Meta learning for time series with missing values

A baseline approach

Author:
Miguel Ángel Caldas
279180

Supervisors:
Rafael Drumond
Dr. Lars Schmidt-Thieme

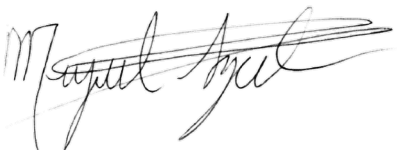
16th September 2020

Thesis submited for
Master of Science in Data Analytics

Wirtschaftsinformatik und Maschinelles Lernen
Stiftung Universität Hildesheim
Universitatsplätz 1, 31141 Hildesheim

Statement as to the sole authorship of the thesis:

META-LEARNING FOR TIME SERIES WITH MISSING VALUES. I hereby certify that the master's thesis named above was solely written by me and that no assistance was used other than that cited. The passages in this thesis that were taken verbatim or with the same sense as that of other works have been identified in each individual case by the citation of the source or the origin, including the secondary sources used. This also applies for drawings. sketches, illustration as well as internet sources and other collections of electronic texts or data, etc. The submitted thesis has not been previously used for the fulfilment of a degree requirements and has not been published in English or any other language. I am aware of the fact that false declarations will be treated as fraud.



16.09.2020, Hildesheim

Abstract

Time series prediction is a field with many real-life applications. However, real-life data is significantly different than the usual datasets. There are many missing values, the time series do not have the same length and the number of dimensions is also different. Our approach is to use meta-learning to pre-trained models that perform accurately in time series data with missing values to have k-shot an even zero-shot meta-learning models for time series. The results were tested on multivariate as well as univariate time series. For multivariate time series, we failed to obtain conclusive results in the Phisyonet dataset, but for univariate time series with missing values, we managed to get our GRU with skip connections model to perform at a baseline level with zero-shot meta-learning on the M4 dataset.

Keywords: Meta-learning, Time Series, GRU-D, MAML

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Time-series prediction is a very useful field in the enterprise environment. It reflects information of one or many variables that change constantly as time advances, like financial data, water-consumption data, traffic data, etc. Many models perform well in time-series, and they vary depending on attributes of the time-series like periodicity and trend. However, a problem most time-series have and most models do not handle properly are missing values and irregularly spaced time-series.
A single time-series consists of the time variable and the dependent variables. These can be one for univariate or more for multivariate time-series. The latter is usually the most complex of both of them given that the variables usually have dependencies between them that affect the prediction. All problems aim to predict a variable from the time-series or a function described by the combination of all its variables.

$$y = F(v_1^{t_1}, v_2^{t_2}, ..., v_{n-1}^{t_{n-1}}, v_n^{t_n})$$

The prediction y is depicted by a function that depends on all the variables that are time-dependant ($v_i^{t_j}$). All models that are built to predict time-series try to find the function F or approximate it as close as possible. This usually needs a big enough dataset for the model to learn the behavior of a time-series. In real-life, this kind of information is not as organized and complete as in public datasets. Businesses need to collect the information in real-time which more often than not ends up missing information or the complete lack of it. Having missing entries within the time-series requires a new model that is equipped to deal with this case. Usually with imputation methods in order to have artificial data that affects as minimum as possible the outcome of the model.

The first naive approach towards having missing and/or not enough data would be using meta-learning. Teaching the model with some pre-training how to learn faster the behavior of the time-series. This would end up in a model learning properly even with the limitations previously described. However, the problem with this naive approach is the differences between the time-series. This method resembles how humans learn to do tasks. The example used in the MAML paper[4] was that if a person sees one dog, or last one breed of dog, that person will always recognize that breed of dog. When meeting a new breed of dog, the person will get used and recognize this dog in a significantly faster way. The same applies to a model with meta-learning. Normal models use thousands of examples to identify one dog, yet when the model needs to learn a new dog it requires again thousands of iterations. Therefore with meta-learning, the model has the capacity to learn a new dog in significantly fewer iterations. Regarding the number of variables that depend on time, they are required to be the same, for the model to be pre-trained. That inevitably forces the solution to create more artificial data or to erase it in order to make the dimensions match between the different time-series datasets. To the best of our knowledge, there are no approaches towards handling this issue other than padding and or truncating time-series in order to have matching dimensions and being able to use the MAML framework.

When the time-series have the same dimensions coming from different datasets is usually for univariate time-series. That is why we found existing work handling meta-learning for univariate time-series[2]. The results are promising and the authors leave as future work experiments on multivariate time-series and time-series with missing values. Our experiments will handle both scenarios jointly and separately while testing if our model can perform at the same scale of success as N-BEATS[2] on the same dataset. For this, we use two different datasets. Phisyonet[5], a multivariate time-series with missing values for binary classification, and M4[6], a univariate time-series for forecasting.

The model we will use is inspired by GRU-D[3], a variation of a GRU network that uses a pre-processing function to create two extra features as input for the model. These new inputs inform the model of the missingness within the time-series. A *masking* vector of binary-nature; that contains a one if the entry is not a missing value and a zero in the opposite case; and a *decay* vector that informs the GRU gate for how much time has passed since the last missing value (based on the timestamp).

In this work, we create three different experiments to deal with the meta-learning idea of time-series, both in a multivariate as a univariate environment. We use our model of a GRU-D[3] with skip connections architecture that is proven to be good in predicting time-series with missing value; and then we propose to combine our implementation with the MAML framework[4] to perform k-shot meta-learning with multivariate time-series and zero-shot[2] meta-learning with univariate time-series.

Chapter 2

Related Work

To the best of our knowledge, there is no significant literature done on meta-learning for times series with missing values. Some literature was found but no literature from a relevant conference of machine learning. Since our baseline is MAML[4] and it was published in 2017, we filtered our search from 2018 until today. We browsed NIPS, IEEE, ECCV, AAAI, and ICLR. Research on the field of meta-learning has been done mostly in image processing. Most of these publications have the same goal in mind. Get an optimal initialization of the weights to have a fast-learning model.

Research on the field of meta-learning has been done mostly in image processing. Most of these publications have the same goal in mind. Get an optimal initialization of the weights to have a fast-learning model. The closest paper found in NIPs used meta-learning with relational information for short-sequences. This paper focuses on capturing the underlying pattern of relationships among the points distributed in different short sequences. Using a hierarchical Bayesian the authors outperform existing methods when predicting numerical future events within the sequence. However, this solution was designed for short sequences and the authors do not have experiments in a different set up like a long sequence or a sequence with missing information[16].

The meta-learning variants we want to focus on are those that allow the model to have few-shot meta-learning. The premise of our study is that in real-life data we do not possess enough information to train the solution for every new task; therefore the solution should be able to be trained with as few data as possible. Average Regret-Upper-Bound Analysis (ARUBA) uses transfer learning and a new online optimization process to learn over the losses stored when performing the meta-learning task. Per every task used

on meta-training there is a process in which the model regrets over the losses to archieve the optimal weights among all tasks. This work is done in image processing and univariate sequence forecasting[17].

In a similar research, we found FFORMS (Feature-based FORecast-model Selection), a study that uses meta-learning for univariate time series forecasting with a wide variety of models. In the study, the authors show the performance of the various statistical and machine learning models and compare their results. The authors apply meta-learning through a random forest that identifies the best model using the time-series features. This work is useful to show the advantages of different models according to the characteristics of the time series, like seasonality and trend[18].

2.1 N-BEATS

We decided to go for baselines both for the time-series with missing values problem, as well as for the meta-learning problem. As mentioned before, to the best of our knowledge, we do not have a specific baseline for meta-learning for time-series with missing values. However, there is a meta-learning model that is applied to univariate time-series[8]. This paper has very promising results in the area with numerous experiments. The model is called N-BEATS and it is an ensemble architecture of feed-forward networks with stacked residual blocks of forecasts[8] that is built in to be a Zero-shot meta-learner on univariate time-series. The difference to regular residual blocks is that N-BEATS does not have all the backcasts in its blocks. But a simple skip connection that resembles the forget gate of an LSTM architecture as seen in Figure 2.1. It is known that the forget gate that connects to the input of the LSTM cell, is what makes them so useful for sequence data, especially long sequence data[1]. This feature allows this kind of RNN to avoid the vanishing and exploding gradient issue.

N-BEATS was published as a baseline for time-series problems, but also, a second paper was done in which the authors show that using transfer learning the model can be a Zero-shot learning model for univariate time-series[2]. This makes it a very good paper to take into consideration for this study, even though it is not availed by a major conference. From this technique, we used mainly the experiments part where they show a zero-shot learning model in the M4 dataset[6]. There we used our model to try to replicate the results achieved by N-BEATS. An important observation for N-BEATS is that as future work, the authors consider whether their model is also use-

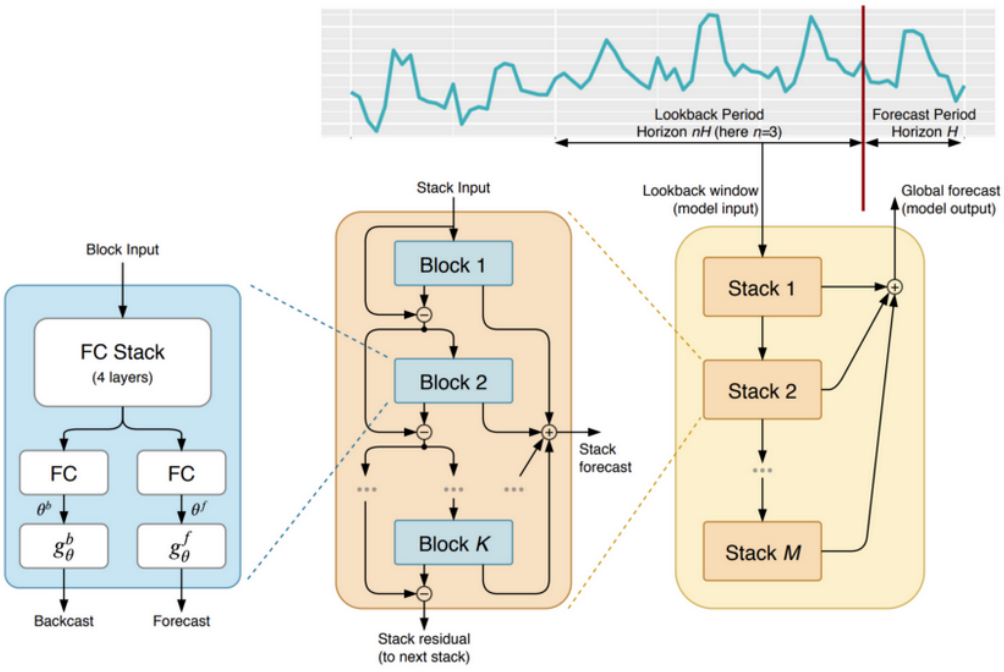


Figure 2.1: N-BEATS Residual block. Source [8]

ful in the multivariate time-series environment (which is a main part of our work).

2.2 GRU-D

As mentioned above, N-BEATS targets meta-learning for univariate time-series forecasting, therefore we require a model that solves the missing values within a time-series. The work we focused our research on, is an implementation of a pre-processing function and input augmented input of a GRU network. GRU-D[3] is a GRU network that tackles missing information in multivariate time-series using a pre-processing function to the input of the GRU gate. The authors declare that missing information is information on itself. Referred as information missingness, the authors try to input an analysis of this missingness in the GRU gate by adding two features to the time-series. These are a masking vector that informs whether there is data or not in the time-series. And a decay vector that carries the information of the timestamps of the entries based on the mask. The authors describe the

problem as:

$$X \subset R^n \text{ The original input}$$
$$M \subset R^n \text{ The masking vector of the original input}$$
$$\Delta \subset R^n \text{ The decay vector}$$
$$m_t^d \in M,\ \delta_t^d \in \Delta$$

$$m_t^d = \begin{cases} 1, & \text{if } x \text{ is observed} \\ 0, & \text{Otherwise} \end{cases}$$

$$\delta_t^d = \begin{cases} s_t - s_{t-1} + \delta_{t-1}^d, & t > 1, m_{t-1}^d = 0 \text{ Previous entry was missing} \\ s_t - s_{t-1}, & t > 1, m_{t-1}^d = 1 \text{ Previous entry was an actual value} \\ 0, & t = 1 \end{cases}$$

As shown above, for every possible input for the model, a masking and a decay vector are calculated. The masking simply is a binary codification of where there is missing data and where there are values for the model. While the decay vector is based on the timestamps of the original input which in the case of the paper[3] is the Phisyonet dataset[5]. This dataset has time-series data with their corresponding timestamps in seconds after the initial timestamp of zero. For any sequence dataset, it would be simply a time delta of one for every entry. The decay brings the information of the time passed between non-missing values in order for the model to now how much time it elapsed between actual values.
Figure 2.2 illustrates the architecture of the GRU-D model in which we can see how the masking vector and the decay vector are introduced into the GRU cell. Apart from these two new dimensions, the GRU gate is exactly as it is supposed to be. The idea behind the GRU-D[3] is to create a new method in which no data is imputed but rather the model receives as input which data is missing and for how long it has been missing. That is why the masking vector and decay vector are built for. Apart from the architecture of GRU-D, we are going to use the same dataset which is Phisyonet[5], a medical dataset that contains information of thousands of patients that were admitted into a UCI and whose vitals were taken in different timeframes. This dataset was part of a competition that GRU-D won as the most accurate model.
An example as described on the paper[3] of the masking and decay vector is the following:

X: Input time-series (2 variables) M: Masking for X
s: Timestamps for X Δ: Decay for X

$$X = \begin{bmatrix} 47 & 49 & NA & 40 & NA & 43 & 55 \\ NA & 15 & 14 & NA & NA & NA & 15 \end{bmatrix} \qquad M = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$s = \begin{bmatrix} 0 & 0.1 & 0.6 & 1.6 & 2.2 & 2.5 & 3.1 \end{bmatrix}$$

$$\Delta = \begin{bmatrix} 0.0 & 0.1 & 0.5 & 1.5 & 0.6 & 0.9 & 0.6 \\ 0.0 & 0.1 & 0.5 & 1.0 & 1.6 & 1.9 & 2.5 \end{bmatrix}$$

We can appreciate how the decay Δ is letting the model know for how long in each entry there has been missing values, using the values of the timestamps. And the masking plain tells where are the missing values in each entry too. With this approach, the model will have the information that the authors describe as informative missingness. This means that when a value is missing is not a problem but actually additional information for the model. Now, the model will know where values are missing from the time-series and for how long they have been missing.

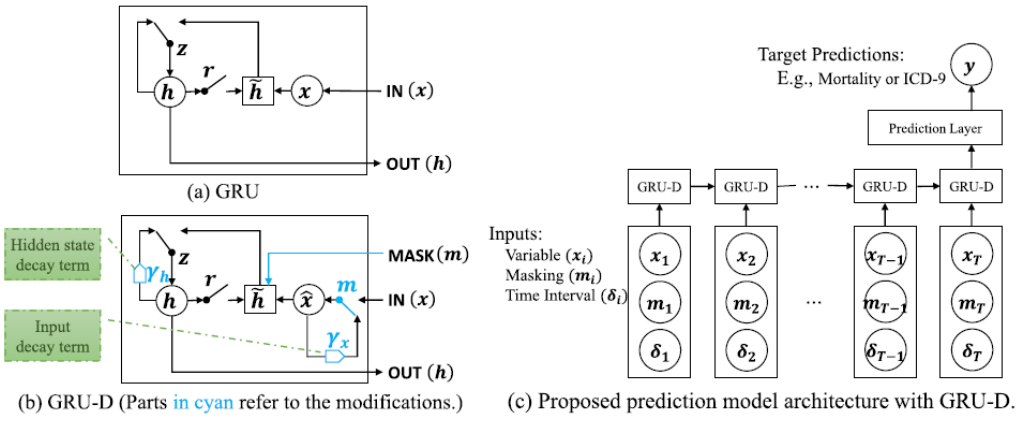


Figure 2.2: GRU-D cell with masking and decay inputs. Source [3]

2.3 MAML

After researching methods for handling missing values in multivariate time-series, we have decided which method to use for meta-learning. To date and the best of our knowledge, there are no papers on meta-learning for time

series with missing values. Model Agnostic Meta-Learning (MAML) is a framework that was designed to calculate the optimal initialization of weights by pre-training on multiple tasks. A "simple" yet promising algorithm that creates a baseline for meta-learning and few-shot training for any model that is trained with gradient descent. The main idea of this technique is to create different tasks to pre-train a model to get an optimal initialization of the weights for any kind of problem. Same as a human can learn to do a task and then use this knowledge to learn faster a completely new task, the idea of this technique is to give a machine learning model the ability to do the same. As mentioned above, real-life data is few and with missing information. Therefore, having an optimal initialization of weights that allows the model to learn through a few instances[4] and a model that can handle missing information properly[3] is all we need to make our experiments.
One of the most important assumptions this framework does is that for every new problem, it should exist numerous different problems that have similarities with it. With that in mind, any model can learn to do the other problems and use this knowledge to learn faster the new problem. This translated to machine learning means that the model will have a good performance using less data and fewer iterations in its actual training. Therefore, the paper is focused on k-shot learning, which means that a model will have k samples to learn how to solve the task.

As we can see in Figure 2.3, MAML takes the model's weights and trains

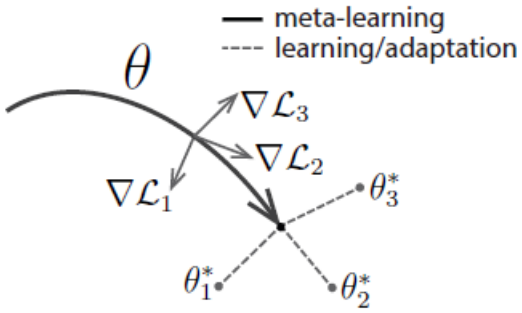


Figure 2.3: Diagram of MAML. Source [4]

them in different tasks. This will give optimal weights for each of those tasks and by the end of pre-training on all tasks, MAML will calculate the optimal initialization weights with all of the optimal task-specific weights.
The most problematic aspect of using MAML for time-series is that since the goal is to achieve the optimal weights for the initialization, then the tasks must have all the same dimensions. For univariate time-series is easy and it only requires to pad or truncate the sequence to match a window size in the

one variable. However, for multivariate time-series, it needs to be done in two dimensions. One for the length of the sequence as before, but another for every additional or missing variable between tasks.
X_u: Input univariate time-series X_m: Input multivariate time-series

$$X_u \subset R^{N \times M \times 1}, \qquad X_m \subset R^{N \times M \times P}$$

For X_u the only dimension that needs to be padded/truncated is M. Which is the window size.
For X_m the window size M needs to be padded/truncated as well, but also, every variable in P needs to be padded/truncated.

This will be explained in depth on the methodology section as the *curse of dimentionality*.
Tha MAML algorithm can be seen in the paper[4], however, here is without so much text in the algorithm.

Algorithm 1: MAML(T, α, β, K) :

Result: Returns the optimal initialization weights for the model
Θ random initialization of weights;
while *Not done* **do**
  T = (T˙1, T˙2,...T˙i) random batch of tasks of lenght k
  for $T\dot{i} \in D$ **do**
    $\theta_i' = \Theta - \alpha * \nabla_\Theta L_{T_i}(F_\Theta); L_{T_i}$
  end
  $\Theta \leftarrow \Theta - \beta \nabla_\Theta \sum_{T_i} L_{T_i}(f_{\theta_i})$;
end

The algorithm consists of two loops. The outer loop where the meta-update is done and the inner loop where the optimal weights for specific tasks are calculated. The inner loop begins with a random sample of tasks. Over these tasks, the initial weights Θ are trained for K iterations and store in an array of optimal weights per task $\theta = (\theta_1, ..., \theta_i)$. Then in the outer loop, the initial weights Θ are updated using the meta-update learning-rate β with the gradients of each optimal weights stored when calculated with its task. The result is the initial weights updated with all the optimal weights gradients of each task.

HYDRA

HYDRA is a meta-learning approach that evolves MAML by making a more robust solution. In this research, the authors prove that MAML is not robust enough to handle a dynamic target variable range; and that if added

in between, a "master neuron" could fix this issue. The purpose of this new component in the MAML framework is to learn a common ground for different numbers of outputs. It clearly shows its superiority t when compared to MAML as the number of target classes grows in an image classification problem. The accuracy of MAML drops significantly after a certain threshold; while HYDRA keeps a steady decrease in accuracy as the number of classes grows[19]. The mindset behind the idea of this paper could prove useful to manage the differences of dimensions on the multivariate time-series.

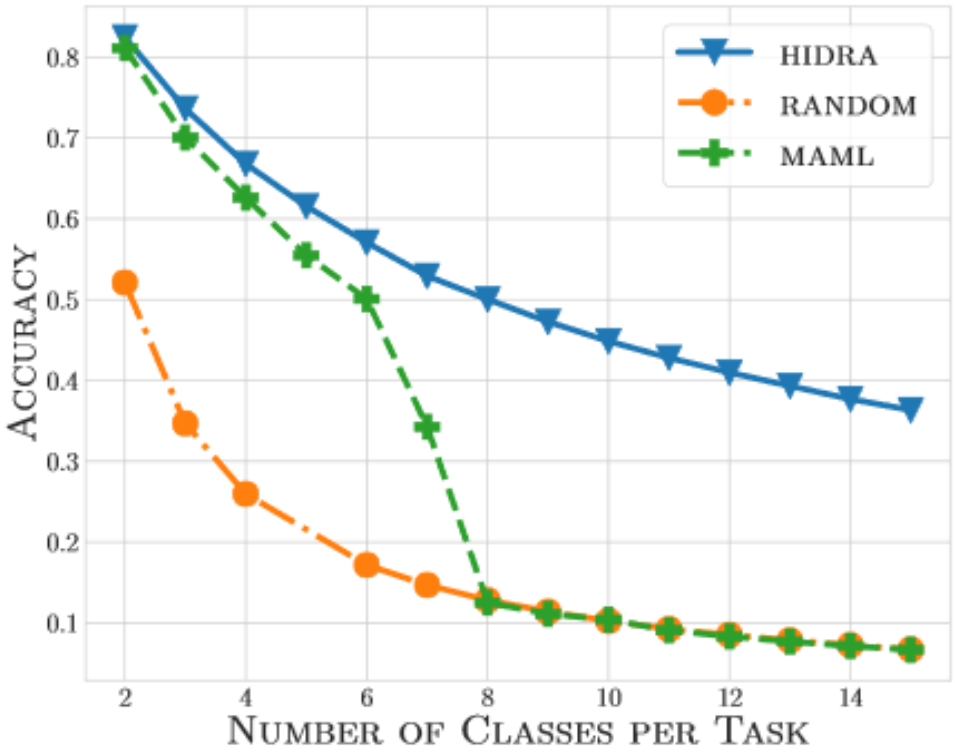


Figure 2.4: Accuracy of HYDRA compared to MAML as the number of outputs grows. Source[19]

Chapter 3

Methodology

Real-life time-series data is usually hard to gather and to store. This results in companies having enough data to train properly a model to make predictions out of the time-series data. On top of that, the gathered data is prone to having missing values within the time-series. These two problems do not share common solutions. MAML is a framework that gets the optimal initialization of weights for any model that learns through gradient descend. This allows such a model to learn with fewer data points and perform at the best level of performance. Imputation is a method often used when handling missing information on time-series. There are many variations for this method [1] with different pros and cons. A different approach than filling blanks with artificial data is using informative missingness [3][15]. Where the model is informed of a missing value instead of replacing the missing value with a made-up value. Our method proposes to merge these two solutions into one that can handle both problems.
The whole project is divided into two different experiments. The first one is to use our modified version of the GRU-D architecture to improve its performance in the multivariate time-series with missing values tasks, and then use the MAML algorithm to get K-shot learning with our model in this task. The second experiment is divided into two small experiments. The first one, is based on [2] and the idea is to achieve the same or better results in Zero-shot learning for univariate time-series forecasting. In summary, comparing our model to N-BEATS in the zero-shot meta-learning for univariate time-series forecasting. The second part is to test our model and the zero-shot meta-learning in time-series with missing values, which is an experiment that N-BEATS did not do. The most important part of this second sub-experiment is to measure the potential of using meta-learning as a solution for time-series forecasting with missing values, a topic that, to the best of our knowledge, has no significant literature up until today.

Since we did not find any baselines for meta-learning for time-series with missing values, we have to compare results on separated problems. We can compare meta-learning with N-BEATS yet, in the same paper, the authors mention that as future work they would like to try their solution in multivariate time-series. We use MAML with our implementation of GRU-D and use it on both univariate as well as multivariate time-series. On top of that, if a model is not designed to handle missing data, it will most likely have a decreased accuracy in this case. Our evaluation will be done in two parameters. The first one, the learning process. Both in univariate and multivariate, we will have baselines that will have access to the full training set of the corresponding dataset. Then these baselines will be compared to our model on the same testing data and we can directly compare the results using the same metric. The second one, the meta-learning process. We compare in the univariate time-series environment the performance of our meta-learning method against N-BEATS. If we can achieve the same results they have on their experiments. We can conclude that the experiments we perform on multivariate time-series with missing values would have similar results as N-BEATS.

3.1 Multivariate Time-series

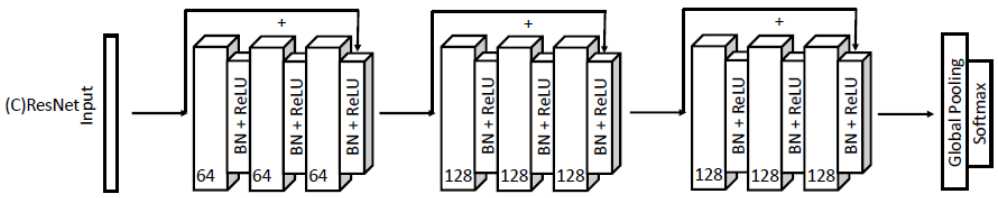


Figure 3.1: Resnet architecture. Source [7]

For these experiments to succeed, we first need a proper model that can work with time-series with missing values. Therefore, a first approach is to use GRU-D[3] to tackle the Phisyonet dataset[5]. After having that baseline we can focus on our own solution. Taken the Resnet architecture from [7] as inspiration we aim to take the best of both worlds and join them into one. From GRU-D we take the masking and decay vectors that allow the model to have informative missingness, and from Resnet we will use the skip connections between blocks that will allow our model to understand relations between distanced points within the time-series.

The Resnet architecture provides a deep study of why using skip connections between blocks of RNNs is beneficial for analysis of long sequences, allowing the model to understand long-time dependencies between different points in the time-series with a similar idea of how dilated kernels work in convolutional neural networks[9].
We can see in figure 3.1 how the authors designed their model so that the output of each block was passed on to the next in order to be added up to the sequence output. This paper got state-of-the-art results in 2017 for time-series problems and showed that this skip connections were helping the NN to learn dependencies between points further away within the time-series by giving the deeper layers of the network the unaltered input that the earlier layers were also seeing. As for our model, it has GRU blocks which by being types of RNN will not necessarily have the same success as the CNN blocks of [7], however, the same idea was published using regular RNNs in [10] back in 2018 and they achieved very good results by implementing the skip connection in an RNN architecture. Their idea, similar to us is to tackle the problems of RNNs when dealing with long time-series or sequences. Vanishing gradients and not capturing long term dependencies between points in the time-series. With the use of skip connections the results show that the model is capable of understanding these possible long term dependencies more often than the regular version without skip connections[10].
Our model is a Neural network with GRU modules that will receive as input both the masking and decay vectors additionally to the time-series. This is calculated before and added into one big input for the time-series. And also between the GRU-D blocks, we implemented skip connections that go to an addition of outputs of posterior layers. This way we have a perfect merge of two baselines that will help in the time-series tasks. After these GRU-D blocks with skip connections, the output goes to conventional Dense output layers.
For GRU-D[3], the dataset in which they made the experiments is only partially publicly available. This is because it belonged to a competition and therefore only the train and validation sets are open for everyone. Also, the authors do not have a public Github with their code which is why our baseline was implemented a 100% on our own. We came up with two architectures for our own GRU-D. The reason behind this was that the authors did not specify how they concatenate their input with the two newly created vectors of masking and decay.

$$X \subset R^{B \times W \times V} \text{ Input vector}$$
$$M \subset R^{B \times W \times V} \text{ Masking vector}$$

$\Delta \subset R^{B \times W \times V}$Decay vector

All RNNs take as input a three-dimensional array. One dimension for the batch-size, another for the window size of the time-series observations, and the last one for the multiple variables of the time-series (equal to one in univariate time-series). When calculating the masking and decay vector, this will always have the same number of dimensions as the input vector, which will be three as well.

The naive approach of merging these three vectors would be to append

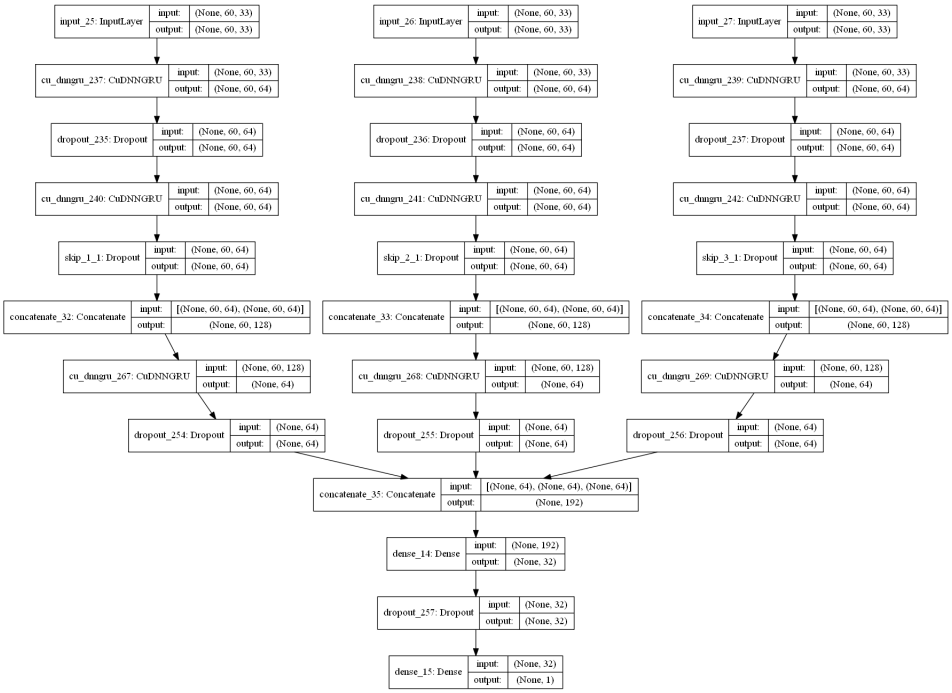


Figure 3.2: Vanilla version of our GRU-D with 3 inputs

them into a new vector that will consequentially be a four dimension array with the same three dimensions as before and the new dimension being of size three which will have the values of the three different vectors. However, this makes it impossible for the RNN to handle the input given that has too many dimensions. Therefore, the two approaches that we prove in this project are:

- Append the masking and decaying vector to the last dimension as new features. This will cast the following result:

$$X \subset R^{B \times W \times 3V}$$

These two different approaches will obviously have two different model architectures. One that resembles more the single flow network of Resnet[7], and a second one in which we create three different inputs for the network and it will receive the vector separately instead of all together in one big input.

- Creating three inputs for the model and merging the results with a concatenation layer. This means that X, M and Δ will be inputs of three different layers that will process them and then merge the results like an autoencoder and then work with this resulting vector in order to make the prediction[11].

These two different approaches will obviously have two different model architectures. One that resembles more the single flow network of Resnet[7], and a second one in which we create three different inputs for the network and it will receive the vector separately instead of all together in one big input.
This can be seen clearer in Figure 3.2, the input vector is pre-processed in order to calculate the masking and decay vectors that will be introduced into the model in their own Input layers. All of them will go along the GRU gates and then the final outputs of the GRU modules will be concatenated in order to be analyzed in the Dense layers that will give the last prediction. In this figure, we do not show the skip connections that are part of our final model. This was one of our baselines and it went through the whole experiment with its own hyperparameter optimization.
Figure 3.3 shows our final version of GRU-D in which we implemented the skip connections along with the GRU-D architecture that better suited our experiment after doing a fine tunning and comparison of all our architectures. Similar to Figure 3.2 but with a skip connection on after every GRU gate in order to give the deeper blocks information that was only observed on top blocks. Apart from that is the same idea, 3 different inputs to manage the 3 different sources of information of the time-series.

3.2 Univariate time-series

The models that were proven in the multivariate environment were also used for the univariate one. This was to compare if there is a significant difference and if they are compatible. Also, every model was fine-tuned in order to perform at its best in each task.

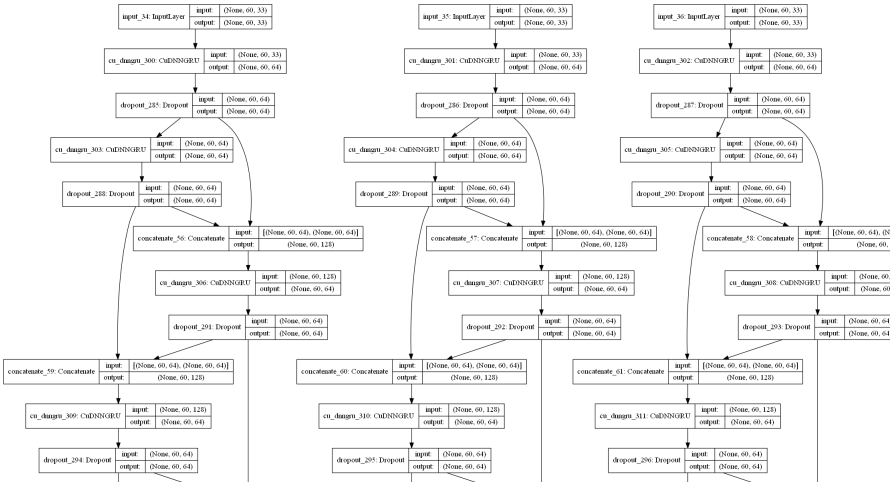


Figure 3.3: Portion of GRU-D with 3 inputs and skip connection

The biggest difference between this experiment and the previous one is the size of the last dimension. In this experiment, we can use our pre-processed inputs as new features in that last dimension. This converts our univariate time-series with missing values into a multivariate time-series with informative missingness. As we can see in Figure 3.4, the masking and decay vectors

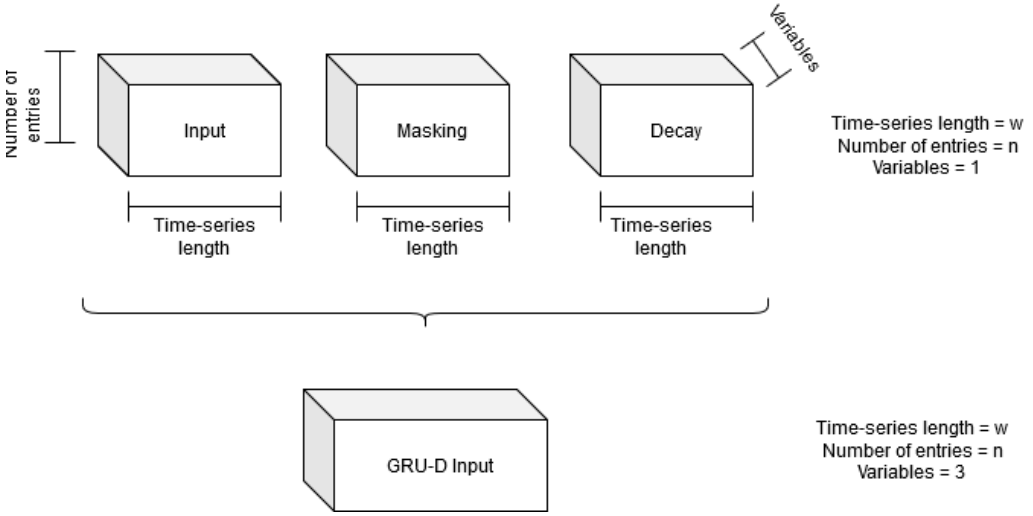


Figure 3.4: GRU-D input for the univariate time-series

can be concatenated into one final input. This is something we could not do with the multivariate time-series because it would create a fourth dimension.

3.3 Curse of dimentionality

We expect to have volatile results in this experiment given the curse of dimensionality. As mentioned before, in this experiment we have two dynamic dimensions and that will result in a high amount of artificial or missing information that might impact significantly the results. We have three dimensions in our input, and two of those need to be the same across all tasks. When we fill/truncate in the window size of the univariate time-series, we are affecting $N \times 1$ values. But when we do it for the multivariate time-series we affect $N \times P$ plus the affected values when fixing the third dimension which will be $N \times M$ values. The increase in accuracy is not linearly dependent on the number of dimensions affected, but higher than that[1].

As seen in Figure 3.5, the number of affected values by padding or truncating grows significantly with every new dimension. In this example, we are removing the first and last elements of each dimension. For the one dimension example given that the size is 10, the percentage of lost data is 20%. However, this is not constant as dimensions grow. For a two dimensional array, the percentage grows to 36% given that the first and last element of each dimension will be the edges of a square. The more dimensions we have the more information is lost with padding and therefore for the multivariate time-series it is particularly difficult to do meta-learning.

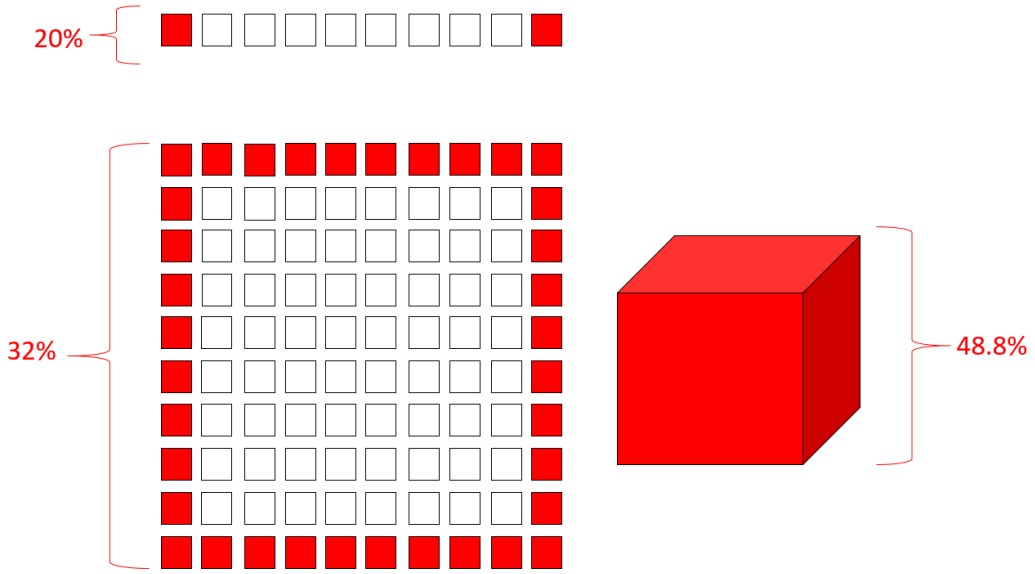


Figure 3.5: Missing information growth per additional dimension.

Chapter 4

Experiments

Having decided on the three research questions for the proposed work, we set up three different experiments to answer each one of these.

4.1 First experiment

For the first experiment, we use the Phisyonet Dataset[5] in order to do a multivariate time-series classification model. We implemented a version of the GRU-D[3] architecture to use as a baseline and try to replicate results. Since there is no open code for the paper, we had to do our implementation from scratch. The results that our implementation achieves are not comparable to the ones in the paper, given that the challenge testing data is not publicly available. However, using a split in the training data we created a validation sub-dataset and achieved close results to the ones reported in the paper. The worst results were archived by trying to pad and/or truncate different datasets (tasks) in order to match the dimensions of the Phisyonet dataset. The lost information is too much and therefore, it does not allow for proper transfer learning. After failing with this experiment, we focused on dividing the original Phisyonet into different datasets in order to have multiple "tasks" with the same amount of variables.

In figure 4.1 we can see an example of how different multivariate time-series have different dimensions and therefore require a trimming and/or padding task before performing the pre-training. Also as mentioned above in the methodology, this task when done in multiple dimensions results in a big loss of information. This is just an illustration to show what was done in the end and not the actual measures in which was done. In the figure, every cuboid figure represents a multivariate time-series, with their three dimensions. n for the number of entries in the dataset. w for the length of each

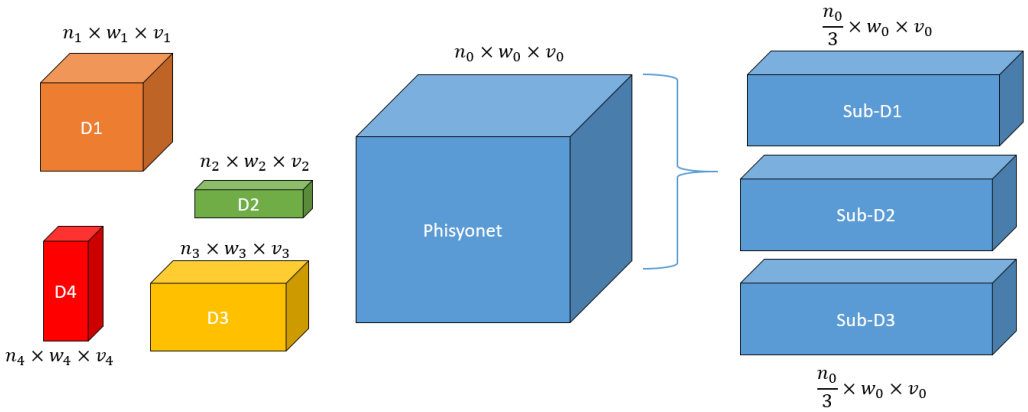


Figure 4.1: Tasks creation for the Phisyonet dataset. A partition of the original dataset results in the several tasks used for MAML.

time-series in the dataset. This dimension is present is usually altered using a sliding window in order to have all of the time-series with the same length. Finally, v for the different variables that are time-dependent in the multivariate time-series. It is extremely rare to find datasets with the same dimensions, especially the second and third which should be the same in order to perform any kind of transfer learning. Our final solution is illustrated by the blue figures. We take the original Phisyonet dataset and we split it into numerous disjunctive subsets that we will use as tasks for the meta-learning part. This is obviously not optimal to compare for a meta-learning task, but if we make sure that the model never had the actual training data in its other tasks, it could be a good first approach.

In order to train MAML in Phisyonet and use it in a fair comparison with a regular baseline model, we divided the dataset into two parts, training and testing. Then we performed the MAML algorithm using our own models in tasks taken from the training set and then compared them against baseline models using tasks from the testing set. This way we ensure two things:

1. That our models were tested using data they never saw in the training phase.

2. That the comparison between the baseline and our MAML model is fair.

For this experiment, we used GRU-D with three input layers (GRU-3D) and GRU-d with a super vector as input which contains triple the variables of

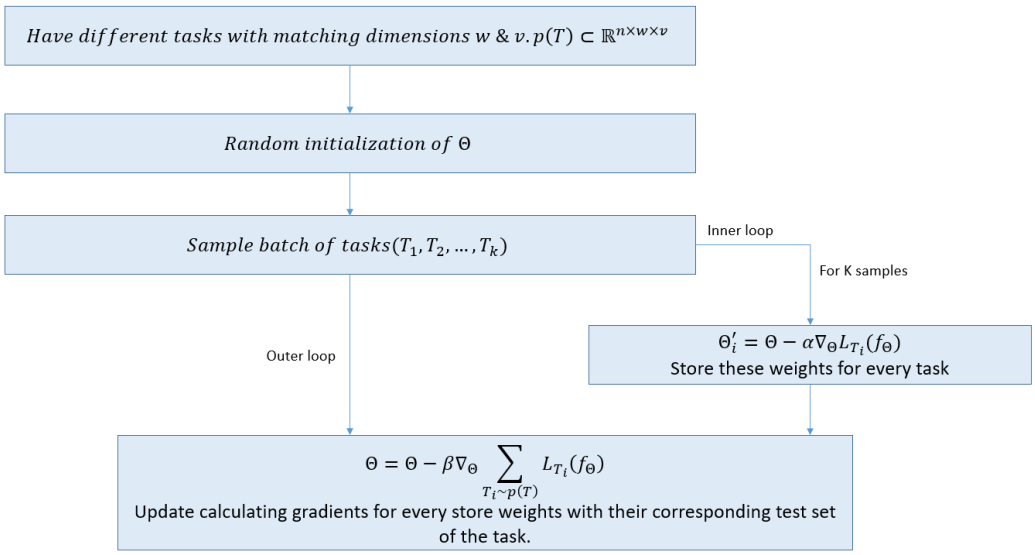


Figure 4.2: Algorithm of MAML in Phisyonet

the time-series. For both models, we require to do a pre-processing task. This takes the original input vector of the Phisyonet dataset and creates the masking and decaying vector which is ultimately used to create the input of both models. Both of these models are also implemented with skip connections and compared against each other. In addition to these four, a Resnet model inspired by [7] is used to compare results.
As seen in Figure 4.2, the algorithm is applicable to any model that uses gradient descent to update its weights. Therefore, we can try to use MAML for our version of GRU-D, its variation with skip connections, and the Resnet model that we use as a time-series baseline. All these models were be put through MAML and also were be compared with their vanilla version with the full training set of Phisyonet.

The results for these experiments are measured using the F1 score. This is a good metric given that the dataset is class-imbalanced[1]. This means that the dataset has a class that is significantly more present than all others and therefore it is easy for any model using a binary cross-entropy loss to simply predict always the class that has a higher percentage of entries.

4.2 Experiment 2

For the second experiment we worked on the M4 dataset[6] which is a known data set that contains over 10.000 time-series with different time deltas (yearly, monthly, daily, quarterly and weekly measurements). All these time-series are univariate, and this dataset, in particular, was very important in the experiments of N-BEATS[2] as a meta-learning algorithm for time-series. The model tries different datasets to achieve zero-shot learning in any univariate time-series. For the experiment I use the same models designed for experiment one and after few modifications implemented them along with a MAML version to compare their performances. The idea was to replicate the results of the zero-shot model paper[2] which was published a month into this work.

In figure 4.3 we can see how for the univariate time-series the dimensions

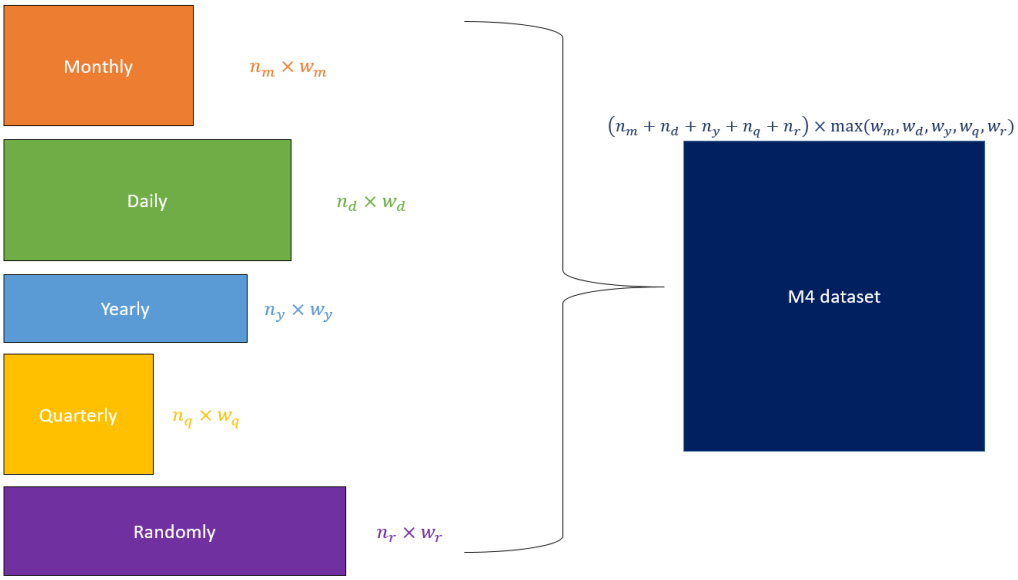


Figure 4.3: Content of the M4 dataset. The 5 partitions that come within M4 dataset and have different time deltas between points. This are used as tasks in our MAML implementation.

are reduced to two and what is more important, the only dimension that needs to be truncated and/or padded is the second one by using the sliding window. This will make our results less volatile and the amount of lost information significantly smaller. Also, this technique is of standard use when training a deep learning model in time-series[1] to do data augmentation or simply making the model predict the next value using several inputs different

of different lengths than the original time-series.

In figure 4.4 we take a random selection of the time-series that are in the

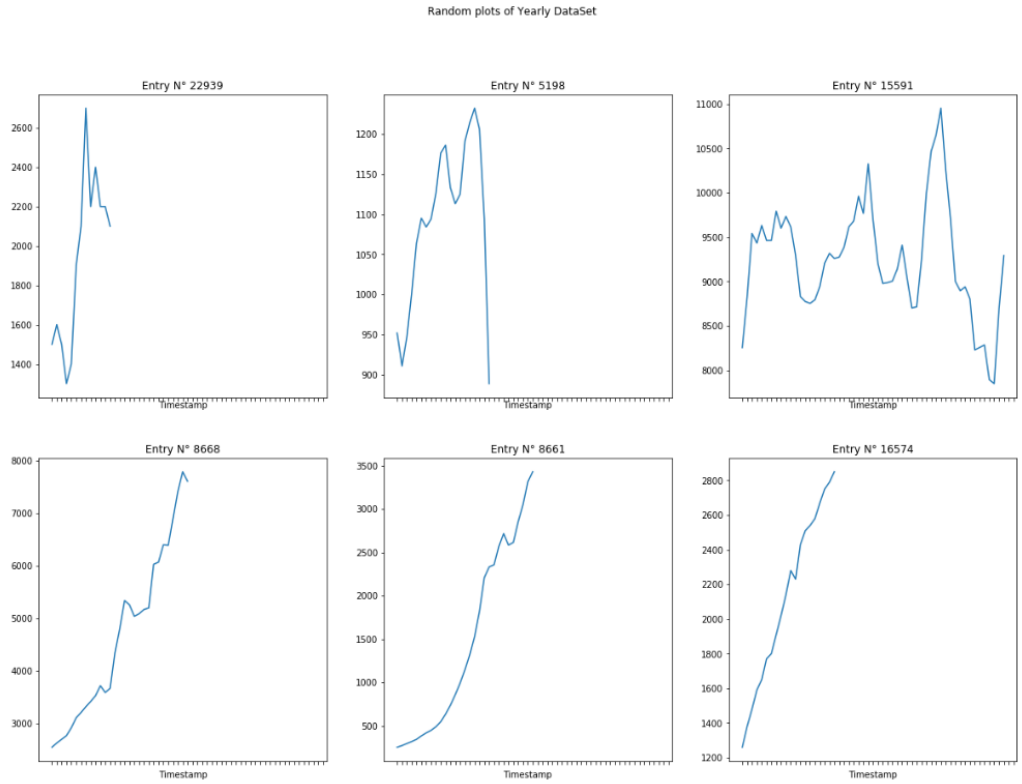


Figure 4.4: Random sample of time-series within the M4 dataset

whole M4 dataset. Here we just show a couple, however we did this experiment with up to 20 plots and many different times to guarantee that the selection was random. We concluded then that the time-series within the dataset are not necessarily the same length. Also and more important, the time-series do not share trends or periodicity between them. Every time-series is independent of all others and although some may have similar patterns, all of them present their own behaviors. Therefore, we safely conclude that this dataset split how it is with time-series with different time deltas is enough to attempt a meta-learning evaluation on univariate time-series.

Algorithm 2: MAML for M4(T, α, β, K) :

T=$\{T_{quarterly}, T_{yearly}, T_{daily}, T_{randomly}, T_{monthly}\}$;
for $T`i \in T$ **do**
    $T_{Pre} = T \setminus T_i$; *Pre-training set.*
    $\Theta = MAML(T_{Pre}, \alpha, \beta, K)$;
    $\widehat{y}_{T_i} = f(\Theta)$; *Testing with the subset that was not in pre-training.*
end

Algorithm 3: Regular M4(T, α) :

T=$\{T_{quarterly}, T_{yearly}, T_{daily}, T_{randomly}, T_{monthly}\}$;
for $T`i \in T$ **do**
    $T_i = T_i^{test} \cup T_i^{train}$; *Training and test set for current subset of T.*
    $\Theta = Model(T_i^{train}, \alpha)$; *Regular training for the current task.*
    $\widehat{y}_{T_i^{test}} = f(\Theta)$;
     Testing with the optimal weights after training in the current subset.
end

We can see the process of this experiment on both algorithms displayed above. There are two types of models we will be using for this experiment and both of them will give us a resulting $\widehat{y}$, which is the prediction of the model for a certain test subset of the M4 dataset. A very important fact of our process is that for every subset of the M4 dataset (yearly, monthly, etc.), we have a training and testing set. For the MAML version, the model was never trained in the corresponding training set of its testing subset. Instead, it had a K-shot pre-training on the other subsets to achieve a Zero-shot learning like in [2]. The vanilla version of the model was fully trained in the training subset before being tested on its corresponding testing subset. At the end, we have the two predictions, one for a regular model fully and properly trained on the training set before doing predictions on its corresponding testing set, and a second model that had few training iterations on other "datasets" before doing its predictions without ever experiencing the data of the corresponding subset. This is illustrated for a more clear view of the experiment on figure 4.5. We can see how both models are predicting the monthly subset of the M4 dataset. The vanilla version trained for 200 EPOCHS on the monthly training data and then predicted the monthly test data. While the MAML version was pre-trained on the training data of quarterly, daily, randomly, and yearly for 10 EPOCHS and then immediately tested doing predictions for the monthly test data. Both of these predictions are compared between

different models to conclude if using Zero-shot meta-learning is viable for univariate time-series.

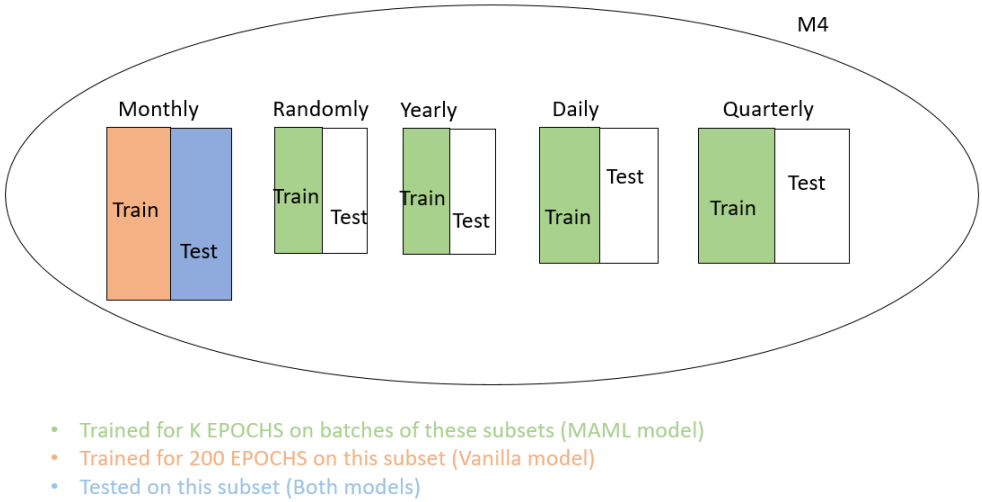


Figure 4.5: Venn diagram of the process of experiment 2. Illustrates the subsets in which each model was trained and evaluated.

More detailed into the models used, the process of this experiment was to use batches of time-series from the dataset to pre-train a MAML version of our GRU-D with skip connections. With the optimal initialization weights, we compare forecasting a time-series against a prediction from a baseline model for time-series like Resnet[7] trained on the data from the corresponding time delta. The final results show how padding and truncating time-series with lower dimensionality, one dimension, in this case, allows for very promising results. The MAML GRU-D with skip connections that were pre-trained in only 10 time-series from different time deltas is doing predictions that rival and even outperform baselines with proper and longer training in the actual training set of the time-series.

4.3 Experiment 3

Finally, for the third and last experiment, we wanted to go further with meta-learning for time-series. We already got close to N-BEATS[2] results on M4,

but now we want to address our research question of handling missing data.

For this experiment, we will keep using the M4 dataset given its useful-

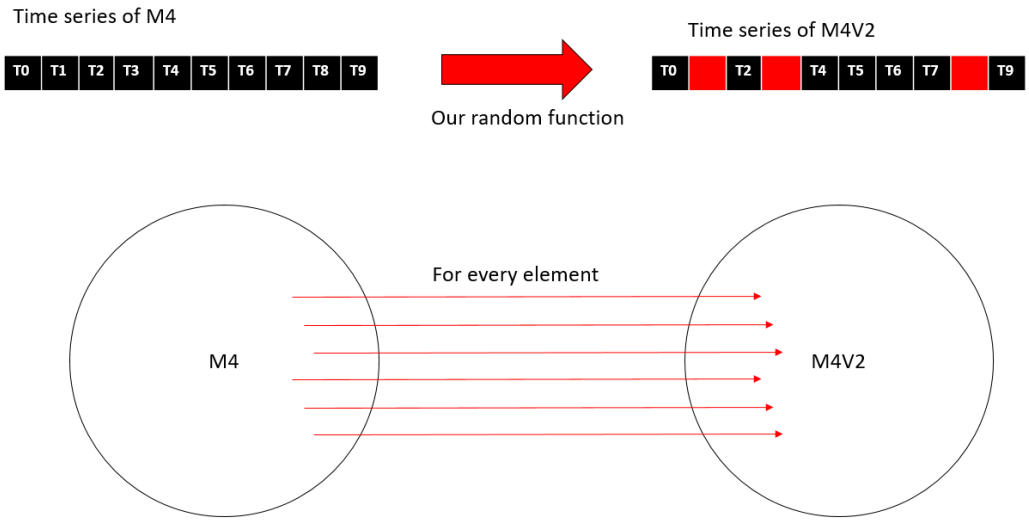


Figure 4.6: Transformation of the M4 dataset for experiment 3. It illustrates our random function to create missing data within the time series of M4.

ness for meta-learning in the univariate time-series environment. However, this experiment is meant to tackle meta-learning for time-series with missing values. Therefore, the solution is to create a random function that creates patterns for missing information but in different ways for every time-series and not always the same missing pattern.

We created such function with a simple algorithm that relies on the possible randomness produced by a computer [12] and it is used independently for every time-series to create different and independent missing patterns on each time-series of the dataset. As a result, we created a second dataset that we will call M4V2. M4V2 and M4 have the same dimensions and the same indexes. It is very important to highlight that there is a bijection between the two datasets given that the elements in one dataset correspond exactly to the elements on the other. The only difference is that M4V2 is the result of transforming M4 with our function that generates random values within a time-series. This is illustrated in Figure 4.6, where we created our function that receives a time-series and returns the same time-series with up to 30% of its values missing. As mentioned above, these values are selected randomly. This function is applied to every element in the M4 dataset without exceptions and store into a new dataset that we call M4V2. Then we have that this new dataset is a transformation of the original M4 and now consists of

over 10.000 univariate time-series but with missing values.

Algorithm 4: Creation of M4V2(M4):

```
R(n) Returns a random value lesser than n;
M4V2 = [];
for T˙i ∈ M4 do
    for i in range(int(T˙i.size*0.3)) do
        index = R(T˙i.size-1); random index within the time-series
          different than the last number
        T˙i[index] = Nan;
    end
    M4V2.append(T˙i);
end
returns M4V2;
```

The algorithm is pretty self-explanatory but there are a couple of facts that need to be justified. The algorithm is made to produce up to 30% of missing values in a time-series, but to give the problem more complexity we do not necessarily want every time-series to have 30% of its data missing. sometimes it could be less than that, and that is why the random index selected to become a missing value is not stored and checked if it's repeated. This way there is the chance that the random number has appeared before and it won't be a new missing value. This is left to chance and it could mean that we might end up with a time-series with just one missing entry, but given the length of the time-series (200 entries), it is a rather unlikely event $(\frac{1}{200})^{60}$. Finally, the reason why we chose a random index within the time-series different than the last entry is simply because we need that last entry as our ground truth. It would make it impossible to use the whole dataset if some entries had a missing value in the ground truth.
Now with a new dataset, we can set up the environment for the experiments that will help us answer our third research question. How does missing data affect meta-learning for time-series? In order to tackle the entirety of this problem, we divided this into two sub-experiments, each one with its own setup.

First setup

This first experiment is to measure the direct impact of missing values on the performance of the models. Taken the same models we designed for the second research question, we want to test them against the M4V2 dataset. Similar to the previous research question, we will have vanilla versions and

MAML versions of our GRU-D model and the baseline Resnet. And the idea of this setup is to measure how much impact the missing information has no the basic experiment (vanilla version). Even before experimenting, we expect the accuracy to drop with our vanilla models. We are using the same dataset but now we have less information on each time-series, therefore it should have a significant negative impact on the performance. Also, the model now has to predict with missing values and that makes the task more complex. All in all, we already know the predictions will be worse, but the important metric in here that we want to prove is, whether using Zero-shot meta-learning with data from M4 will still give acceptable results in M4V2.
In figure 4.7 we can see that for this setup, we use both the M4 and M4V2

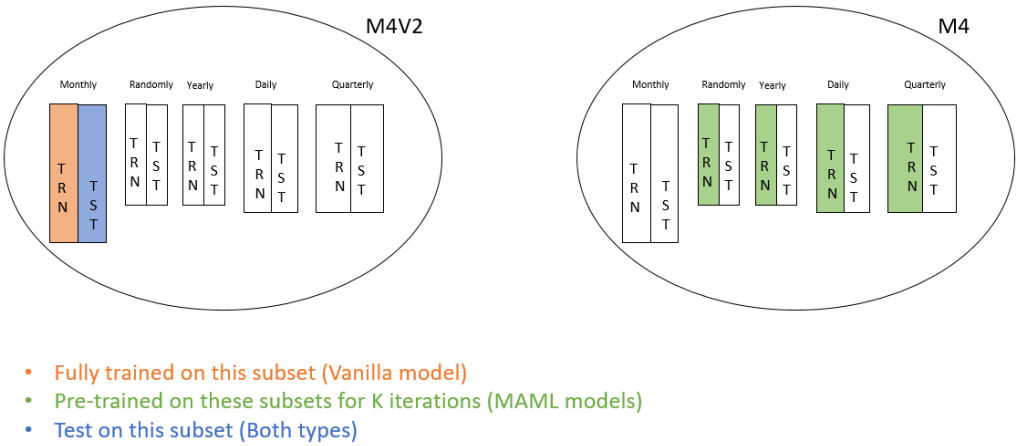


Figure 4.7: First setup of the experiment with missing data. Shows which data from M4 and M4V2 was used to do the first part of experiment 2.

datasets. The idea behind this setup is to evaluate if using meta-learning yields acceptable results compared to a baseline model trained on the training set of the M4V2. It is the scenario where we use the information with its missingness and try to predict properly despite that using the standard training. While with our MAML version we want to pre-train on time-series without missing data, that is the M4 dataset, and then test against the baseline that was fully trained with missing information. It is important to remember for this experiment that the M4 and M4V2 datasets are equivalent and therefore our pre-training was done using the same guidelines as the previous experiment. If the test subset is on the monthly subset, then our pre-training will never occur with the monthly subset entries.
This experiment is very important thinking towards real-world applications

and, as part of the motivation to do this research. In real-life many companies have missing entries in their data. Usually, that ends up in having insufficient data to properly train a model. A success in this experiment would mean that for this kind of task, we could have a meta-learning K-shot model that would grasp the overview of a dataset with the very few information that a company would have and still perform at the highest level. And more importantly, the pre-training section could be done using the best datasets available, given that comparing M4 to M4V2, we can conclude that M4V2 would be the incomplete information that we want to predict but we do not possess enough training data. And M4 is just a well-built dataset without missing values that gives our model an overview of how to handle time-series to quickly learn how to best handle the future tasks.

Second setup

We came up with this second setup as a curiosity, by aiming to improve the results of the first setup. The previous setup as mentioned is trying to prove that using a well-built dataset as pre-training to predict a faulty dataset with missing entries is a useful tool. That fixes the real-life problem when companies want to predict a value in the future but do not have enough information. Instead of doing data augmentation techniques that might conclude in an overfitted model, we propose using a similar dataset to do pre-training and achieving good results with few iterations of actual training. However, this only tackles half of the problem. Now that we do not require data augmentation techniques, we still have the missing information as a main difference between both our datasets. That is where this setup comes into play.

Figure 4.8 illustrates how the pre-training is now done using only the M4V2 dataset. At a simple sight, it would look like we are repeating experiment 1 using M4V2 instead of M4. And that is true, however, the reason why it is different and important to do it whatsoever is deeper that. We should remember that M4V2 was built using our random generator function independently on each time-series. That means that the patterns of missing data are with a high probability different on every time-series. Not only in the proportion of missing information, but on the distribution of this missingness. That is important because it allows us to create an artificial real-life scenario. Missing data across different datasets is not necessarily the same in terms of those two same attributes, distribution, and proportion to the full time-series. Yet, M4V2 is an artificial dataset made from a real one using a transformation function. Being successful in this experiment would mean that for real-life problems where the dataset has missing values and fewer

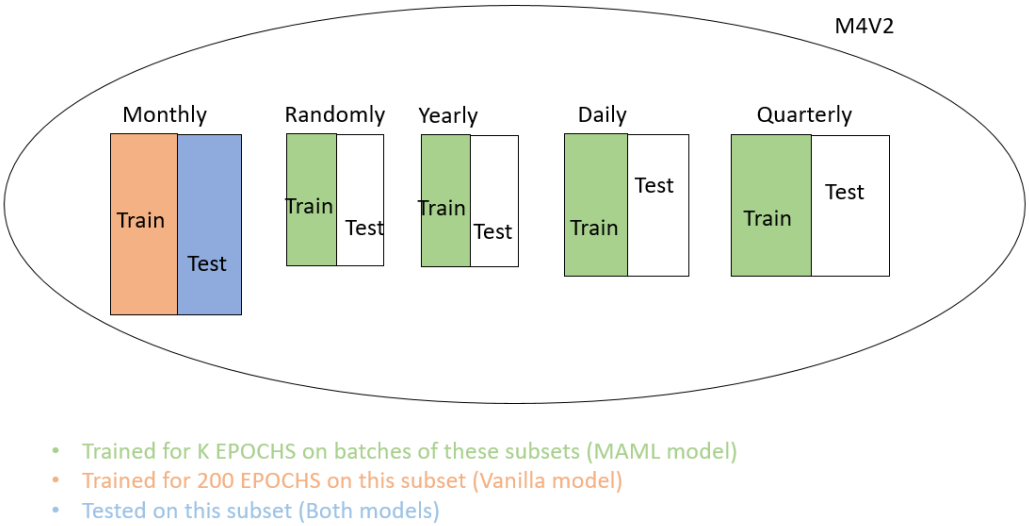


Figure 4.8: Second setup of the experiment with missing data. This corresponds to the second part of experiment 2 done only in M4V2.

data, we could use a well-built dataset, transform it and achieve promising results that could outperform the previous setup, given that the model was pre-trained using time-series with missing values and could learn to ignore them as opposed to the previous setup.

N-BEATS has a big influence on our research even though it was released after we began to work. Yet, two major points the authors of N-BEATS do not experiment on in their work are using their method in a multivariate time-series environment and also for time-series with missing values[2]. With our experiments, we are mostly covering their future work and giving a comparing model for them to take into consideration for their future experiments.

4.4 Hyper-parameter optimization

In machine learning is never enough to just try a model with the first randomly generated hyper-parameters and then commit to those for the entirety of the experiments. It is always necessary to do some kind of hyper-parameter optimization technique[1][13].
It is a simple yet important process. The first step is to identify which hyper-parameters should be tuned. The second one is deciding which optimization technique will be used. Lastly, the third step is to perform the

hyper-parameter optimization and then stored the optimal results. For our experiments, we have hyper-parameters for the learning task and hyper-parameters for the meta-learning task. Although they do not have a clear dependency, there is a one-sided relationship of impact between them. The hyper-parameters of the learning task will affect the performance of the meta-learning process but not vice-versa. This because with better hyper-parameters for the learning process, the meta-learning process will be faster and/or better, allowing for the results in meta-learning to be also better. Given this relation, we decided to do first an optimization for the hyper-parameters in the learning process and then do them for the hyper-parameters in the meta-learning process.
The hyper-parameters for the learning process that we optimized were:

- Learning rate (α): This hyper-parameter is the rate that multiplies the gradient when updating the model's weights.
- Dropout rate (d): In all our models we used as regularization a dropout layer after every GRU gate. The dropout rate indicates which percentage of weights should be ignored before the next layer.
- Number of layers (l): The universal approximation theorem states that any problem can be approximated with the proper neural network. That network may have more or less hidden layers depending on the complexity of the problem and the dependencies between variables. Therefore is a very important hyper-parameter to optimize.
- Number of neurons per layer (n): This parameter defines how many neurons or perceptrons a layer has. That also affects the accuracy of the result.
- Batch size (b): The batch size is how many entries at the same time does the network uses before updating weights. This affects the performance of the neural network both in time and accuracy. However, we only used the accuracy metric to optimize this parameter.

Every model has also an optimization technique in which it may change its hyper-parameters during the optimization and not only in the beginning. For all of our models we used *Stochastic Gradient Descend* (SGD), *Adagrad* and SGD with *Nesterov momentum*. This selection was done prior to the actual hyper-parameter optimization for time constraint reasons.
The technique we decided on using for the hyper-parameter optimization was a grid search with the most common values seen in other networks for the hyper-parameters we selected. For every parameter, there is a different

decision boundary that we needed to define. The idea behind this technique is to have numerous different and controlled experiments where one variable is being tested at the time. That means that we run the optimization task of the network with all hyper-parameters locked but one, which is the one being evaluated for its optimal value.

Algorithm 5: Grid-Search for Learning Rate Optimization

Result: Optimal learning rate

```
L = inf;
A=(α_1, ..., α_n) different learning rates;
d, l, n constants;
α̂ = None;
for α_i ∈ A do
    L_(α_i,d,l,n) f(Θ) Loss with the current hyper-parameters;
    if L < L_(α_i,d,l,n) f(Θ) then
        L = L_(α_i,d,l,n) f(Θ;
        α̂ = α_i;
    else
end
return α̂
```

This is an example of the grid-search we used for the learning rate optimization. We lock all other hyper-parameter $d, n,$ and l, and then, we try different values for α and keep the one that has the lowest result in the Loss function. This algorithm was used in multiple loops in which we tried different values for each hyper-parameter. In the end, the algorithm tried the following values:

- For *alpha*: [0.1, 0.001, 0.0001, 0.00001]
- For d: [0, 0.1, 0.2, 0.3]
- For l: [3, 4, 5, 10]
- For n: [32, 64, 128, 256, 512] and [8, 16, 32, 64, 128]

Before running the whole process to calculate all the combinations we tested individually every variable to know at what point increasing or decreasing it had always a negative effect. This allowed us to decided what are the ranges that should be tested and which simply are a waste of time. Also, you may notice that for the number of neurons we have two lists of values. That is because we did it independently for the GRU layers and the dense layers. Their contributions to the model are different and therefore their values can be too. In summary, we compared 1600 different combinations to evaluate

which one was the best one for our model. However, this task had to be done for every dataset, given that the data affects directly how the hyper-parameters should be tuned. In the end, the best setup for our model in the Phisyonet dataset was:
$\alpha = 1e^{-5}, d = 0.2, l = 5, n = 128$ GRU-D and 32 Dense.
That was just for the learning routine. Now that we have the optimal hyper-parameter for our model to predict better we can proceed to do the grid-search for our meta-learning hyper-parameters. These hyper-parameter are:

- β Meta-learning rate.
- k Defines the k-shot learning.

The procedure is the same, we use the same routine only this time we compare MAML versions of our models instead of the loss of the optimization done in training. The values we used for our hyper-parameter were:

- For β: [0.1, 0.001, 0.0001]
- For k: [10, 15, 20]

As for the M4 dataset, the best results in our hyper-parameter optimization was the following combination:

- For α: 0.0001
- For d: 0.2
- For l: 5
- For n: 256 and 64
- For β: 0.0001
- For k: 20

As curiosities, we can appreciate that the hyper-parameter change between datasets. However, we see that k is the same and also the biggest possible value in the range we allowed. This is obviously because the more the model trains with data (before any overfitting) the most suited it is to handle a test. Yet, that goes against the purpose of this experiment in which we are supposed to use a few data entries to do the whole training of the model. The second curiosity is the learning rate (α) that is two orders of magnitude smaller in the M4 dataset as it is in the Phisyonet dataset. This is most likely given the complexity of the task. Phisyonet is the multivariate time-series

dataset that has among its variables many dependencies that affect the end result. The model needs to take smaller steps towards the optimal approximation given that these dependencies among variables make the predictions very volatile. While our univariate time-series dataset allows our model to take bigger steps towards an optimal prediction, given that there cannot be as many dependencies among variables if there is only on time-dependent variable.

Chapter 5

Evaluation and Discussion

In this section, we will analyze and illustrate deeply all the results we achieved in our experiments. Boarding each research question separately with its corresponding experiment. Determine whether the research question was answered of we require future work in it. Also, illustrating the results in order to show any highlights achieved and to have an easy way to compare the numerous different models we used. Finally, we will give our feedback on the results, what was achieved what the next steps will be and what could be redone differently.

5.1 Results for Phisyonet Dataset

The first research question was if we could manage to create a MAML version of a model that can handle a multivariate time series with missing values. The experiments contain numerous models that were tested against a testing subset of the Phisyonet dataset. We mentioned in the previous chapter how we pre-trained the MAML versions with different subsets of the whole dataset and tested all of them against the same test subset along with the vanilla version of the models that had access to more training data. Figure 5.1 shows two plots. The plot on the left is the evolution of the F1 score per EPOCH. As we mentioned before, the dataset is class unbalanced, and therefore using the standard binary cross-entropy would have resulted in a model that learned to predict the same class regardless of the input. With the implementation of the F1-score both as a metric as a key element in the loss function, we can improve the predictions of our model. Also, we did data augmentation in the Phisyonet dataset for the class that was less prominent within it. These two techniques improve greatly the end result of our models.

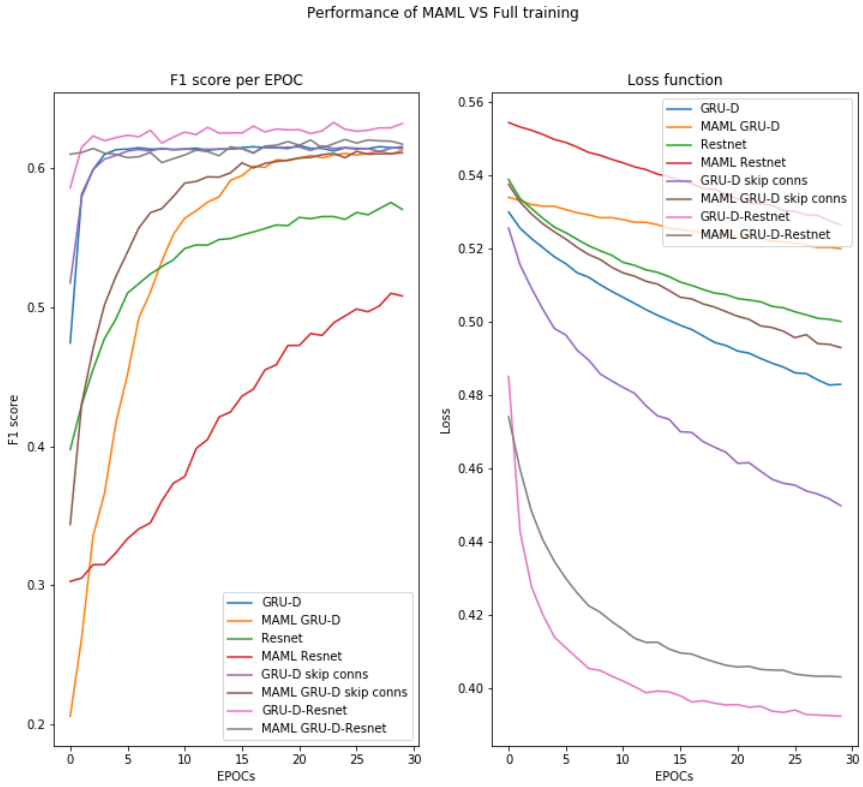


Figure 5.1: Results of MAML and vanilla version of every model.

The plot on the right displays the same models but this time it illustrates the evolution of the loss function per EPOCH.
It is important to note that for the F1-score per EPOCH the higher the model is the best it is considered. While in the second plot it is the opposite of that. The model with the lowest loss per EPOCH will be the best performing model. What every plot represents and illustrates is different from the other, but using both to make a conclusion is the best policy given what they reflect.

We can appreciate that in the F1-score there are many models with close or equal results at the end of their training. This makes it harder to decide if one of these models is actually better than the others. The only evident difference is for our baseline model that was created for traditional time series classification without missing values[7]. The Resnet model even after the 30 EPOCHS we use for our experiment does not achieve the same ac-

curacy as all of the other models. Its F1-score is considerably lower than all of the other models. And we are referring to both the vanilla version and the MAML version that even though it improved the F1-score is below all other models. As for the other model, some of them took a handful of EPOCHS to achieve the peak accuracy, while there are at least two models that achieve this peak accuracy in less than 2 EPOCHS. Yet, the interesting observation here is that our results regarding to the research question are insufficient. We already pointed out that using MAML for our Resnet model improved the F1-score, as it is the case with our GRU-3D with skip connections. The accuracy in the end seems to be the same, yet we can appreciate that the MAML version reaches this accuracy faster than the vanilla version. However, for our GRU-3D without any skip connections, the vanilla version actually begins with a higher score than its MAML counterpart. Finally, our most complex model reaches a slightly higher accuracy than all other models in its vanilla version. The inconvenience with this model is that it took considerably longer to learn given that it possesses more layers than any other model we ran. Therefore, a MAML version of this model was simply not feasible. Also, given the mixed results that we achieved with the other MAML versions, it was not a priority to spend the computing time on doing the MAML version of our ensemble method.

As for the loss per EPOCH plot, we can see that the differences between models are on a smaller scale and therefore are smaller. Again our baseline is performing the worst in terms of the total loss at the end of the training. Again followed by its MAML version. So far the behavior is the same as in the previous plot. However, now we see the main differences. Starting with our MAML version of GRU-3D without skip connections, it has a worst performance than its vanilla version. Ideally, the MAML version can start with a higher loss however it should learn the task quicker than the vanilla version. But if we compare the MAML version against its vanilla counterpart, we can see that the *inclination* of the loss function is pretty much the same. By EPOCH 30 a model that was pre-trained with MAML should show evidence of faster learning. Yet our two GRU-3D models seem to learn at the same pace, yet our MAML version started with a higher loss. This would point to the conclusion that using MAML for this experiment was a failure. But then it comes our model with skip connections and shows differently. The MAML version begins with a lower loss function and maintains it through the 30 EPOCHS. Yet again, the learning speed seems to be the same as the vanilla version.

Our conclusion for this experiment is that we simply cannot deny nor confirm the fact that using MAML is viable in a multivariate time series with missing values. By using different datasets as tasks and performing padding

and truncating pre-processing to it, lead to random models that could not make any acceptable predictions. Using subsets of the same dataset as tasks led to random results where some models seem to have an improvement in the overall performance, yet the models do not evidence the benefits of a pre-trained meta-learning model that learns faster than its vanilla version.

5.2 Results for M4 Dataset

The second research question is regarding using MAML as a zero-shot forecasting for univariate time series. Based on the results of N-BEATS[2], we will use our model to compare the results to those of N-BEATS. The model was pre-trained on different tasks of time series and tested against a model fully trained on the task to be tested.

Figure 5.2 shows the predictions of every task done by our baseline GRU. Every row in the plot is a pair of graphs that illustrate the predictions against the real values. Since this is the controlled environment of our experiment, every row represents the results of different learning processes done by our model. Our model was trained on the training subset of each of the corresponding tasks of M4. Then tested with the corresponding testing subset of the same task. Therefore, we can argue that the model learned some of the patterns that should be representative of the task it was trained on. it is noticeable that the shape of every prediction matches that of every task. Only by observing the values, we can see that there are small differences between the prediction and the real values, however the pattern of the time series is understood by the model.

The second plot we have for this experiment reflects the results of the predictions done by our baseline, the Resnet model. Also fully trained on each task to provide the model with the necessary data to understand the patterns of the time series is going to be tested on. The results are in Figure 5.3 and we can clearly see the differences in the predictions on this model compared to the previous one. The baseline model is outperformed by our GRU model on every task.

The final model on this experiment will be our MAML version of our GRU model. There is no point in doing a MAML version of the baseline given its inferior performance compared to our model. Figure 5.4 illustrates the results of our MAML model on every task. As mentioned before, this model was pre-trained for 20 EPOCHS in random batches of the tasks it is nor being tested on. For example for the first row, the model was pre-trained on the other four tasks and never saw time series from the first task. At first sight,

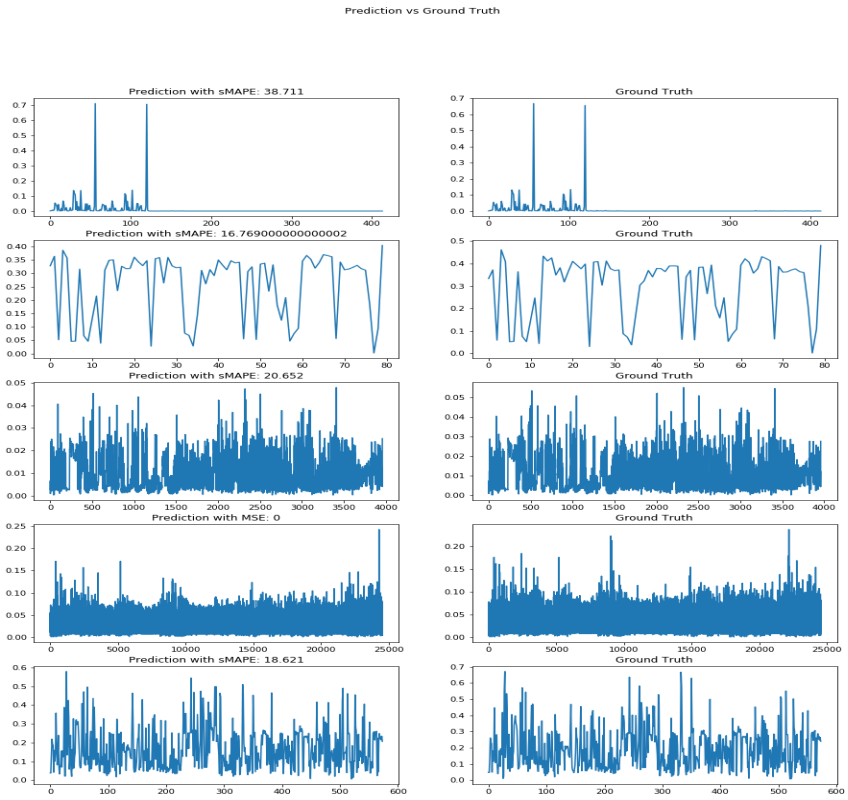


Figure 5.2: GRU results vs real values in M4 dataset. Every plot is the comparison of the real values vs the prediction done by GRU on every subset of M4.

it is evident that our Zero-shot model outperforms the baseline model. Yet, it does not outperform the vanilla version of GRU on any task. However, it is important to remember that the MAML version had zero training on the data it was being tested and still managed to beat a simple baseline and get close results to the same model fully trained on the training subset of the corresponding test task. Therefore we can say that the experiment is as successful as the N-BEATS one[2].

This experiment is heavily influenced by N-BEATS[2], yet there is no comparison available for it. We do not possess the exact experiment to perform in N-BEATS. We use the same metric to evaluate the symmetric mean absolute percent error (sMAPE) but as we can see in our numerous experi-

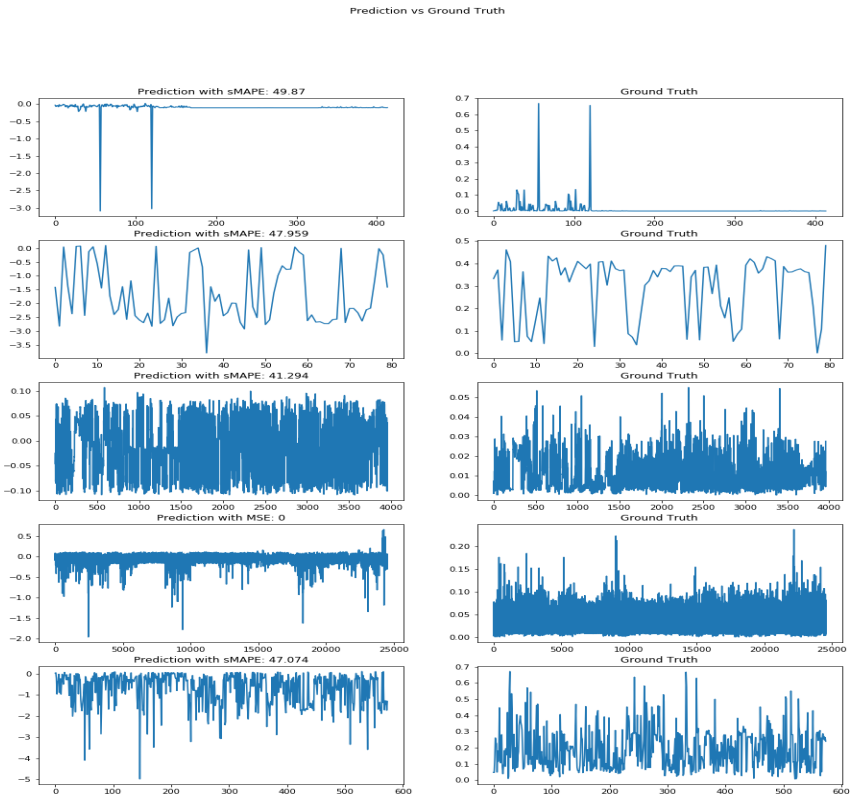


Figure 5.3: Resnet results vs real values in M4 dataset. Every plot is the comparison of the real values vs the prediction done by Resnet on every subset of M4.

ments, the values change significantly depending on which task we evaluate it. Therefore without the specifics of their metric measurement, we cannot do a fair comparison. However, let it be noted as a curiosity that in their results, they report a sMAPE of 11.135 and our best-reported sMAPE is 16.76. That gives the warning that N-BEATS might be performing better than our model.
Now that we have shown the results of every model in all tasks it is time to gather all of them into one single table in order to just focus on the numbers and compare every model with the other on every task they were tested on.

It is more clear now in Table 5.1 that our Zero-shot MAML version of GRU is inferior as we said to the vanilla version that was trained for over

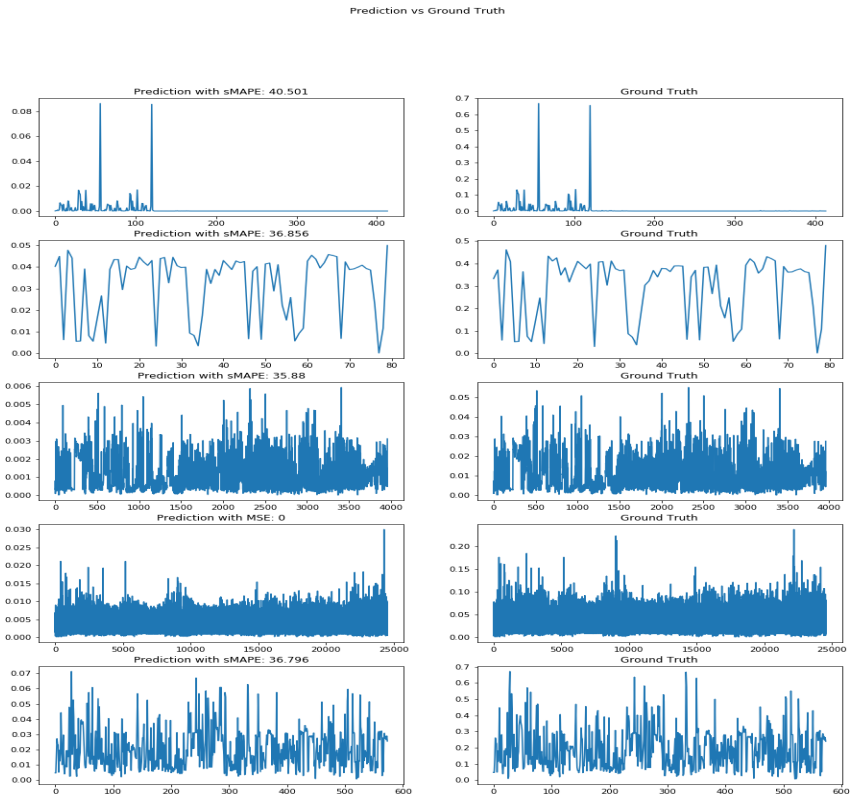


Figure 5.4: MAML GRU results vs real values in M4 dataset. Every plot is the comparison of the real values vs the prediction done by our MAML GRU on every subset of M4

200 EPOCHS on every task it was tested. But also, it is superior not just in one but in all tasks when compared to a vanilla Resnet trained also for over 200 EPOCHS on every task it was tested. From these results, we can quickly draw a conclusion.
It was to be expected that a model properly trained on a task performs better than the same model pre-trained on different tasks without any further training. However, the results obtained by the Zero-shot model are close enough to the fully trained vanilla version. That means that pre-training for univariate time series for problems where data is scarce is a good valid approach instead of data augmentation. If data is not scarce there is no reason to use transfer learning, however that was not our hypothesis when investigating our research question. We wanted to prove if we could make

Model	Task	Validation 1	Validation 2	Validation 3	Validation 4	Validation 5	Average
Vanilla GRU	Randomly	48.565	39.362	41.42	40.985	38.711	41.81 ±3.52
	Yearly	17.2919	17.407	50	18.12	16.769	23.91 ±13.04
	Daily	27.533	19.508	49.997	24.32	20.652	28.401 ±11.16
	Quarterly	18.87	18.718	50	20.454	18.621	25.332 ±12.35
Vanilla Resnet	Randomly	49.434	49.53	47.438	49.579	46.142	48.42 ±1.39
	Yearly	37.419	30.409	29.654	45.438	37.188	36.02 ±5.72
	Daily	42.859	39.105	37.219	48.703	38.677	41.312 ±4.13
	Quarterly	34.329	27.362	28.28	43.1539	41.1419	34.85 ±6.45
MAML GRU	Randomly	45.383	41.114	40.501	41.865	40.501	41.8728 ±1.82
	Yearly	49.247	39.645	36.856	42.123	36.856	49.247 ±1.82
	Daily	49.467	38.75	35.88	41.323	35.88	40.24 ±5.04
	Quarterly	49.573	39.69	36.796	42.169	36.796	40 ±4.73

Table 5.1: Table of results on M4 dataset using random cross validation.

something similar to N-BEATS in the univariate time series environment with what we previously used for multivariate time series. As stated it is not exactly the same success, but it is very close to it. Yet, in the multivariate time series experiment, it was not a successful result. That means that in the future work the authors of N-BEATS proposed, it might not be as easy as just converting the exact same model to a multivariate time series problem; but that this problem requires a different approach with meta-learning.

For one random time series

We wanted to go further than just getting a model that is generally good for all the time series in the M4 dataset. Therefore we randomly selected a time series from the dataset and used the traditional approach of using the sliding window as a data augmentation technique to reconstruct the time series using our GRU with skip connections. In the end, we reconstructed the time series with the fully trained GRU with skip connections as well as the MAML version of it without any specific training on the windows subtracted from the selected time series.

We can see in Figure 5.5 that the model had a not so smooth descent through the loss function yet in the end it had an early stop before the 200 EPOCHS it was programmed to do. Apparently, it took it 80 EPOCHS to fully understand the time series. That shows that the univariate time series is not a very complex problem and that might be the reason why zero-shot learning works so well in it.

Finally, Figure 5.6 shows the predictions of the fully trained GRU with skip connections, the predictions of the MAML GRU with skip connections we pre-trained for the previous results, and the real values of the time series we wanted to reconstruct. It perfectly shows how the MAML version un-

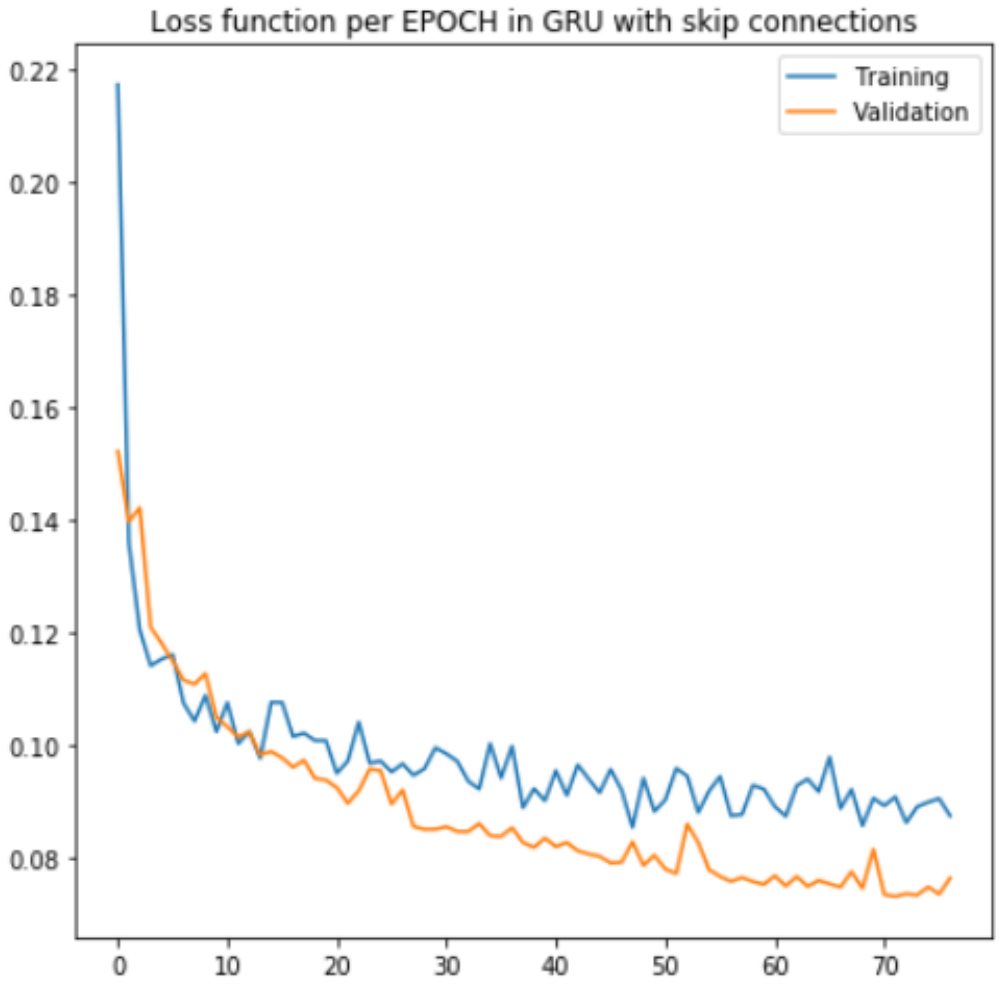


Figure 5.5: Loss function of GRU with skip connections

derstands the behavior of the time series even without any training with the windows created in the data augmentation. The only problem our MAML model seems to have is that the ranges on which it is changing the data are not close to the actual values. This shows that the MAML version understand which changes will occur in the time series but is not yet ready to be precise in how big these changes will be. This might be linked to what we showed in the previous chapter when we plotted randomly selected time series from the dataset and we could see that this time series not only have different lengths and patterns but their values range were also very diverse. Therefore the Zero-shot learning allows the model to understand how the patterns of the time series will be, yet when it comes to the differences between values, the results are not that accurate. Which is something that k-shot learning might

quickly fix, once the model understands how the time series will behave it only should learn which number scale it should work on.

In conclusion, this experiment was a success in recreating similar results to

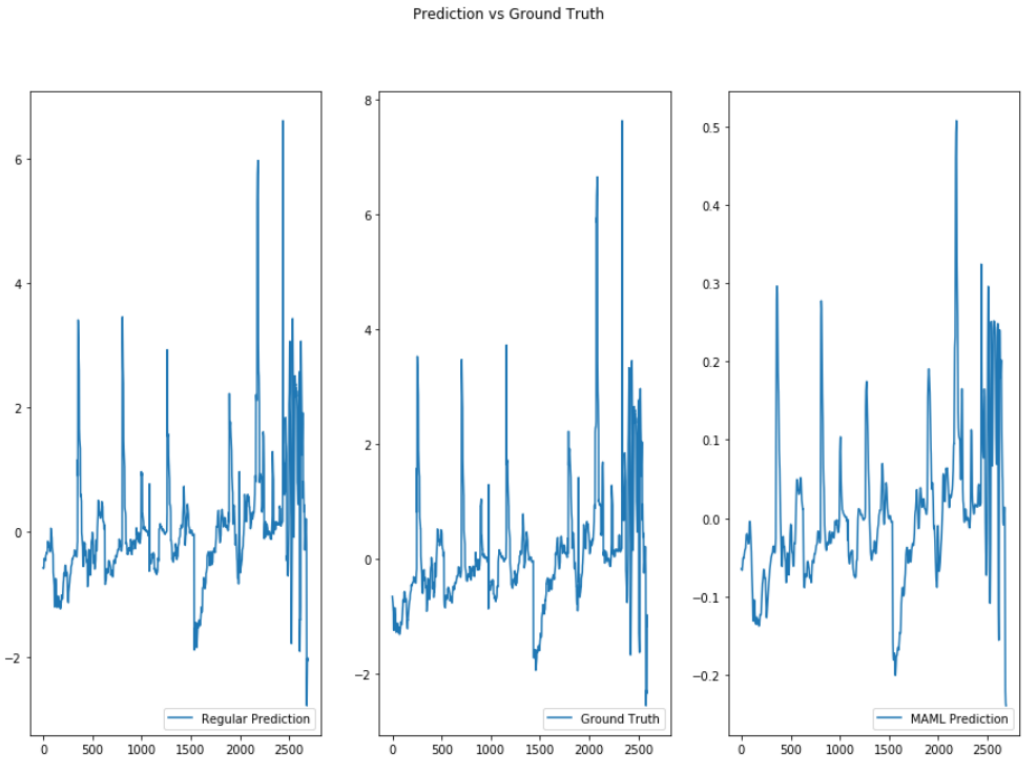


Figure 5.6: Comparisson of results in one time series for the vanilla and MAML version

N-BEATS in the univariate time series. Even without complete certainty that our GRU with skip connection has the same performance with time series as N-BEATS, we can assure they are on a similar scale. Yet, our model does not allow for good results when doing meta-learning for multivariate time series with missing values. Given the amount of lost information and the volatility of the missing values. The next experiment is going to measure the impact of missing values in the univariate time series environment.

5.3 Results for M4V2 Dataset

This experiment is aimed to answer the research question of how big is the impact of missing values when using meta-learning. We already tested that our model performs at the same scale of accuracy of N-BEATS. Their future

work includes doing tests in multivariate times series, which is covered in our first experiment. Something the authors did not contemplate for future work was the impact of missing values in the time series. This is what we aim to test in this experiment. As mentioned above, the setup of the experiment is to independently create random missingness on each time series of the dataset and then similar to the previous experiment, test our vanilla and MAML version of the GRU with skip connections. Since our Resnet baseline underperformed on the previous experiment, there is no reason to try it on this one. Finally, since we have again missing values in our data, we can implement our version of the GRU-3D with skip connections to test against the GRU from the previous experiment. The means using again the pre-processing functions to create the masking and decay vectors.

The first model we are going to analyze is our GRU with skip connections by training on the training subset of the corresponding task. Same as the previous experiment, we have our model with full training on each task in order to test on each one to have the test in order to reach the best results and then compare to the Zero-shot meta-learning model. We can see in Figure 5.7 that the results of the model are significantly affected by the missing values present in the M4V2 dataset. The tests on every task have worse values than they did on the M4 dataset. We can see the impact of having up to 30% of missing data on our time series. Not just the sMAPE increased but the shape of the predictions compared to the real values is visibly different. Which means our model is now having trouble understanding the behavior of the time series.

The next experiment we attempt is to use MAML with our GRU with skip connections that we trained in the previous experiment and test it against the tasks of M4V2. We previously witnessed how our model with zero-shot learning managed to understand the behavior of the time series yet not the deltas between values. However considering the accuracy drop of the vanilla GRU with skip connections, there is the chance that the MAML version might perform better without training on the corrupted data. Figure 5.8 shows the results of this attempt. It is clear that having a pre-trained model with time series that do not have missing data, allows our MAML version of GRU with skip connections to give slightly better predictions that our fully trained vanilla version of the same model using the training of M4V2. However, it is important to notice that our model was not precisely built to deal with missing values in a specific way any different than other regular neural networks. Since the difference in the results are not significantly different in all tasks, we wan to experiment to see if using MAML with pre-training on the M4V2 tasks produces better results. We must not forget that using MAML we did not train for many EPOCHS and therefore the

chance of our model learning bad patterns from it are slim.

The guidelines are the same as the previous experiment. The pre-training will be done with batches of the other tasks, not the one the model is being tested on. This will produce different weights than the MAML version pre-trained on M4, and we can evidence the difference in Figure 5.9. The sMAPE dropped even further than the original MAML version we had. This is an interesting finding, knowing that all the random missingness is independent from time series to time series. It creates a foundation to argue that for any univariate time series with missing values dataset from a real-life, a valid approach would be to create random missingness in a complete dataset and pre-trained a model in it in order to get acceptable results on the real-life dataset.

As we mentioned our GRU was not thought specifically for handling missing values. Therefore, we decided of applying GRU-D to univariate time series problems. This is done with a new model that we could not use on the Phisyonet dataset. As we mentioned the GRU gate only takes three-dimensional inputs (Batch size $\times$ window size or sequence length $\times$ variables or channels). If we decided to create a fourth dimension using the masking and decay of each three-dimensional input the result was a four-dimensional input that the GRU gat could not handle. However, in the M4V2 dataset, our third dimension is actually a single value, making it possible to flatten the input into two dimensions. Now we can construct a three-dimensional input with the masking and decay vector as a new channel of the time series. This will be an application of GRU-D to univariate time series with missing values. Also, to have a comparison, we implemented our GRU-3D from the Phisyonet experiment and tested it against the tasks in M4V2.

Figure 5.10 illustrates the results of creating the three-dimensional input with the masking and decay vector and passing it into our GRU-D. The results are somehow promising. It shows the best performance so far in two out of three tasks compared to all our previous models. However, in the tasks that perform better, the difference is significant. These results when compared to the ones from our GRU with skip connections show that clearly the baseline results we should consider when comparing to the MAML versions are these.

The results of the final model can be seen in Figure 5.11. This model is our GRU-3D that we used on the Phisyonet dataset. It basically has three input layers in order to process the values of the time series separately from the corresponding masking and decay vectors. This model in a univariate problem does not have much sense given the dimensionality of the input is

not a problem here. However, in order to compare fairly to all models previously used, we decided to include in the experiments. As the results reflect on Figure 5.11, this model does not perform any better than the previous one, on any of the tasks. The only noticeable remark on these results is the result we achieve on the second task. We achieve a sMAPE of 50, which is the maximum you could get in this measure. That means that our GRU-3D performs the worst in comparison to all previous models in this task. It basically has the behavior of the Resnet on the M4 dataset experiment. It is a proven model that simply does not perform the best in this experiment, yet it serves to compare against the zero-shot models.

We can see clearer in table 5.2 all the results of all models on every task

Model	Yearly	Daily	Quarterly
Vanilla GRU	49.88	49.576	49.99
Vanilla GRU-D	29.156	40.613	19.99
Vanilla GRU-3D	40.548	50	45.401
MAML GRU M4	49.26	33.192	47.346
MAML GRU M4V2	49.209	28.238	47.274

Table 5.2: Table of results on M4V2 dataset

of M4V2. As previously stated, the converted GRU-D performs the best on every task and it plays the same role as the GRU with skip connections did on the previous experiment on M4. That same model that was the best performing on the M4 dataset is having significantly worse results on M4V2. This allows us to argue that Zero-shot meta-learning for univariate time series is significantly different when used on time series with missing data. Therefore, N-BEATS could add to their future work not only the multivariate time series as a different problem, but also the missing information in the dataset. There is not much to say about our GRU-3D except that is is definitely not suited for handling missing information when the data is so simple. although it was the best-performing shape of GRU-D in our experiments in Phisyonet, it falls quite short when tested on M4V2. As for our MAML versions, since they were variations of our GRU with skip connections, the results are definitely not as promising as they were on the M4 dataset. It would be the equivalent of having done the MAML version of the Resnet on the previous experiment. We used the wrong model with this algorithm and therefore the zero-shot learning paid the price for a poor choice of model. We can see two interesting results in the MAML versions. They are outperformed by the GRU-D model on two out of three tasks. However, the task where they perform the best is the task where the vanilla models had the worst results.

This shows that there is definitely something different in the MAML versions of our model. Something all the vanilla versions were not able to learn. Also, on the two tasks where the MAML versions are outperformed, the results are better than the ones on the vanilla model. This confirms the fact that MAML is not necessarily a bad idea to handle missing values, but that the model which we decided to use with MAML was a poor choice.

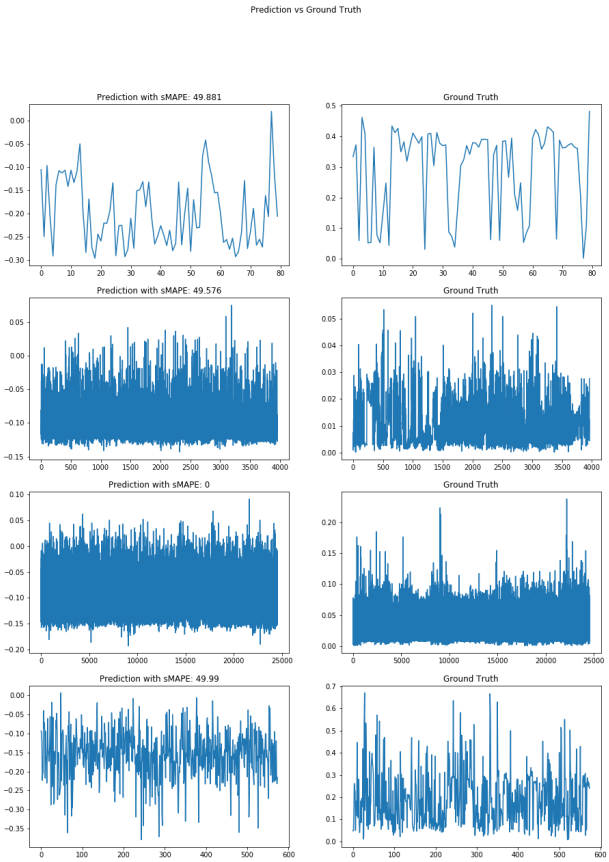


Figure 5.7: GRU results vs real values in M4V2 dataset. Every plot is the comparison of the real values vs the prediction done by our GRU on every subset of M4V2

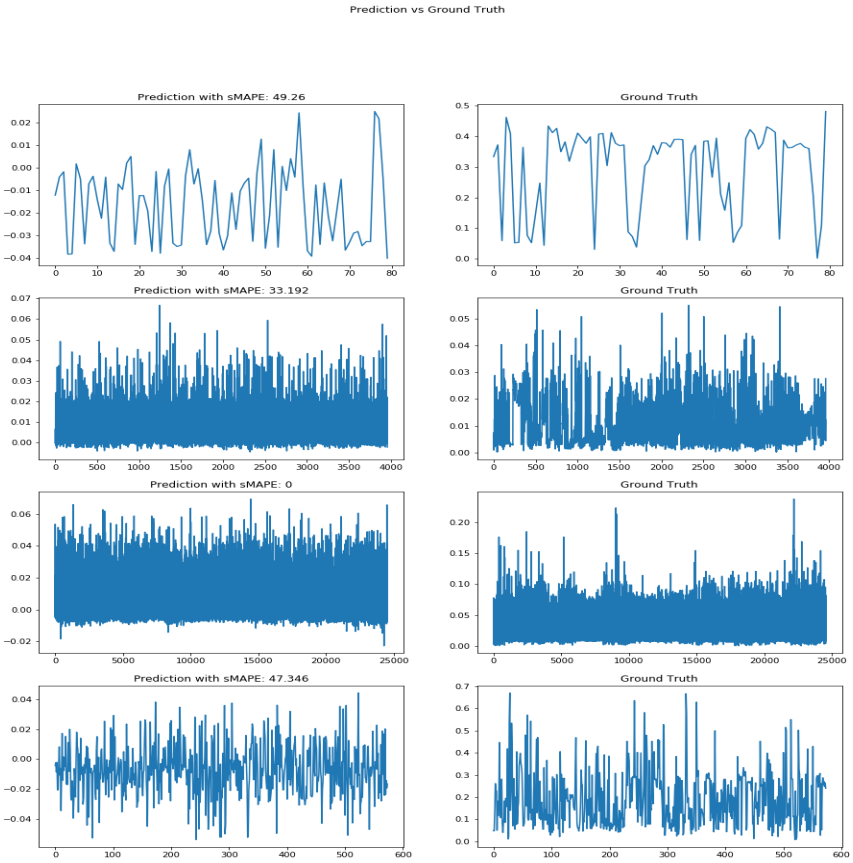


Figure 5.8: MAML GRU results vs real values in M4V2 dataset. Every plot is the comparison of the real values vs the prediction done by our MAML GRU on every subset of M4V2 pre-trained on M4

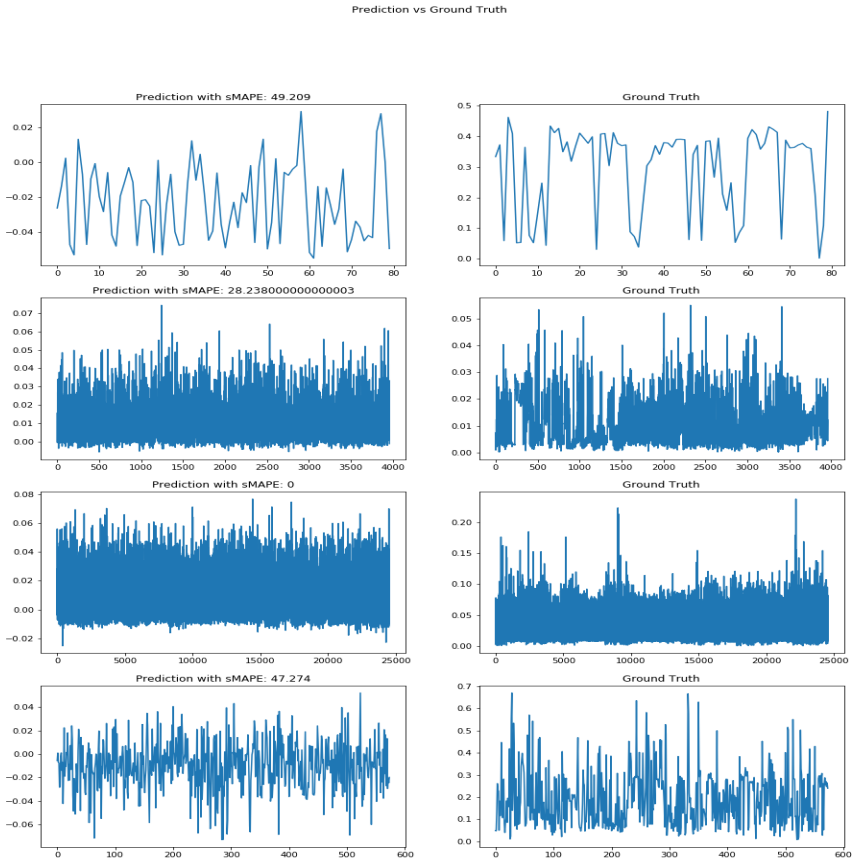


Figure 5.9: MAML GRU results vs real values in M4V2 dataset. Every plot is the comparison of the real values vs the prediction done by our MAML GRU on every subset of M4V2 pre-trained on M4V2

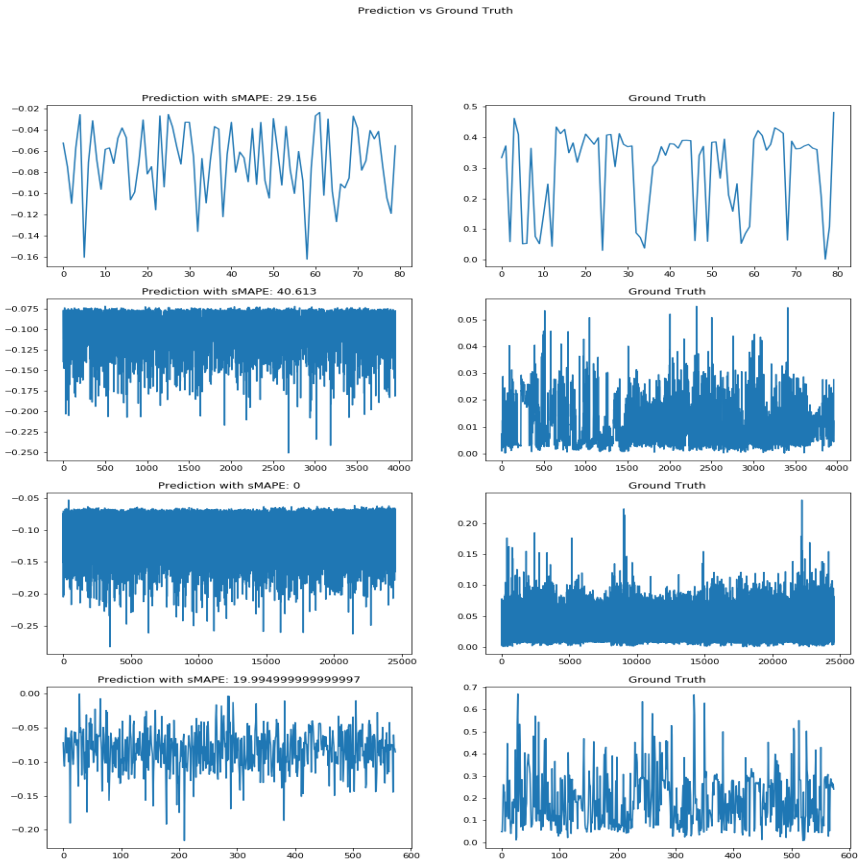


Figure 5.10: Results of GRU-D train and tested on each task of M4V2

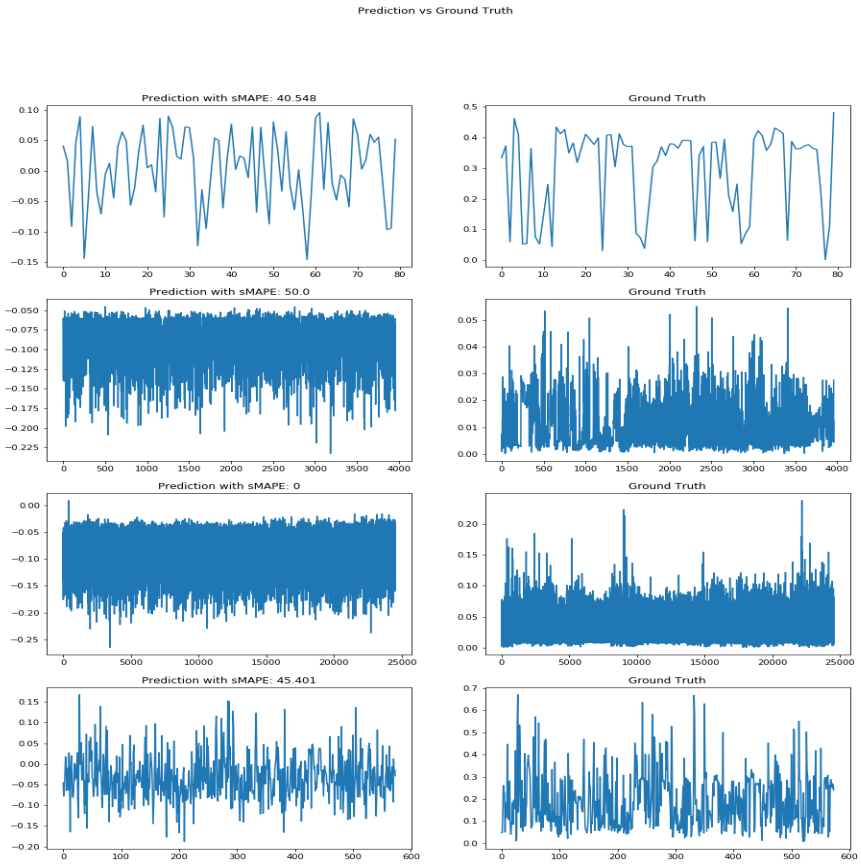


Figure 5.11: Results of our GRU-3D trained and tested on each of M4V2 tasks

Chapter 6

Future Work

After all the results we reached while researching our three questions there are clear points which we could address in a further study. Regarding multivariate time series with missing values, we can conclude that the results are simply not enough to conclude. Therefore, we need a different approach altogether. Among discussions with other colleagues, using an encoder-decoder to perform meta-learning on the encoded vectors came to light and might help towards handling the difference between dimensions sizes. As for the other two research questions, the path is clearer towards future experiments. the most obvious now is to change the vanilla model to be used with MAML. Instead of using our GRU with skip connections for handling missing values, we can use the GRU-D we adapted as the vanilla model. In this case, we expect to find significantly better results and be able to conclude without a doubt that meta-learning is a plausible solution towards handling missing data in univariate time series.

Chapter 7

Conclusion

In this work, we implemented a novel approach towards meta-learning for multivariate time-series with missing values. We used our implementation of GRU-D with skip connections with the MAML framework to show that the curse of dimensionality makes the process inaccurate. The results of the vanilla version of every model used in the Phisyonet dataset outperform their MAML counterparts. Also, we implemented a model that, although inferior to N-BEATS, performs at a similar scale of accuracy in meta-learning for univariate time-series. Beyond that, we used said model to experiment in areas where N-BEATS did not. We used our model for meta-learning for univariate time-series with missing values; proving that a regular network used in meta-learning is outperformed by a model designed for handling missing values. Finally, converting a univariate time-series with missing values into a multivariate time-series with informative missingness, we proved that our GRU-3D with the MAML framework significantly outperforms the GRU that performed at the same scale of N-BEATS.

Bibliography

[1] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. Massachusetts Institute of Technology, 2012

[2] Boris N. Oreshkin, Dimitri Carpov, Nicolas Chapados & Yoshua Bengio. *Meta-Learning framework with applications to zero-shot time-series forecasting*. Element AI, 2020

[3] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag & Yan Liu. *Recurrent Neural Networks for Multivariate Time Series with missing values*. Published by Nature, 2018.

[4] Chelsea Finn, Pieter Abbeel & Sergey Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. International Conference on Machine Learning, 2017.

[5] Phisyonet Dataset. *Predicting mortality of ICU Patients* - The Phisyonet Computing in Cardiology Challenge, 2012. `https://physionet.org/content/challenge-2012/1.0.0/`

[6] M4 Dataset. *Over 10.000 time series from different time deltas* `https://mofc.unic.ac.cy/the-dataset/`

[7] Zhiguang Wang, Weizhong Yan & Tim Oates. *Time Series Classification from scratch with Deep Neural Networks: A strong Baseline.* International Joint Conference on Neural Networks (IJCNN), 2017

[8] Neo Yi Peng. *N-BEATS - Beating statisthical models with pure Neural Nets* Towards Data Science, 2019. `towardsdatascience.com/n-beats-beating-statistical-models-with-neural-nets-28a`

[9] Liang-Chieh Chen, George Papandreau, Iasonas Kokkinos, Kevin Murphy & Alan L. Yuille. *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and fully connected CRFs.* Accepted by TPAMI IEEE May 2017.

[10] Victor Campos, Brendan Jou, Xavier Giró-i-Nieto, Jordi Torres and Shih-Fu Chang. *Skip RNN: Learning to skip State Updates in Recurrent Neural Networks.* Accepted in ICLR 2018.

[11] Tung Kieu, Bin Yang, Chenjuan Guo and Christian S. Jensen. *Outlier Detection for Time Series with Recurrent Autoencoder Ensembles.* Presented in IJCAI-19 2019.

[12] Shahryar Rahnamayan, Hamid R. Tizhoosh and Magdy M.A. Salama. *Opposition versus randomness in soft computing techniques.* Published by Applied Soft Computing in 2008.

[13] Gladilin Peter and Maria Matskevichus. *Hyperparameters Tuning for Machine Learning Models for Time Series Forecasting.* Published by IEEE in 2019.

[14] Milind Sahay. *Neural Networks and the Universal Approximation Theorem.* Towards data science, June 2020. `https://towardsdatascience.com/neural-networks-and-the-universal-approximation`

[15] Rolf H. H. Groenwold. *Informative missingness in electronic health record systems: the curse of knowing.* Published by nature in 2020.

[16] Yujia Xie, Haoming Jiang, Feng Liu, Tuo Zhao and Hongyuan Zha. *Meta Learning with Relational Information for short sequences.* NIPS 2019.

[17] Mikhail Khodak, Maria-Florina Balcan and Ameet Talwalkar. *Adaptive Gradient-Based Meta-Learning Methods.* NIPS 2019.

[18] Thiyanga S Talagala, Rob J Hyndman and George Athanasoupoulos. *Meta-learning how to forecast time series.* Published in 2018.

[19] Rafael Rego Drumond, Lukas Brinkmayer, Josif Grabocka and Lars Schmidt-Thieme. *HYDRA: Head Initialization across Dynamic targets for Robust Architectures.* Accepted for publication on SIAM 2020.