Artificial Neural Networks and Deep Learning: Second Challenge.

Giacomo Da Re, Filippo Betti, Francesco Attorre.

December 2021

1 Purpose

The main goal is to solve a multivariate time series forecasting of future samples for 7 different classes.

2 DataSet

2.1 DataSet distribution

We can see, when plotting the distribution of our 7 classes, that some of them aren't equally distributed around the mean. We can also see some outliers in some of the kernels (e.g. the density estimation of Meme creativity).

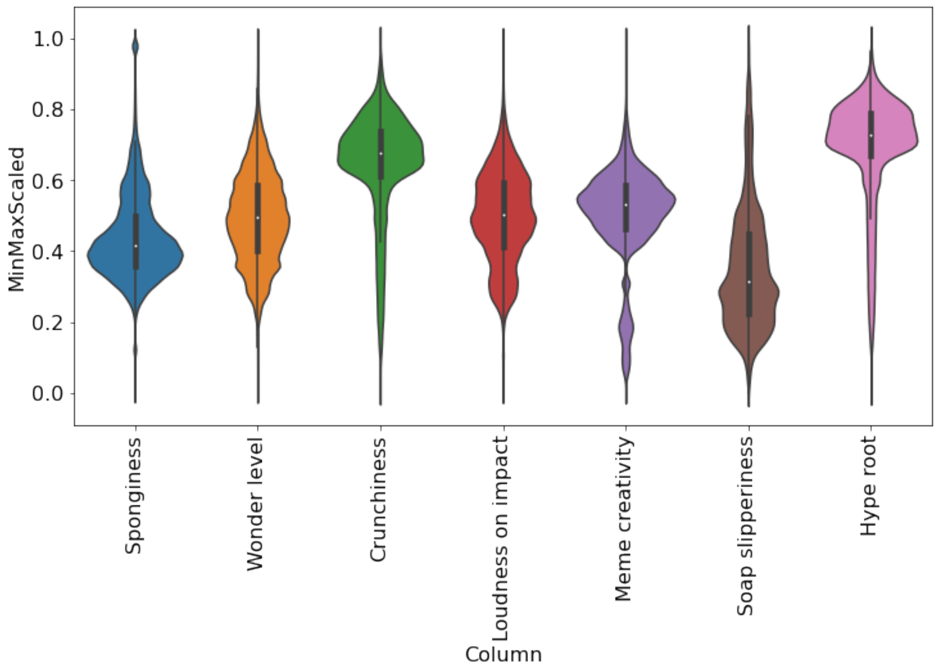


Figure 1: Violin plot of our dataset.

2.2 Data preprocessing

Printing the timeseries, we noticed that the values of our data are not equally bounded. For this reason we decided to perform some min-max normalization on our dataset. We also used the function "build_sequences" in order to create the sequences to train our LSTM model. We obtained good results focusing on the stride of our input sequence generator, in fact lowering the dimension of this parameter, our models' performances improved by far.

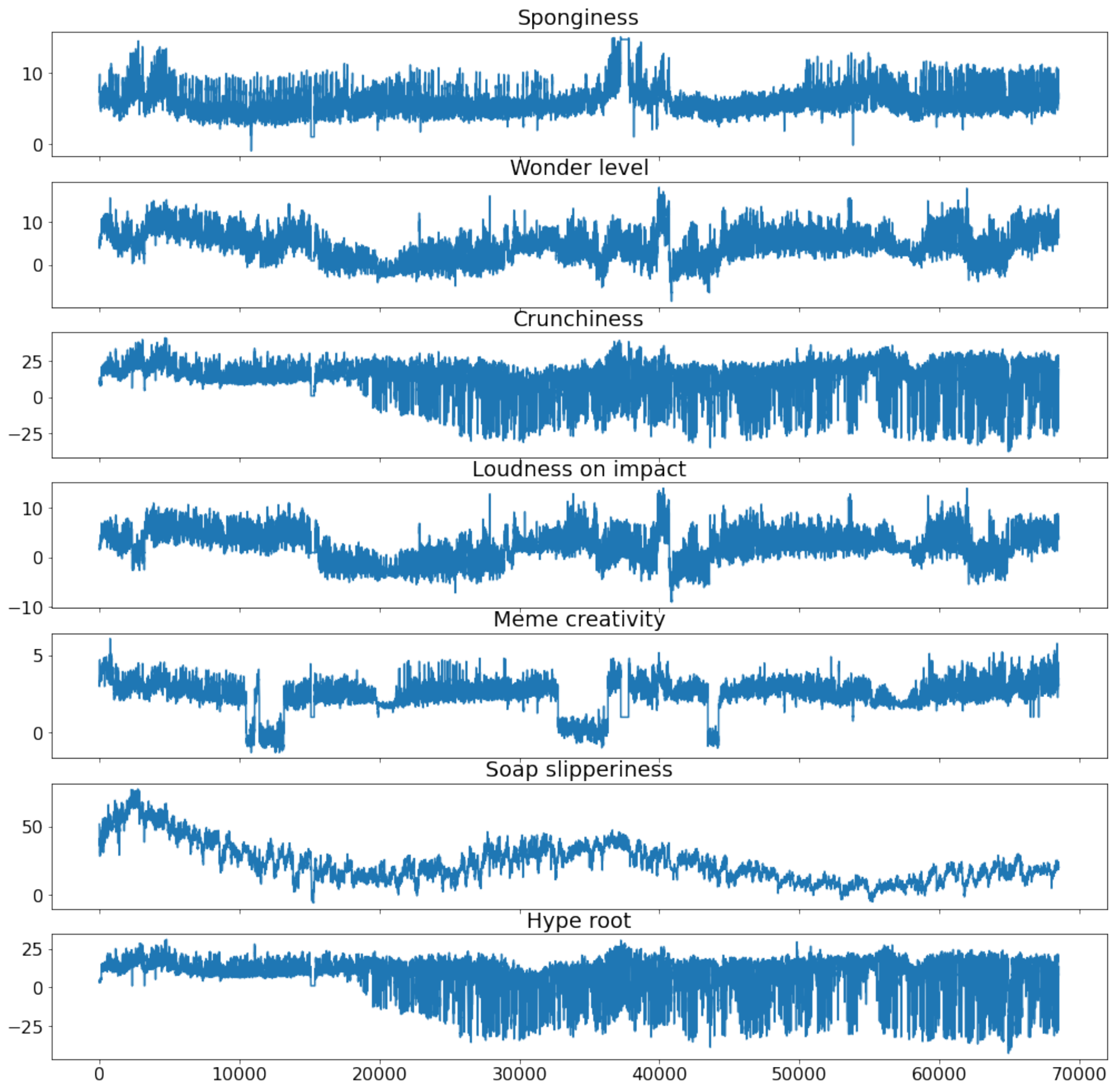


Figure 2: Timeseries values plot.

3 Architecture

The architecture of the network was shaped after the comparison of multiple trials, using different model settings and different types of layers. All of the final models that we delivered carried out some similarities. An example network will be examined:

3.1 LSTM Model

Most of the models provided were using the LSTM. In particular we found out the model that could best perform was bidirectional LSTM. We can see from the following figure that bidirectional LSTM (toghether with some other hidden layers that will later be examinated) had better results when used in our network. All the models we provided were trained using "validation_MSE" as evaluation metrics.

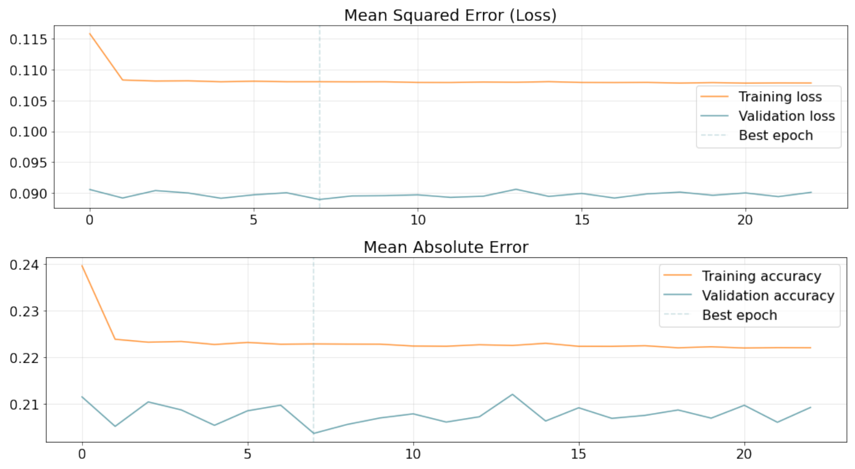


Figure 3: These results were obtained throughout a simple model using LSTM, uploaded on codalab as "SimpleLSTMModel.zip".

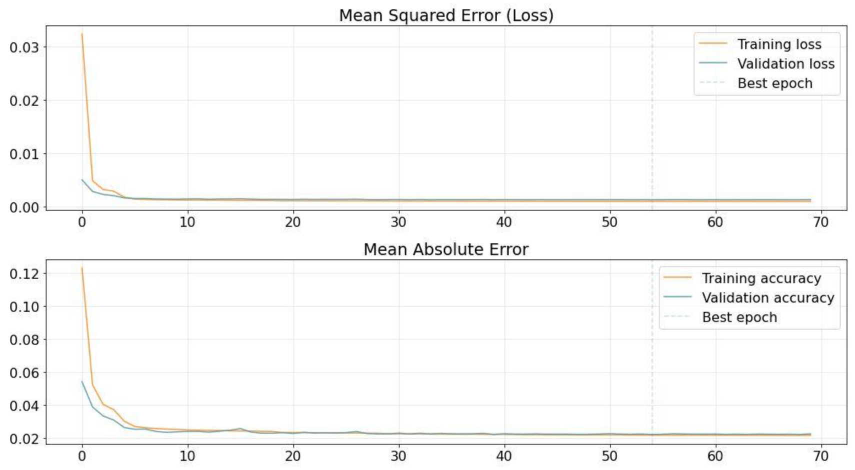
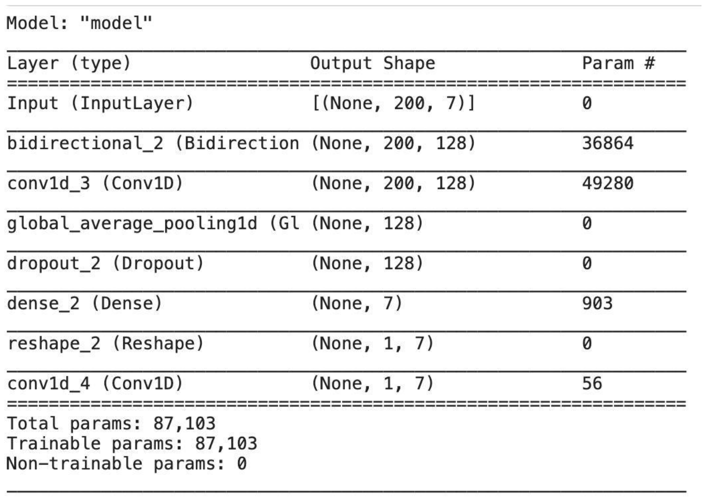


Figure 4: Trend of Loss and Absolute Error during epochs. ("SimpleAR_Stride1.zip" model).

After trying some different architectures we noticed the best results were obtained by the simpler models neither using deep networks after the LSTM nor using more stack of LSTM layers. The following picture represent the "SimpleAR_Stride1.zip" model



```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           [(None, 200, 7)]          0
_________________________________________________________________
bidirectional_2 (Bidirection (None, 200, 128)          36864
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 200, 128)          49280
_________________________________________________________________
global_average_pooling1d (Gl (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 903
_________________________________________________________________
reshape_2 (Reshape)          (None, 1, 7)              0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 1, 7)              56
=================================================================
Total params: 87,103
Trainable params: 87,103
Non-trainable params: 0
_________________________________________________________________
```

3.2 Callbacks

All the callbacks used were provided by tf.keras.callbacks

3.2.1 EarlyStopping

This function was very helpful to prevent overfitting. We also tried to use the Keras Tuner on the parameter of this callback but the result obtained were almost identical.

3.2.2 ReduceLROnPlateau

This callback monitors the evaluation metric and, if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced, until a minimum value.

4 Hyperparameter Tuning

During the development of our networks we tried to use the Keras Tuner. Even if it didn't lead to remarkable performances, we managed to test it on different parameters, such us learning rate, dropout probability and number of neurons of the LSTM layers. We also had the confermation of what was previously affirmed, in fact, when we learned whether or not adding other layers, the outcome was that the best networks were the simplest ones.

5 Conclusions

We've achieved good results with a simple autoregressive model, but to improve further our performance and to experiment other networks we also tried a transformer model.
Even though this network would have had better results, this didn't happen. The failure of this model is probably caused by some implementation errors in the transformer structure itself. One of our attempts implementation is provided in the notebook.
We could have tried a seq2seq model approach with attention layers, fitted for a time series forecasting problem.