ANNDL 2nd CHALLENGE REPORT

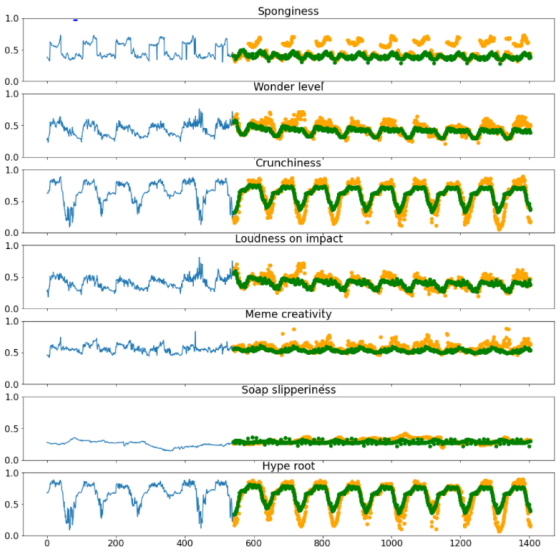
Team ColleDellOrso
Alberto Tavini 10578559 - Simone Reale 10616980

First Trials

The first thing we did was explore the dataset and have a look at different sequences after applying the train test split on them, in order to extrapolate some “expert” knowledge that could be useful for approaching the problem. We noticed that a window with around 600 datapoints had interesting features: it would show enough information about the cyclic behavior of the attributes while not being excessively long (and as a consequence, repetitive). So at the beginning we explored the space of parameters in that surroundings.
We used in the majority of cases models that had a paired combination of conv1D and Bidirectional LSTMs that had the same number of units, so that our network would be able to grasp the trends in the data series both from a temporal and a “visual” perspective. When the number of layers of each type didn’t match the network seemed not able to learn. More than a couple of each would have too many parameters and overfit, while exactly that number seemed to be the best compromise between capacity to learn something more than the average and the ability to generalize. After some initial trials Autoregressive models seemed more promising so we stuck with those. In particular we noticed that pairing stride and window size brought better results. Also a kernel size matching those values in certain cases ameliorated the results.
Apart from Modelcheckpoint and Tensorboard to monitor evaluation metrics and not lose data about training, we applied EarlyStopping and Reduce Learning Rate on Plateau to optimize the learning procedure.

Some thoughts

We tried training different models for predicting different features, but that did not lead to any improvement, as models that performed well on a feature appeared to do the same on all of them.



Changing the Optimizer to ones that looked promising on a theoretical basis, like Adadelta, Adagrad or SGD, worsened the results with respect to Adam.
We thought about setting *use_bias* in the conv1d to False but it worsened results.
We also tried to implement a batch normalization layer, but its usefulness has been proved to be minimal, probably because the dynamic range of the input variables at each layer was already restricted.
Dropout at 0.3 seemed to be the best compromise between consistent learning and contrasting overfitting. After a few rounds varying parameters it looked like the best model

we could find was one which had 540 window, 12 stride and telescope, with 256-512 units for the paired Conv1d and Bidirectional, 12 kernel size, with Max poolings in between,and GAP at the end of the network. Above an example of its predictions.

Attention is all we would have needed!

Not satisfied with the results found with these models, we tried to look if the implementation of transformer that we had seen at the latest exercise session had a pretrained for data series forecasting, but scrolling the dozens of models present on the library's documentation we weren't able to find one. So we found a paper done discussing a similar task (https://medium.com/mlearning-ai/transformer-implementation-for-time-series-forecasting-a9db2db5c820) and we tried to decipher their implementation to see if a tfk compatible version was possible to develop, but it looked way too different from the things we had seen (Still an interesting lecture). More details about this can be seen at the end of the 2ND HM Transformer notebook.

Ensemble methods

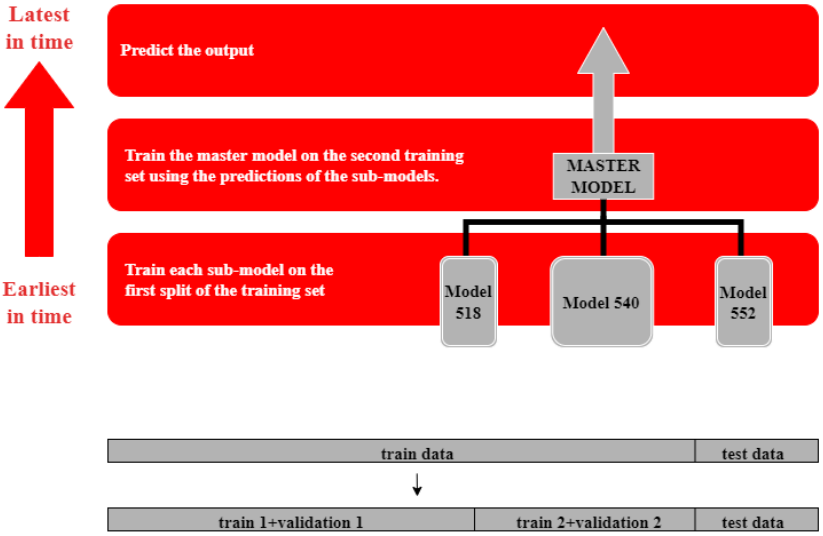
After countless attempts with autoregressive models we started to look forward to the fascinating world of model ensembling considering that we had interesting results in the previous challenge by applying *bagging*. We resorted to this technique in order to escape the impasse we found ourselves in after trying dozens of different model/parameters combinations.
A rudimentary first version of the model includes three distinct sub-models trained with different 'window' parameters but same stride and telescope, we chose to use 528, 540 and 552 as window and 12 as stride and telescope.
The idea is to achieve better results by mediating the prediction of the three different sub-models (we give more weight to the central one, which yielded the best results up to that moment), the image provided will explain in practice how we managed to do all that in the model.py file.

```python
class model:
    def __init__(self, path):
        self.model1 = tf.keras.models.load_model(os.path.join(path, 'SubmissionModel1'))
        self.model2 = tf.keras.models.load_model(os.path.join(path, 'SubmissionModel2'))
        self.model3 = tf.keras.models.load_model(os.path.join(path, 'SubmissionModel3'))

    def predict(self, X):
        # MODEL CONSTANTS
        NUMBER_OF_PREDICTIONS_TO_DO = 864
        TELESCOPE = 12
        WINDOWS_SIZE1 = 528
        WINDOWS_SIZE2 = 540
        WINDOWS_SIZE3 = 552
        # Conversion of the tensor to a numpy array
        X = X.numpy()
        # Computation of minumum and maximum
        X_min = X.min(axis=0)
        X_max = X.max(axis=0)
        # Min max rescaling
        last_window1 = X[-WINDOWS_SIZE1:]
        last_window1 = (last_window1 - X_min) / (X_max - X_min)
        last_window1 = np.expand_dims(last_window1, axis=0)
        last_window2 = X[-WINDOWS_SIZE2:]
        last_window2 = (last_window2 - X_min) / (X_max - X_min)
        last_window2 = np.expand_dims(last_window2, axis=0)
        last_window3 = X[-WINDOWS_SIZE3:]
        last_window3 = (last_window3 - X_min) / (X_max - X_min)
        last_window3 = np.expand_dims(last_window3, axis=0)
        out1 = np.array([])
        for reg in range(0, NUMBER_OF_PREDICTIONS_TO_DO, TELESCOPE): ...
        out2 = np.array([])
        for reg in range(0, NUMBER_OF_PREDICTIONS_TO_DO, TELESCOPE): ...
        out3 = np.array([])
        for reg in range(0, NUMBER_OF_PREDICTIONS_TO_DO, TELESCOPE): ...
        # Min max descaling
        out1 = out1 * (X_max - X_min) + X_min
        out2 = out2 * (X_max - X_min) + X_min
        out3 = out3 * (X_max - X_min) + X_min
        out = (out1 + out2*2 + out3) / 4
        out = np.reshape(out, (NUMBER_OF_PREDICTIONS_TO_DO, 7))
        out = tf.convert_to_tensor(out)
```

The results were not that different from the one of our best simple models. But we thought we could do better.

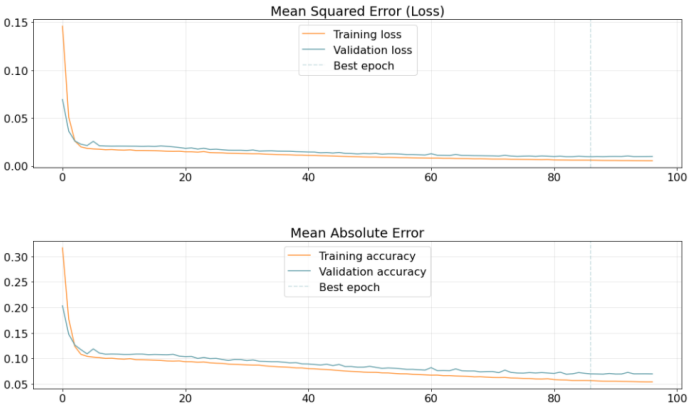
Stacking



A much more sophisticated way of doing this would be to stack another neural network (a standard ffn actually) on top of the other three with the aim of discovering the correct weight to give to each one of the sub-models when it comes to predict targets, so applying *stacking*.
We quickly found out that in this case splitting the dataset is a pretty complicated task and that in order to achieve good results the amount of data needed is way way bigger than that requested by a single model, so we gave up and turned our attention back to simpler models. A graph we plotted better illustrating this idea on the side.

Final Discoveries

A little bit down in morale because it seemed we could not reach better results, we were scrolling through keras documentation when we found KerasTuner, through which we did some hyperparameters research about the layer's units and kernel sizes for some new configurations, where we dared increase the telescope to higher numbers, that seemed unpromising at beginning. (More at Hyperparameter Search Notebook).
With these we finally found a model which improved our RMSE, gaining us our best result of 4.24. Model used 200 window, 20 stride and 144 telescope, with 256-512 Bidirectionals and 512-512 Conv1Ds with kernel 8. On the side the MSE and MAE plots.
The prediction results were very similar to the ones shown for the first best solution we found.



Conclusion

This homework made us once again realize how even the slightest shift in parameters can cause noticeable changes in performance: patience and trial and error are key in facing such fascinating tasks, no result should be considered as definitive and one should never think about any choice as bad before giving it a try.
We're both looking forward to using the knowledge about NN's we developed here in the new stimulating challenges the world will present to us!