# AN2DL Challenge 2

Francesco Colabene, Andrea Gerosa, Christian Lisi, Giulio Saulle

## 0 Introduction

The project's goal was to design and implement forecasting models to learn how to exploit past observations in the input sequences, in order to correctly predict the future. The provided dataset's actual meaning was unknown. It consisted of a collection of independent monovariate time series, belonging to six different domains not to be understood as closely related to each other, but only as collected from similar data sources. In the following report, we will discuss how we dealt with the problem in 3 stages:

1. Data analysis and input preparation
2. Testing on a plethora of models
3. Hyperparams fine-tuning to elevate the performance of the best model found in step 2

## 1 Data analysis and input preparation

### Dataset esploration

The provided dataset contained 3 files:

- One with 48000 time series of different lengths, padded to the left with as many zero-values as needed to reach the length of 2776 values
- One containing the valid periods of each time series, represented by the first and the last points
- One containing a character ranging from *A* to *F*, representing the domain each series belongs to

The longest time series had indeed a length of 2776, while the shortest was 24. Further inspection of the time series length distribution revealed that ~99% of sequences in the dataset had a period below 650. All the values of the time series were already normalized to be in the interval [0,1].

### Cleanup and input preparation

Firstly, we conducted a check to see if there were multiple instances of the same time series in the dataset and deleted the duplicates.
We then moved to the sequence preparation task. Due to the nature of the data provided, we thought of two different strategies to tackle this issue.
The first idea was to simply delete all the time series with a length lower than window+telescope (where *telescope* is defined as the number of future samples to predict) or higher than 700, in order to speed up training. We also added some padding back to every time series to avoid losing values at the end.
The second strategy aimed to keep all the unique time series: starting from the end of each time series, we selected the range of values belonging to the window and the value of the telescope. We kept doing so going backward in the time series until we encountered some padding, and then we stopped and moved our attention to the next time series. With this method, we were able to select only the useful values of the time series for learning the different trends, leaving out the possible window with only padding values that would have hurt the models' training.
That said, the best-performing model uses the first approach.

### Sets split and imbalance in the domains

When deciding how to divide the dataset into train, validation, and test sets, we quickly realized that the imbalance of the time series in the different domains could lead to training problems for our model. So, taking this problem into account, we performed a stratified split, maintaining invariant the distribution of the time series for each domain also in test and validation sets. This way we boosted our models' performances.

## 2 Implemented Models

### 2.1 LSTM

The first choice for our model was a standard LSTM, which is a common starting point for time series forecasting due to its native support for sequences. In our case, we also added some convolutional layers in order to squeeze out some more info from the dataset. The performance was okay, but being a starting point, we quickly moved on.

Firstly, we tried to add more complexity to the base model by concatenating various LSTM layers together and by adding bi-directionality to them. To our surprise, the performances were worse, and the model was more prone to overfitting.

### 2.2 ResNet

Following a hint given by the TA, we tried to adapt the ResNet model for the time series forecasting problem. We followed this architecture found on a paper online (link).

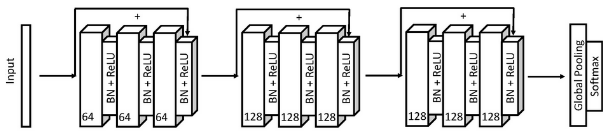


Figure 1: ResNet architecture for time series

The model is composed of 3 blocks, each of which is in turn composed of 3 sets of the following: a convolutional 1D layer, a batch normalization layer, and an activation (ReLU in this case). For the first block, the number of filters was set to 64 while for the following two, it was set to double.
After training, we found an improvement in performance over the basic LSTM model, but not by much. Thus, we continued to experiment with other models.

### 2.3 Encode-Decode LSTM

During our research, we stumbled across this new interesting model and put it to a test.
Encode-Decode LSTM model, different from the based LSTM model, will not output a vector sequence directly. Instead, the model consists of two sub-models: the encoder to read and encode the input sequence, and the decoder to read the encoded input sequence and make a one-step prediction for each element in the output sequence.
The difference is subtle, as in practice both approaches do in fact predict a sequence output.
The important difference is that an LSTM model is used in the decoder, allowing it to both know what was predicted for the prior data in the sequence and accumulate internal state while outputting the sequence.
The result is an improvement over the ResNet model.

### 2.4 GRU and Bi-GRU

Another model encountered during our research is the GRU model, which stands for *gated recurrent unit*. It's a gating mechanism in recurrent neural networks (RNN) similar to an LSTM unit. The main differences between them are the following:

- GRU has a simpler architecture with two gates (reset and update gates) against the three gates the LSTM
- GRU has fewer parameters, making it computationally less expensive and faster in training
- GRU can perform well on certain tasks and is sometimes preferred in scenarios where computational resources are limited.

In our research, we have observed that this kind of model performs usually a bit worse than LSTM, but due to the fact that the performance of these architectures can vary depending on the specific task and dataset, we put it to a test.
We found out that it performs better than every other model tested before, maybe due to the limited amount of data for the training, which is a known factor of better performance for GRU vs LSTM. To make the GRU model a little more robust to change, the dropout function has been used.

As a last attempt, we tried also to add the bi-directionality with the same pattern as the LSTM model, worsening a bit the performance on CodaLab. We took this last experiment as a confirmation of the fact that a simpler model seemed to perform much better on our dataset.

### 2.5 GRU Attention

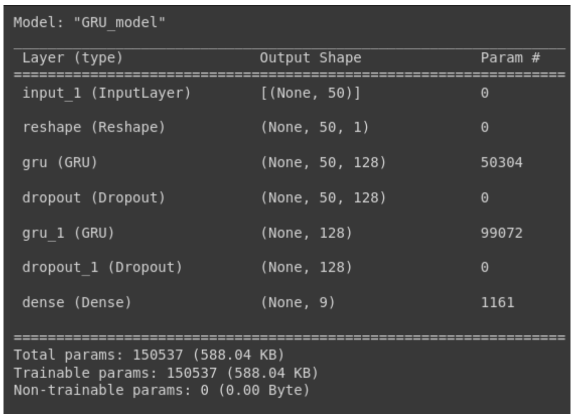
We experimented with the attention layer by adding it before the GRU decoding layer. Before doing so we placed another GRU encoding layer, concatenating it with the Attention layer output, keeping a feasible model to be trained (stretching the hyperparams has resulted in failure both in time and memory terms). The results were close to the best model, but the model did not perform well enough to surpass it, thus it hasn't been improved.

### 2.6 GRU Autoregression

We also tried a GRU model with autoregression: telescope was set to 1 and consecutive values were computed by adding the previous prediction to the input. As we expected, the MSE was much lower for a single prediction but it was also higher than the normal one when predicting 9 values in a row.

## 3 Final Model + Hyperparams tuning

With the previous result in mind, our final proposed model is the GRU model with the following architecture.



```
Model: "GRU_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 50)]              0

 reshape (Reshape)           (None, 50, 1)             0

 gru (GRU)                   (None, 50, 128)           50304

 dropout (Dropout)           (None, 50, 128)           0

 gru_1 (GRU)                 (None, 128)               99072

 dropout_1 (Dropout)         (None, 128)               0

 dense (Dense)               (None, 9)                 1161

=================================================================
Total params: 150537 (588.04 KB)
Trainable params: 150537 (588.04 KB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 2: Final GRU model summary

We tested the effects of various hyperparameters on overall performance, and we found that the sweet spot is the following:

- **Window** : 50
  We started with a window of 200 but after multiple training sessions done with lower values (ranging from 100 to 30), we experienced a better learning of the trends by the model with this specific value set as window. Since the test set had 200 values per time series, we only took the last 50.
- **Stride** : 5
- **GRU units** : 128
- **Dropout rate** : 0.2
- **Batch size** : 256
- **Epoch** : 30
- **Callbacks** : We implemented
  - **Early stopping**
    Patience : 6
  - **Reduce learning rate on plateau** :
    Patience : 4
    Factor : 0.1
    Minimum LR : 1e-5

## 4 Conclusions

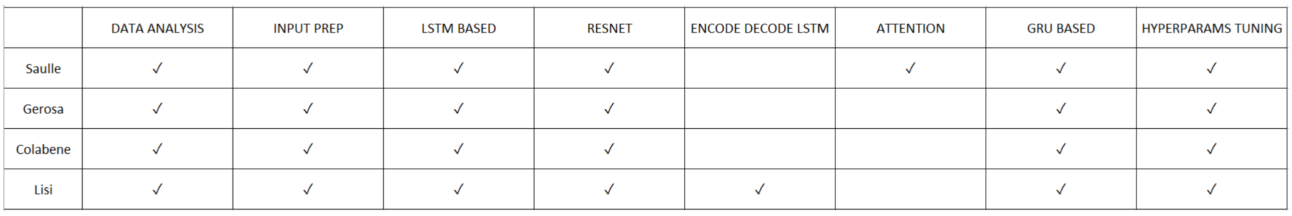
In the second phase of the competition, we tested again our models, achieving the following final score.

| Model | MSE-value | MAE-value |
|---|---|---|
| GRU | 0.00931830 | 0.06643198 |

It's important to note that the best-performing model in this second phase is exactly the same model as the first one. We used it similarly to how an Autoregression model works, by making it predict twice and adjusting the input between the two predictions. Instead, training again the same model with a telescope equal to 18 led to worse end results.

During the competition, we experimented with a lot of different models, and some others were on our radar as well, but due to time constraints, we were not able to dive into them. We attach to this report the notebooks with all the models described previously.

## Contribution



| | DATA ANALYSIS | INPUT PREP | LSTM BASED | RESNET | ENCODE DECODE LSTM | ATTENTION | GRU BASED | HYPERPARAMS TUNING |
|---|---|---|---|---|---|---|---|---|
| Saulle | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ |
| Gerosa | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ |
| Colabene | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ |
| Lisi | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |