Artificial Neural Networks and Deep Learning - Homework 2 Report

Alberto Aniballi, Lorenzo Campana, Michele Pio Prencipe (Group: Alan Tuning)

December 22, 2023



# Dataset Inspection

Our first step in this assignment was to examine the dataset to determine the nature of the problem we were trying to address, whether there was a disparity in the classes and whether there was some anomalous value that would interfere with our model’s ability to learn. The objective was to develop a forecasting model capable of using historical time series data to make accurate future predictions. In particular, our dataset was composed of 48000 not correlated time series, for this reason the main challenge for the model was to be capable of generalizing in the context of input belonging to different domains. Each time series belongs to a specific category within a group of 6 categories. By creating plots to graphically visualize the numerosity of the categories belonging to the 6 different categories, we immediately noticed that there was a category in the clear minority: category 'F' had only 277 time series, while the others were around 10000 values, apart from category 'A' which had around 5700 values. Each time series belongs to a specific category within a group of 6 categories. By creating plots to graphically visualize the numerosity of the categories belonging to the 6 different categories, we immediately noticed that there was a category in the clear minority: category 'F' had only 277 time series, while the others were around 10000 values, apart from category 'A' which had around 5700 values. Furthermore, again using the graphical power of plots, we saw that our time sequences were characterized by significantly different lengths: from the longest with 2776 values, to the shortest with less than 50 values. To overcome this problem, we subsequently performed data augmentation both from a perspective of increasing the temporal length of the series, and from a perspective of increasing the numerosity of the sequences in certain categories, e.g. by jittering or scaling the series already present. The time sequences already had values scaled between 0 and 1, yet we tried using a further scaling technique called Robust Scaler which uses the interquartile range, the difference between the third quartile and the first quartile, to scale the values. However, models driven on data that underwent a transformation through Robust Scaler did not perform significantly better. We then studied the stationarity of the data, including analyses on seasonality, trends and residuals. With regard to the study of the series trend, we analyzed the autocorrelation function 'ACF' and the partial correlation function 'PACF'. The Augmented Dickey-Fuller test revealed that most of the series were non-stationary and therefore it was not possible to exploit stationarity conditions to our advantage. Decomposing the series into seasonality, trend and residuals we would then create three models to predict each of these and then sum the different components.

1. **Anomaly Detection**

To prevent the predictive capabilities of our model from being worsened by anomalous or erroneous data, we decided to include analyses of anomalies or outliers in our work. In particular, three outlier detection techniques were performed to evaluate their goodness-of-fit with respect to our dataset: Z-score, Isolation Forest, Local Outlier Factor. The time series were studied independently of each other so that time sequences belonging to different categories could not influence each other. The first technique is based on calculating the Z-score for each point in the time sequence, this metric calculates the distance of a data point from the mean and divides it by the standard deviation. Values that are more than 3 standard deviations away from the mean can be considered suspect. In order to apply this analysis, it is necessary that the data follow a normal distribution, otherwise the validity of the result is lost, so we performed two tests to find out whether the data in each series had a normal distribution: Shapiro-Wilk and Kolmogorov-Smirnov tests. Most of the time series in our dataset were not able to pass the above two normality tests, so in order not to draw nonsignificant conclusions, we decided to identify outliers using the Z-Score. Wanting to continue the analysis on outliers, we implemented two additional unsupervised learning techniques. The Local Outlier Factor algorithm uses the notion of local density deviation of a point from its neighboring points. Points that have a significantly lower density than their neighbors are considered outliers. The Isolation Forest algorithm attempts to isolate outliers from our of the dataset by creating an ensemble of decision trees. This algorithm randomly selects a split value to create partitions within the data until the various points are separated, the intuition behind the Isolation Forest algorithm is that outliers data are more easily isolated and therefore will have a shorter path between the root and them than normal data which precisely are more difficult to separate. Both unsupervised methods require the inclusion of a parameter called 'contamination' that tells the algorithm a congruent value of outliers that might be hiding in the data. Although the time series did not have a proper Gaussian distribution, we decided to use the number of data with Z-score greater than 3 as an approximate value at which to set the contamination parameter of the two unsupervised methods mentioned above. We noticed that the two methods identified outliers in common with each other, often also common with the method using the Z-score, however, the outliers were almost never points completely isolated from the others but rather small sets of points were identified as suspicious. Since we did not have a thorough understanding of the domain and semantics from which the data came, we decided not to discard these short sets of points identified as anomalous since they were not isolated points and could still represent plausible points.

# Data Augmentation

Data augmentation is often a relevant step to improve the performance of models as it may make the model visit previously unexplored areas of the input space and at the same time reduce overfitting. A small amount of data is often not enough to properly learn many of the parameters that populate deep neural networks. One solution to this problem could be to collect new data, however, in addition to being an expensive task in terms of time and money, it was not really a viable solution in our case. For this reason, we tried using three different methodologies to augment the dataset: scaling, jittering and data interpolation. Scaling allows us to change the magnitude of the data by multiplying it by a scalar, the idea being however not to drastically change the overall shape of the time series. In our case, we multiplied the time series by a vector of equal length containing numbers generated by a normal distribution centered at 1 and a standard deviation of 0.1 or 0.2. Jittering was instead used to introduce additive noise at each value, this noise was generated by a Gaussian centered in 0. For the standard deviation we tried the values 0.01,0.02 and 0.03 so a reduced value in order not to distort the original sequence too much. With these two techniques described above we tried to increase the numerosity of the dataset, while with data interpolation we tried to increase the length of the individual series. With the aim of not introducing too much misleading data between one dataset and the next, we decided to use a simple interpolation technique: we took two consecutive datasets, calculated the mean value and added it between the two. By using this interpolation we approximately doubled the length of each series, so by doing so we were able to keep in our training dataset those sequences that initially seemed too short and avoid the risk of losing such information. The validity of this reasoning was then also found by an improvement in the performance of the model: before applying the interpolation on CodaLab we reached a value of about 0.013, after increasing the length of each series we were instead able to break the 0.010 barrier reaching a result of 0.0073 on CodaLab.

1. **Models Development**

The first model we built was relatively simple and consisted of the following layers: a bidirectional long short term memories layer with 128 units, a 1D convolutional layer with 256 filters and a kernel with size 5, and finally an additional 1D convolution layer to make our model output the correct shape. In order to limit the problem of overfitting, we clearly applied the Early Stopping criterion to the training of our model, making it stop the learning phase if there was no improvement in the validation loss for more than 12 epochs. Being an early model, we did not expect it to achieve remarkable results, in fact it approached a test loss of around 0.1 locally. After this first model we decided to create deeper models by combining various convolutional blocks and long short term memories, however we realized that although the model was much deeper than the initial model, after a few epochs the neural network could no longer learn. For this reason, we decided on these blocks formed by convolution and LSTM with the residual connection technique. Residual connections in fact help to combat the problem of vanishing gradient by using direct connections, called shortcuts, which do not go through all the layers, so that the gradient is somehow preserved. This approach using residual connections led to the creation of models that achieved values of 0.013 and 0.010 on CodaLab's hidden test set. After trying different configurations of learning rate, batch size, window length, and tuning of our model's layer parameters, the results obtained locally on our test set seemed promising however were not accompanied by as much improvement on the hidden test set in CodaLab. Therefore, we thought the problem was caused by overfitting our model and implemented data augmentation techniques by introducing new variability to our time series. As described in Section 3, we applied three different augmentation techniques, the most effective of which was the lengthening of the sequences through the use of data interpolation that allowed us to achieve a value of 0.0073 on CodaLab. Once we introduced this new data and obtained the new best result, we again tried to increase the complexity of the model by increasing its depth, either by inverting the position of the convolutional layers with respect to the layers containing LSTM, or by introducing parallel branches, for example: 3 parallel branches exploiting convolution, normal LSTM, bidirectional LSTM and a final concatenation layer. Despite these efforts directed toward the complexity of the model, the results did not seem better, indeed in many cases the performance was worse than in simpler models we had used previously. After this evidence, we changed our approach by returning to simpler models but exploiting a new type of layer: attention layer. After seeing the Attention mechanism in class, we immediately perceived the potential of this technique and included an attention layer before each layer of Long Short Term Memories, at the end of the blocks containing residual convolution and connection. This new approach allowed us to achieve a result of 0.006 in CodaLab, making us realize that creating a deeper model does not always help but sometimes it is simply necessary to make sure that the existing model is able to identify which elements of a series are most relevant in order to focus one's attention on these parts at the time of prediction and at the same time reduce the influence of irrelevant or redundant information. Despite the latest improvements, we attempted to achieve new performance peaks by combining the Attention mechanism with the Masking mechanism. This layer makes sure that the learning process is not influenced by tokens that in fact do not incorporate information, combined together with Attention allows us to focus the computational capabilities of our model only on the actual relevant parts of the series. We achieved our best result 0.0056 on CodaLab using data augmentation, attention, masking, convolution, and residual connections. During the final phase, we implemented an autoregressive forecasting in model.py to predict 18 values and no longer 9 as we did for the first stage, reaching a final result of 0.0112 on CodaLab.

## CONTRIBUTIONS

**Alberto Aniballi**

-Led the dataset inspection phase and implemented outlier detection technique

-Contributed to the development of residual connections and convolutional models

-Collaborated in the implementation of parameter tuning of various training models

-Structured the report, ensuring a clear presentation of the workflow

-Performed data augmentation

**Lorenzo Campana**

- Coordinated the organization of the team

- Studied time series decomposition and stationarity using statistical tests

- Implemented masking to exploit an higher number of batch sequences

- Collaborated in the design of a ResNet-like RNN model with attention

**Michele Pio Prencipe**

-Participated in the initial dataset inspection

-Implemented a version of Resnet Model with Attention

-Engaged in the analysis of the competition results, performing some hyperparameter tuning.

-Performed interpolation for the data augmentation part.

## 