Recurrent Neural Network: Time Series Forecasting

Artificial Neural Networks and Deep Learning - a.y. 2021/2022

Fabio Tresoldi
M.Sc. Computer Science and Engineering
Politecnico di Milano - Milan, Italy
E-mail: fabio1.tresoldi@mail.polimi.it
Student ID : 10607540
Codalab Nickname: "fabioow"
Codalab Group: "artificial_comrades"

Mirko Usuelli
M.Sc. Computer Science and Engineering
Politecnico di Milano - Milan, Italy
E-mail: mirko.usuelli@mail.polimi.it
Student ID : 10570238
Codalab Nickname: "mirko"
Codalab Group: "artificial_comrades"

***Abstract*—In this report, we explore and compare the development process we had to design a Recurrent Neural Network able to forecast a multivariate time series composed of 7 different components.**

1. Introduction

The proposed dataset has been split into a Training Set (first 90% time steps) and Test Set (last 10%) in purpose of estimating forecasting conclusions. During the training we adopted a Cross-Validation 90% - 10% at run time.

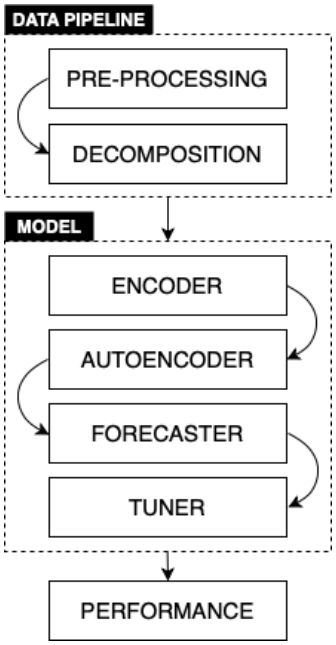

Figure 1. Development Process approach.

2. Data Pipeline

2.1. Pre-Processing

First of all, we adopted a Min-Max scaling with respect to the Training Set in order to normalize the data. Next, we applied it to the Test Set.

2.2. Decomposition

We developed from scratch an additive STL decomposition using just `numpy` and we have been able to extract – from each time series – their respective *Trend*, *Seasonal* and *Residual* components. Before applying it we needed to discover the characteristic period of each time series: we exploited Moving Average properties related to the Autocorrelation Function to identify this information on the entire dataset.

$$periods = [192, 96, 96, 96, 95, 86, 96]$$

This achievement was made in purpose of gaining better forecasting capabilities by working on a modular architecture rather than a compact one with a more sophisticated input to deal with.

3. Model

The design approach that we had was purely incremental as it is going to be explain in the model section: starting from a baseline we progressively improved it by trying several techniques and made decisions according to RMSE as our primary metric.

All the results that will be shown were obtained with $window = 200$, $stride = 10$ and direct forecasting of 864 timesteps on the Test Set.

3.1. Encoder

3.1.1. RNN Module. We first discovered what was the best RNN cell to focus in, hence we compared them through a simple sequential layer followed by the forecaster:

	RMSE	*MAE*
GRU	0.1252	0.1027
LSTM	0.1246	0.1012
Bi-LSTM	**0.1229**	**0.0943**

3.1.2. Encoder Architectures. We then compared 5 baseline architectures starting from a basic RNN – namely Bi-LSTM – then improved through a CNN. We replicated the well-known architecture WaveNet consisting in just Convolutional layers to obtain further hints on the direction to take into account. At the end, we also tried the Transformer Encoder – including a temporal Positional Encoding – in order to fully exploit its architectural benefits.

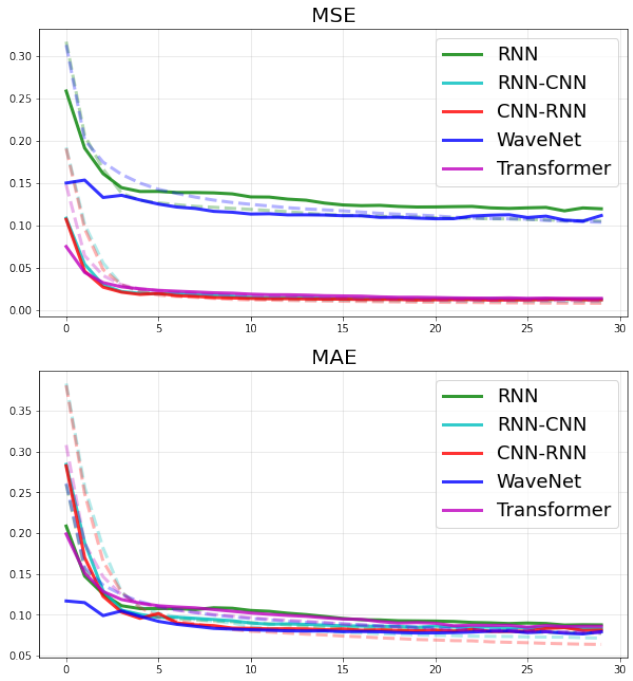

Figure 2. Encoder Architectures Comparison (Validation Set).

As shown by the two metrics, the lowest convergence has been achieved by the CNN-RNN Encoder, thus we decided to improve this baseline with further experiments.

3.1.3. Multi-Branch Encoder. We decided to split the selected Encoder into 7 branches – one branch for each time series – through a Lambda layer able to address each time series to gain a better feature extraction learning:

	RMSE	*MAE*
Baseline (CNN-RNN)	**0.1198**	**0.0855**
MultiBranch-Baseline	0.1310	0.1030

Nevertheless, this attempt failed to improve our baseline.

3.1.4. Attention Mechanism. This recent technique has been a breakthorugh within the Deep Learning community, so we decided to make several attempts on it; more precisely we focused on Spatial Attention (ATT-Encoder), Temporal Attention (Encoder-ATT) and Dual Attention (ATT-Encoder-ATT). The following results were obtained through the Keras layer `MultiHeadAttention` set as self attention mechanism:

	RMSE	*MAE*
Baseline (CNN-RNN)	**0.1198**	**0.0855**
ATT-Baseline	0.1460	0.1094
Baseline-ATT	0.1277	0.0992
ATT-Baseline-ATT	0.1371	0.0980

However, the Attention Encoders did not outperform the starting baseline, thus we decided to move towards other techniques.

3.1.5. Decomposition. Due to Time Series defintion expressed as:

$$Series = Trend + Seasonal + Residual$$

Our goal was to improve prediction capabilities on the residual component because of its stochastic behaviour with respect to the others. At the end of prediction – due to the additive nature of the decomposition – it was just needed an `Add` layer to bring the time series back.

	RMSE	*MAE*
Baseline (CNN-RNN)	**0.1198**	**0.0855**
STL-Baseline	0.1205	0.0988

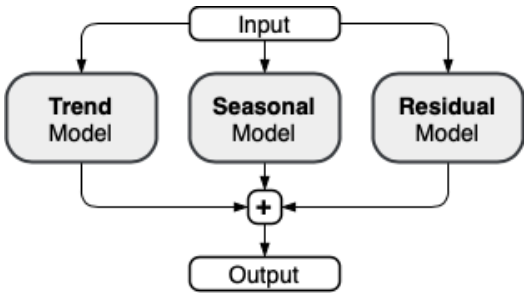

Figure 3. Overall STL Architecture

Despite that, there was no relevant improvement as shown by the data.

3.2. Autoencoder

Afterwards, we decided to furtherly improve feature extraction performance through an Autoencoder of given window and then unplug the trained Encoder to be joined with a forecaster in the following section:

	RMSE	*MAE*
Baseline (CNN-RNN)	0.1198	0.0855
AE-Baseline	**0.1189**	**0.0834**

3.3. Forecaster

For the forecaster we just applied a simple Dense layer preceeded by a Dropout that we tuned as follows; then we reconstructed back the output shape for the final prediction as shown in section 4.1.

	RMSE	*MAE*
No Dropout	0.1189	0.0834
Dropout(0.1)	0.1126	0.0848
Dropout(0.2)	0.1198	0.0836
Dropout(0.3)	**0.1105**	**0.0809**
Dropout(0.4)	0.1134	0.0824
Dropout(0.5)	0.1170	0.0857

3.3.1. Predictive Approach.

	RMSE	*MAE*
Direct	**0.1105**	**0.0809**
Autoregressive	0.1411	0.1044

3.4. Tuner

We adopted a Bayesian Optimization for the model hyper-parameters in purpose of finding the best possible performance with the design choices we discussed. This process has been done with `KerasTuner`.

4. Conclusion

Resuming, we started from a basic CNN-Bi-LSTM, afterwards we trained an Autoencoder in order to improve pattern recognition performance in an unsupervised way. Later on, we unplugged the Encoder to be joined with the Forecaster to estimate final predictions in a direct approach.

4.1. Final Model

A the end, the Bayesian optimization tuning produced the following model:

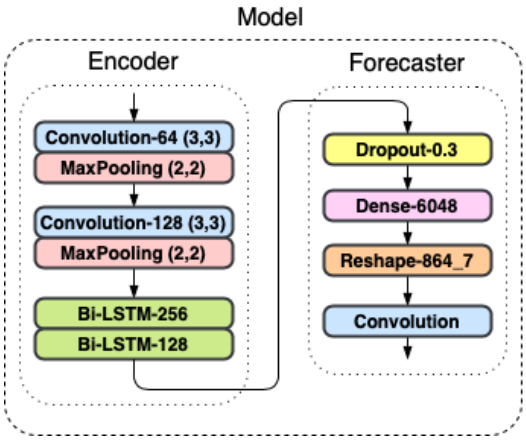

Figure 4. Model Architecture

4.2. Performance

	RMSE	*MAE*
AE-CNN-Bi-LSTM	**0.1105**	**0.0809**

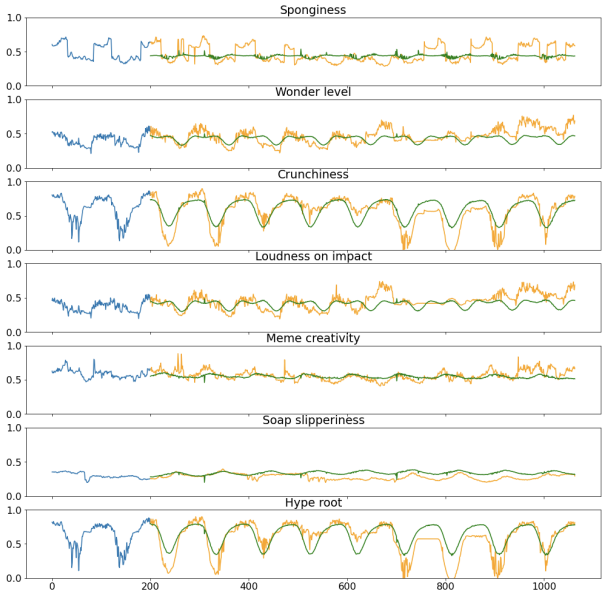

Figure 5. Predictions on the Test Set highlighted in green.

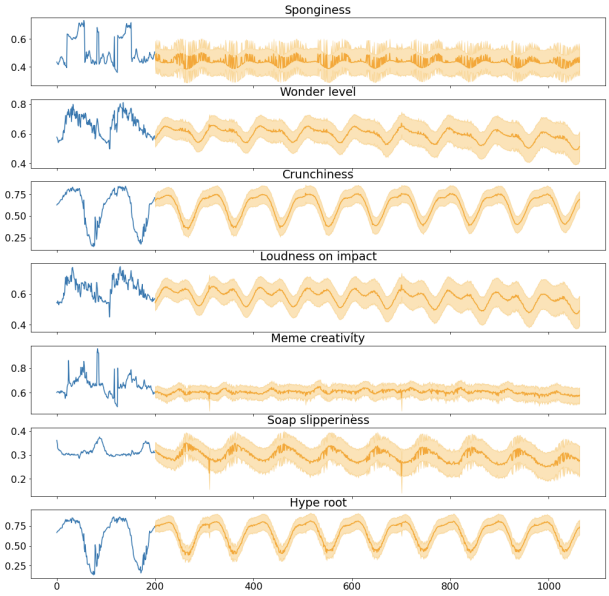

Figure 6. Futures with prediction error.

4.3. Leadboard Evaluation

- Development phase RMSE : **4.1252**
- Final phase RMSE: **—**