AN2DL – Assignment #2

Time Series Forecasting

This report presents different approaches developed to predict future samples of a multivariate time series using deep neural network techniques. We are going to see what we did for data preprocessing, model development, what were the results and what we are going to do in future works.

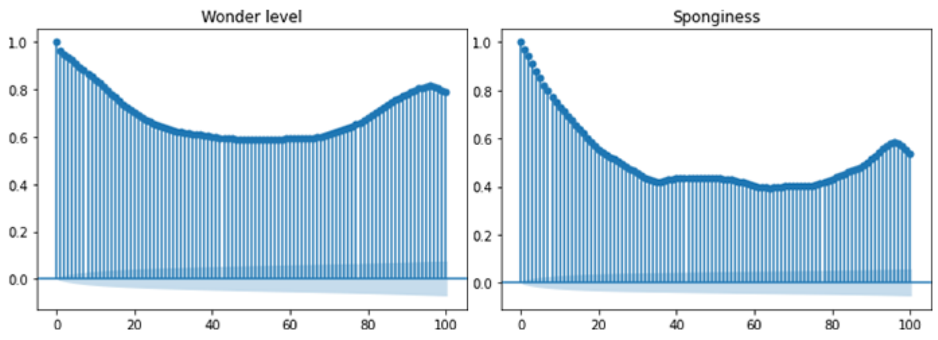
1 Data Preprocessing

Data visualization

The dataset is composed of a multivariate time series, composed of 7 features, each having 68528 samples. We visualized every feature in the training set separately (but we do not put them here to make the report no longer than it should be) and then after training and predicting our validation set, we also plotted the predicted points in another plot but together with the ground truth to see if our model is really predicting the correct values and is not just trying to lower the loss. But since we are in preprocessing section, we do not put them here.

From plotting the time series, it came that our data is not stationary, meaning that the data does not have constant mean, constant variance and constant covariance between periods of identical distance. However, as LSTM architecture is used in our model, stationarity is not important here. Otherwise, we would have to perform some operations like distancing to make it stationary.

Another thing that might be interesting to observe, is the autocorrelation graph. Autocorrelation refers to a degree of similarity between a time series and a lagged version of the same time series. In our dataset, we have a high autocorrelation, meaning that this data is good for predicting future samples from previous samples. Below we can see the autocorrelation graph for two randomly chosen features.



Building sequences

In time series forecasting, we need to predict the future based on a chunk of past data. The size of these chunks should be equal and it is exactly what the window parameter is meant for. To find the best window size, we cannot do a grid search here, since it takes too long for our model to get trained on our dataset. However, after trying a few times, we found out that bigger window sizes are working better than small values and of course, we expected that, since the autocorrelation graph was telling us our dataset benefits from a high correlation among different lags. To make our dataset divided into equally sized windows, we write a function, in which we add as many zeros as we need at the beginning of our dataset.

Data normalization

For data normalization, we perform min-max normalization. We do it without utilizing scikit-learn functions because by using scikit-learn functions we get an ndarray which is not desirable here and produces errors.

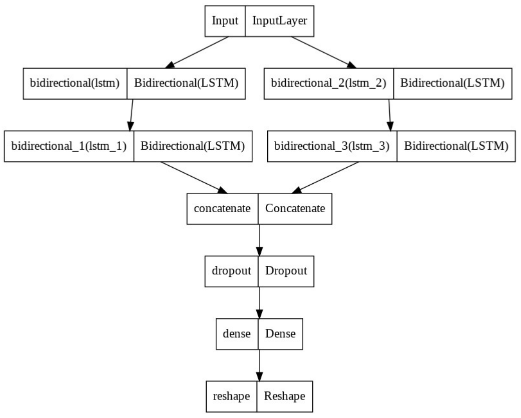
2 Model Development

To tackle this problem, several methods have been investigated in order to create and train different models.

Forecasting strategies

One of the critical aspects of our problem was to decide the number of points predicted by our model. Indeed, we thought about three different strategies: autoregressive forecasting, in which only one point is forecast, direct forecasting, by which we predict directly the 864 samples in the future and a third strategy, consisting in the forecasting of an intermediate number of points (100, 200, 288 (864/3), 442 (864/2)). This number of points is named telescope and plays an important role in the performance of different models. The choice among the detailed strategies and their associated results will be presented later in the report.

Models


Input | InputLayer
bidirectional(lstm) | Bidirectional(LSTM)
bidirectional_2(lstm_2) | Bidirectional(LSTM)
bidirectional_1(lstm_1) | Bidirectional(LSTM)
bidirectional_3(lstm_3) | Bidirectional(LSTM)
concatenate | Concatenate
dropout | Dropout
dense | Dense
reshape | Reshape

This step is the core of our project. Indeed, in order to achieve the best possible performance, we developed and worked on many points:

- Different gating mechanisms: LSTM, Bidirectional LSTM and GRU
- Presence or absence of convolutional layers in the model
- Addition of dropout layers and their rate tuning
- Implementation of attention layer in order to take advantage of the attention mechanism
- Concatenation of layers

While always monitoring our models performances, thanks to a validation set, a high number of models have been developed by combining the previously listed points. First, very simple models with only few layers of LSTM followed by a dense layer were trained. From those models, convolutional and pooling layers (Conv1D, GlobalAveragePooling1D and GlobalMaxPooling1D) were added in the feature extraction part of the network. From different attempts we made and the results on the validation set and on the hidden test set, we inferred that the aforementioned layers are not leading us to better results. Moreover, we added dropout layers, which helped in reducing overfitting. By playing with different gated units, the bidirectional LSTM turned out to give the best results. Finally, another idea was developed: to concatenate two blocks of bidirectional LSTM layers. Since it improved the results on the hidden test set, a final tuning of the number of units and blocks has been done. Above, the figure presents the structure of our best model.

Some attempts of attention layer implementation were made in order to add the self-attention mechanism in our model, but it did not lead to better results locally and on the hidden test set. However, it would be a key point to further investigate in order to help the network in understanding the important times series features to improve the prediction.

Some hyperparameters and design choices

Among the large number of possible hyperparameters (more generally, architecture alternatives) available, the following points have been developed:

- The **window**, **stride** and **telescope**: those three hyperparameters turned out to be key elements. Indeed, for example, a too large window leads to a constant prediction but a too small window does not take full advantage of the past data, or a too large stride reduces too drastically the number of available training sequences. We finally found that a window of **500**, a stride of **1** and a telescope of **864** (indeed, direct forecasting) gave the best results for our model. *An attempt with a higher telescope (1152, corresponding to the final number of predicted samples asked) was made but it did not give better results on the hidden test set.*
- The **validation_split**: different values were tried: 0.1, 0.05 and 0.01. Even if the smallest value gave the best results locally, 0.05 was the kept choice, due to the results obtained on the hidden test set. Possibly, more promising results of validation rate of 0.01 during training were due to the validation set being far smaller (and then less diversified) than a split of 0.05.
- The **learning rate**: reducing the learning rate from 10^{-3} to 10^{-4} improved the performance slightly, therefore we kept this value.
- The **batch_size**: a value of 32 led to best results in our case.
- The **loss function**: the Mean Squared Error appeared to be the most logical choice for the training.

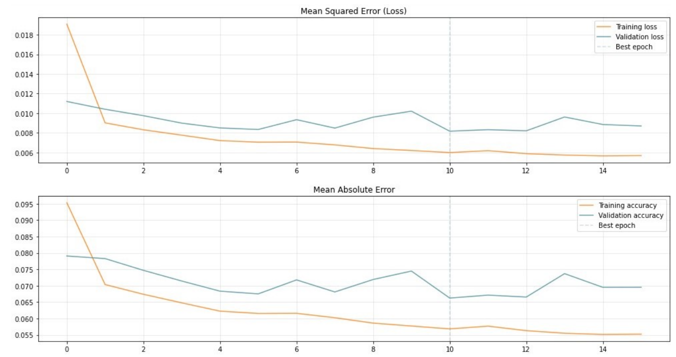
3 Results

In this section, we present the most important results of the work in order to further understand the behavior of the models.

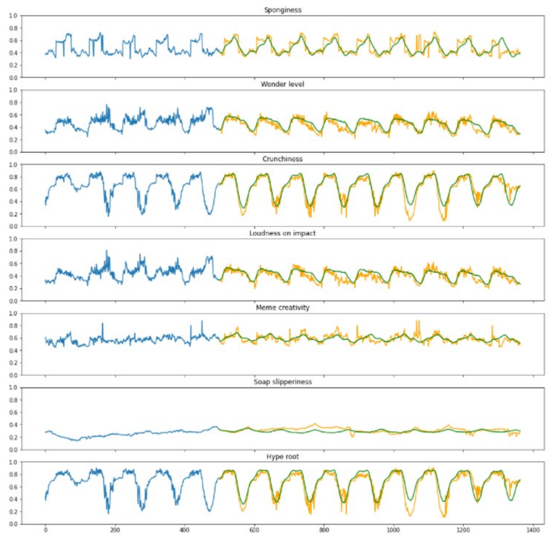
As explained in the previous section, we found that the best model was a concatenation of two blocks of two bidirectional LSTM layers. In the following table the RMSE value obtained on the hidden test set for this model in addition to the RMSE value for the same architecture but for different hyperparameters choices is presented, in order to understand the impact of some of them on the results. For all of those configurations, a stride of 1 and a telescope of 864 (direct forecasting) were used.

	Window size	Validation split	Batch size	Learning rate	Hidden test RMSE
Configuration 1	350	0.1	64	10^{-3}	4.077
Configuration 2	500	0.1	64	10^{-3}	3.859
Configuration 3	500	0.05	32	10^{-3}	3.649
Configuration 4	**500**	**0.05**	**32**	$\mathbf{10^{-4}}$	**3.645**
Configuration 5	700	0.05	32	10^{-4}	3.757

Please note that the training and/or validation RMSE values are not reported in the table, since different validation splits have been used while training those models. However, we noticed that a better validation RMSE did not imply necessarily a better hidden test RMSE. This phenomenon might be due to overfitting and therefore to a lower capacity of the network to generalize the hidden test set, especially if it is slightly different from the training one.



From the table we understand that the window is a key hyperparameter, having a strong influence on the hidden test RMSE. Moreover, reducing both the batch size and the validation split helped too. It is understandable since it increases the amount of training data. Finally, reducing the learning rate resulted in a minuscule improvement. Since it takes more time to train with a learning rate of 10^{-4}, the very small improvement induced might not be worth it.



Above are presented the Mean Squared Error and the Mean Absolute Error on the training and validation sets. We notice that the validation curves are less smooth (probably due to the lower validation split) and that the lowest validation RMSE is reached at the 10^{th} epoch.

Another important result is to compare the prediction of our model with the ground truth. On the right predictions for a random sequence are presented. Moreover, the table just below presents the RMSE values on each time series in the hidden test, along with the normalized ones in order to compare them (obtained by dividing by the range of each feature in the given training set):

	Sponginess	Wonder Level	Crunchiness	Loudness on Impact	Meme Creativity	Soap Slipperiness	Hype Root
RMSE	1.3745	1.3528	5.9605	1.2724	0.5378	3.2943	6.4026
Norm. RMSE	0.0855	0.0509	0.0757	0.0552	0.0733	0.0395	0.0868

The best results have been achieved on the variables *Wonder Level*, *Loudness on Impact* and *Soap Slipperiness*. *Sponginess* and *Hype Root* are the two most critical features, which is understandable looking at their signals. For example, the *Sponginess* fluctuates highly and the difficulty in forecasting this variable was present during the whole work: a lot of trained networks were predicting the whole telescope by an almost constant signal.

4 Future Work

Obviously, there is still a lot of things to be developed in this project. The following list details the main points which may require a further development:

- Investigate more carefully the attention mechanism in order to implement it since it should theoretically improve the performance.
- Apply a more structured hyperparameter search. Indeed, an important conclusion of this work is that hyper-parameters play a key role and can influence the obtained performance a lot. It would, therefore, be interesting to tune some of them even more finely, such as the window, the stride, the telescope and the learning rate.