Artificial Neural Network and Deep Learning

Challenge 2

NetCom

POLITECNICO
MILANO 1863

Alessia Luoni

Lorenzo Bancale

INTRODUCTION

The goal of the project is to build a forecasting model which is able to predict a multivariate time series by observing the past behaviour.
This challenge is a typical regression problem.

THE DATASET

We have been provided with a CSV file containing a dataset composed by seven time series which features are the following:

1. "Sponginess"
2. "Wonder level"
3. "Crunchiness"
4. "Loudness on impact"
5. "Meme creativity"
6. "Soap slipperiness"
7. "Hype root"

The type of values of the tensor in input is float64.
By inspecting the data, we observed that there were 68528 values for each feature, and we observed that null values were absent.

MODELS

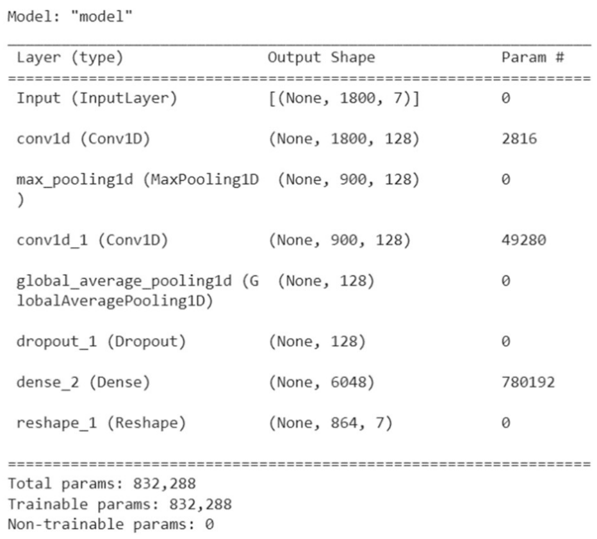
We started our work by importing the necessary libraries among those that were allowed, setting the seed and loading the dataset.

PRE-PROCESSING

The first thing that we did in this phase was to split the dataset into the test and train set in the following way: ~10% for the test set and ~90% for the train set.
Then we normalized the data and since we decided to use the sliding window approach, we set the window's size at 1800 points and the stride at 10 points. With these parameters we built a function, called build_sequence which permitted to cycle over the time series using the windows already defined.
Finally, since we needed to predict 864 values for each feature, we set the telescope to this value.

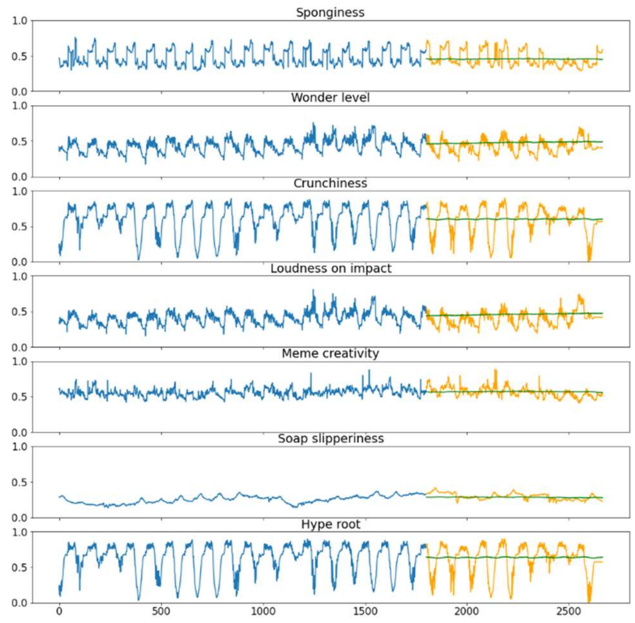
BUILDING AN INITIAL MODEL

We started by building the following initial model:



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 1800, 7)]         0

 conv1d (Conv1D)             (None, 1800, 128)         2816

 max_pooling1d (MaxPooling1D  (None, 900, 128)         0
 )

 conv1d_1 (Conv1D)           (None, 900, 128)          49280

 global_average_pooling1d (G  (None, 128)              0
 lobalAveragePooling1D)

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 6048)              780192

 reshape_1 (Reshape)         (None, 864, 7)            0

=================================================================
Total params: 832,288
Trainable params: 832,288
Non-trainable params: 0
```

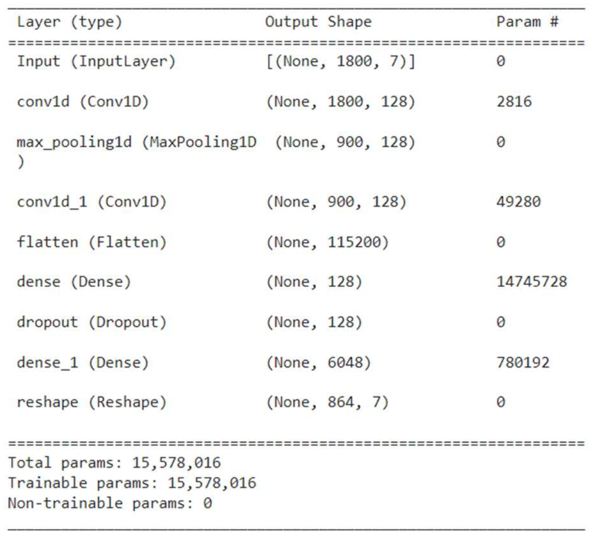
The first model we developed was a simple 1D convolutional neural network with a final reshape layer which was able to predict the next 864 values for each of the seven features.
We have decided to use the ReLu activation function both in the convolutional layers and in the output layer. We also inserted a dropout layer to prevent the overfitting fixed at 0.5.
Then, we proceeded by compiling the model with the mean squared error since it's a regression problem and with Adam as the optimizer algorithm.
Afterward we fitted the model with a batch size of 64, a callback for the early stopping and another one which automatically reduces the learning rate after 5 epochs of patience.
We then plotted the results of the training and by looking at them we found out that our model was predicting the mean.



This model had a score of ~44.3. This bad result was due also to the fact that we had an error in the denormalization inside the model.py file.

After fixing the error in the model.py file, we noticed that the global average pooling layer was causing the network to predict the mean, so we substituted it with a fully connected layer.

Moreover, we took inspiration from our previous model of the challenge 1 and we applied it on this dataset.



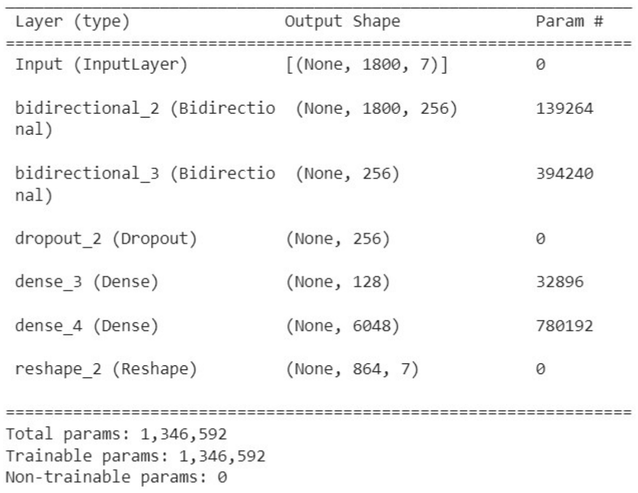
```
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 1800, 7)]         0

 conv1d (Conv1D)             (None, 1800, 128)         2816

 max_pooling1d (MaxPooling1D  (None, 900, 128)         0
 )

 conv1d_1 (Conv1D)           (None, 900, 128)          49280

 flatten (Flatten)           (None, 115200)            0

 dense (Dense)               (None, 128)               14745728

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 6048)              780192

 reshape (Reshape)           (None, 864, 7)            0

=================================================================
Total params: 15,578,016
Trainable params: 15,578,016
Non-trainable params: 0
_________________________________________________________________
```

From it we obtained a better score.

BILSTM

We decided to try different models in parallel to see which one was performing better.

Therefore, we built a second model completely different to the previous one, in fact instead of being a convolutional neural network it was a recurrent neural network composed by bidirectional LSTM layers, and in particular, it was designed as follows:



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 1800, 7)]         0

 bidirectional_2 (Bidirectio  (None, 1800, 256)        139264
 nal)

 bidirectional_3 (Bidirectio  (None, 256)              394240
 nal)

 dropout_2 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 128)               32896

 dense_4 (Dense)             (None, 6048)              780192

 reshape_2 (Reshape)         (None, 864, 7)            0

=================================================================
Total params: 1,346,592
Trainable params: 1,346,592
Non-trainable params: 0
```

Also this time the model was not following the mean, but unluckily, because of the error we got ~43.8.

Since we didn't notice the error, we modified the model by adding an additional BiLSTM layer with 64 units before the dropout layer. This little change improved the score up to ~43.5 and by fixing the error in the model.py file we got ~4.48.

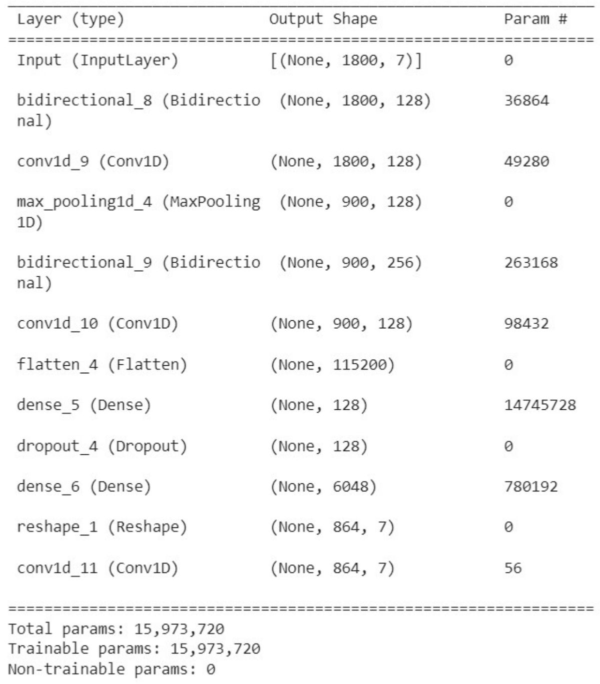
We then tried to modify some hyper-parameters, but none of these changes improved our result (e.g., we decreased the dropout value from 0.5 to 0.2, obtaining ~8.07).

In parallel, we inserted also a GRU layer after the BiLSTM layers, but it didn't improve your best result.

From our best model we modified the stride by setting it to 8 and this made us reach ~4.42.

MIXED MODEL

Another model that we experimented was a mixed model composed both by 2 BiLSTM layers and some layers belonging to a convolutional neural network:



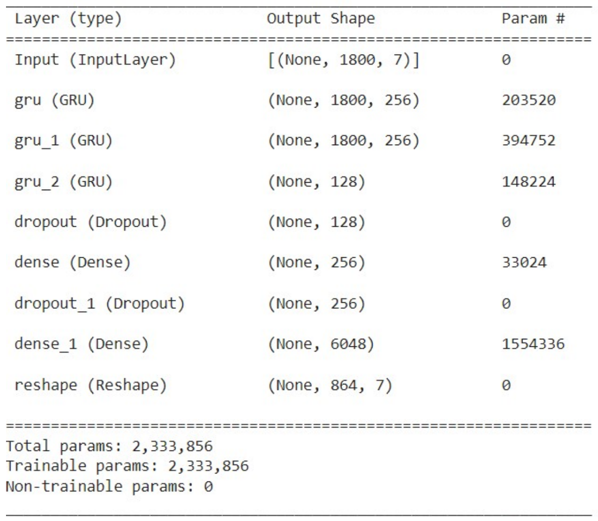
```
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           [(None, 1800, 7)]         0

bidirectional_8 (Bidirectio  (None, 1800, 128)         36864
nal)

conv1d_9 (Conv1D)            (None, 1800, 128)         49280

max_pooling1d_4 (MaxPooling  (None, 900, 128)          0
1D)

bidirectional_9 (Bidirectio  (None, 900, 256)          263168
nal)

conv1d_10 (Conv1D)           (None, 900, 128)          98432

flatten_4 (Flatten)          (None, 115200)            0

dense_5 (Dense)              (None, 128)               14745728

dropout_4 (Dropout)          (None, 128)               0

dense_6 (Dense)              (None, 6048)              780192

reshape_1 (Reshape)          (None, 864, 7)            0

conv1d_11 (Conv1D)           (None, 864, 7)            56

=================================================================
Total params: 15,973,720
Trainable params: 15,973,720
Non-trainable params: 0
```

Probably this network was too complex for our task because we reached only ~6.15.

GRU MODEL

Until now the model with the highest score has been the one with the BiLSTM layers, so we decided to create a new model with GRU layers taking inspiration by the BiLSTM model.

Here it is the structure of the model:



```
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           [(None, 1800, 7)]         0

gru (GRU)                    (None, 1800, 256)         203520

gru_1 (GRU)                  (None, 1800, 256)         394752

gru_2 (GRU)                  (None, 128)               148224

dropout (Dropout)            (None, 128)               0

dense (Dense)                (None, 256)               33024

dropout_1 (Dropout)          (None, 256)               0

dense_1 (Dense)              (None, 6048)              1554336

reshape (Reshape)            (None, 864, 7)            0

=================================================================
Total params: 2,333,856
Trainable params: 2,333,856
Non-trainable params: 0
```

This attempt let us hit a better score of ~4.11.

Starting by this result we tried it again with some changes:

- Firstly increased and then decreased some hyperparameters such as the stride and the window size
- We tried to add a batch normalization layer as a regularization technique (both maintaining the dropout layer and removing it)
- We introduced the skip connections in order to help the following layers to learn features on the top of the input value

However, none of these modifications increased the score of the model.

ANOMALY DETECTION

Taking as base the BiLSTM model, we decided to inspect the dataset looking at the distribution of samples for each feature. We noticed that there were possible outliers, so we decided to perform an anomaly detection analysis.

We set an upper and a lower different threshold for each feature based on its distribution, we then substituted the outliers with the upper or the lower bound.

We first submitted the model with a strong outlier detection, in fact we were discarding almost 10% of the samples for each feature and this caused a worsening of the score up to ~11.4.

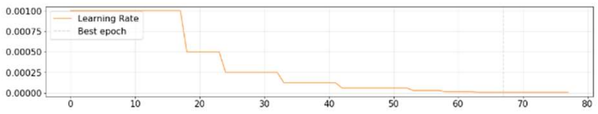
So, we made the outlier detection softer by raising the upper threshold and decreasing the lower threshold and this mean that we were discarding less than 3% of samples for each feature. This change made the score reach ~4.57.

Before applying the anomaly detection at the GRU model, that was our best model up to now, we tried to substitute the outliers with the mean values of each feature instead of the upper and lower bound with poor results.

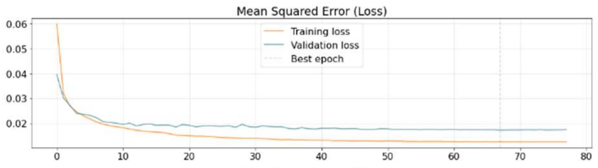
We finally tested this analysis on the GRU model with a softer version. Initially we substituted the outliers with the mean value obtaining the same score, then by replacing the outliers with the upper and

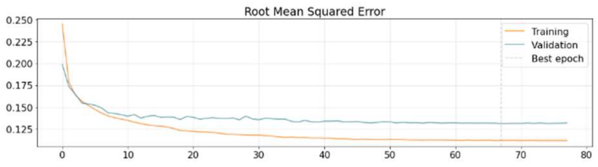
lower boundaries, we improved the score to ~4.10.
Further changes were tried without significant enhances.

We report below the graphs of our best model.

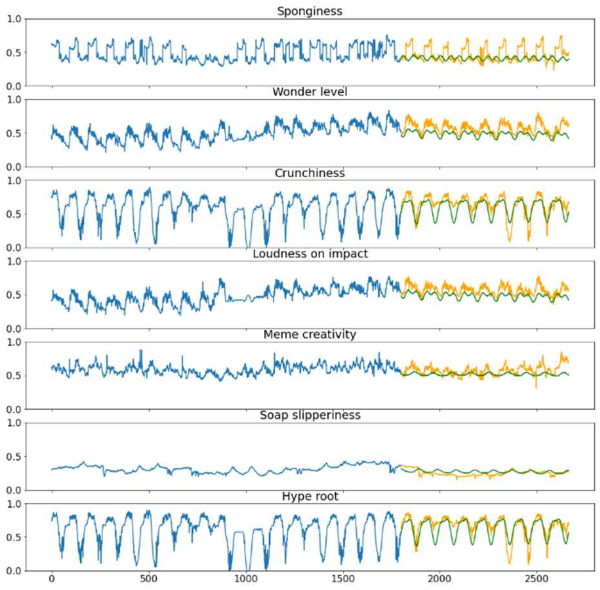


This (↑) is learning rate trend that decreased because of the callback that we introduced to reduce it.

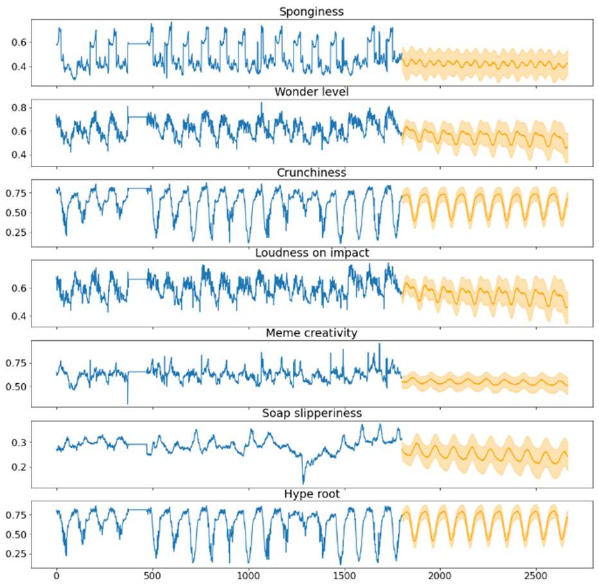




Instead, those two graphs (↑) show the trends of the mean squared error and of the root mean squared error.
The following graph (↓) exhibit the test and the evaluation of the model, and we can see that for some features is pretty accurate.



Finally, we report the prediction graph (↓).


Sponginess
Wonder level
Crunchiness
Loudness on impact
Meme creativity
Soap slipperiness
Hype root