Artificial Neural Network and Deep Learning
Homework 2

Bita Rahmat Zadeh-10758773
Héloïse Viossat-10847709
Gabriele Bruni-10804805

January 22, 2022

1 Introduction

Aim of the homework is to forecast future samples of a multivariate time series. We briefly introduce the dataset and discuss some statistics about it. Next we present the performed work. Instead of presenting the work subdivided in Model, Experimnet and Result as for the last homework, we decided to illustrate directly the work developed subdivided by notebook. This is because this homework required for all of us a more trial and error approach rather than a systematic one as in the first homework.

2 Dataset

Our dataset is composed of 68528 timesteps and 7 feature.

3 Model, Experiments and Result

We tried several different model for this homework. We wil try to organize the work done in this report as best as possible. Two approach were adopted:

1. trying to predict the entire target sequence entirely (all the 864 timesteps in a single shot)
2. proceding with an autoregression approach

We produce 5 different notebooks were we adopt one or both the approach. For better organize this report, we will explain directly the work performed on the notebooks rather that explaining separatley the models, the experiment conducted and the results obtained. We will report only the root mean squared error metric as is the one utilized in the competition.

3.1 Notebook 1

In Notebook 1 we follow the basic Keras tutorial on time series forecasting (https://www.tensorflow.org/tutorials/structured_data/time_series). We use the sliding window methodology for preparing the training set. The last 2000 timesteps were used to predict the next 864 timesteps. 20 % of the dataset was preserved for the validation set. Samples were previously normalized. Alle the result refer to the score of the model on the validation set and with normalized samples.

We first defined two simple baseline model. In the first we predict the future 864 timesteps using the last timestep available in the window (costant prediction). in the second we use the last 864 samples available in the window and projected as forecast of our future samples. We obtain respectively an error (with normalized samples) of

```
baseline1: 0.16906396194467568
baseline2: 0.12163786391307785
```

We thus select baseline 2 as standard for comparing all the other model.
Then we try the single-shot approch. We train, in order, a single layer neural network, a multilayer percepetron, a CNN+dense network and a LSTM+dense network. In CNN the number of output mask was imposted to 256. In LSTM the number of unit for the hidden state was imposted to 32. The result obtainde are below:

```
one layer neural network: 0.12518413958036348
two layer neural network: 0.12057452248835265
CNN+dense network: 0.11180651508543824
LSTM+dense network: 0.10247052490690034
```

CNN + dense network and LSTM+dense network clearly beats the baseline. The two layer dense network did only slightly better.

Following the keras tutorial, we try also to built a autoregressive model. Predictions were made using only the last predicted step here. The rmse was of

```
autoregressive: 0.13514594992490542
```

but looking at the prediction, the model was learnig only the average of the time series after few timestep.

3.2 Notebook 2

In notebook 2, we try to tune the LSTM model who performed best increasing the number of hidden units progressively from 32 to 64,128,256 and 512. We also try to stack more LSTM layer. Unfortunately, these changes does not decrease the error score. By way of example, we report the rmse of a model obtained stacking two LSTM layer having both 32 hidden units

```
2 LSTM+dense network: 0.10443024715742501
```

We try also to perform some other tuning, such decrease or increase the window length from which predicted but not help.

3.3 Notebook 3

In notebook 3, we look at the varius autocorrelation and lag properties of the multivariate time series, using the panda library. The autocorrelation plot of "crunchness" and "hype root" features shows how these two sequences are the more "chaotic" ones. As for the lag plot, also "sponginess" showed a strange behaviour.

Despite all, the majority of the feature seemed to show good autocorrelation and lag properties. Based on that, we decide to add to the dataset, for each of the feature, a "diff" feature, calulated using the homonymous function of panda data frame. The function simply make difference between two adjacent timesteps. The model utiled is a single LSTM+dense network like the one in notebook 2. The only difference is in the number of neuron, increased to 64. The result is reported below:

```
LSTM+dense network (feature engineering): 0.1045183599384125762
```

Adding more feature does not help with our problem and this model.

3.4 Notebook 4

At this point it looked like the simple LSTM model's capabilities were saturated. We decide thus to explore other model and hypothesis space. The first model we tried in notebook was an already simple bidirectional LSTM + dense layer. The result

```
bidirectional LSTM: 0.1064467033351123789
```

does not beat the LSTM model of the notebook 2.

We then decide to implement an endoer-decoder sequence to sequence model. We follow the guide at https://www.analyticsvidhya.com/blog/2020/10/multivariate-multi-step-time-series-forecasti
We implement first a sequence to sequence Model with one encoder layer and one decoder layer. Then we try incresing the number of layer in both decoder and encoder to 2. The number of hidden unit was mantained costant to 32. Here the result:

```
1 Encoder 1 decoder: 0.1284455465469848748
2 Encoder 2 decoder: 0.1292124594739576946
```

Even these two model doen't seems to beat the simpler lstm model of notebook 2.

We try also the model introduced in lanoratory lesson in this notebook, without luck.

As ultimate attempt, in this notebook we also try to implement two mixed encoder-decoder architectures composed of CNN and LSTM module, mostly based on work found at https://towardsdatascience.com/cnn-lstm-based-models-for-multiple-parallel-input-and-multi-step-forecast-
The model required too much time to be trained and we decide to don't investigate further.

3.5 Notebook 5

All the model seen to far have been trained trying to predict the entire 864 sequence in one shot. Now is arrived the time to using the autoregressive approach seen in lesson.
Here, we utilize 288 samples to predict the next 48 time-steps, and then we iterate in an autoregressive fashion. The first layer we tried was a 3 layer LSTM network, with respectively 128,64 and 32 hidden unit. The result reported the score for the single shot 48 time samples prediction and the autoregressive prediction for the entire 864 sequence:

```
3 LSTM autoregressive 48 time steps: 0.09102590936669086
3 LSTM autoregressive 864 time steps: 0.11023311249965034
```

Then we build an encdoer decoder model, 1 layer with 128 hidden units for both the encoder and the decoder,we a batch normalization layer between them. The model obtain a score of

```
encoder-decoder autoregressive 48 time steps: 0.08975138269889467
encoder-decoder autoregressive 864 time steps: 0.11393476120225136
```

While perfomring better on single shoit prediction than the previous one, the result seems to be worse on the autoregressive prediction. Our last attempt was to implemnt some attention mechanism in our model. We follow the guide found at https://levelup.gitconnected.com/building-seq2seq-lstm-with-luong-attention-in-keras-for-time-series-forecasting-1ee00958decb to implement the Luong attention on previous encoder-decoder model. The result is:

```
encoder-decoder autoregressive luong 48 time steps: 0.08745158795503243
encoder-decoder autoregressive luong 864 time steps: 0.10484014492118862
```

While the score are better than previous model in the notebook, also this model does not beat the simple LSTM foun on notebook 2

4 Conclusion

We tried serveral model for this work, using all the resources explained during the lesson. Unfortunately we have not been able to improve over the simple LSTM model. For this reason, this is the only model that we have upload on codalab competiton, obtain a rmse score of 4.9678. Probably the work was a bit disorganized in the initial stages, and not all the time was spent in the most efficient way, trying to optimize models that were seen to be unsuccessful even immediately after a quick test. At the time of writing this report, the one thing we can think of to do that we haven't tried, (other than trying to train more appropriate models), is the use of a single model for every feature time series. Furthermore, probably looking for an architecture that was capable of exploiting the spatial correlation between the various time series would have helped. In spite of everything we are satisfied with the work done, and we look to improve in the future.