

Multivariate Timeseries Forecasting Challenge

Artificial Neural Networks and Deep Learning @PoliMi

Antonio Ercolani
Francesco Romanò

A.Y. 2021/2022

1 Validation and testing

During all the phases of this challenge we have followed the following steps to make local evaluation of our models before trying submissions on the challenge test data. First of all we've monitored the validation loss (MSE metric used) during the training of the model, using a hold-out validation approach (we have considered a 90-10 split, 90% used for training, 10% for validation) and Early Stopping to avoid overfitting. This type of evaluation has revealed to be somehow poor since it was really noisy and often in contradiction with respect both to our local and the challenge test set. An other technique used was looking at our local test set loss. This approach seemed a little bit more reliable, but also this appeared noisy if models performance was similar. We also noticed that repeating the training of the exact model could bring to different validation metric also using seed for reproducibility, probably due to the intrinsic non-determinism of the GPU.

Our solution: In order to have a less noisy local evaluation we have then used 10-fold cross validation of our models. This approach gave us evaluations less correlated with the hold out split and so more stable. In particular we have fixed 30 epochs of training for each iteration of cross validation and we monitored the MSE once again (taking the average MSE over the 10 trained models). Our actual best model gave the best average score also with cross-validation and also other comparisons seems coherent with results on the challenge test set. The code used can be found in the notebook.

A last very qualitative approach we have used was human inspection of the model predictions plots. We have directly look at the model predictions in order to understand what the model was trying to learn. Below we just show two plots of two models with a very different behaviour:

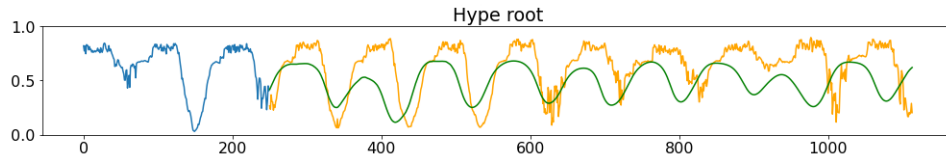


Figure 1: Prediction graph of an autoregressive model

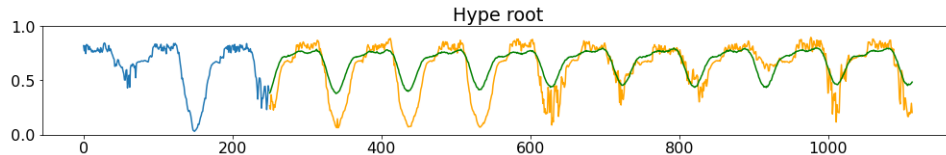


Figure 2: Prediction graph of our best model

2 Model Design

After the usual data exploration and sequence building we started approaching the forecasting problem with the following steps.

Direct vs Autoregressive forecasting At first we decided to focus our effort on one of these two approaches and after some trials with many networks we discovered that the Direct forecasting models ($MSE \sim 4.5$) perform better, on average, than the autoregressive ones ($MSE \sim 8$). We then went on exploring direct forecasting models, but we also compare some of them with their autoregressive counterpart (confirming the behaviour highlighted before).

Simple vs Complex models Then we thought to improve our predictions exploiting more complex architectures. We tried different combinations of networks with many layers, both recurrent and convolutional combined in different ways. The results of these first kind of models were very bad, the more we added complexity the more the models performed poorly, so we decided to continue with simple architectures. We improved performances both removing layers and decreasing the number of units of the LSTM layer up to a certain point. We instead notice that an intermediate dense layer before the final one increase a lot the performance. This layer adds a lot of parameters to the model. So our best model has a simple architecture (just one LSTM and two final Dense layers) but it is also pretty complex (around 12M parameters)

Recurrent vs convolutional models Since complex models with the combination of these two kind of layers brought poor result we decided to follow two ways: build models with just convolutional or recurrent layers.

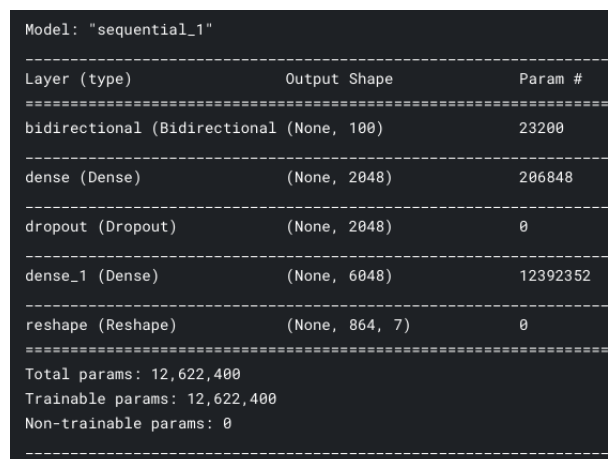
The convolutional approach is covered by a dedicated chapter.

Regarding the recurrent one, we tried different combinations of the LSTM and GRU layers, both bidirectional and not. After a lot of trials with a progressive performance improvement we came to our best model composed by a single Bidirectional LSTM layer with 50 units and one intermediate Dense layer of 2048 units before the final Dense one. This result agrees with what said in the previous paragraph: this task doesn't need a too complex model, and simpler ones perform better.

Dropout The usage of dropout has a relevant impact on the overall performances: it improves them significantly in particular after our intermediate Dense layer, this is line with what we expects since Dense layer are prone to overfitting and dropout helps reducing it. The dropout rate was handled in the tuning phase of the model.

Last layers To match the task domain we use in all our model a Dense layer with size $\#samples$ to predict(our telescope) * $\#variables$ (the 7 variables of the problem) followed by a Reshape layer which create an output with shape $[\#samples \text{ to predict}, \#variables]$. The activation function used in the last dense layer is the ReLu which correctly match the domain of the output prediction.

The final model obtained is the following:



```
Model: "sequential_1"
-----
Layer (type)                 Output Shape              Param #
-----
bidirectional (Bidirectional) (None, 100)               23200
-----
dense (Dense)                 (None, 2048)              206848
-----
dropout (Dropout)             (None, 2048)              0
-----
dense_1 (Dense)               (None, 6048)              12392352
-----
reshape (Reshape)             (None, 864, 7)            0
-----
Total params: 12,622,400
Trainable params: 12,622,400
Non-trainable params: 0
-----
```

Figure 3: Final model summary

3 Convolutional approach

After the exploration of recurrent models we also tried a pure convolutional approach, using Conv1D layers to extract the time series features. We noticed that results appeared to be good looking at prediction plots, but we faced the following problem: the predictions present some isolated samples with very large error as shown in the following image:

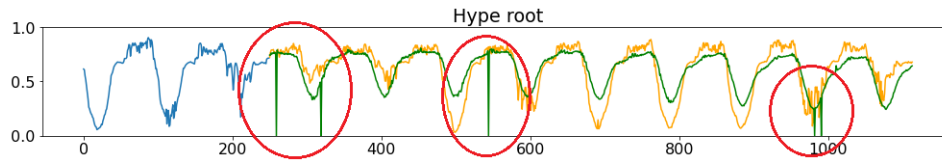


Figure 4: VGG model prediction graph with highlighted mispredicted samples

We initially thought that this problem was due to very high weights absolute value, but also using dropout or weight decay layers we had the same problem. So we tried to use a more deep network, hoping to perform a better feature extraction, a monodimensional VGG model designed for time series data was used. This model suffered the same problem: very high error on isolated samples, even more accentuated than before, while the general prediction plot appeared quite good. Since this anomalous peaks affected a lot the overall performances of the model we decide to not insist with this not-recurrent convolutional approach, thinking that it was probably not suitable to a forecasting task.

4 Tuning

The majority of our experiments were about parameters tuning. Every time we discovered a promising model we started an intensive tuning before going ahead with something new. We focused on the units of the dense and recurrent layers and on the percentage of their dropout, if provided.

Sometimes, when we noticed an "overfitting behaviour" as explained in the convolutional approach chapter, we applied a normalization to some layers and obviously this adds parameters to be tuned.

The other crucial point were the values of **Stride** and **Window**, key parameters in sequences building and therefore in the model training. At first we thought that a larger window and a small stride would have improved the predictions since in our mind the model would have "seen" better the patterns of the signals, but it turned out to be wrong. After many experiments on different networks we have came up to these values for our best model: Stride = 10 and Window = 250.

5 Summary

Finally the reader can find a table that summarizes the progression of the most significant models (we avoid insertion of lots of similar variants of same model) that we have trained and the relatives results on the CodaLab test set (development phase test set) and the 10-fold score (values can seem "strange" since it is the MSE computed with normalized data).

Model	Test RMSE on CodaLab	10-fold score
Best model with autoregression	9.76	-
VGG1d	7.92	0.01906
LSTM + dense + LSTM + Dense	4.49	0.00757
GRU + dense	4.02	0.00688
Best model: LSTM + Dense	3.85	0.00687