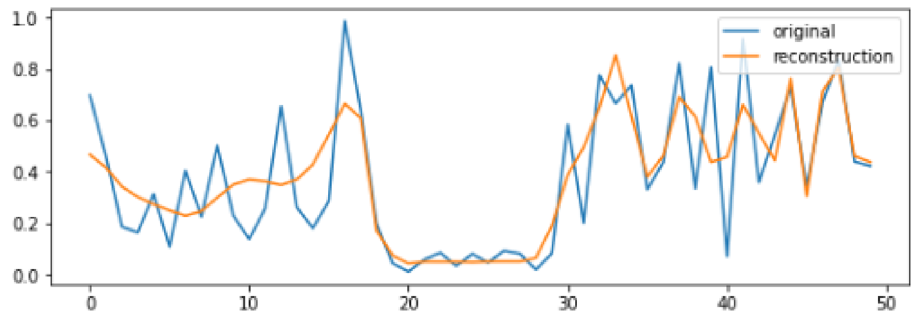
Batch size: 32

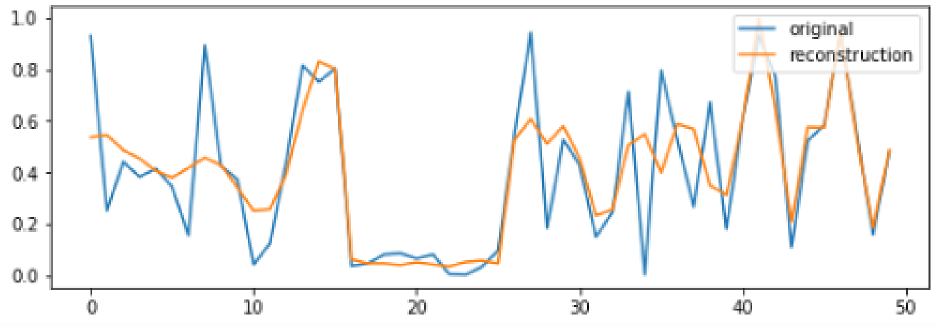
Hidden size: [20,30,40]

Clipping value: [1,10]



	optimizer	lr	epochs	batch_size	hidden_size	clip_val	val loss
0	adam	0.001	200	32	40	10	0.029722
1	adam	0.010	200	32	30	10	0.031361
2	adam	0.001	200	32	40	1	0.033297
3	adam	0.001	200	32	30	10	0.041333
4	adam	0.001	200	32	30	1	0.042177
5	adam	0.010	200	32	20	1	0.042629
6	adam	0.001	200	32	20	10	0.047343
7	adam	0.001	200	32	20	1	0.052323
8	adam	0.010	200	32	40	1	0.057653
9	adam	0.010	200	32	40	10	0.062149
10	adam	0.010	200	32	30	1	0.062470
11	adam	0.010	200	32	20	10	0.064168
12	SGD	0.010	200	32	30	10	0.096629
13	SGD	0.010	200	32	20	1	0.096714
14	SGD	0.010	200	32	40	1	0.096901
15	SGD	0.010	200	32	30	1	0.097310
16	SGD	0.010	200	32	20	10	0.097455
17	SGD	0.010	200	32	40	10	0.097761
18	SGD	0.001	200	32	20	10	0.098460
19	SGD	0.001	200	32	30	10	0.098947
20	SGD	0.001	200	32	40	10	0.099040
21	SGD	0.001	200	32	20	1	0.099090
22	SGD	0.001	200	32	30	1	0.099350
23	SGD	0.001	200	32	40	1	0.099361

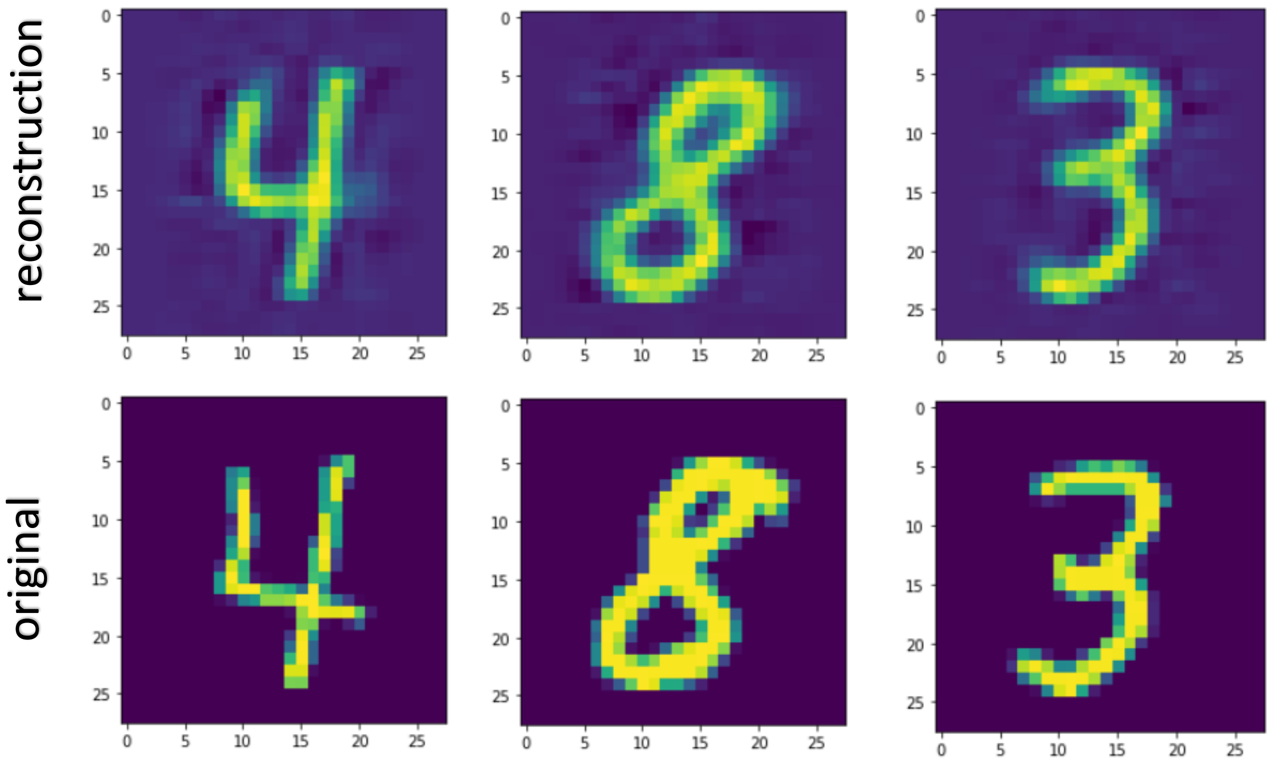




3.2.1

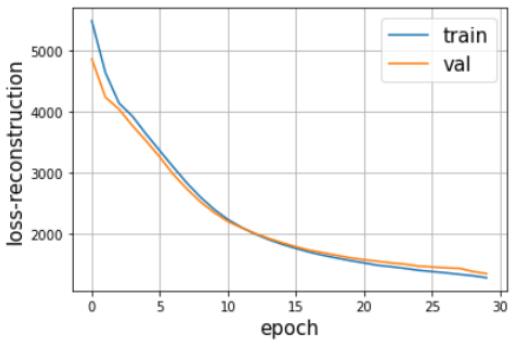
In this section we use the same architecture described above with hyper parameter variations. The following images are reconstructions from the test set.

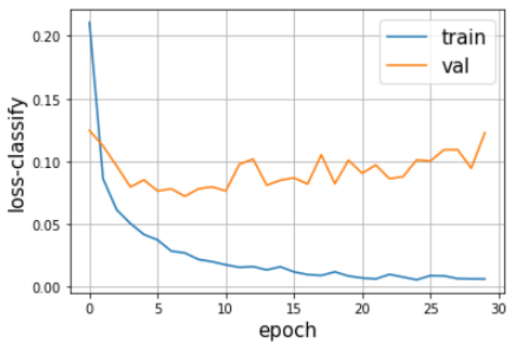
Hyper-params: epochs – 30, lr – 0.001, hidden size – 200, clipping – 1000, optimizer – adam, batch size - 64.

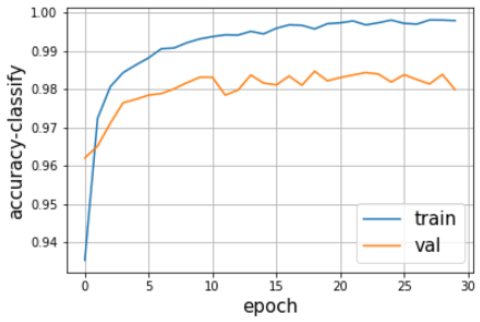


3.2.2

To apply a classification task, we add an additionally fully connected linear layer that outputs the 10 possible class which will then be plugged in the cross-entropy function to return 10 probabilities for each label. Therefore, we add another criterion function – cross entropy. The loss from both the reconstruction and classification will be adjusted (penalized) with the parameter lambda1 = 1/1000 and lambda2 =1 respectively.



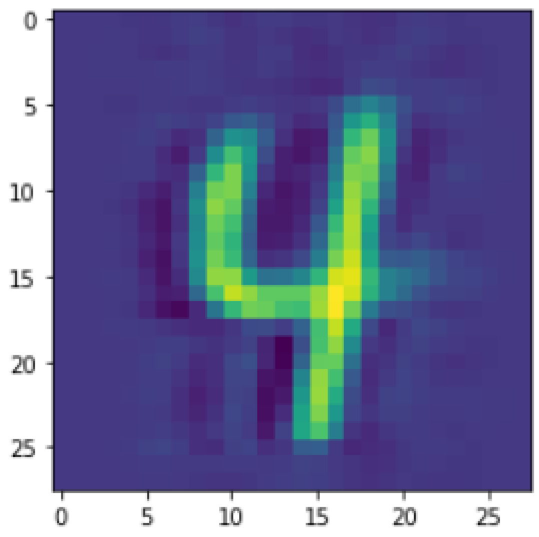




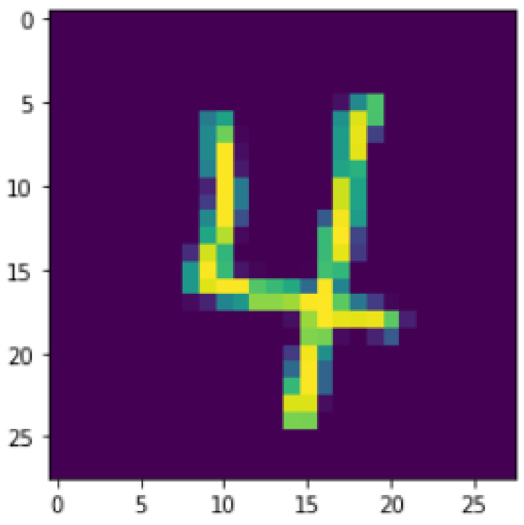
Test accuracy (from print):

```
tensor(0.9837, device='cuda:0')
```

Reconstruction:

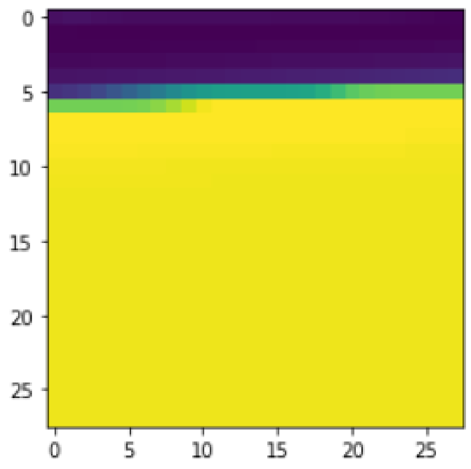


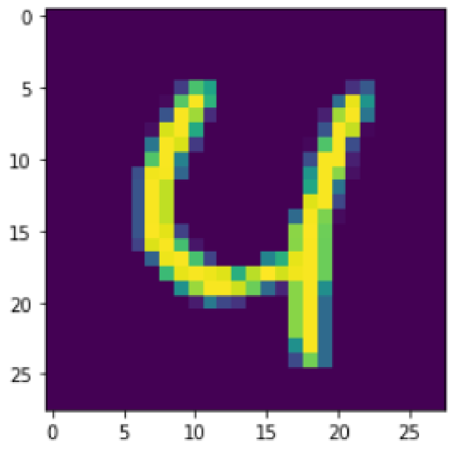
Original:



3.2.3

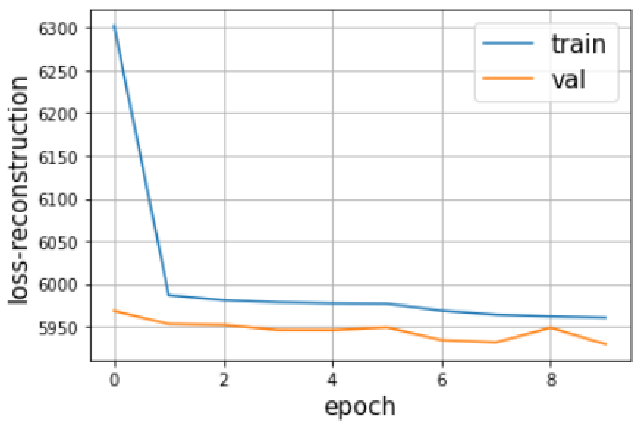
We repeated the same with a flattened version of the data – (batch = 16, T = 784, d = 1). Hyper-params: epochs – 10, lr – 0.001, hidden size – 200, clipping – 1000, optimizer – adam, batch size - 64, lambda1 = 1/1000, lambda2 = 1. As seen below the reconstruction is not defined, perhaps it requires many more epochs for training.

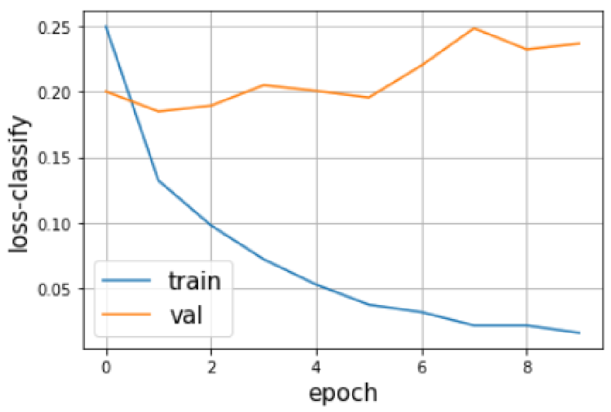


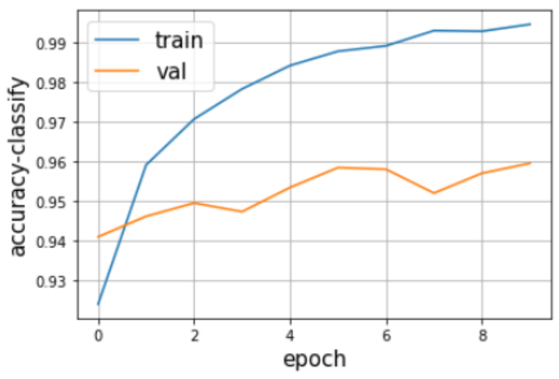


The test accuracy:

```
test acc 0.9633333086967468
```

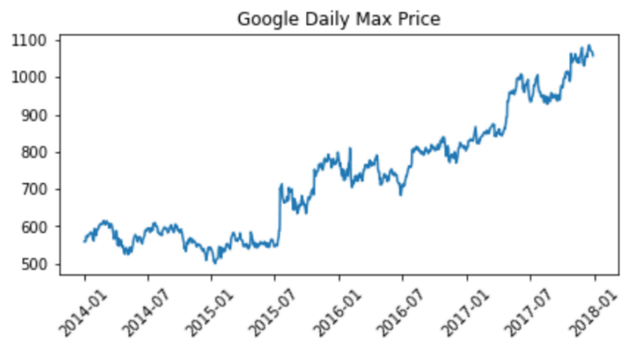

loss-reconstruction
6300
6250
6200
6150
6100
6050
6000
5950
0
2
4
6
8
epoch
train
val

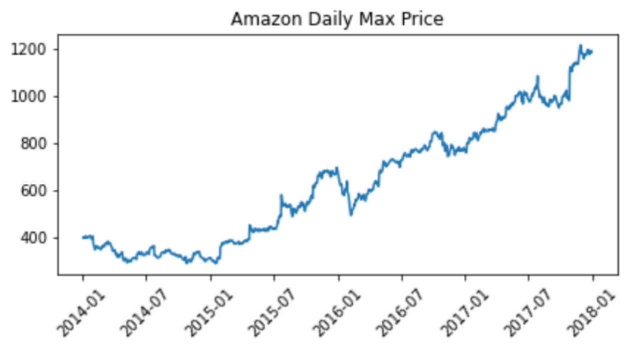

loss-classify
0.25
0.20
0.15
0.10
0.05
0
2
4
6
8
epoch
train
val


accuracy-classify
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0
2
4
6
8
epoch
train
val

3.3.1

In this section we used a similar architecture to reconstruct time-series data with a slight modification to the forecasting element, in the form of the number of hidden state and the output layers for the forecasting task. The following images show the plain daily highs of the Google and Amazon stock between 01.2014 – 01.2018.



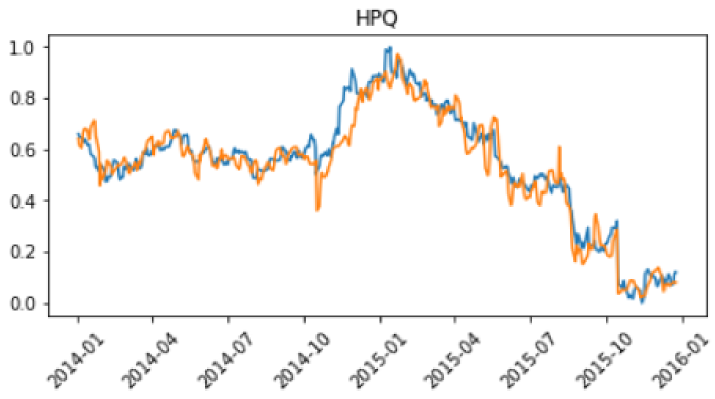


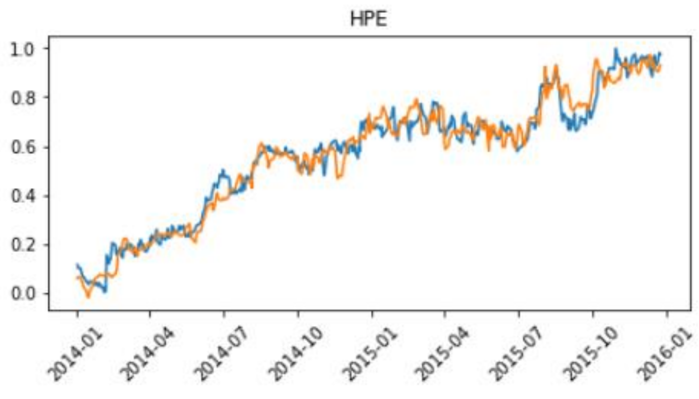
3.3.2

Here we are asked to reconstruct stock prices with the Auto-Encoder, due to computational limitations we learn a subset of the stock as well as only half the time span. Each stock is divided into lengths of 50 timesteps which will be learnt and then reconstructed.

Hyper-params: epochs – 5000, lr – 0.0025, hidden size – 100, clipping – 100, optimizer – adam, batch size - 32.

Here are scaled reconstructions of the stocks HPQ and HPE from the test set.





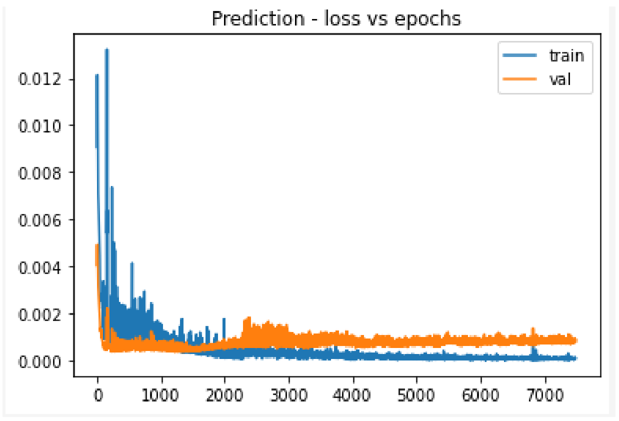
3.3.3

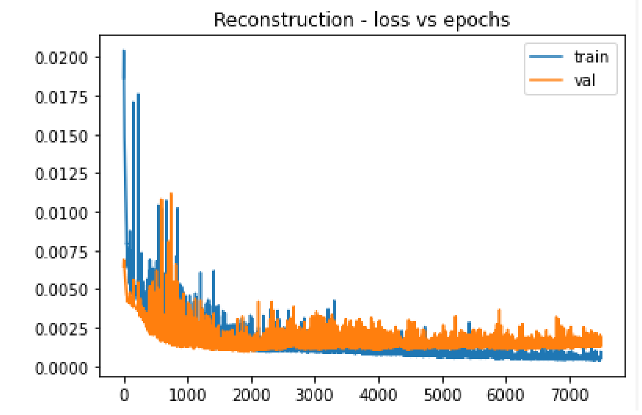
In this section we attempted to forecast the HP stock in the required manner. Because our task is forecasting, we penalized the reconstruction loss with a greater magnitude. We preprocessed the data such that the whole length is divided to overlapping examples of length T=20 with the task to predict X_(T+1). All exampled are scaled with sklearn-minmax scaler.

For the prediction we used the following hyper – params:

Hyper-params: epochs – 7500, lr – 0.00025, hidden size – 100, clipping – 1, optimizer – adam, batch size - 16. Lambda1 (reconstruction) – 1/100, lambda2 (forecast) – 1.

Loss plots:

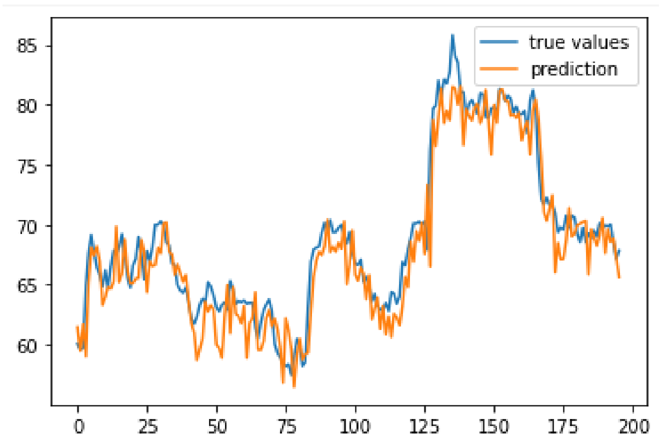


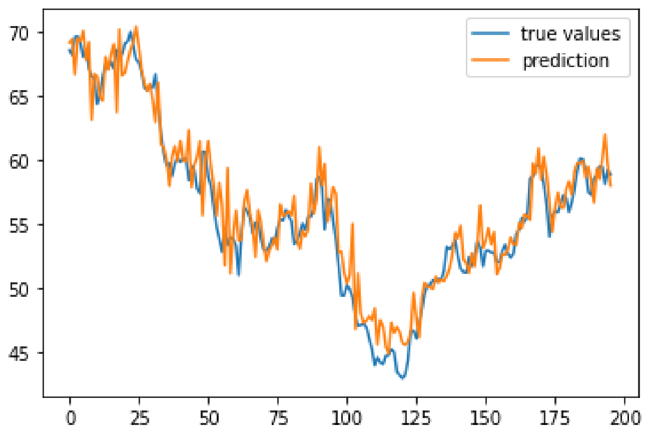


1 step prediction – HP stock:

Validation - forecast

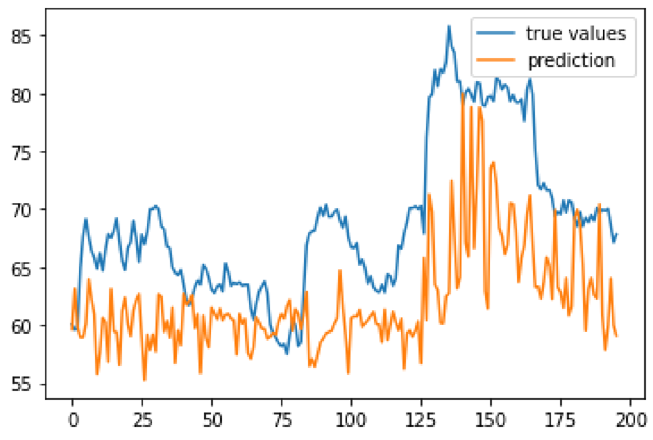
Test - forecast

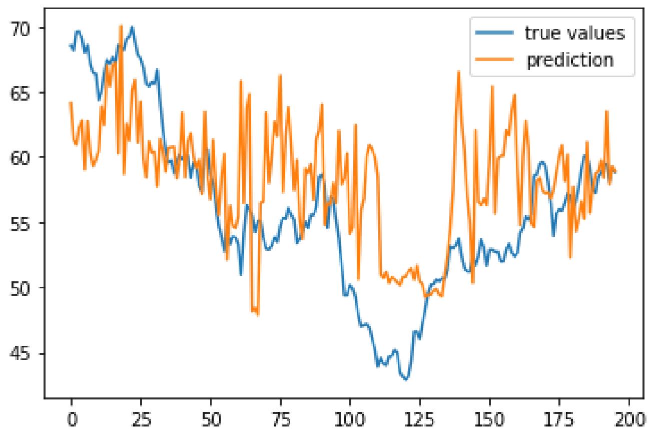




<u>3.3.4</u>

Multistep prediction with the same model as in **<u>3.3.3.</u>** Here we predict X_(T+1) in an auto regressive manner such that the latter of the series of length T is predicted using the former outputs until X_(T+1) is predicted. Here is a plot of the validation and test sets.





From observing the plots of both the 1-step and multistep forecast we find that the 1-step is more accurate with less noise, whereas the multistep forecast in the test set fails to achieve a similar result. This phenomenon can be explained by the auto-regressive nature of the second task. The first part of the series is a true value whereas the second is an approximation and finally the output X_(T+1) = y is the output given the approximated data. Perhaps lower the fraction of the series X_(T/2 + 1) , X_(T/2 + 2).. X(T) to less than half will most likely return better results closer to the ones we observed in **3.3.3.**