POLITECNICO
DI MILANO

AN2DL: Challenge 2

Authors:
Gabriele De Santis - 10820992
Giulia Maria Sarcina- 10581803
Michela Capasso - 10786184

November 2021

1 Preparation

The aim of this challenge is to predict future samples of a multivariate series.
First of all, we prepared the dataset in order to train the model over it. We divided the dataset into training set, which is the 90% of the entire set, and validation set, which is the 10%. At the beginning, we used a window of 200 and a stride of 10. During the challenge, we fine-tuned these two parameters.

2 Models

The firsts models we made are all autoregressive models. We tried the most of the models both with autoregression and without. Before each model, we will state whether it has been trained with autoregression, without or both.
The first two models we made were:

```
Trained only with autoregression
Model1-a: lstm(64) + drop(0.5) + lstm(128) + gap + drop(0.5)

Model1-b: bidir(lstm(64)) + drop(0.5) + bidir(lstm(128)) + gap + drop(0.5)
```

We decided to start with these very a simple models in order to evaluate if bidirectionality would be a good solution or not for this time series forecasting.
As expected both validation and test results of the model without bidirectionality were better than the bidirectional ones. So we decided to continue with the simplest one which gave us a RMSE of 5.059 on Codalab.

```
Trained only with autoregression
Model2: lstm(64) + drop(0.1) + lstm(128) + drop(0.1)
```

After that, we decided to lower the dropout. Indeed, our concern, was that this heavy dropout could lead in erasing important context information.

Locally, we had a slightly improvement. Once submitted, we obtained a RMSE of 4.3563, which is way better than the RMSE of the Model1-a.

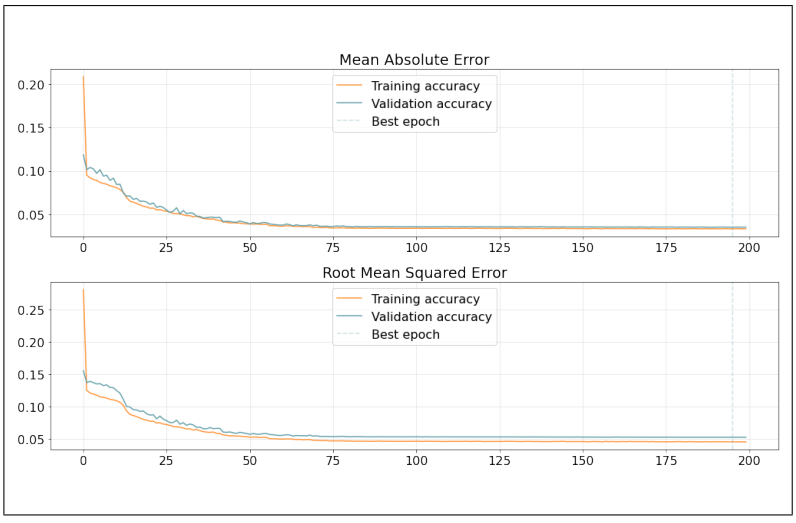

Figure 1: MAE and RMSE accuracy of model2

Observing the MAE and the RMSE of the model2, we decided trying to lower even more the overfitting, which however seems to be very slight.
Because of that we decided to try first the *recurrent dropout* and then some layers's regularizer.
About the recurrent dropout, we thought to apply the dropout directly to the recurrent input signal on the LSTM units in order to improve the robustness of the model. Our new model was:

```
Trained with and without autoregression
Model3: lstm(64, rec_drop = 0.2) + drop(0.1) +
            lstm(128, rec_drop = 0.2) + gap + drop(0.1)
```

With this model, with autoregression, we reached a RMSE of 4.0342 and because of that we decided to keep both the dropout and the recurrent_dropout. Without autoregression, the RMSE reached on Codalab was of 4.01.

Going on, we tried to regularize the layer's weight and the recurrent connections on each LSTM unit by using respectively the *kernel regularizer* (for the sake of clarity, it'll be called *kr* in the model description) and the *recurrent regularizer* (which will be called *rk*). At first, we tried to regularize by using the L2 class with a value of 10^{-4}:

```
Trained with and without autoregression
Model4: lstm(64, rec_drop = 0.2, kr=l2(1e-4), rk = l2(1e-4)) + drop(0.1) +
        lstm(128, rec_drop = 0.2, kr=l2(1e-4), rk = l2(1e-4)) + gap + drop(0.1)
```

We soon noticed that the MAE and the RMSE accuracy were worst than before, as the graphs show. Indeed, on Codalab, with autoregression, it gave us 4.14. We tried to fine-tune the values of the regularizers and we even tried to use the L1 class, but the results were always similar or worst. Because of that, we decided to not use the layer's regularizers.

We tried also different approaches, but we did not present them as models since they didn't gave us great results. These approaches were:

- *GRU layers*, used insted of the LSTM layers. They were efficient in the training, but bad in the prediction;
- *LayerNormalization*, to normalize the activations of the previous layer. it gave slightly worst results;
- *Conv1D*, used in combination with LSTM, it gave better results locally but worst results on Codalab;
- *Stacked LSTM*, it gave worst results and it was totally inefficient since it required too much time to train.

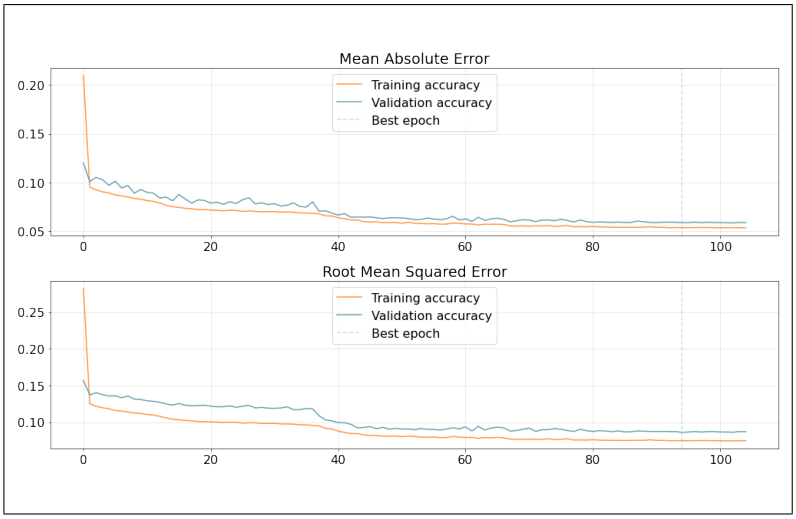

Figure 2: MAE and RMSE accuracy of model4

In order to improve our model we have tried to get better results by developing an encoder-decoder model adding the attention mechanism. We implemented it without using the keras Attention layer. The model summary was:

Layer (type)	Output Shape	Param #	Connected to
Input (InputLayer)	[(None, None, 7)]	0	
lstm (LSTM)	[(None, None, 64), (	18432	Input[0][0]
batch_normalization (BatchNorma	(None, 64)	256	lstm[0][1]
repeat_vector (RepeatVector)	(None, 864, 64)	0	batch_normalization[0][0]
batch_normalization_1 (BatchNor	(None, 64)	256	lstm[0][2]
lstm_1 (LSTM)	(None, 864, 64)	33024	repeat_vector[0][0] batch_normalization[0][0] batch_normalization_1[0][0]
dot (Dot)	(None, 864, None)	0	lstm_1[0][0] lstm[0][0]
activation (Activation)	(None, 864, None)	0	dot[0][0]
dot_1 (Dot)	(None, 864, 64)	0	activation[0][0] lstm[0][0]
batch_normalization_2 (BatchNor	(None, 864, 64)	256	dot_1[0][0]
concatenate (Concatenate)	(None, 864, 128)	0	batch_normalization_2[0][0] lstm_1[0][0]
time_distributed (TimeDistribut	(None, 864, 7)	903	concatenate[0][0]

Total params: 53,127
Trainable params: 52,743
Non-trainable params: 384

After that we used this basic model to obtain a stacked version of the model repeating the block for 4 times. However neither the base nor the stacked model obtained good results on the test set when trained with autoregression. For this reason we did not train it without autoregression.

3 Conclusion

At the end of our analysis we decided to present as the best model the model4 without autoregression. It gave us the smallest error on both the training and the test sets that was about 3.8, training it over all the set. Furthermore, we noticed that all the models without autoregression behaves better than the models trained with autoregression.