On-the-fly Data Augmentation for Forecasting with Deep Learning

Vitor Cerqueira[a,b,*], **Moisés Santos**[a,b], **Yassine Baghoussi**[a,d] **and Carlos Soares**[a,b,c]

[a]Faculdade de Engenharia da Universidade do Porto, Porto, Portugal
[b]Laboratory for Artificial Intelligence and Computer Science (LIACC), Portugal
[c]Fraunhofer Portugal AICOS, Portugal
[d]INESC TEC, Porto, Portugal

Abstract.

Deep learning approaches are increasingly used to tackle forecasting tasks. A key factor in the successful application of these methods is a large enough training sample size, which is not always available. In these scenarios, synthetic data generation techniques are usually applied to augment the dataset. Data augmentation is typically applied before fitting a model. However, these approaches create a single augmented dataset, potentially limiting their effectiveness. This work introduces `OnDAT` (On-the-fly Data Augmentation for Time series) to address this issue by applying data augmentation during training and validation. Contrary to traditional methods that create a single, static augmented dataset beforehand, `OnDAT` performs augmentation on-the-fly. By generating a new augmented dataset on each iteration, the model is exposed to a constantly changing augmented data variations. We hypothesize this process enables a better exploration of the data space, which reduces the potential for overfitting and improves forecasting performance. We validated the proposed approach using a state-of-the-art deep learning forecasting method and 8 benchmark datasets containing a total of 75797 time series. The experiments suggest that `OnDAT` leads to better forecasting performance than a strategy that applies data augmentation before training as well as a strategy that does not involve data augmentation. The method and experiments are publicly available.

1 Introduction

Time series forecasting is a relevant problem with vast real-world applications. Increasingly, deep neural networks are emerging as a powerful alternative to well-established approaches such as ARIMA or exponential smoothing [14]. Neural architectures such as NHITS [7], N-BEATS [25], or ES-RNN [31], have recently exhibited state-of-the-art forecasting performance in benchmark datasets and competitions.

It is widely accepted that deep neural networks require large amounts of data for an adequate generalization [2]. However, time series datasets sometimes have a limited number of time series. Or, each time series in the database may contain only a small number of data points. In effect, a sufficiently large dataset may not be readily available for training deep learning models.

In these scenarios, data augmentation techniques can be used to tackle data scarcity and increase the sample size. Various data augmentation methods have been developed for time series data. These range from simple approaches such as jittering to more sophisticated techniques involving generative models [32]. In this work, we address univariate time series forecasting problems with datasets containing multiple time series. For these tasks, data augmentation processes have been shown to improve forecasting performance of recurrent neural networks [2]. Besides time series problems, data augmentation is also common in tasks such as image or text classification [29].

Data augmentation is typically done apriori, i.e. before training a model (e.g. [2]). A synthetic dataset is created and combined with the original data, leading to an augmented dataset. This augmented dataset is larger and more diverse, thereby improving the performance of models. We hypothesize that this approach is limited because only a single augmented dataset is created. The resulting dataset may not cover the data generation process well, due to either randomness or inadequacy of the augmentation technique. Besides, we may need to store and load the augmented dataset in memory, which is computationally demanding in some cases.

We can tackle these issues by doing data augmentation during the training process, i.e., on-the-fly data augmentation [24]. By augmenting the original data with a new synthetic dataset at each iteration, the model is exposed to a constantly changing dataset variations. This process can be beneficial because it enables a better exploration of the underlying data generation process. This has the potential to prevent the model from overfitting spurious patterns, thereby improving forecasting performance. On-the-fly data augmentation has been recently used in speech recognition problems with promising results [26, 24]. To our knowledge, this is the first application of this approach to time series data and forecasting tasks.

This paper presents a novel approach for training deep learning forecasting models using data augmentation called `OnDAT` (On-the-fly Data Augmentation for Time series). `OnDAT` works by augmenting the mini-batch of time series at each iteration of a deep neural network. We formalize `OnDAT` using a data augmentation technique based on seasonal decomposition and moving blocks bootstrapping (MBB) [3], though other methods can be applied. The randomness of the bootstrapping procedure ensures that the augmented dataset is different at every run. Moreover, `OnDAT` involves creating a augmented dataset during both the training and validation steps. The validation stage benefits from data augmentation to obtain better performance estimates, which also improves the early stopping and model

* Corresponding Author. Email: vcerqueira@fe.up.pt.

checkpointing mechanisms.

We validate the proposed method using 75797 time series from 8 benchmark databases. The results of the experiments suggest that OnDAT leads to better forecasting performance than apriori data augmentation or no augmentation. In summary, the contributions of this work are the following:

- The formalization of on-the-fly data augmentation for univariate time series forecasting problems;
- OnDAT, a method that applies on-the-fly data augmentation based on seasonal decomposition and bootstrapping for training deep learning global forecasting models;
- An empirical study comparing OnDAT with different training strategies and data augmentation approaches, including a discussion about the implications in terms of forecasting performance and computational scalability.

All methods are implemented using the neuralforecast Python library, which is based on PyTorch, and the experiments are fully reproducible[1]. The rest of this paper is organized in 5 sections. Section 2 provides a background to our work, including the definition of the predictive task and an overview of the related work. The following section (Section 3) formalizes the proposed method, OnDAT, for data augmentation during the training process of neural networks. The experiments are presented in Section 4. We describe the experimental design, including the methods and datasets. We also summarise the results, which are discussed in Section 5. The conclusions of this work are presented in Section 6.

2 Background

This section provides a background to our work. We start by describing the task of univariate time series forecasting (Section 2.1). Then, we explore how deep learning is used to tackle this problem, focusing on how these methods leverage multiple time series to train a model (Section 2.2). Finally, in Section 2.3, we overview the related work on data augmentation in the context of time series.

2.1 Univariate Time Series Forecasting

A univariate time series is a time-ordered sequence of values $Y = \{y_1, y_2, \ldots, y_t\}$, where $y_i \in \mathbb{R}$ is the value of Y observed at time i and t is the length of Y. Forecasting is the process of predicting the value of upcoming observations $y_{t+1}, \ldots, y_{t+h}$, where h denotes the forecasting horizon. The importance of forecasting spans several domains, including applications in business areas such as inventory management or operations planning [16].

Machine learning methods tackle forecasting problems using an auto-regressive type of modeling. According to this approach, each observation of a time series is modeled as a function of its past q lags based on time delay embedding [4]. Time delay embedding is the process of transforming a time series from a sequence of values into an Euclidean space. In practice, the idea is to apply sliding windows to build a dataset $\mathcal{D} = \{\mathrm{X},\mathrm{y}\}_{q+1}^{t}$ where y_i represents the i-th observation and $X_i \in \mathbb{R}^q$ is the i-th corresponding set of q lags: $X_i = \{y_{i-1}, y_{i-2}, \ldots, y_{i-q}\}$. Accordingly, the objective is to train a regression model to learn the dependency $y_i = f(X_i)$.

Forecasting problems often involve datasets with multiple univariate time series. We define a collection of n time series as $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_n\}$. Traditional approaches build a forecasting model for each univariate time series within a collection $\mathcal{Y}$. Such models are commonly known as local models [15]. Leveraging the data of all available time series can be valuable to build a forecasting model. The dynamics of the time series within a collection are often related, and a model may learn useful patterns in some time series that are not revealed in others. Models that are trained on collections of time series are referred to as global forecasting models [13].

For global approaches, the time delay embedding framework described above is applied to each time series in the collection. Then, an auto-regression model is trained on the joint dataset that contains all transformed time series. In effect, for a collection of time series the dataset $\mathcal{D}$ is composed of a concatenation of the individual datasets: $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$, where $\mathcal{D}_j$ is the dataset corresponding to the time series Y_j.

2.2 Forecasting with Deep Learning

In most cases, global forecasting models are trained using deep neural networks. Various types of neural architectures have been recently developed for time series forecasting. Architectures based on recurrent neural networks, such as the LSTM [30], are more common due to their intrinsic capabilities for sequence modeling. However, deep neural networks based on convolutional layers [33], transformers [34, 20], or multi-layer perceptrons (MLPs) [7] have also shown competitive forecasting accuracy.

The adoption of deep learning to solve forecasting problems surged when Smyl [31] presented an LSTM-based model that won the M4 forecasting competition that features 100.000 time series from various application domains. The method, known as ES-RNN, combines exponential smoothing with an LSTM neural network. Exponential smoothing is used to de-seasonalize and normalized time series, while the neural network is trained on the transformed data in a global fashion. Another popular recurrent-based architecture is DeepAR [27]. DeepAR leverages a stack of auto-regressive LSTM layers and Markov Chain Monte Carlo sampling to produce probabilistic forecasts.

Several deep learning forecasting methods adopt the transformer architecture, which has been driving important advances in natural language processing tasks. Examples include the Temporal Fusion Transformer [20], and the Informer [34].

The MLP is one of the simplest neural network architectures, which is composed of stacks of fully connected layers. Neural networks based on MLP have been used to tackle forecasting problems for several years. Recently, two particular architectures have shown promising performance and computational scalability: N-BEATS [25] and NHITS [7]. Both are based on stacks that contain blocks of MLPs along with residual connections. NHITS [7], short for Neural Hierarchical Interpolation for Time Series Forecasting, extends N-BEATS [25] by including multi-rate input sampling that models data with different scales and hierarchical interpolation for better long-horizon forecasting. NHITS has shown state-of-the-art forecasting performance relative to other deep learning forecasting methods, including several transformers, and recurrent-based neural networks [7]. Challu et al. [7] also provide evidence that NHITS is more computationally efficient than transformer-based methods by a factor of 50.

2.3 Time Series Data Augmentation

One key motivation for adopting a global approach is the additional data available for training a model. Machine learning algorithms tend

[1] https://github.com/vcerqueira/ondat

to perform better with larger training sets [6]. However, an adequate sample size may not be readily available. This issue can be solved using data augmentation techniques [2].

There are several methods for time series data augmentation. Flipping or jittering the value of observations are among the simplest approaches to create a synthetic time series from an original one [32]. More sophisticated methods use seasonal decomposition [8] or generative models [17] to drive the data augmentation process.

Recently, Bandara et al. [2] tested three different data augmentation methods that improved the forecasting performance of an LSTM neural network. These three methods are the following: dynamic time warping barycentric averaging (DBA) [11], moving blocks bootstrapping (MBB) [3], and GRATIS (GeneRAting TIme Series with diverse and controllable characteristics) [17]. DBA, proposed by Forestier et al. [11], involves averaging multiple time series to create a synthetic one. The average is weighted based on dynamic time warping. Bergmeir et al. [3] use a method based on seasonal decomposition and MBB to create a synthetic version of a given time series. We will describe this method in detail in the next section. Finally, GRATIS [17] leverages a Gaussian mixture of auto-regressive models to create a set of time series with diverse and controllable features. Most data augmentation methods create new observations based on the original ones. On the other hand, GRATIS creates synthetic data disregarding the data generation process of the original time series.

Bandara et al. [2] used these data augmentation methods to create a larger training sample size. They first augment the original time series dataset using one of these methods and then train an LSTM on the augmented dataset. We hypothesize that this approach is limited because only a single synthetic dataset is created, which may miss relevant aspects of the data generation process.

Another relevant data augmentation approach is time series morphing, developed by Santos et al. [28]. Morphing consists in gradually transforming a time series into another, which the authors use for understanding the performance of forecasting models.

3 OnDAT

This section formalizes the methodology behind `OnDAT`. We aim at increasing the sample size for training and validating deep neural networks with the ultimate goal of improving their forecasting performance. We focus on univariate forecasting problems, particularly on datasets involving multiple time series where models are trained on a global fashion (c.f. Section 2.2).

We start by describing the data augmentation technique behind `OnDAT` (Section 3.1), which is based on seasonal decomposition and bootstrapping. Then, we describe how this method is applied in each iteration of the fitting process of a neural network (Section 3.2). Finally, we present some of the benefits of data augmentation applied to the validation data (Section 3.3).

3.1 Data augmentation technique

We formalize `OnDAT` using a particular data augmentation technique based on seasonal decomposition using STL (Seasonal and Trend decomposition using LOESS) [8] and MBB [19], following Bergmeir et al. [3]. The idea is to extract the remainder component of a given univariate time series, and resample it using a bootstrapped-based technique. The resampled remainder is then combined back with the other components, leading to a bootstrapped version of the original series. This approach has been shown to improve forecasting performance both in terms of apriori data augmentation [2] as well as bagging forecasting models [3].

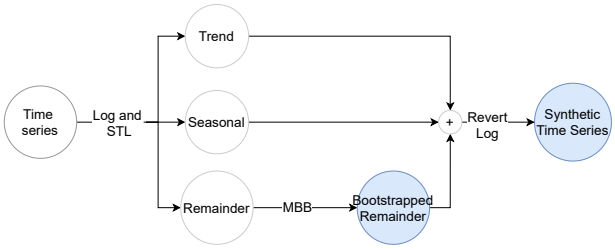

Figure 1. High-level workflow of the data augmentation process behind `OnDAT` based on STL and MBB for a single time series. The blue shaded circle represent synthetic data.

We follow the particular approach by Bergmeir et al. [3] to create synthetic time series. The process, which we illustrate in Figure 1, is carried out as follows for a given univariate time series:

1. First, we stabilize the variance of the time series by taking the logarithm transformation. The logarithm ensures that the time series seasonal decomposition is additive [14].
2. Then, we decompose the log-transformed time series into trend, seasonal, and remainder components using STL [8], a state-of-the-art method for seasonal decomposition. This stage is important to handle non-stationary time series.
3. The remainder (stationary) component is resampled using MBB [18]. MBB is a variation of the bootstrapping technique [10] that is tailored for time-dependent data. Essentially, the remainder series is split into contiguous subsets (blocks) based on a sliding window of size l. We set l equal to the seasonal period for capturing a full seasonal cycle, for example $l = 12$ for monthly time series. This process leads to $t - l - 1$ overlapping blocks, where t is the size of the time series. Then, we apply bootstrapping and randomly draw (with replacement) blocks until we reach t observations. This process results in a bootstrap version of the remainder component.
4. The boostrapped remainder time series is combined back with the trend and seasonal components, reverting the seasonal decomposition process.
5. Finally, we revert the logarithm transformation by taking the exponential. This process leads to a bootstrapped version of the original series.

Figure 2 shows a time series (top) and two examples of synthetic time series generated using the process described above.

In this work, we assume that the time series contains a well-defined seasonal period for seasonal decomposition. Nonetheless, for non-seasonal time series, the decomposition can be carried out using LOESS [9], as described by Bergmeir et al [3]. Alternatively, we can use a different data augmentation technique that does not require time series decomposition (c.f. Section 2.3).

3.2 On-the-fly data augmentation

The typical way of applying data augmentation methods, such as the one described in the previous section, is before the training process. The proposed method involves augmenting the original dataset during the training process.

Figure 3 shows a high-level diagram that details the workflow in a given training step of a deep neural network for univariate time series forecasting. First, we get a mini-batch of time series, following

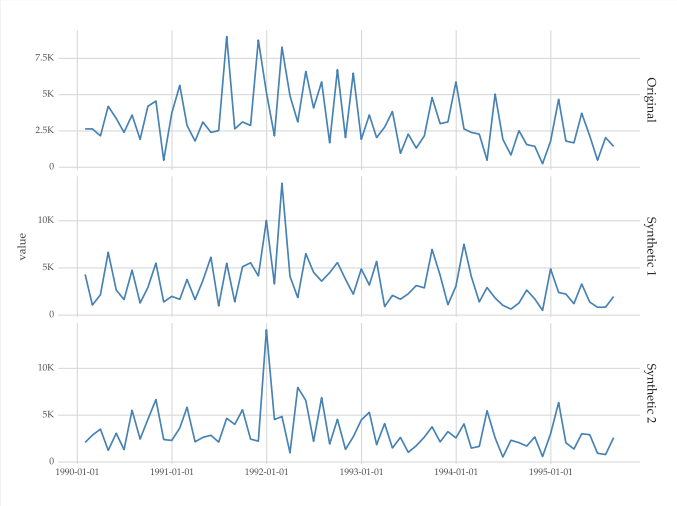

Figure 2. Example application of MBB in the time series with id "M1" from M3 dataset. Two runs of the synthetic data generation process lead to two different synthetic time series.

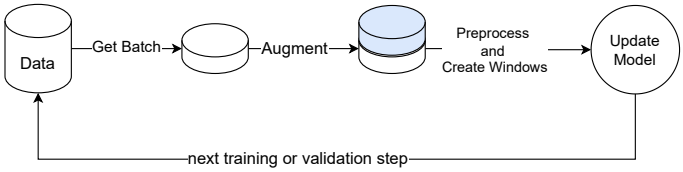

Figure 3. High-level workflow of the data augmentation process behind `OnDAT` for a given iteration of a deep neural network for forecasting.

the standard mini-batch approach for training neural networks. We augment this batch using the MBB-based method described in the previous section. For each time series in a given batch, we create a single synthetic variation of it, doubling the number of time series in the batch. If necessary, data prepararion is done in the new, synthetic series, in the same way as for the original ones, for example the process of creating windows for auto-regression (c.f. Section 2.1). Finally, the processed data is used to train or validate the model. This process repeats until the training process finishes.

In general, the data augmentation method should have two characteristics to be applied on-the-fly:

1. Be non-deterministic, so that each run leads to a different augmented dataset. In our case, this is guaranteed by the random nature of bootstrapping;
2. Be computationally efficient, so it is feasible to apply the method at each training or validation step of a neural network.

3.3 Data augmentation in validation

The development of machine learning models usually involves evaluating these models in samples that are not available during the training process to avoid overly-optimistic performance estimates [5]. Similarly, algorithm selection and hyperparameter optimization is best done using a validation set.

With i.i.d. data, the data partitioning process into training, validation, and testing sets is typically done randomly. For time series, the temporal aspect of data poses an additional challenge. The temporal

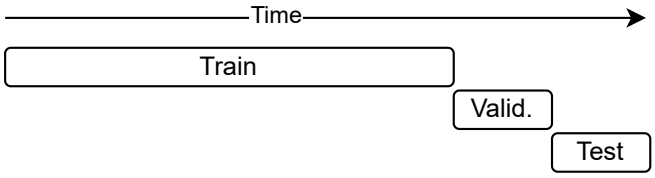

Figure 4. Traditional time series data partitioning process into training, validation, and test sets for developing forecasting models.

order of observations must be preserved for cross-validation to produce reliable performance estimates. In effect, all observations of the testing set are always future to the training samples, as illustrated in Figure 4.

The same rationale also applies for the validation set, which is typically built with the last part of the training set. This leads to a data scarcity issue due to temporal constraints because we cannot use the observation for both the training and validation steps. In effect, even in cases where the number of time series of the original dataset is not small, the validation sample size may be insufficient and data augmentation can be beneficial. In this context, we also apply `OnDAT` during the validation stage. This process is useful to get better performance estimates as well as drive early stopping and model checkpointing mechanisms.

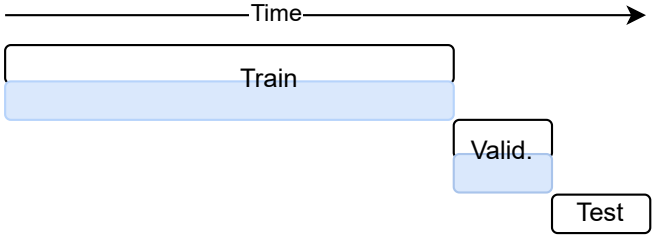

Figure 5. Proposed time series data partitioning process into training, validation, and test sets.

Figure 5 illustrates the additional data when using `OnDAT` at a given training or validation step of a neural network.

4 Experiments

We carried out extensive experiments to evaluate the performance of `OnDAT`. The experiments focus on the following research questions:

- RQ1: How does a neural network trained using on-the-fly data augmentation based on `OnDAT` perform relative to other approaches, such as apriori data augmentation, or benchmark techniques for univariate time series forecasting?
- RQ2: What is the impact of the different elements of `OnDAT` on forecasting performance?
- RQ3: Does applying `OnDAT` during the validation stage improve early stopping and model checkpointing, as well as the model performance estimates?
- RQ4: How does `OnDAT` compare with other approaches for training a neural network in terms of execution time?

4.1 Data

We use 8 datasets from four databases and forecasting competitions: M1 [22], M3 [21], M4 [23], and Tourism [1]. M1, M3, and M4 are

time series databases that come from the Makridakis competitions. These datasets cover different application domains, including industry, demography, and economics. Tourism is a database focused on the tourism domain. These datasets, which represent standard benchmarks for univariate time series forecasting, are summarised in Table 1.

Table 1. Summary of the datasets: average value, number of time series, and number of observations.

Dataset	Average value	# time series	# observations
M1 Monthly	72.7	617	44892
M1 Quarterly	40.9	203	8320
M3 Monthly	117.3	1428	167562
M3 Quarterly	48.9	756	37004
M4 Monthly	234.3	48000	11246411
M4 Quarterly	100.2	24000	2406108
Tourism Monthly	298.5	366	109280
Tourism Quarterly	99.6	427	42544
Total	-	75797	14062121

We focus on databases with low-frequency time series, namely monthly and quarterly time series. We exclude time series with an yearly sampling frequency because these do not have a well-defined seasonal period, which is important for the seasonal decomposition method described in Section 3.2. High-frequency time series, such as hourly or daily, may exhibit complex seasonality which can undermine the performance of STL. Besides, datasets with low-frequency are the ones that tend to comprise a lower number of observations, and where data augmentation can be more useful. Overall, the combined datasets contain 14062121 observations across 75797 univariate time series.

In terms of forecasting horizon, we set this value to 18 and 8, for monthly and quarterly time series, respectively. The input size is set to 24 for monthly time series and 8 for quarterly ones. These values correspond to two seasonal cycles.

4.2 Methods

We compare `OnDAT` with the following approaches:

- `Standard`: The standard training procedure that does not involve data augmentation. The training and validation steps are carried out using the original data.
- `DA`: An approach that does data augmentation before the fitting process, following Bandara et al. [2]. First, we create a synthetic version of each time series in the available data using MBB and merge it with the original data. Then, we train and validate the neural network using the augmented data.
- `SeasonalNaive`: The benchmark technique that uses the last known observation of the same season as the forecast.

Regarding the neural network architecture, we focus on NHITS [7], which we described in Section 2.3. We focus on a single deep learning approach in our study as NHITS has shown state-of-the-art forecasting performance relative to other deep learning forecasting methods, including several transformers and recurrent-based neural networks [7]. Besides a competitive forecasting performance, NHITS is also more computational efficient than other popular architectures [7]. We use the NHITS implementation available on neuralforecast[2] Python library, which is based on PyTorch.

[2] https://nixtlaverse.nixtla.io/neuralforecast/models.nhits.html

We train one global NHITS model for each dataset listed in Table 1. The configuration of NHITS is fixed in all training procedures. We build NHITS models with 3 stacks, each of which with one block of multi-layer perceptrons. Each multi-layer perceptron contains 2 hidden layers, each with 512 units. The activation function is set to the rectified linear unit. These values are default on the implementation available on neuralforecast Python library.

The training process parameter are based on the work of Chally et al. [7]. NHITS is trained for a maximum of 1500 training steps using ADAM optimizer and an initial learning rate of 0.001, which is updated 3 times during training. We also apply early stopping and model checkpointing based on validation performance. Early stopping is a mechanism where the training process stops earlier if the validation scores do not improve for a number of patience steps. We the the patience value to 50 training steps. Model checkpointing involves saving the best model found during training according to validation scores. After the training finishes, the final model is the one currently stored, i.e., the one that maximizes validation performance.

4.3 Data partitioning and evaluation

For performance estimation, we leave the last H observations of each time series for testing. The remaining available observations are used for training and validating the model. The validation set is composed of the final H observations of each time series, similarly to the test split. The data partitioning process is illustrated in Figure 4.

We use SMAPE (symmetric mean absolute percentage error) as evaluation metric. This metric is often used in forecasting competitions, such as M4 [23], and can be defined as:

$$\text{SMAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|\hat{y}_i - y_i|}{(|\hat{y}_i| + |y_i|)/2} \tag{1}$$

where $\hat{y}_i$, and y_i are the forecast and actual value for the i-th instance, respectively.

4.4 Results

This section presents the results of the experiments, which we organize by research question.

4.4.1 RQ1

The main results of the experiments are presented in Table 2, where the best score is shown in a bold font. The last couple of rows of the table report the average score and average rank of each method.

Overall, all NHITS variations outperform the seasonal naive baseline, which validates the forecasting accuracy of the NHITS neural network. Both approaches based on data augmentation (`OnDAT` and `DA`) outperform a standard approach that only uses the original data. This outcome emphasises the importance of data augmentation for forecasting with deep learning. `OnDAT` presents a competitive performance, with the best score in 6 out of the 8 datasets. This culminates in the best average SMAPE and best average rank.

We noticed that the impact of data augmentation is bigger in smaller datasets, which is expected. For example, the `Standard` method (no data augmentation) shows the best score on M4 Quarterly, the second largest dataset. The improvements from data augmentation in M4 Monthly (the largest dataset) are negligible. Overall, the performance scores are systematic in favor of the proposed method. This result suggests that on-the-fly data augmentation is a promising direction for building more accurate forecasting models.

Table 2. SMAPE scores for each method in each dataset. Bold font represents the best method in the respective dataset. The last two rows denote the average score and average rank, respectively.

	OnDAT	DA	Standard	SeasonalNaive
M1 Monthly	**0.15950**	0.15977	0.16053	0.19096
M1 Quarterly	**0.12580**	0.12865	0.13190	0.16330
M3 Monthly	0.14594	**0.14519**	0.15003	0.17638
M3 Quarterly	**0.08139**	0.08167	0.08258	0.10704
M4 Monthly	0.12208	**0.12166**	0.12235	0.15746
M4 Quarterly	0.10518	0.10487	**0.10470**	0.13015
Tourism Monthly	**0.21619**	0.21666	0.21667	0.23916
Tourism Quarterly	**0.18777**	0.19369	0.18887	0.19741
Average	**0.14298**	0.14402	0.1447	0.17023
Average Rank	**1.44**	1.89	2.67	4.0

4.4.2 RQ2

We carried out an ablation analysis to understand the effect of different elements in the performance of OnDAT. We test the following variants:

- OnDAT(tr): A variant of the proposed method in which data augmentation is only carried out during the training steps. The validation stage is done using the original dataset;
- OnDAT(vl): The inverse variant of OnDAT(tr). In this case, data augmentation is only carried out during the validation stage;
- OnDAT(gratis): The implementation of OnDAT using GRATIS for data augmentation. In this variation, we replaced the data augmentation method based on seasonal decomposition and MBB with GRATIS [17], which we described in Section 2.3. Bandara et al. [2] showed that synthetic time series created using GRATIS improve forecasting performance in an apriori data augmentation setting. We apply GRATIS in an on-the-fly setting, as described in Section 3.2. Essentially, we duplicate each training or validation batch with synthetic time series created using GRATIS. The only constraint we set to the synthetic data generation process is the seasonal period, which we set according to the original data (e.g. 12 for monthly time series). Besides that, we also normalize the values of each synthetic time series based on their maximum value.
- OnDAT(fixed): Another variant of OnDAT, in which we tweak the data augmentation method. Instead of applying MBB to resample the remainder component, we apply the standard bootstrap method [10]. As such, observations from the remainder component are selected at random with replacement disregarding the order of observations.

Table 3. SMAPE score of different variants of the proposed method. Values in bold represent the best score in the respective problem.

	OnDAT	OnDAT(tr)	OnDAT(vl)	OnDAT(gratis)	OnDAT(fixed)
M1 Monthly	0.1591	0.1584	0.1626	0.1592	**0.1576**
M1 Quarterly	**0.1246**	0.1259	0.1319	0.1313	0.1270
M3 Monthly	**0.1471**	0.1490	0.1500	0.1488	0.1480
M3 Quarterly	**0.081**	0.0815	0.0835	0.0822	0.0811
M4 Monthly	**0.1215**	**0.1215**	0.1223	0.1217	**0.1215**
M4 Quarterly	0.1051	0.1050	**0.1047**	0.1052	0.1051
Tourism Monthly	**0.2115**	0.2126	0.2167	0.2173	0.2132
Tourism Quarterly	0.1877	0.1879	0.1950	0.1882	**0.1850**
Average	**0.1422**	0.1428	0.1458	0.1442	0.1423
Average Rank	**1.67**	2.56	4.44	4.11	2.22

The results are presented in Table 3. Overall, we found that applying data augmentation in both training and validation stages is important. As expected, the additional data is more impactful when applied during training as suggested by the better performance shown by OnDAT(tr) when compared with OnDAT(vl). Regarding OnDAT(fixed), it shows a comparable performance with OnDAT. This means that the moving blocks aspect within the bootstrapping process is not a critical component in the data augmentation technique. Finally, OnDAT shows a better performance with the MBB-based data augmentation technique than with GRATIS.

4.4.3 RQ3

We also analyzed the impact of OnDAT in the validation stage, specifically in terms of performance estimation. For each approach, we measured the difference between validation and testing SMAPE scores to understand how well the performance is estimated using validation.

Table 4. Difference between validation SMAPE and testing SMAPE for each method and in each dataset. A positive value denotes higher SMAPE on validation. Bold values denotes the best estimation (closest to zero) in the respective dataset.

	DA	Standard	OnDAT
M1 Quarterly	**0.045**	0.059	0.048
M1 Monthly	**0.046**	0.049	0.048
M3 Monthly	0.378	0.382	**0.374**
M3 Quarterly	0.136	0.138	**0.129**
Tourism Monthly	0.036	0.029	**0.022**
Tourism Quarterly	-0.005	**-0.004**	-0.025
M4 Monthly	0.008	0.012	**0.003**
M4 Quarterly	0.034	0.042	**0.028**
Median	0.040	0.045	**0.038**

The results are shown in Table 4, which shows the difference between validation SMAPE and testing SMAPE for each method and in each dataset, along with the median difference. A positive value denotes higher SMAPE on validation. All methods show a positive median value, which means the SMAPE score is larger in validation than in testing, on average. In effect, the validation process is pessimistic (forecasting performance is underestimated). OnDAT shows the results closer to zero most of the time, which suggests that it provides a better estimate for the testing performance of models compared to the other two methods.

4.4.4 RQ4

Finally, we measure the execution time of each strategy for training the neural network. In Table 5, we report the average percentage

difference in execution time of DA and Standard with respect to OnDAT. Negative values denote a quicker execution for the respective method. By execution, we refer to the complete training plus testing cross validation cycle.

Table 5. Average percentage difference in execution time of each approach and OnDAT. Negative values denote a quicker execution by the respective method.

DA	Standard
-2.885%	-35.157%

Overall, OnDAT takes longer to execute than both Standard and DA, though the difference to the latter is small. The Standard approach that does not apply data augmentation runs about 35% faster than OnDAT.

5 Discussion

The main contribution of this work is an approach for applying data augmentation during the training process of neural networks for univariate time series forecasting. We formalized OnDAT using a particular data augmentation technique based on seasonal decomposition and MBB, following the work by Bergmeir et al. [3]. This method has been shown to improve performance of LSTM neural networks [2]. However, we remark that the proposed method is agnostic to the underlying data augmentation method and it is applicable with others, such as DBA. OnDAT is also applicable with other neural network architectures besides NHITS, which we used in the experiments.

We validate the proposed method with an extensive empirical analysis. The experiments provide evidence about the usefulness of on-the-fly data augmentation for forecasting problems. On-the-fly data augmentation based on OnDAT leads to better forecasting performance than a strategy that applies data augmentation before training. Both approaches outperform the approach that does not involve data augmentation, which underlines the benefits of synthetic data. Finally, all strategies show better forecasting performance than a seasonal naive benchmark, validating the forecasting accuracy of models (RQ1).

We controlled the analysis by different elements of OnDAT (RQ2). We found that carrying out data augmentation during both the training and validation steps is important. Yet, the moving blocks component of the bootstrapping-based synthetic data generation process does not bear a significant impact in forecasting performance. The MBB-based data augmentation method shows better performance than GRATIS. This outcome suggests that creating synthetic time series based on the original data generation process can be important. While GRATIS creates realistic time series, it disregards the original data.

We analysed the impact of OnDAT in the validation process of the neural network (RQ3). We found that the performance estimates of OnDAT computed during validation are a better approximation of testing performance relative to DA and Standard. This can also result in better early stopping and model checkpoint mechanisms.

Finally, we studied the relative execution time of each approach (RQ4). On-the-fly data augmentation leads to a larger execution time, though the difference is not considerable relative to apriori data augmentation (DA). Notwithstanding, the increased execution time only affects the fitting process and not the inference stage of the model.

We resorted to 8 datasets which comprise over 14 million observations across more than 75 thousand time series. These datasets only contain two sampling frequencies: monthly and quarterly. The choice for this type of time series data was driven by two main reasons. First, datasets with low-frequency time series usually contain a small number of observations. Data augmentation is usually more relevant in these cases. Besides this, these time series exhibit a well-defined seasonal period which can be important for STL seasonal decomposition behind the data augmentation method applied.

Overall, the experimental results show that on-the-fly data augmentation is a promising direction for time series forecasting. In future work, we will study how to improve the data augmentation process during training. For example, introduce a mechanism that weights time series based on their impact on the training process, akin to boosting methods [12].

6 Conclusions

Data augmentation techniques are useful in contexts where insufficient data is available to train adequate deep learning models. In time series forecasting problems, data augmentation is typically done before the training process. We argue that this approach is limited because only a single augmented dataset is created. In this paper, we proposed a novel approach called OnDAT, which applies data augmentation on-the-fly, i.e., during the training process. OnDAT works by augmenting each batch of time series in each training or validation step using seasonal decomposition and MBB. This process enables a large number of synthetic dataset variations. We carried out extensive experiments that suggests that OnDAT leads to better forecasting performance than a strategy that applies data augmentation before training and a strategy that does not involve data augmentation.

Acknowledgements

This work was partially funded by projects AISym4Med (101095387) supported by Horizon Europe Cluster 1: Health, ConnectedHealth (n.º 46858), supported by Competitiveness and Internationalisation Operational Programme (POCI) and Lisbon Regional Operational Programme (LISBOA 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (ERDF) and NextGenAI - Center for Responsible AI (2022-C05i0102-02), supported by IAPMEI, and also by FCT plurianual funding for 2020-2023 of LIACC (UIDB/00027/2020 UIDP/00027/2020)

References

[1] G. Athanasopoulos, R. J. Hyndman, H. Song, and D. C. Wu. The tourism forecasting competition. *International Journal of Forecasting*, 27(3):822–844, 2011.

[2] K. Bandara, H. Hewamalage, Y.-H. Liu, Y. Kang, and C. Bergmeir. Improving the accuracy of global forecasting models using time series data augmentation. *Pattern Recognition*, 120:108148, 2021.

[3] C. Bergmeir, R. J. Hyndman, and J. M. Benítez. Bagging exponential smoothing methods using stl decomposition and box–cox transformation. *International journal of forecasting*, 32(2):303–312, 2016.

[4] G. Bontempi, S. Ben Taieb, and Y.-A. Le Borgne. Machine learning strategies for time series forecasting. *Business Intelligence: Second European Summer School, eBISS 2012, Brussels, Belgium, July 15-21, 2012, Tutorial Lectures 2*, pages 62–77, 2013.

[5] V. Cerqueira, L. Torgo, and I. Mozetič. Evaluating time series forecasting models: An empirical study on performance estimation methods. *Machine Learning*, 109(11):1997–2028, 2020.

[6] V. Cerqueira, L. Torgo, and C. Soares. A case study comparing machine learning with statistical methods for time series forecasting: size matters. *Journal of Intelligent Information Systems*, 59(2):415–433, 2022.

[7] C. Challu, K. G. Olivares, B. N. Oreshkin, F. G. Ramirez, M. M. Canseco, and A. Dubrawski. Nhits: Neural hierarchical interpolation for time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6989–6997, 2023.

[8] R. B. Cleveland, W. S. Cleveland, J. E. McRae, I. Terpenning, et al. Stl: A seasonal-trend decomposition. *J. Off. Stat*, 6(1):3–73, 1990.

[9] W. S. Cleveland, E. Grosse, and W. M. Shyu. Local regression models. In *Statistical models in S*, pages 309–376. Routledge, 2017.

[10] B. Efron. Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics: Methodology and distribution*, pages 569–593. Springer, 1992.

[11] G. Forestier, F. Petitjean, H. A. Dau, G. I. Webb, and E. Keogh. Generating synthetic time series to augment sparse datasets. In *2017 IEEE international conference on data mining (ICDM)*, pages 865–870. IEEE, 2017.

[12] J. H. Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[13] R. Godahewa, K. Bandara, G. I. Webb, S. Smyl, and C. Bergmeir. Ensembles of localised models for time series forecasting. *Knowledge-Based Systems*, 233:107518, 2021.

[14] R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[15] T. Januschowski, J. Gasthaus, Y. Wang, D. Salinas, V. Flunkert, M. Bohlke-Schneider, and L. Callot. Criteria for classifying forecasting methods. *International Journal of Forecasting*, 36(1):167–177, 2020.

[16] K. B. Kahn. How to measure the impact of a forecast error on an enterprise? *The Journal of Business Forecasting*, 22(1):21, 2003.

[17] Y. Kang, R. J. Hyndman, and F. Li. Gratis: Generating time series with diverse and controllable characteristics. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 13(4):354–376, 2020.

[18] H. R. Kunsch. The jackknife and the bootstrap for general stationary observations. *The annals of Statistics*, pages 1217–1241, 1989.

[19] S. N. Lahiri. *Resampling methods for dependent data*. Springer Science & Business Media, 2013.

[20] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

[21] S. Makridakis and M. Hibon. The m3-competition: results, conclusions and implications. *International journal of forecasting*, 16(4):451–476, 2000.

[22] S. Makridakis, A. Andersen, R. Carbone, R. Fildes, M. Hibon, R. Lewandowski, J. Newton, E. Parzen, and R. Winkler. The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of forecasting*, 1(2):111–153, 1982.

[23] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The m4 competition: Results, findings, conclusion and way forward. *International Journal of forecasting*, 34(4):802–808, 2018.

[24] T.-S. Nguyen, S. Stueker, J. Niehues, and A. Waibel. Improving sequence-to-sequence speech recognition training with on-the-fly data augmentation. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7689–7693. IEEE, 2020.

[25] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. *arXiv preprint arXiv:1905.10437*, 2019.

[26] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*, 2019.

[27] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International journal of forecasting*, 36(3):1181–1191, 2020.

[28] M. Santos, A. de Carvalho, and C. Soares. tsmorph: generation of semi-synthetic time series to understand algorithm performance. *arXiv preprint arXiv:2312.01344*, 2023.

[29] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[30] S. Siami-Namini, N. Tavakoli, and A. S. Namin. A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, pages 1394–1401. IEEE, 2018.

[31] S. Smyl. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1):75–85, 2020.

[32] Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*, 2020.

[33] H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *The eleventh international conference on learning representations*, 2022.

[34] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.