Convolution is all you need

Time series forecasting

Alessandro Gennari, Gianluca Gragnaniello, Lorenzo Brochier

January 20, 2022 Politecnico di Milano

Artificial Neural Network and Deep Learning

1 Possible Data Artifacts

During the competition we realized the presence of several flat regions with a length equal to 97/98 timesteps common to all the features. In addition to the strangeness of the fact itself (due to the oscillatory nature of the features) these regions seemed almost added on purpose as they interrupted the oscillation frequency of the features. We have therefore tried to remove these regions and in some networks this has improved the results and in others it has not. The reason for this is not yet fully clear to us, we would need a more in-depth study.

2 Traditional approach

We initially tested the network used by Professor Lattari in the laboratories, to have an initial evaluation of this type of architecture. The results as expected were not satisfactory (test_score = 8.57). This was to be expected looking at the prediction images on the validation data, since the predictions were an average for each feature without oscillations. At this point we started trying architectures from scratch. As first attempt, a coarse model was built with the Keras Recurrent layers: SimpleRNN, LSTM and GRU. For each of those architectures, three models were created with respectively one, two, and three layers. Furthermore, other 4 model were builded, with one or two layer of, respectively, LSTM or GRU architecture, leading to a totality of 13 models. Each one of those models had a Dropout layer, a Dense layers and Reshape at the end. The original dataset was splitted in training and validation sets, with the last 20% of original data for the validation (13706 values) and the previous 80% for the training. All models were trained with the same training dataset, with the same hyperparameters: Batch size 128, Epoch 200, Window size 2000, Telescope 864, layer units 128, Dropout 0.3, Activation function 'relu'. As metrics to evaluate model's performance, Mean Squared Error was considered, because it was comparable with the test Root Mean Squared Error. During training, an adaptive learning rate was applied, which consists in a reduction of learning rate when the gradient remains stacked in a plateau for more than 5 epochs. Validation mean squared error of each of the 13 models was compared, the best results was obtained with one layer of bidirectional LSTM, the second and the third best models with, respectively, one layer and two layers of bidirectional GRU. Those three best models were then submitted to the test dataset: One Bi-LSTM (RMSE 4.30), One Bi-GRU (RMSE 4.53), Two Bi-GRU (RMSE 4.56).

Hyperparameters tuning Given the previous results, we proceed with the hyper parameters tuning of the best model, composed of one bidirectional layer of LSTM, Dropout layer, a Dense layers and Reshape. Changing one parameters at a time, the model was re-trained and only the test RMSE was considered as metrics for the evaluation, because validation mean squared error was a very small value, so it was difficult to notice a real improvement on the validation (being already near to the 'perfection'). After several different trainings, LSTM model didn't show any hint of performance improvement, hence, an other attempt was done with the second best model. The percentage of neurons to turn off with the Dropout layer was the first parameter tuned, then the Batch size, number of Units per layer, activation function and windows size. The results of this tuning process was a model with one Bidirectional GRU layer with 128 units, Dropout 0.05, ReLU as activation function for the Dense layer, trained with 32 Batch size, and window of 2000. Test RMSE is 4.03.

Data Augmentation An attempt of data augmentation on the original dataset has been made to improve generalization capabilities. The last 18634 values of each time series were copied, added with some noise, and appended to the end of the original time series. Noise was added before normalization, with different scales according to the variance of the 7 series belonging to the dataset: scale = 0.2 in the series # 0, 1, 3 and 4; scale = 0.6 for the series # 2, 5 and 6. The results on the Test set didn't show any improvement, with RMSE 4.36 obtained with the best model so far, hence, data augmentation was discarded.

3 Convolutional-LSTM Nets

We decided to explore also some networks in which Convolutional and LSTM layers were used at the same time. For all the nets discussed, a large number of variations were made considering mainly 5 parameters:

the training dataset, input length, output length, activation functions and batch size. They are not discussed in detail due to the limited space of up to 3 pages.

CoLs Nets The first series of nets was 'CoLs Nets' in which a module consisting of Bi-LSTM + Conv + Pool was used, repeated several times (3/4/5) in a row, connected to the usual GAP layer and then to the final dense layer. From here we can see a hint of cyclicality in the validation test predictions, even if still very 'anchored' to the average of each feature. We also noticed that by training on less data, the network had better scores. In fact we tested the datasets without the first 25000 and 42000 samples, obtaining better scores (about 8.0).

Parallel Nets In this case instead of putting the modules described in the Cols Nets one after the other we parallelized them, assuming that each module could learn the behavior of 1/2 similar features. The outputs of each module were then added together before passing them to the usual GAP layer. We also tried to eliminate the GAP layer, increasing the Conv layers in the modules, obtaining more or less similar results. Finally, seeing the prediction images in the validation set (there were some 'spikes'), we decided to change the activation function by trying sigmoid and tanh, since we almost always applied a normalization between 0 and 1 to the data; this totally changed the prediction images in the validation set by completely eliminating the 'spikes' and results on the test set. The best model was obtained with activation = tanh (test_score = 4.46), dataset with first 42000 timesteps removed.

AutoRegressive Nets At this point we decided to try to build an autoregressive model. The architecture always remained the same but the output and therefore also the parameters were significantly reduced. This meant that at this point instead of adding the outputs of each module we could, instead, concatenate them, thus keeping more information for each module. After several attempts for the input and output lengths we found the optimal one (input = 1000x7, output = 100x7), obtaining an even better result than the one obtained previously (test_score = 3.95). The strength of autoregressive models perhaps lies in the fact that by being able to predict only a small part of the output at a time, they can better fit it. Finally, we further improved the test scores by removing the flat regions found in the dataset, then taking the last 40000 timesteps of the remaining ones (test_score = 3.71).

Missed Nets Two further architectures could not be tested due to Codalab submission problems. The first in which the Bi-LSTM layers were placed towards the end after the Concatenate layer as suggested by some articles (Sainath et al. 2015, Brownlee 2017), due to the capability of feature extraction of the latter. The second with the same architecture as the first but also used a GAP layer (size 1x1000), parallel to the 4 modules, which was concatenated to the network output just before the dense layer. The basic idea was that the GAP was used to identify the main timesteps to predict the output. Furthermore, thanks to another stack of conv+maxpooling after the first concatenation, the advantage of these nets nets was that they had a much lower number of parameters (1.5-2 million) than the best AR net (89 Million), despite having better results in the validation set.

4 Parallel Convnet

Since the classical approaches (RNN, LSTM, GRU) seemed to fail to improve further, we decided to try a completely different architecture. This structure, which we will call Parallel Convnet for simplicity, is made of two parts, a feature extractor consisting only of Conv1d layers (with filters=1) followed by Maxpooling, and a regressor made of a classical Dense layer. There are no memory cells in any part of the network. We thought that instead of choosing a fixed kernel size, we could use many different kernel sizes in parallel. In this way it becomes possible to extract both low level and high level features at the same time, and then pass them all to the regression layer. We made a script that would allow us to decide how many parallel filters to have (a parameter we called n_filters) and for each of them, how many Conv1d+Maxpooling stacks (a parameter we called depth). If for example we set n_filters=3, we will have three filters in parallel, the first with k_size=1, the second with k_size=2 and the third with k_size=3 [1]. For this architecture we have been inspired by FilterNet [Chambers and Yoder 2020]. The optimisation of the model has been very complex. On the one hand, increasing the depth allows the extraction of finer features, but at the same time considerably reduces the number of final parameters and therefore the power of the model. Similarly, increasing n_filters means extracting more features but also increasing the number of parameters, risking overfitting. These, together with window size and various types of regularization (e.g. dropout, early stopping etc.) are just

some of the parameters that had to be evaluated in order to tune the model. For depth the values 2, 3, 5 were tested, while for n_filters 5, 7, 10, 14, 18, 21, 24, 26 and 30. The window size was set to 2000. From this large number of parameter combinations we tried to figure out what the "sweet spot" was. This seems to be achieved with n_filters=10 and depth=3. The corresponding model, called G14 on Codalab, obtained an RMSE of 4.078. To further improve the architecture, we tried adding bidirectional memory layers, at the beginning or at the end of the feature extraction, and increasing the number of filters in each parallel lane from 1 to 64 or 128. In this way, however, the number of parameters exploded. To solve this problem, we introduced a GAP layer. Both of these approaches did not lead to improvements and were consequently discarded. In particular, the second approach caused obvious underfitting, since it went from having 2000 neurons per parallel lane to only 64 or 128 to be passed to the dense layer. Increasing the number of filters in order to have more neurons after the GAP was not possible due to the strongly parallel structure, which caused Out of Memory error for values above 128.

Multi-input ParallelConvnet (MIP-Convnet) This model stems from the observation that the features present in the dataset are very different from each other. Consequently, the model may have difficulty extracting features representative of all 7 columns at the same time. In order to increase flexibility, we created a multi-input model, where each of the 7 columns of the dataset is passed and processed by the feature extraction part separately. The feature extraction is the same as in the Parallel Convnet model, simply applied in parallel to each column. All outputs of the convolutional part are then concatenated and passed to a single dense layer that computes the output. This architecture clearly has many more parameters and this raised concerns that the resulting models might overfit. On the contrary, thanks to this approach and after testing different combinations of hyperparameters, we came up with a further improvement. With n_filters=8 and depth=4, the resulting model, submitted as G16, achieved an RMSE of 4.055. The improvement seems small, but in reality it does not take into account that the G14 score was obtained using the whole dataset, while for G16, part of the dataset was retained for validation. We also tried replacing concatenate() with add() in order to reduce the number of parameters and have a better generalization, but the results obtained were not encouraging and the approach was discarded.

Multi input-Multi processing This third evolution of the architecture stems from the fact that, although the previous models appear to be good, they perform significantly worse on some features, namely Crunchiness and Hype Root. Even processing them separately does not seem to be sufficient. In fact, when you look at the data, there is a very different trend for these two features, which probably makes them more complex to learn [2]. For this reason, we thought that each feature could have different n_filters and depths, in order to increase the power of feature extraction selectively for Crunchiness and Hype Root. We started with the best model, G16, and left the encoding part unchanged for all features except Crunchiness and Hype Root, for which we tried to increase n_filters and modify depth. This approach did not lead to improvements, but it was also the least tested, so it is not out of the question that with more time we could have obtained an even better model. The thing that puzzled us, however, is that when we downloaded the matrix of scores per class from Codalab, we noticed an improvement in the scores of some features that had not been touched (e.g. Sponginess) and at the same time a worsening of other features, including Crunchiness, which we were trying to improve.

Final improvements Having chosen the best architecture, the MIP-Convnet, we made some final modifications to obtain a definitive model. The first major change was to switch to an autoregressive model. This allows us to greatly reduce the number of parameters, while maintaining the same architecture. In this way we have a faster training time and at the same time a better ability to generalize. The best score was obtained using 100 as the number of predicted values. We also noticed that the first part of the dataset has a different trend from the rest [3], and training only on the last 42000 points of the dataset improves the testing error. We tested different activation functions and the "hard sigmoid" was the best. The last two changes were to pass some points of the input and the output of some convolutional layers directly to the regression layer, skipping the feature extraction part. We chose to use the last 100 points from the input and the output of the second convolutional block. Combining all these modifications with a final dropout to limit overfitting, the resulting model obtained an RMSE of 3.4679 on Codalab [4]. Not only did this model scored better than other architectures with memory cells, but it also had a significantly lower running time, with only 2s per epoch. This is due to the strongly parallel structure and exclusive use of convolutional filters.

References

[1] Tara N Sainath et al. "Convolutional, long short-term memory, fully connected deep neural networks". In: *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2015, pp. 4580–4584.

[2] Jason Brownlee. *CNN Long Short-Term Memory Networks*. Aug. 2017. URL: https://machinelearningmastery.com/cnn-long-short-term-memory-networks/.

[3] Robert D. Chambers and Nathanael C. Yoder. "FilterNet: A Many-to-Many Deep Learning Architecture for Time Series Classification". In: *Sensors* 20.9 (2020). ISSN: 1424-8220. DOI: 10.3390/s20092498. URL: https://www.mdpi.com/1424-8220/20/9/2498.

Images

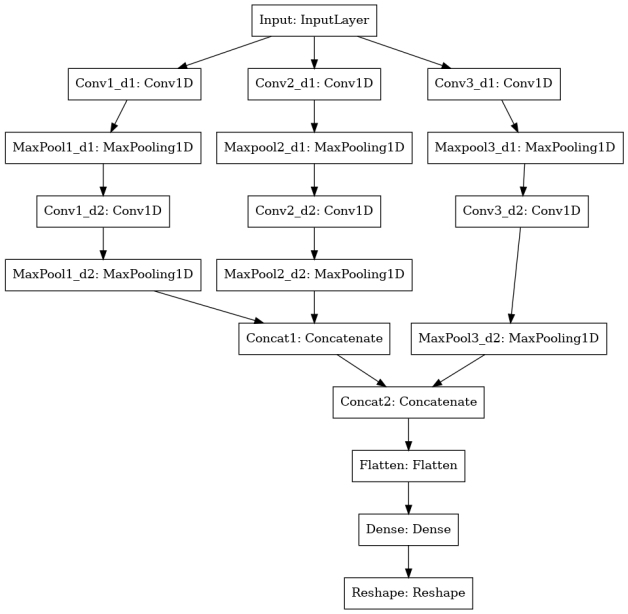

Figure 1: Model example with n_filters=3 and depth=2. Each parallel column has an increasing kernel size

Figure 2: Some features have different oscillations as we can see in this example for Crunchiness (top) and Soap Slipperiness (bottom)

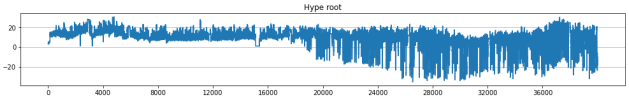

Figure 3: There is a big difference in oscillation range between the first and second part of the dataset as shown for Hype Root

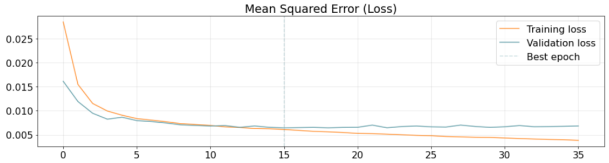

Figure 4: Train and validation MSE of the best model