REPORT AN2DL
Homework2

Tommaso Lucarelli, Emanuele Baldelli, Tecla Perenze
January 2022

Introduction:

In this homework, we were required to predict future samples of a multivariate time series. The goal was to design and implement forecasting models to learn how to exploit past observations in the input sequence to correctly predict the future.

We started by splitting the input data into training and test. The next step was to create and build a series of sequences by setting the values of window, stride and telescope parameters. By doing so, we managed to find a repetitive pattern inside the input data, that could be then exploited to predict the future.

Models:

As a base model, we used the model introduced during the exercise sessions.
The model was compose composed by:

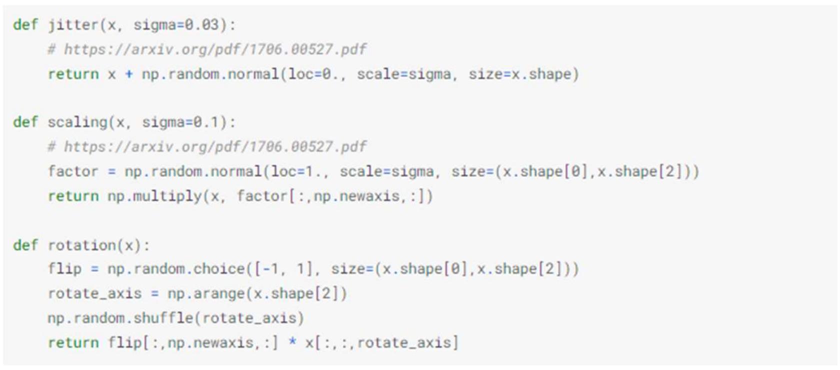
- Bidirectional LSTMs - RNN architecture with memory functions used for sequence classification.
- Convolutional layers with Max Pooling / Average Pooling, are used for pattern extraction.
- Dropout layer to avoid overfitting.
- Dense layer - Reshape - final Convolutional layer.

We tried to modify this model by using non-bidirectional LSTMs, removing the convolutional layers, and lastly adding some skip connections. By doing these changes we didn't manage to improve the best result on the online test obtained with the basic model [8.2].

Tuning & Data Augmentation:

After having noticed that the professor's basic structure was the best one, we decided to tune the different parameters with Keras Tuner. This allowed us to test different values for the parameters of both Conv and BiLSTM layers and also increase or reduce the number of layers. Unfortunately, the best values found with Keras Tuner didn’t give us the best results on the Codalab validator.

Another idea was to perform data augmentation on the data, in order to increase the quantity of data used for the training phase and to obtain better hyperparameters. We used different transformations (jitter, scaling, rotation, permutation, magnitutde_warp, time_warp, window_slice) to augment our data, but also this attempt was ineffective to obtain a better online result, even though some models actually improved after a retrain with augmented data.



```
def jitter(x, sigma=0.03):
    # https://arxiv.org/pdf/1706.00527.pdf
    return x + np.random.normal(loc=0., scale=sigma, size=x.shape)

def scaling(x, sigma=0.1):
    # https://arxiv.org/pdf/1706.00527.pdf
    factor = np.random.normal(loc=1., scale=sigma, size=(x.shape[0],x.shape[2]))
    return np.multiply(x, factor[:,np.newaxis,:])

def rotation(x):
    flip = np.random.choice([-1, 1], size=(x.shape[0],x.shape[2]))
    rotate_axis = np.arange(x.shape[2])
    np.random.shuffle(rotate_axis)
    return flip[:,np.newaxis,:] * x[:,:,rotate_axis]
```

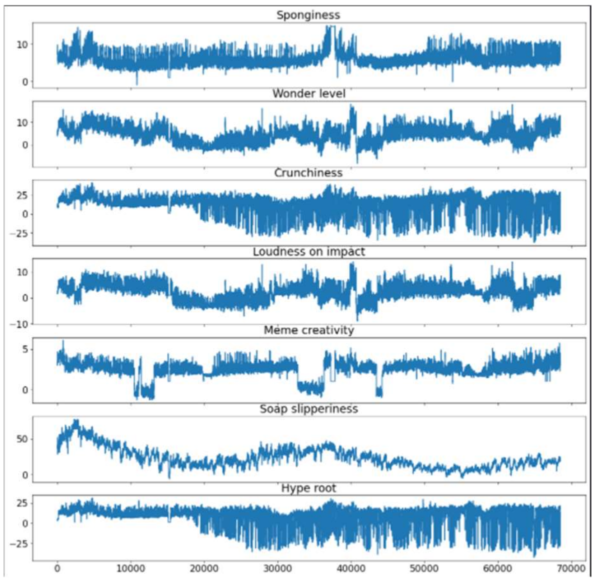
Figure 1 example of transformation

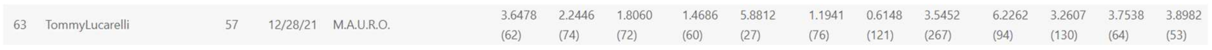
Autoregressive:

Finally, we tried to predict with an autoregressive strategy. So, instead of predicting the entire sequence of 864 values, we started predicting just 1 value at a time, by sliding the window and predicting the next value, exploiting already predicted values as input for further predictions. This process is repeated until the model has produced 864 values. In this way, we had our first improved result on the online test [4.2].

At this point it was clear for us that a trade-off between predicting 1 and 864 values could be the best solution. Therefore, we started trying different values for window and telescope (sequence generation), testing them both locally and online. Eventually, we obtained the best result on the online test [3.64] with window=200 and telescope=27.

Dataset Analysis:





63	TommyLucarelli	57	12/28/21	M.A.U.R.O.	3.6478 (62)	2.2446 (74)	1.8060 (72)	1.4686 (60)	5.8812 (27)	1.1941 (76)	0.6148 (121)	3.5452 (267)	6.2262 (94)	3.2607 (130)	3.7538 (64)	3.8982 (53)

Figure 2 Our results on the online test. The labels are in the same order of the above graphic

Analyzing the dataset we understood that the “Crunchiness” and “Hyperoot” series would be the most difficult to predict, because they are very variable and dense. Therefore, in order to improve the results, we tried to rescale and normalize the dataset. Normalization in particular helped us to achieve a good result for those two series compared to the other teams.
It is worth highliting that our worst performances, compared to other teams, came in the "Meme Creativity" and "Soap slipperiness" series. Infact the model had problem in understanding more complex patterns, that eventually occur in the series, beacuse probably the window we chose was too short.

Conclusions:

After carrying out this project we realized the potential of the RNN in exploiting the memory to make predictions about the future.
Achieving good results was not complex, a few tricks were enough. The main problem occured when it came to refine and improve those results: we have made several attempts, but only some of these have had a positive response. In addiction another challenging problem was tuning all the parameters that changed every time a modification of the implementation choise was made. That is due by the fact that the local test did not give many guarantees, despite the graphical representation of the prediction gave a big help in identifying the strengths and weaknesses of the various models.