Attilio Sbrana

N-BEATS-RNN: deep learning for time series forecasting

Sorocaba, SP

February 2nd, 2021

Attilio Sbrana

N-BEATS-RNN: deep learning for time series forecasting

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC-So) da Universidade Federal de São Carlos como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação. Linha de pesquisa: Inteligência Computacional.

Universidade Federal de São Carlos – UFSCar

Centro de Ciências em Gestão e Tecnologia – CCGT

Programa de Pós-Graduação em Ciência da Computação – PPGCC-So

Supervisor: Prof. Dr. Murilo Coelho Naldi

Co-supervisor: Prof. Dr. André Luis Debiaso Rossi

Sorocaba, SP

February 2nd, 2021

Sbrana, Attilio

N-BEATS-RNN: deep learning for time series forecasting / Attilio Sbrana -- 2021.
47f.

Dissertação (Mestrado) - Universidade Federal de São Carlos, campus Sorocaba, Sorocaba
Orientador (a): Murilo Coelho Naldi
Banca Examinadora: Gustavo Enrique de Almeida Prado Alves Batista, Tiago Agostinho de Almeida
Bibliografia

1. Previsão de séries temporais. 2. Aprendizado de máquina. 3. Aprendizado profundo. I. Sbrana, Attilio. II. Título.

Ficha catalográfica desenvolvida pela Secretaria Geral de Informática (SIn)

DADOS FORNECIDOS PELO AUTOR

Bibliotecário responsável: Maria Aparecida de Lourdes Mariano - CRB/8 6979

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências em Gestão e Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato Attilio Sbrana, realizada em 27/01/2021.

Comissão Julgadora:

Prof. Dr. Murilo Coelho Naldi (UFSCar)

Prof. Dr. Gustavo Enrique de Almeida Prado Alves Batista (UNSW)

Prof. Dr. Tiago Agostinho de Almeida (UFSCar)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.
O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

To my parents,

who will never read this,

but supported me to pursue yet another masters degree.

To all my family members and close friends,

who will not read this either.

Acknowledgements

I would like to express my deep gratitude to Prof. Dr. Murilo Coelho Naldi, my research supervisor, and Prof. Dr. André Luis Debiaso Rossi, my co-supervisor, for their patient guidance, enthusiastic encouragement and useful critiques of this research.

I would also like to thank Prof. Dr. Tiago Agostinho Almeida, Prof. Dr. Katti Facelli, and Prof. Dr. Sahudy Montenegro González, for their support during my pursuit for this masters degree.

My special thanks are extended to scholarship programs Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brazil (CAPES – Finance Code 001) and Fundação de Amparo a Pesquisa do Estado de São Paulo(FAPESP - Grant2019/09817-6 and 2013/07375-0 - CEPID-CeMEAI) for their generosity in funding of my mastering degree.

"People respond well to those who are sure of what they want."
(Anna Wintour)

Resumo

Este trabalho apresenta o N-BEATS-RNN, uma versão estendida de um conjunto de redes neurais de aprendizagem profunda para previsão de séries temporais, N-BEATS. Para tal, aplica-se uma busca de arquitetura neural estado-da-arte, baseada em uma pesquisa de compartilhamento de peso rápida e eficiente, para solucionar uma arquitetura de rede neural recorrente ideal a ser adicionada ao N-BEATS. Avalia-se a arquitetura N-BEATS-RNN proposta no conjunto de dados amplamente conhecido da competição M4, que contém 100.000 séries temporais de uma variedade de fontes. O N-BEATS-RNN alcança resultados comparáveis ao N-BEATS e ao vencedor da competição M4, empregando apenas 108 modelos, em comparação com os 2.160 modelos originais empregados pelo N-BEATS. Assim, a maior contribuição do N-BEATS-RNN está na redução do tempo de treinamento, que é da ordem de 9 vezes em comparação com os conjuntos originais do N-BEATS.

Palavras-chaves: previsão de séries temporais, competição M4, aprendizado profundo, pesquisa de arquitetura neural, divisão de peso

Abstract

This work presents N-BEATS-RNN, an extended version of an ensemble of deep learning networks for time series forecasting, N-BEATS. We apply a state-of-the-art Neural Architecture Search, based on a fast and efficient weight-sharing search, to solve for an ideal Recurrent Neural Network architecture to be added to N-BEATS. We evaluated the proposed N-BEATS-RNN architecture in the widely-known M4 competition dataset, which contains 100,000 time series from a variety of sources. N-BEATS-RNN achieves comparable results to N-BEATS and the M4 competition winner while employing solely 108 models, as compared to the original 2,160 models employed by N-BEATS, when composing its final ensemble of forecasts. Thus, N-BEATS-RNN's biggest contribution is in its training time reduction, which is in the order of 9 times compared with the original ensembles in N-BEATS.

Key-words: Time series forecasting, M4 competition, deep learning, neural architecture search, weight sharing

List of Figures

List of Tables

List of abbreviations and acronyms

ARIMA	Autoregressive Integrated Moving Average
BO	Bayesian Optimization
CNN	Convolutional Neural Network
DAG	Directed Acyclic Graph
DARTS	Differentiable Architecture Search
DL	Deep Learning
ENAS	Efficient Neural Architecture Search
ES	Exponential Smoothing
ES-RNN	Exponential Smoothing Recurrent Neural Network
ETS	Error, Trend, Seasonality Decomposition
LSTM	Long Short-Term Memory
MASE	Mean Absolute Scaled Error
ML	Machine Learning
MLP	Multi-layered Perceptron
NAO	Neural Architecture Optimization
NAS	Neural Architecture Search
NLP	Natural Language Processing
OWA	Overall Weighted Average
RL NAS	Reinforcement Learning Neural Architecture Search
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RS NAS	Random Search Neural Architecture Search
sMAPE	Symmetric Mean Absolute Percentage Error

List of symbols

H	Horizon, in Days
T	History Length, in Days
m	Seasonal Period
y	Observable Values in a given Time Series
$\hat{y}$	Forecast Values in a given Time Series
N	Number Of Nodes in a Dag
θ	Weight Matrix of a Neural Network

Contents

1 Introduction

Forecasting is the process of modeling a data generation process by observing a sequence of its past states, and predicting the next few. Forecasting techniques are commonly applied to a range of fields in which estimates of future conditions can be advantageous, such as prediction of weather and natural phenomena, control engineering, energy management, stock market, human activity, among others (AFOLABI et al., 2017).

There are several challenges associated with modeling data series, and forecasting techniques are often confronted in open forecasting competitions. Forecasting competitions have become popular for their emphasis on large-scare empirical validations of different forecasting methods and for allowing for a fairer comparison of state-of-the-art methods and newly proposed solutions (MAKRIDAKIS; SPILIOTIS; ASSIMAKOPOULOS, 2019b). Through such forecasting competitions, methods such as Exponential Smoothing (ES) (GARDNER, 2006) and the Theta method (ASSIMAKOPOULOS; NIKOLOPOULOS, 2000) rose to prominence as state-of-the-art models in the forecasting world. Further, the competitions' datasets are designed to represent reality as closely as possible, and often become important testbeds that enable direct benchmarking of new studies against past studies (SPILIOTIS et al., 2019). These datasets also allow for easier reproducibility, since the entire code of the submissions is usually made available by the competitions' organizers.

Historically, time series forecasting has been dominated by statistical approaches where model parameters are estimated per individual time series. This has been evidenced in the forecasting competitions organized by Prof. Dr. Spyros Makridakis, the Makridakis competitions, or, abbreviated, the M-competitions. In the three initial M-competitions, all Machine Learning (ML)-based algorithms significantly underperformed simpler statistical local models (MAKRIDAKIS; SPILIOTIS; ASSIMAKOPOULOS, 2019b). In fact, for over half a century, in spite of the technological and methodological advances, the academic world observed limited progress in our ability to forecast the future accurately (GILLILAND, 2019).

The M4 competition, which was the fourth M-competition and which was held in 2018, marked a turning point, as for the first time a ML method with radically different properties emerged as the winner. Based on the mix of a statistical model and a Recurrent Neural Network (RNN) model, the winning submission of Smyl (2019) employs a single global model over an entire collection of time series. Not only the model is characterized as ML, but also implements deep neural networks, that is, deep learning (DL) techniques that have established themselves as top-performing methods in areas such as computer

vision and natural language processing (HE; ZHAO; CHU, 2019).

It was not long until Oreshkin et al. (2019) proposed N-BEATS, a "pure" DL method without *a priori* hypotheses of time series' theoretical properties. Although it was submitted several months after the competition had ended, as a post-fact submission, N-BEATS was shown to considerably outperform statistical models and their combinations on the datasets of the M4 competition. N-BEATS achieved a performance improvement of 11% over the competition's Official Benchmark, that is, the official baseline forecast produced by a combination of traditional statistical techniques. This figure compared favorably against the winning entry, which achieved an improvement of 8.6% over the Official Benchmark, and the best statistical model, which achieved a 4.1% improvement. To attain such impressive results, however, Oreshkin et al. (2019) applied significant computational resources, with a cost of nearly 90 GPU-days just for the training phase of an ensemble of 2,160 models in total.

While the work of Oreshkin et al. (2019) achieved a remarkable accuracy improvement from the competition winner, its DL architectural solution was largely independent from the work of Smyl (2019), and did not employ RNNs to process the residuals of the decomposed signal. As RNNs have been successful in applications such as energy forecasting (WERON, 2014), where the series are large and multivariate, we would expect RNNs to add value to the proposal in Oreshkin et al. (2019) by further capturing non-linear trends and cross-learning across the multivariate residual signals produced by N-BEATS.

In this work, inspired by the successful use of RNNs in Smyl (2019), we aspire to take the work of Oreshkin et al. (2019) a step further, exploring its DL architectural solution by adding RNNs to its architecture while seeking to optimize it through state-of-the-art Neural Architecture Search (NAS) methods. Therefore, we propose an extended version of the model in Oreshkin et al. (2019), named herein N-BEATS-RNN, and compare its performance on the same set of univariate time series represented by the M4 competition's dataset.

1.1 Purpose and Problem Definition

Deep Learning-based solutions have been recently found to be highly successful when applied to the univariate time series forecasting problem, as illustrated by the works of Smyl (2019) and Oreshkin et al. (2019), which both significantly outperformed traditional statistics-based models in the large and varied datasets of the M4 competition.

While Oreshkin et al. (2019) applied a DL-based signal decomposition method, Smyl (2019) applied an RNN-based neural network to the residuals of a statistical signal decomposition method. Therefore, an opportunity presents itself to combine these two DL-based methods, by applying the decomposition of Oreshkin et al. (2019) to the RNN-based

residual processing of Smyl (2019), in a structure that we named herein N-BEATS-RNN.

Further, recent advancements in NAS for RNNs have allowed for the efficient discovery of superior RNN architectures that resulted in state-of-the-art performance in Natural Language Processing (NLP) benchmarks (WISTUBA; RAWAT; PEDAPATI, 2019; HE; ZHAO; CHU, 2019; ELSKEN; METZEN; HUTTER, 2019). As such, these methods present an opportunity to apply a cost-effective search for better RNN architectures to integrate the residual processing phase of the proposed forecasting model.

Therefore, this research proposes a new "pure" DL method which extends the model presented in Oreshkin et al. (2019), by solving for an ideal Recurrent Neural Network architecture to be incremented into the model for a residual processing of the signal decomposition. Thus, we seek to extract the full forecasting potential of the N-BEATS model while offering insights into the influence that the addition of RNNs and NAS can have in its results.

1.2 Goals and Limitations

As demonstrated earlier and by its name, this work has the explicit objective of improving the state-of-the-art model of Oreshkin et al. (2019) by implementing a newly proposed RNN structure to it. Such improvement can be presented both in terms of **overall accuracy** performance improvement or **overall time reduction** to reach the same performance.

The forecasting performance of our proposed model will be compared not only with the performance of the original N-BEATS, but also with the performance of some of the most important entries in the M4 competition. These include, for example, the winning submission, the best "pure" statistical method, the best ML-statistical combination method, the competition's Official Benchmark, and so forth. These methods will be better described in the sections to come, but for the purposes of this chapter, we will refer to them, collectively, as the M4 Entry Benchmarks.

It is important to note that this research does not possess the ample computational resources of the private research group of Element-AI that enabled the impactful work in Oreshkin et al. (2019). While the training phase of N-BEATS required nearly 90 GPU-days distributed in several GPU-enabled servers, a likely larger undisclosed amount of GPU-days were required for NAS and hyper-parameter tuning of N-BEATS, according to the authors.

The present work, however, will bear the limitations of training several N-BEATS-RNN models sequentially, in a single computer, but with a comparably powerful GPU for a GPU-days metric comparison. Therefore, this research's efforts will be partially driven by its computational resource limitations.

1.3 Hypotheses

Given these stated goals and limitations, we state the following two hypotheses:

Hypothesis 1. *A pure-DL model ensemble can be constructed* ***to outperform*** *the forecasting performance of the original N-BEATS ensemble and the M4 Entry Benchmarks on the set of univariate time-series problems presented by the M4 competition.*

Hypothesis 2. *A pure-DL model ensemble can be constructed* ***to be more computationally cost-effective*** *than the original N-BEATS ensemble on the set of univariate time-series problems presented by the M4 competition, without a significant loss of predictive performance.*

1.4 General Approach

In order to test these two hypotheses, this research's experimentation phase is done in two parts:

1. **A RNN NAS phase:** in which a NAS is conducted to find the ideal RNN architecture for the N-BEATS-RNN model; and
2. **An N-BEATS-RNN ensembling phase:** in which several models are trained to compose a single final forecast for the univariate problems presented by the M4 competition.

Finally, the final output of the experimentation phase will provide the two metrics that will enable us to evaluate the two hypotheses tested herein: (i) the overall forecasting performance of the N-BEATS-RNN ensemble in the M4 problems and (ii) the overall computational cost for its training.

The performance evaluation criteria will be the same of those used in ranking criteria of the M4 Competition, which are thoroughly explained in Chapter 3. The computational cost will be measured by GPU-days.

1.5 Success Criteria

This research will be measured in its success by:

- Partially or completely validating **Hypothesis 1**, that is, by outperforming some or all of the M4 Entry Benchmarks; and/or

- Validating **Hypothesis 2**, that is, by proposing a solution that is comparably cost-efficient when compared with both the performance and GPU-days cost of the original N-BEATS ensemble, without a significant loss of predictive performance.

1.6 Structure

This work is devised as follows:

- In Chapter 2 - we cover the relevant literature surrounding both the application of deep learning for time series forecasting, as well as Neural Architecture Search (NAS) for RNNs;
- In Chapter 3 - we describe the methods necessary to implement an RNN for residual processing into N-BEATS and propose a NAS strategy for such RNN;
- In Chapter 4 - we describe the aforementioned two-phase experiment;
- In Chapter 5 - we describe the results of the experiments.
- In Chapter 6 - we present our conclusions based on the results and analysis of the experiment, and state the contributions and possible future work regarding this study.

2 Literature Review & Background

Two main distinct areas of research are of interest to this work. The first concerns the emergence of pure deep learning methods in the context of time series forecasting. The second concerns the state-of-the-art in deep learning neural architecture search. Both topics are separately accounted for in the subsections below.

2.1 Deep Learning as State-of-the-Art in Forecasting

The history of forecasting research is surrounded by controversies, and much of such accounts have been thoroughly documented in Hyndman (2019). Forecasting competitions emerged as a remedy to the historically heated dispute among academics on which model more properly captured the data generation process in time series. Ever since the first open forecasting competition, the Makridakis Competition (M1), in 1979, forecasters have had the opportunity to empirically demonstrate their aptitude to the forecasting task, regardless of the theoretical properties of their methods, and in a fair and objective setting. These competitions have since played an important role in advancing the academic knowledge and practice of time series analysis and forecasting (MAKRIDAKIS; SPILIOTIS; ASSIMAKOPOULOS, 2018).

The most recent competition, the M4 Competition, ended in May, 2018, and entailed predicting 100,000 time series from a range of sources (MAKRIDAKIS; SPILIOTIS; ASSIMAKOPOULOS, 2019b). The competition itself is a good reference for the historically significant methods, as it chose to include them as performance benchmarks. The organizers justified this decision given that these benchmarks have now well-known properties and would serve as a great baseline for comparison against more complex methods (MAKRIDAKIS; SPILIOTIS; ASSIMAKOPOULOS, 2019b). These baselines include traditional methods such as Error, Trend, Seasonality (ETS) (or Exponential Smoothing (ES)) and Auto-Regressive Integrated Moving Average (ARIMA), naïve methods, as well as ML methods such as the Multi-Layer Perceptron (MLP) and RNNs. These blaselines are outlined in Table 1.

In the M4 competition, a total of 61 methods were ranked, whereby 49 belonged to submissions, 10 were benchmarks, and 2 were standard methods for comparison (an ARIMA standard and an ES standard). The submissions were officially categorized as “purely” statistical, “purely” ML, a combination of statistical and ML methods, and the winning submission was tagged as a “hybrid" statistical-ML method, which is a combination of statistical and ML methods.

Table 1 – The benchmarks and standards for comparison of the M4 Competition (MAKRIDAKIS; SPILIOTIS; ASSIMAKOPOULOS, 2019b).

Benchmark type	Methods	Description
Statistical benchmarks	Naïve 1	A random walk model, assuming that future values will be the same as that of the last known observation.
	Naïve S	Forecasts are equal to the last known observation of the same period.
	Naïve 2	Like Naïve 1 but the data are seasonally adjusted, if needed, by applying a classical multiplicative decomposition. A 90% autocorrelation test is performed to decide whether the data are seasonal.
	SES	Exponentially smoothing the data and extrapolating assuming no trend. Seasonal adjustments are considered as per Naïve 2.
	Holt	Exponentially smoothing the data and extrapolating assuming a linear trend. Seasonal adjustments are considered as per Naïve 2.
	Damped	Exponentially smoothing the data and extrapolating assuming a damped trend. Seasonal adjustments are considered as per Naïve 2.
	Theta	As applied to the M3 Competition using two Theta lines, $\vartheta_1 = 0$ and $\vartheta_2 = 2$, with the first one being extrapolated using linear regression and the second one using SES. The forecasts are then combined using equal weights. Seasonal adjustments are considered as per Naïve 2.
	Comb	The simple arithmetic average of SES, Holt and Damped exponential smoothing (used as the single benchmark for evaluating all other methods).
ML benchmarks	MLP	A perceptron of a very basic architecture and parameterization. Some preprocessing like detrending and deseasonalization is applied beforehand to facilitate extrapolation.
	RNN	A recurrent network of a very basic architecture and parameterization. Some preprocessing like detrending and deseasonalization is applied beforehand to facilitate extrapolation.
Standards	ETS	Automatically provides the best exponential smoothing model, indicated through information criteria.
	ARIMA	An automatic selection of possible ARIMA models is performed and the best one is chosen using appropriate selection criteria.

Table 2 illustrates some of the most relevant entries and benchmarks in the M4 Competition. The competition's Official Benchmark, *Comb*, is a combination of basic statistical methods, and is used as the true baseline for the comparison of different methods. The Theta method (ASSIMAKOPOULOS; NIKOLOPOULOS, 2000) was the winner of the M3 Competition, held in 1998, and its performance was equivalent to that of the Official Benchmark. The relatively strong performance of more complex and ML-driven models led organizers to finally discard a decades old belief that complex models are no better than simpler ones (MAKRIDAKIS; PETROPOULOS, 2019).

Table 2 – Performance of selected entries and benchmarks in the M4 Competition.

Method	Entry Type	sMAPE	OWA	OWA % improvement over Official Benchmark	Rank
M4 winner, DL-statiscal	Entry	11.374	0.821	+8.6	1
Best ML-statistical	Entry	11.720	0.838	+6.7	2
Best pure statistical	Entry	11.986	0.861	+4.1	8
Theta	*Benchmark*	*12.309*	*0.897*	*+0.1*	18
Comb	*Official Benchmark*	*12.555*	*0.898*	*0.0*	19
ARIMA	*Benchmark*	*12.669*	*0.903*	*-0.6*	20
ETS	*Benchmark*	*12.726*	*0.908*	*-1.1*	23
Best pure ML	Entry	12.894	0.915	-1.9	25
Naïve (repeats last entry)	*Benchmark*	*14.208*	*1.058*	*-17.8*	41

While the competition provided supporting evidence to the use of more complex

ML in forecasting, the most striking results in favor of ML-based models, however, were presented by the winning entry, which employed a DL-statistical method. Smyl (2019) presented an application of a common Holt-Winters statistical pre-processing method of level and seasonal decomposition mixed with stacks of RNNs. Yet, the method introduced one significant innovation: the author applied an ES class where the seasonal and smoothing components that normalize the series are fitted in the context of the RNN's learning through gradient descent and backpropagation. Since the ES class implies *a priori* knowledge of time series modeling and the method entailed applying ML to learn a statistical model's parameters, it was thus labeled as a hybrid method. Impressively, however, the competition winner performed 8.6% better than the Official Benchmark, more than twice the performance of the best "pure" statistical method, and significantly ahead of the second entry, which posted a 6.7% improvement.

While the competition winner succeeded by way of a hybrid of statistical and DL method, all pure ML submissions failed to outperform the M4 competition's baseline statistical benchmark. The best pure ML method, a Convolutional Neural Network (CNN) method, performed -1.9% worse than the Official Benchmark, and underperformed most classic statistical benchmarks. The competition's organizers highly regarded this as a validation of their notion that pure ML methods, standalone, are not the way forward in improving forecasting accuracy (MAKRIDAKIS; SPILIOTIS; ASSIMAKOPOULOS, 2019a).

As a direct response to the organizers' statements, N-BEATS was proposed in Oreshkin et al. (2019) several months after the end of the M4 competition in order to challenge the notion that a pure ML/DL method could not outperform statistical models and their combinations with ML on the univariate forecasting tasks in the M4 dataset. The model proposed in Oreshkin et al. (2019) indicated state-of-the-art performance in the M4 competition's benchmarks through a proposed deep neural network architecture based on backward and forward residual links and stacks of fully-connected layers. The method uses no time series specific components and, instead, provides a hierarchical decomposition of the input signal through the intermediate outputs of the architecture's blocks of layers. The work is the first to empirically provide a proof of concept for the use of pure DL in time series forecasting by outperforming well-established statistical approaches and hybrid ML approaches. On the M4 competition benchmarks, N-BEATS outperform the statistical benchmark by 11%, the best statistical entry by 7%, and the competition winner by 3%.

2.2 Neural Architecture Search

The NAS surveys done in Wistuba, Rawat e Pedapati (2019), He, Zhao e Chu (2019), and Elsken, Metzen e Hutter (2019) well document recent research efforts to

automate the search process of neural network architectures and, as a result, expand on the current success of deep learning. While, historically, the search for neural network architectures had already been explored through Evolutionary Algorithms and Bayesian Optimization (ELSKEN; METZEN; HUTTER, 2019; WISTUBA; RAWAT; PEDAPATI, 2019), NAS became a mainstream research topic after the seminal works of Zoph e Le (2017) and Baker et al. (2017) framed the search as a Reinforcement Learning (RL) problem and identified newly automatically designed architectures that outperformed handcrafted architectures in image classification and language model benchmarks, giving birth to RL NAS.

Whilst early RL NAS methods demonstrated impressive results, those proved to be too computationally expensive for any practical application. As an example, Zoph e Le (2017) use 800 GPUs for 3-4 weeks to reach their final solution, and, if replicated in a single GPU, the experiment would require over 50 years to completion. Based on these early methods, NAS was limited to researchers with access to extraordinary computational power. Naturally, a number of works followed in quick response seeking to democratize NAS by promoting methods that could reduce the search's computational costs while maintaining or improving performance (ELSKEN; METZEN; HUTTER, 2019; HE; ZHAO; CHU, 2019).

The perhaps most groundbreaking research emerged in the work of Pham et al. (2018) through a method named therein Efficient Neural Architecture Search (ENAS). For the authors, much of the computational bottleneck of early NAS research is attributable to the training of each architecture attempt from random initialization to convergence, only to measure its accuracy whilst throwing away all the trained weights. The authors, then, propose a new search structure where all architecture attempts seek to reuse the weights of the search's previous attempts, aiming to eschew training each architecture from scratch. The work is inspired in earlier works on transfer learning (ZOPH et al., 2016) and multitask learning (LUONG et al., 2016) that indicated that parameters learned for a particular task can be successfully reapplied to other models and tasks. Pham et al. (2018) empirically demonstrate that sharing parameters among architecture leads to radical efficiency, achieving state-of-the-art results in the CIFAR-10 and Penn Treebank benchmarks in less than 16 hours while using a single GPU. The RL-based weight-sharing method, ENAS, obtained a GPU-hour reduction in the order of more than 1000x when compared to many of the early works in NAS.

Other compute time-efficient models based on ENAS' weight-sharing training method quickly succeeded while trying to propose an alternative to a RL-based search. These include search methods based on:

- Gradient-based optimization, such as Differentiable Architecture Search (DARTS) (LIU; SIMONYAN; YANG, 2019);

- Random Search, such as Random Search NAS (RS NAS) (LI; TALWALKAR, 2019; LIU et al., 2017); and
- Surrogate models, such as Neural Architecture Optimization (NAO) (LUO et al., 2018).

Table 3 illustrates some of the main search methods applied during RNN NAS. All these methods were tested by Li e Talwalkar (2019) and Luo et al. (2018) on single GPU resources, for comparison, and applied to the PTB dataset, which is a widely known language modeling benchmark in the Natural Language Processing (NLP) field. These methods where, thus compared in terms of Test Perplexity and GPU-days. Perplexity is an entropy-based measure of information commonly used in NLP to indicate how well the probability modeling of a language model predicts the distribution of entire sentences or texts (BROWN et al., 1992), whereby the lower the value, the better the model predicts a sample from such distribution. Equation 2.1 defines Perplexity as the inverse probability of the test set, normalized by the number of words.

$$PP(W) = 2^{-\frac{1}{N}\sum_{i=1}^{N} logp(s_i)} \tag{2.1}$$

It is necessary to note that the GPU used in Li e Talwalkar (2019), the NVIDIA Tesla P100, has 30% fewer CUDA cores (3584 cores) when compared to the GPU used by Luo et al. (2018), the NVIDIA Tesla V100 (5120 cores). Therefore, when comparing these methods, the GPU-days metric is not directly comparable.

Table 3 – Comparison of different RNN search methods on the PTB benchmark in terms of Test Perplexity and GPU-days. Lower values are better.

Method Name	Method Source	Reproduction Source	GPU Used	Weight Sharing	Search Method	Test Perplexity	Cost (GPU-days)
Vanilla LSTM	Zaremba et al. (2014)	Zaremba et al. (2014)	K20	No	None	78.4	N/A
RL NAS	Zoph e Le (2017)	Li e Talwalkar (2019)	P100	No	RL	64.0	10,000
ENAS	Pham et al. (2018)	Li e Talwalkar (2019)	P100	Yes	RL	56.4	0.50
DARTS	Liu, Simonyan e Yang (2019)	Li e Talwalkar (2019)	P100	Yes	Gradient	55.7	2.00
RS NAS	Li e Talwalkar (2019)	Li e Talwalkar (2019)	P100	Yes	Random	**55.5**	1.25
RS NAS	Luo et al. (2018)	Luo et al. (2018)	V100	Yes	Random	58.8	0.40
NAO	Luo et al. (2018)	Luo et al. (2018)	V100	Yes	Surrogate	56.6	0.40

As the table illustrates, the RNN NAS driven by Weight-Sharing promoted not only a significant Test Perplexity improvements vs. the initial works of Zoph e Le (2017), but also a drastic improvement in GPU-days from 10,000 days to at most 2 days.

Importantly, the table indicates that the two Random Search-based methods demonstrated fairly comparable results with other search methods, and, in fact, the best performance was achieved by a Random Search-based method. The independent works of Yu et al. (2019), Li e Talwalkar (2019), and Adam e Lorraine (2019) reportedly found that, under a weight-sharing search, the best empirical strategy for finding an optimal RNN is through a simple Random Search. The authors in Adam e Lorraine (2019) more

carefully studied these findings, postulating that, most likely given the weight-recycling nature of the weight-sharing training, none of the controllers proposed thus far, be it RL or gradient descent-based, manage to learn meaningful links between a given architecture and its performance during training.

3 Methods

In the methodology here proposed, a few principles must be observed. Firstly, we maintain N-BEATS' original objective of employing neither time series specific *a priori* knowledge into the model, nor input normalization. Secondly, we also seek to maintain N-BEATS' objective of providing a single architecture solution for all different time series modeling tasks. Further details of the methodology are described in the sections below.

3.1 N-BEATS-RNN

The model here proposed, N-BEATS-RNN, relies on a combination of three main methods: (i) decomposition, (ii) residual processing, and (iii) ensembling.

The first method relies entirely on the neural architecture proposed in N-BEATS (ORESHKIN et al., 2019), which provides a deep learning-based method for decomposing the signal. The second method is inspired on the winning entry of the M4 competition, ES-RNN (SMYL, 2019), and entails processing the residuals of the decomposition through an RNN. Lastly, the ensembling method used is, once again, the same used by Oreshkin et al. (2019) and used by most entries in the M4 competition. These methods are described next.

3.1.1 Decomposition through N-BEATS

N-BEATS heavily relies on the well-established principle that passing residual modifications of the input signal through stacks of layers provides clear advantages when training deep neural network architectures (He et al., 2016). In N-BEATS, the authors introduce the notion of backward forecasts, that is, "backcasts", where the model attempts to reconstruct the time series that it received as an input. Thus, the authors propose a model where a residual branch of backcasts are subtracted from the output of the previous fully-connected block (or the input signal in the case of the first block) and then passed to the next block. A second branch of block outputs, the forecasts, is stored and summed up to form the final predictions.

Figure 1 demonstrates N-BEATS' block residual flows expressed by the blue arrows. Each block is a 4-layer MLP that outputs its best reconstruction of the series as a backcast and its best prediction of the future as a forecasts. Intuitively, N-BEATS is a model that combines several simple MLP neural networks and applies them sequentially over the residuals of the previous one. The method is akin to the traditional method of ETS, in which the signal is decomposed in terms of error, trend and seasonality. Hence, the authors

also label N-BEATS as a decomposition method. However, the advancement in N-BEATS is that each MLP block is not bound to finding the preconceived theoretical concepts of trend and a seasonality, and instead the neural network is free to model any pattern recognition as it may see fit. Further, differently from ETS, the decomposition efforts of the MLP blocks are not independent of each other, given the dynamics of backpropagation, which, during training, introduce iterative interactions in the parameters among the different layers.

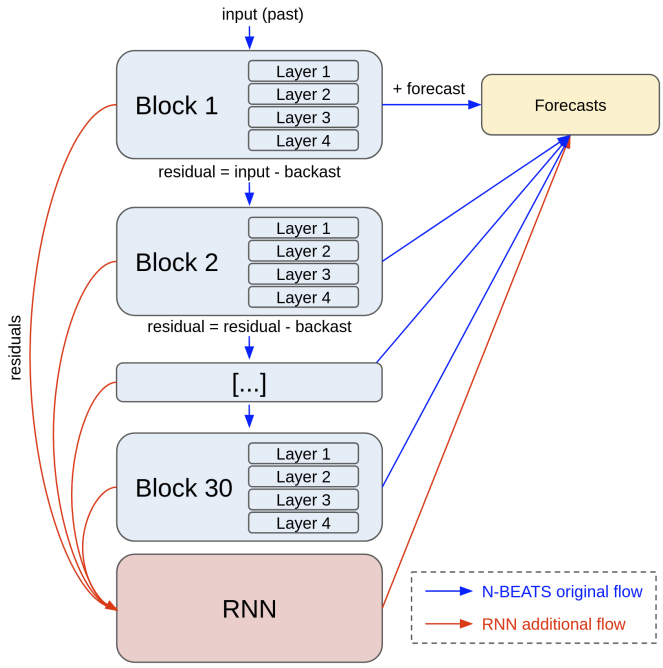

Figure 1 – Architecture of the proposed N-BEATS-RNN. The blue blocks and blue arrows indicate the framework of N-BEATS. The red arrows and the red RNN block indicate the additions made in N-BEATS-RNN. Every block outputs a partial forecast output and a partial reconstruction of the time series, that is, the backcast. All the blue arrows represent these outputs from each block. In the case of the forecasts, the outputs are summed up to form the new forecast, whereby, in the case of the backcast, they are subtracted from the input to produce a residual.

3.1.2 Residual processing

Inspired by ES-RNN, in which Smyl (2019) successfully applied RNNs to the residuals of a ES statistical decomposition of the time series, an opportunity arises to incorporate a RNN to the signal decomposition of the pure-deep learning N-BEATS model. This structure would enable the RNN to capture time patterns in the residuals that were not available through the decomposition. Figure 1 demonstrates the proposed modified architecture as N-BEATS-RNN, with the additional flows being represented by red arrows. Here, the RNN architecture acts as a terminal layer that recurrently modifies a multivariate

input signal derived by the network's intermediate residual outputs, outputting a final residual forecast to be added to the network's prediction.

By adding a new RNN layer for a combined residual processing, we are also introducing new backpropagation dynamics. In the original N-BEATS, blocks are not influenced by the modeling carried by the blocks preceding it. Let's take the most extreme examples, block 30 and block 1. In the original N-BEATS, block 30 only receives a weight update limited to its portion of the output, since its backcast is discarded. At the same time, block 1 receives updates not only through its own forecast, but also from all the other blocks below it, which are also influenced by its backcast. By wrapping all backcasts into a last RNN layer, now, block 30's output also interacts with the outputs of block 1, creating new interdependent backpropagation dynamics.

While, in ES-RNN, Smyl (2019) sought to handcraft the architectures of the model through unique LSTM stack configurations and hyperparameters for each seasonal period, this work takes a step further by employing a RNN design search strategy to avoid any handcrafting. As such, a general architectural solution is automatically sought for the RNN component of the N-BEATS-RNN, for all periods. The automatic search maintains Oreshkin et al. (2019)'s original goal of proposing a pure deep learning solution whose single architectural solution is reusable and generalizes well across time series of different nature.

3.1.3 Ensembling

Finally, all top entries in the M4 competition sought to apply model ensembling (MAKRIDAKIS; SPILIOTIS; ASSIMAKOPOULOS, 2019b), a technique that has been prominently applied in forecasting for centuries (HYNDMAN, 2019). In line with this notion, N-BEATS is no different, and its authors state that ensembling was found to be much more powerful than any other regularization technique.

Therefore, we also apply model ensembling to be comparable. We apply a similar technique to that of Oreshkin et al. (2019), in which several models, with slightly different characteristics, such as lookback horizon and loss function, are combined to form a single forecast composed by the median forecast point of the ensemble. This implementation is covered in details in Chapter 4.

3.2 Designing RNNs through Weight Sharing

The weight sharing NAS strategy first proposed in Pham et al. (2018), and replicated in Li e Talwalkar (2019) and Liu, Simonyan e Yang (2019), is based on a design search where any Directed Acyclic Graph (DAG) architecture attempt is a subgraph of a larger search graph. On Figure 2, the left graph represents the entire search space of a RNN

DAG architecture where the number of nodes $N = 12$. Every node is a local computation denoted by an activation function, and all edges constitute the information flow through a weight matrix multiplication, indicated by θs. The subgraph on the right demonstrates a possible model in the search space, making the left chart the superposition of all the possible architecture attempts.

This design allows for the θ parameters to be shared among the architecture attempts within the search grid, since the grid is never reinitiated, that is, the future architecture attempts will start with a combination of the thetas left by the previous searches. The overall search space is represented by an exponential number of configurations, which, in the case of the here employed $N = 12$, is in the order of 10^{15}.

In Pham et al. (2018), every subgraph and architecture is expressed as a list of integers, which indicate a certain path within the grid and with four types of possible activation functions (tanh, ReLU, identity and sigmoid). This list of integers can be generated by an optimizer, as in Pham et al. (2018) and Liu, Simonyan e Yang (2019), who apply a reinforcement learning and a gradient descent-based controller, respectively, when searching for architectures through weight sharing. However, here, all architecture attempts, represented by these lists, are chosen at random, including its activation functions. This is in line with the work of Li e Talwalkar (2019), whose reproductions of the different NAS methods indicated that all of the optimization-based search methods actually underperformed a simple random search when compared under the same RNN search space.

When the random search generates a new integer list, it decides: (i) which edges should be active and (ii) which computations should be performed at each node in the DAG of the architecture (PHAM et al., 2018). Finally, in the DAG, all leaves, that is, loose ends, are averaged for the processing of the final output.

As this RNN design search, based on weight sharing, has resulted in performance improvements over traditional RNN architectures, such as LSTM, in the natural language processing domain (PHAM et al., 2018; LI; TALWALKAR, 2019; LIU; SIMONYAN; YANG, 2019), this work explores this method in the context of the proposed N-BEATS-RNN architecture, applying NAS directly to the additional RNN component of the structure.

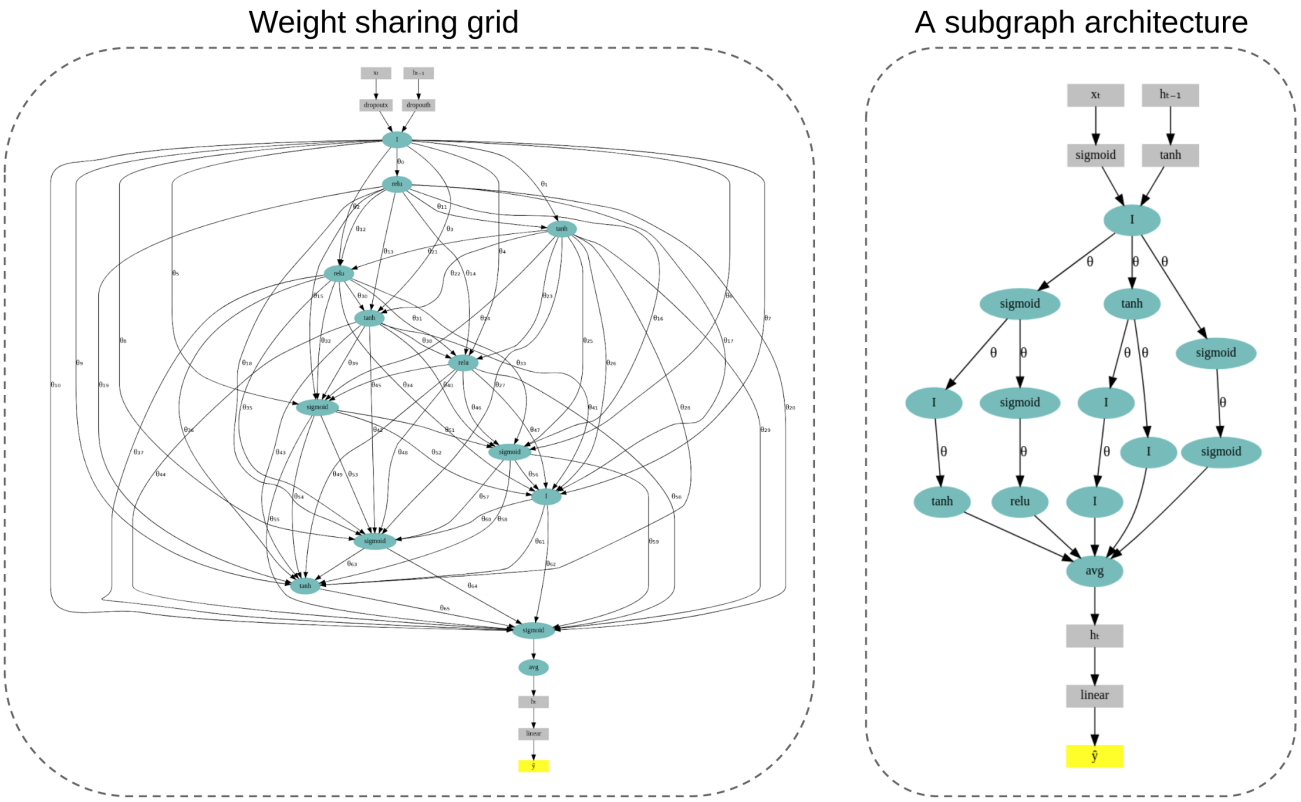

Figure 2 – RNN cell search space via parameter sharing. On the left, the graph represents the whole grid of possible node connections of the DAG that represents the RNN. On the right, a single possible RNN architecture is illustrated as a subgraph.

4 Experiments

In this section, we describe the dataset and the evaluation criteria of the experiment. Then, we delineate the proposed N-BEATS-RNN methodology in two steps:

- Firstly, an RNN architecture had to be sought through the proposed weight-sharing search. Only the architecture with the highest validation performance was selected to be part of the final N-BEATS-RNN model.
- Secondly, the N-BEATS-RNN models would be trained, all from different initialization points, in order to build an ensemble of final forecasts.

4.1 Datasets

Table 1 outlines the composition, across different domains, forecast horizons, and frequencies, of the 100,000 time series in the M4 competition database. These datasets aims at representing real world problems as much as possible (M4-TEAM, 2018a), and the data come mainly from the Economic, Finance, Demographics and Industry areas, while also including data from Tourism, Trade, Labor and Wage, Real Estate, Transportation, Natural Resources and the Environment. Whereas the series represent a wide variety of lengths, as indicated by the summary statistics, the forecast horizons are fixed for each respective seasonal frequency. Further, all series have positive values at all time-steps. The data are publicly available for download in the official repository of the competition (M4-TEAM, 2018b).

4.2 Evaluation Criteria

The M4 competition ranking criteria is the Overall Weighted Average (OWA) (4.3) of two normalized accuracy metrics: the Mean Absolute Scaled Error ($MASE$) (4.2) and symmetric Mean Absolute Percentage Error ($sMAPE$) (4.1) (M4-TEAM, 2018a). The $sMAPE$ metric compares the error of a forecast normalized by the mean between the forecast and the ground truth and it is, thus, a point-based error metric. The $MASE$ metric, instead, normalizes the forecast error with the most obvious seasonal guess, that is, the last available seasonal datapoint, being a seasonality-based error metric.

A combination of $sMAPE$ and $MASE$, named $sMAPEvMASE$ (4.4), is also here introduced by multiplying the resulting $sMAPE$ and $MASE$ losses, as a proxy of the OWA minimization objective. Since OWA is the competition's official ranking metric and

Table 4 – Composition of the M4 dataset: number of time series, sampling frequency, types and length statistics.

	Frequency / Horizon					
Type	Yrly	Qtly	Mthly	Wkly	Daily	Hrly
Forecast Window	6	8	18	13	14	48
Demog.	1,088	1,858	5,728	24	10	0
Finance	6,519	5,305	10,987	164	1,559	0
Industry	3,716	4,637	10,017	6	422	0
Macro	3,903	5,315	10,016	41	127	0
Micro	6,538	6,020	10,975	112	1,476	0
Other	1,236	865	277	12	633	414
Total	23,000	24,000	48,000	359	4,227	414
Min. Leng.	19	24	60	93	107	748
Max. Leng.	841	874	2,812	2,610	9,933	1,008
Mean Leng.	37.3	100.2	234.3	1035	2371.4	901.9
SD Length	24.5	51.1	137.4	707.1	1756.6	127.9

considers both *sMAPE* and *MASE* in its formula, we choose to introduce *sMAPEvMASE* as an alternative loss function that needs to balance both point-based and seasonality-based forecasts.

Given a forecast horizon length H, a series history length T, a seasonal period m, a vector of historical values $[y_1, y_2, ..., y_T]$, a vector of future values $[y_{T+1}, y_{T+2}, ..., y_{T+H}]$, and a vector of predictions $[\hat{y}_{T+1}, \hat{y}_{T+2}, ..., \hat{y}_{T+H}]$, we can derive the accuracy metrics employed herein as follows:

$$sMAPE = \frac{1}{H}\sum_{i=1}^{H} 2\frac{|y_{T+i} - \hat{y}_{T+i}|}{|y_{T+i}| + |\hat{y}_{T+i}|} \tag{4.1}$$

$$MASE = \frac{1}{H}\sum_{i=1}^{H} \frac{|y_{T+i} - \hat{y}_{T+i}|}{\frac{1}{T+H-m}\sum_{j=m+1}^{T+H}|y_j - \hat{y}_{j-m}|} \tag{4.2}$$

$$OWA = \frac{1}{2}\left[\frac{sMAPE}{sMAPE_{naive2}} + \frac{MASE}{MASE_{naive2}}\right] \tag{4.3}$$

$$sMAPEvMASE = sMAPE * MASE \tag{4.4}$$

In *MASE*, its denominator is composed by the mean absolute error of a naïve predictor that uses the actual value from the prior season as the forecast, whereby $\hat{y}_{T+1} = y_{T+1-m}$. In the case of *OWA*, both *sMAPE* and *MASE* metrics are normalized by a naïve forecast, named naïve2, which is seasonally-adjusted through a classical multiplicative decomposition. The naïve2 series are static and are provided by the competition, together with the training series, for the purposes of calculating the *OWA* metric (M4-TEAM, 2018a).

Because *OWA* is reliant on the naïve2 forecasts, the *sMAPE*, *MASE*, and *sMAPEvMASE* metrics are instead applied as loss functions during training. *OWA* is then only calculated for the final test forecast evaluation for comparison and ranking of the models.

4.3 RNN Architecture Search

Since our search optimizer is random, we have chosen to apply a search by tournament. As such, we initially search for a large-enough number of 256 random architectures, electing the top 1/8 (32) architectures for retraining and comparison. Finally, we retrain the top 1/8 (4) architectures for a final winning architecture. With such tournament, the winning architecture should be able to consistently outperform the other architectures in three consecutive rounds, while initializing at three different states of the weigh-sharing grid. Therefore, we expected this search to produce a reasonably robust architecture that was unlikely to be a product of initialization chance.

Since the NAS would entail testing hundreds of architectures, we opted for a smaller dataset for training and validation during the search. For that, the Quarterly frequency was chosen, since the forecasting horizon of 8 steps was neither as short as the Yearly horizon of 6 steps, nor as long as the Monthly horizon of 18 steps. Therefore, the Quarterly frequency provided a dataset that was large enough, representing 25% of all the time series, and in the middle ground of the forecasting horizons. We did not expand across different frequencies because of computational resource limitations, since running the search on individual frequencies would require the search to be done multiple times.

Moreover, the largest lookback window of $7H$ was also selected for the search. Because the sampling procedure during training adds paddings of zeros if the sampled window does not include all 7 horizons, the search would seek to solve for a more versatile architecture in terms of lookback windows, since the architecture would be trained in a mix of windows from only $1H$ of data up to $7H$.

The training data was composed of batches of 1,000 random samples of the Quarterly frequency, at random windowed intervals, for training each architecture attempt. The validation set was determined as a separate, fixed, dataset of 48,000 random samples of the Quarterly series, at random intervals, and it was used for validating the performance of the attempts across the rounds and selecting the winners. The true test set of the submission was also tested for comparison purposes, and it comprised all 24,000 forecasts of the Quarterly frequency. However, none of the test information had influence over the search. The training data was evaluated based on one of the three valuation criteria ($sMAPE$, $MAPE$, $sMAPEvMASE$), which were randomized for each architecture during the search, whereby the valuation set was measured by $sMAPEvMASE$ and the test set was measured by OWA.

The 256 random RNN architectures were trained with a validation early stopping criteria of 1,000 batches. The validation was based on a randomly selected validation set and measured by the *sMAPEvMASE* metric. We have chosen 1,000 batches by analysing the typical pattern of the learning curve of the original N-BEATS. The authors of the original model established that training for a total of 15,000 batches was optimal based on cross-validation, and we observed that N-BEATS hardly ever surpassed 1,000 batches without reaching a new trough. Therefore, 1,000 batches was assumed to provide enough room for model convergence.

After the training of the 256 models, the top 32 performing architectures were tested for an early stopping patience of another 4,000 batches. At this stage, we would expect the weight sharing grid to be better calibrated when revisiting the top architectures. Therefore, 4,000 batches should provide enough room for the top models to plateau on training.

Finally, the final 4 architectures would be run for a total of 15,000 batches, just like for the original N-BEATS, except that, in this case, the initialization was not random, but was, instead, the last state of the weight sharing grid. Of those, the best performing RNN architecture was then selected out of the 292 runs in the search.

4.4 Ensemble Training

In Oreshkin et al. (2019), N-BEATS is separately trained for each of the 6 seasonal frequencies (daily, weekly, monthly, etc) available in the M4 dataset. For a forecast horizon of length H, a lookback window comprehends a history of points of lengths $2H$, $3H$, ..., up to $7H$, which are multiples of the forecast horizon. On each seasonal frequency, a model is trained for each of these 6 lookback windows, and for three different loss functions, totalling 18 different models. This process is repeated 10 times, for two slightly different architectures, for a total of 360 models, which constitute a final ensemble. Since each of the seasonal frequencies demands one final ensemble, 2,160 different models are trained in total to produce the final submission of the 100,000 time series.

Similarly to N-BEATS, an independent N-BEATS-RNN ensemble was trained for each of the 6 seasonal frequencies. All models are trained from scratch. For each of those 6 ensembles, an individual N-BEATS-RNN model was trained from scratch for each of the 6 lookback window, and for each of three loss functions (*sMAPE*, *MASE*, and *sMAPEvMASE*), totalling 18 models.

Combining all ensembles, 108 models were trained to compose the final forecasts, which was based on the median forecast for each frequency type. Figure 3 depicts the ensemble training and forecasting logic in a graphical manner.

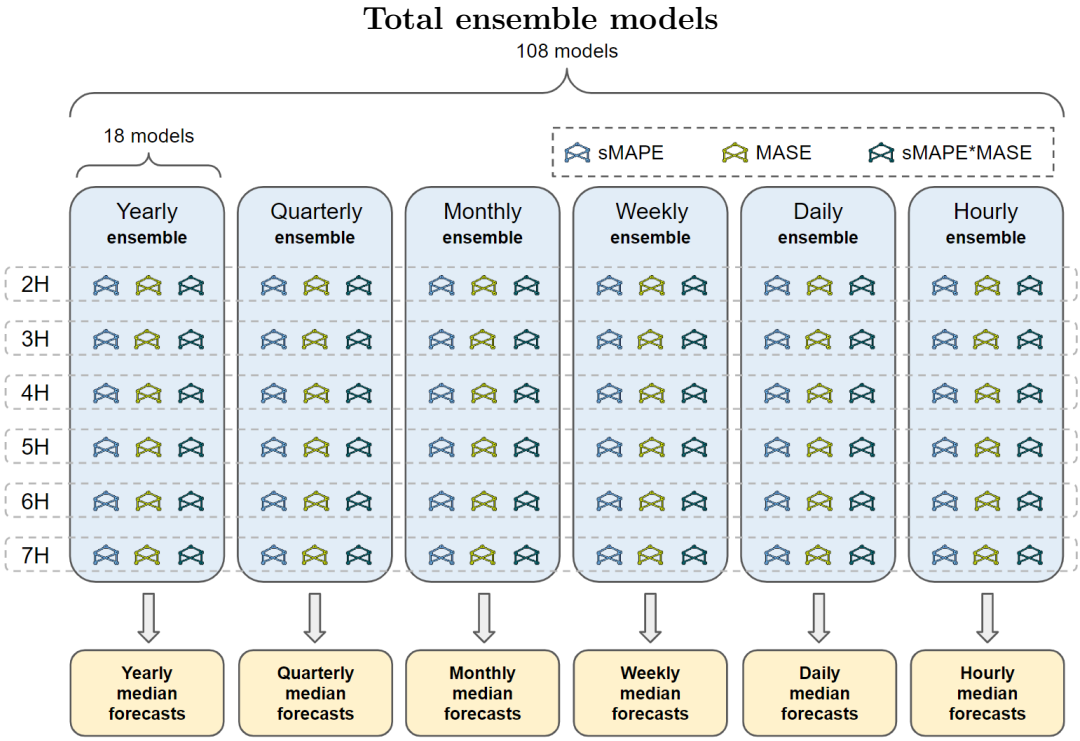

Figure 3 – Each neural net represents a model. In total, 6 ensembles were created, one for each frequency type. Each ensemble contained 18 models, one for each lookback horizon and each loss criteria. Finally, the ensemble forecast is based on the median forecast value of the 18 models. All mdoels are trained from random initialization.

Differently from the original N-BEATS, this process was not repeated 10 times, and each ensemble only contained 18 N-BEATS-RNN models, as opposed to 180.

Since these 18 models are trained each in different combinations of lookback horizon and loss function, they have slightly different training targets, and, thus, promote a diversity of biases when building the ensemble. For example, in Figure 3, in the Yearly ensemble, the top left model is trained while having inputs that are two times the number of the forecast horizon and while minimizing a point-based error metric, *sMAPE*. However, the bottom right model of the Yearly ensemble observes 7 times the forecast horizon while minimizing both *sMAPE* and *MASE*. Both models have different training and objectives, whereby both will contribute to the final Yearly forecast together with the other 18 models.

4.5 Computational Resources

All models of the N-BEATS-RNN architecture search and final ensemble were tested on a single machine, sequentially. The NAS search and the ensemble models were implemented and trained in Pytorch, and the model training was parallelized in two NVIDIA RTX 2080 Super graphics cards. The two cards, combined, amount to 6,144

CUDA cores, which is slightly higher to the 5,120 CUDA cores of the NVIDIA Tesla V100 used in the NAS experiments in Luo et al. (2018), and almost twice the 3,584 CUDA cores of the NVIDIA Tesla P100 used in Li e Talwalkar (2019).

5 Results

Given that our experiments were accomplished in two different phases, one for the architecture search and another for building the ensemble, we describe their respective results in the next two sections.

5.1 RNN Architecture Search

The training of the 292 model runs of the RNN architecture search required about 4 full days in our computational resources. Figures 4 and 5 illustrate the entire evolution of the validation and test losses, measured respectively by *sMAPEvMASE* and *OWA*, across the different RNN architecture attempts, as described in Chapter 4. Note that we are not here displaying the training loss of the entire NAS search for two main reasons: (i) the loss function is changed randomly across the three evaluation criteria and (ii) the training loss refers to randomly selected batch sizes of time series and is not as directly comparable among the architectures. Thus, the training data did not offer as much information as the validation and test losses.

Most of the early attempts of the architecture search can be interpreted as a calibration of the weight sharing search grid, since any architecture attempt seems to have helped improve both test and validation losses for most of the training. Only after circa 1.5 million training steps is that the performance of different architecture attempts started to become more apparent, as both validation and the test losses diverged from a trend of continuous improvement.

The final model elected by the search, based on the best validation loss, achieved a *sMAPE* of 9.706% and *OWA* of 0.840. The median ensemble of all the 292 models from the search led to a 2% improvement on each metric, to a *sMAPE* of 9.514% and *OWA* of 0.824, and it is illustrated in Figure 6. Both the individual and ensemble results were superior to the M4 competition winner's performance in the "Quarterly" seasonal frequency, as the winning submission achieved a sMAPE of 9.679% and a 0.847, but inferior to the results achieved by N-BEATS of 9.213% and 0.800, respectively.

As we expected, the search itself can provide a strong solution to the problem, as it generates several models that individually showcase a strong validation and test performance. However, given these model attempts share weights across training, it turns out that their forecasts are too correlated for a meaningful ensemble improvement. Therefore, in the next session, we were able to generate stronger ensemble results by training new models, which were based on the winning architecture, from random initialization.

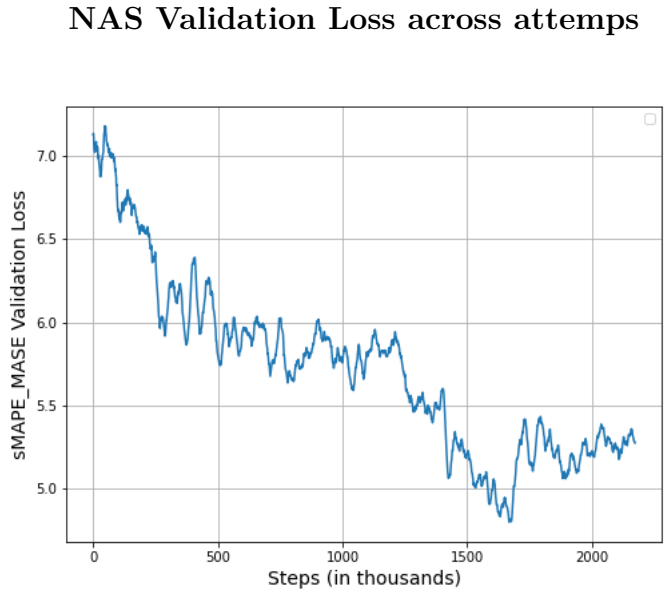

Figure 4 – Smoothed *sMAPEvMASE* Validation Loss across the 292 attempts of the search. Smoothing is done through convolutional-smoothing.

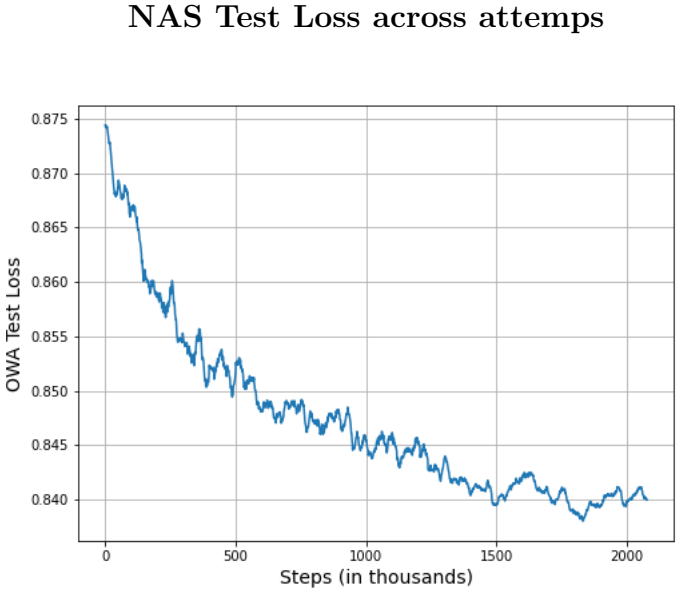

Figure 5 – Smoothed *OWA* Test Loss across the 292 attempts of the search. Smoothing is done through convolutional-smoothing.

5.2 N-BEATS-RNN Ensemble Results

5.2.1 Overall Ensemble Results

Table 5 illustrates the *sMAPE* and Table 6 illustrates the *OWA* of all the main submissions of the competition, the N-BEATS best model, and our model, N-BEATS-RNN. N-BEATS-RNN's results were achieved with significantly fewer models than N-BEATS, with ensembles of 18 models as opposed to 360 models, respectively, for each seasonal

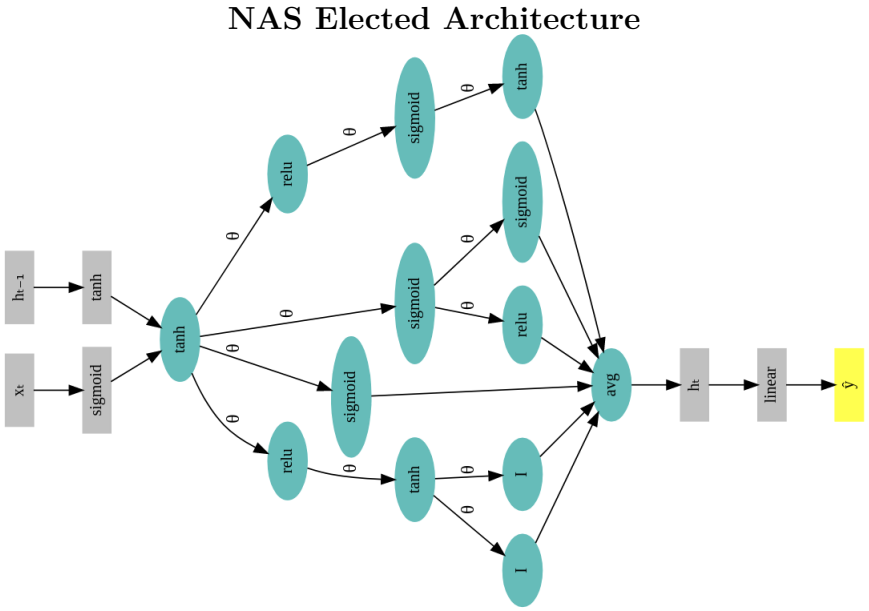

Figure 6 – The final elected RNN-cell architecture during the three rounds of NAS. This architecture was incorporated to be part of N-BEATS-RNN.

frequency. Naturally, it was not easy to surpass the performance of such a large ensemble of models. Therefore, N-BEATS-RNN's overall performance was slightly lower than N-BEATS'.

Nonetheless, the overall results were still superior to all other submission types from the original competition. N-BEATS-RNN achieved a *sMAPE* and *OWA* results of 11.242% and 0.810, which were 1% and 1.8% higher than N-BEATS' respective results, and 2% and 1.3% lower than the M4 competition winner.

Not surprisingly, the only case where N-BEATS-RNN was superior to N-BEATS, in the *OWA* category, was on the "Quarterly" time series, which was the specific data chosen for the architecture optimization. This suggests that perhaps N-BEATS-RNN could benefit from an architecture optimization search for each specific seasonal frequency, as opposed to generalizing the architecture from search results across different sets.

Table 5 – Performance on the M4 test set *sMAPE*. Lower values are better.

	Yearly	Quarterly	Monthly	Others	Avg.
Best pure ML	14.397	11.031	13.973	4.566	12.894
Best statistical	13.366	10.155	13.002	4.682	11.986
Best ML/TS	13.528	9.733	12.639	4.118	11.72
M4 winner (DL/TS)	13.176	9.679	12.126	4.014	11.374
N-BEATS	12.913	9.213	12.024	3.643	11.135
N-BEATS-RNN	13.134	9.214	12.140	3.646	11.242

Table 6 – Performance on the M4 test set, *OWA* and M4 rank. Lower values are better.

	Yearly	Quarterly	Monthly	Others	Avg.
Best pure ML	0.859	0.939	0.941	0.991	0.915
Best statistical	0.788	0.898	0.905	0.989	0.861
Best ML/TS	0.799	0.847	0.858	0.914	0.838
M4 winner (DL/TS)	0.778	0.847	0.836	0.920	0.821
N-BEATS	0.758	0.800	0.819	0.840	0.795
N-BEATS-RNN	0.773	0.799	0.823	0.900	0.810

5.2.2 Ensemble Sensitivity

We have also observed the impact of incrementing the N-BEATS-RNN ensemble with more models, which were all trained from different random initializations. As shown by Figure 7, we have observed that increasing the ensemble size has a considerably meaningfully effect for up to 8 models. Further improvements were generally observed, but became rather limited. Therefore, we have opted not to pursue the strategy for more than 18 models for each seasonal frequency.

Similarly, we tested N-BEATS-RNN vs. our own reproduction of N-BEATS for the Quarterly frequency in terms of OWA. As illustrated by Figure 8, N-BEATS-RNN consistently outperformed the original N-BEATS in that segment of the data for up to 18 models.

We have also combined the 18 models from N-BEATS-RNN and the 18 models from N-BEATS for comparison purposes, and we managed to improve the forecasting accuracy even further, posting an OWA performance of 0.794, a 0.75% improvement over the original N-BEATS and a 0.63% improvement over our N-BEATS-RNN model.

5.2.3 Computational Cost

Each N-BEATS-RNN model required, on average, 2 hours and 20 minutes of training. Therefore, the total training time of 108 models for the N-BEATS-RNN ensembles amounted to about 10 full days. A sample of the original N-BEATS models was tested in the same computational resources for comparison. Each original model architecture took 30 minutes to 2 hours of training time, depending on the seasonal frequency, with an average of one hour, in line with the reported time by the authors. If the original N-BEATS models were trained in the same computational resources, circa 2,160 hours would have been required, or 90 consecutive days.

The results illustrated in Figure 8 give an interesting insight about the computational cost of the models based on the validation and test curves of the NAS. To smooth the validation and test curves, we applied a convolution function that takes the entire signal against a desired linear window of 10 data points, thus, returning a moving average

box by convolution (HUI; GRATZL, 1996). Based on these charts, firstly, we can conclude that the original N-BEATS can be used with fewer models, without compromising its performance. Secondly, given that N-BEATS-RNN costs a little over double the amount of time to train N-BEATS, we can see that, with an 8-model ensemble, N-BEATS-RNN is able to match the results of an 18-model ensemble of N-BEATS. This is insightful because the time to train each of those ensembles should be about the same, and, thus, N-BEATS-RNN did not seem to be costlier vs. the original N-BEATS.

It is important to note that the NAS computational cost phase, which took about 4 days, is not being considered here for a fairer comparison with N-BEATS, as there is little to no information on the architecture search and hyper-parameter tuning carried in Oreshkin et al. (2019). Much like in the N-BEATS research, we solved for a single architecture that is applied across the board for all types of time series.

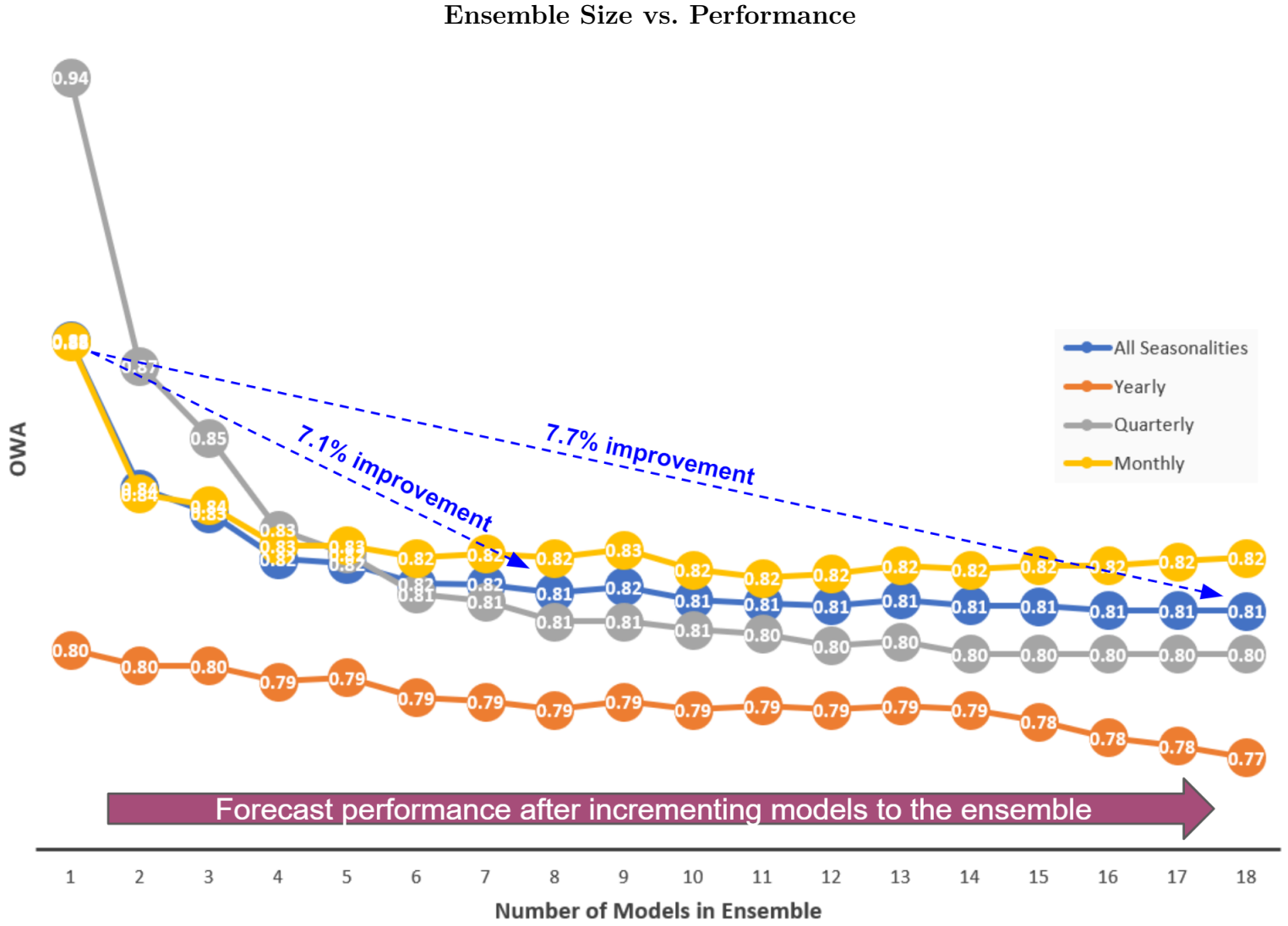

Figure 7 – The graph shows the OWA test set performance of all the seasonalities, as well for the largest three seasonalities, standalone, vs. the size of each ensemble. Incrementing N-BEATS-RNN with more models has had positive effect across the board, but it proved to be limited after 8 models.

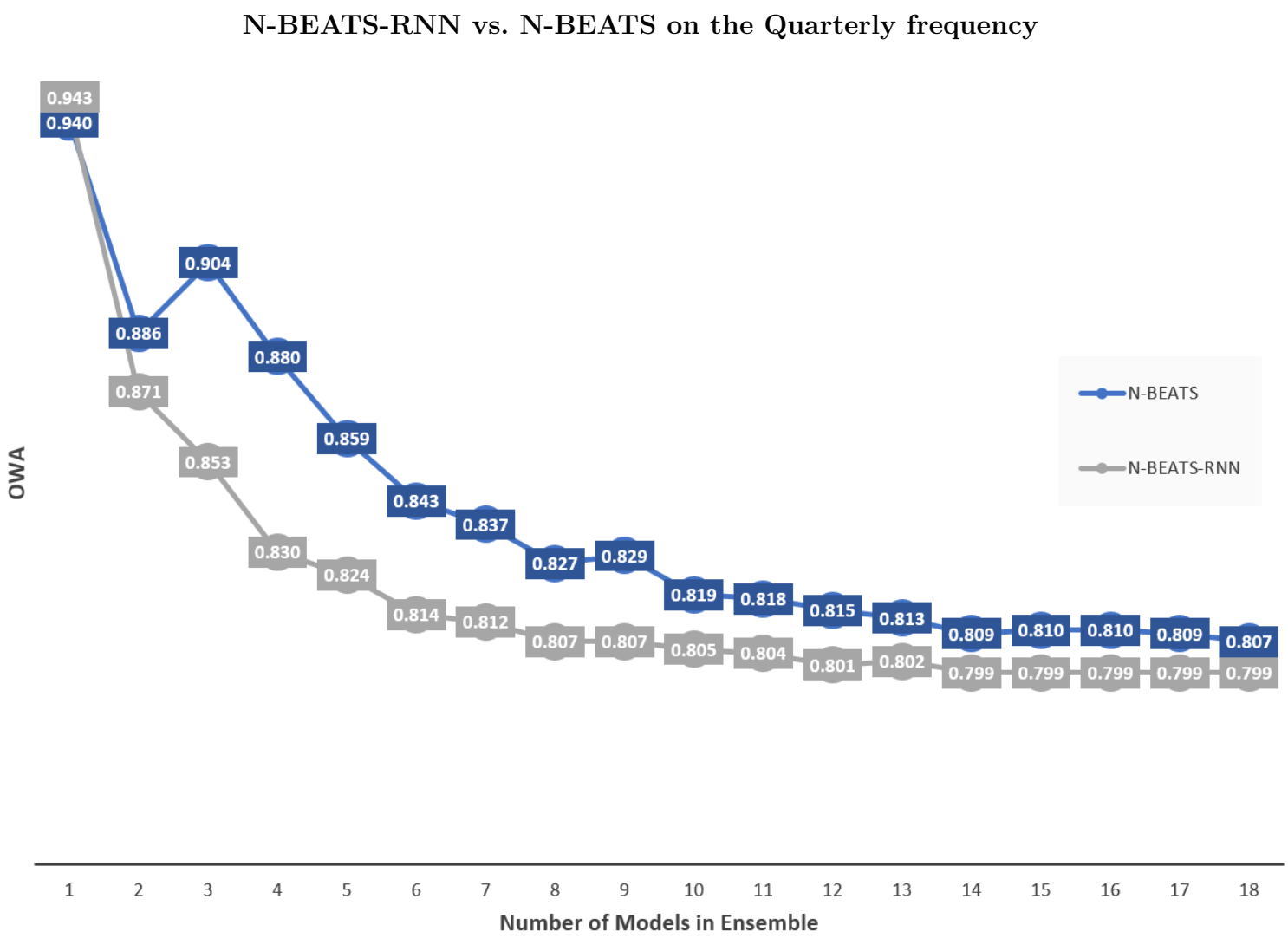

Figure 8 – The graph shows the OWA test set performance of N-BEATS-RNN vs. N-BEATS on the Quarterly frequency.

Conclusion

In this research, we have proposed an extension of N-BEATS by adding a Recurrent Neural Network (RNN) to process its residual outputs, named N-BEATS-RNN. An ideal RNN architecture was searched through a random optimization process based on a weight sharing technique in which all architecture attempts were able to share weights with previous attempts. We trained an ensemble of the best N-BEATS-RNN architecture resulting from the search on the M4 competition dataset and achieved state-of-the-art performance with comparable results to the competition winner and the original N-BEATS model. N-BEATS-RNN achieved an improvement of 9.8% over the competition's benchmark, vs. 11% by N-BEATS and 8.6% by the competition winner.

Therefore, with regards to **Hypothesis 1** we were able to partially validate it by demonstrating that it was possible to build a N-BEATS-RNN model that outperforms all official entries both in terms of *sMAPE* and *OWA*.

We were also able to build a model that outperformed the original N-BEATS and all other models in at least one category, the Quarterly seasonality, in terms of *OWA*. The Quarterly seasonal frequency was the one chosen for the architecture search to take place. While it was possible to partially validate **Hypothesis 1** in the experiments herein, the results of N-BEATS-RNN in the Quarterly seasonality still leave room for the hypothesis to be fully validated. In order to test such hypothesis further, the architecture search could be extended to the other frequencies individually, seeking for results that are better to those of the original N-BEATS on the individual frequencies, although this would come at the expense of computation resources. An investigation of this phenomenon should be the subject of future work.

On the other hand, we trained significantly fewer models, 108 as opposed to the original 2,160 models, resulting in a reduced training time of 9 times when compared with the original 2,160-model N-BEATS ensemble. Thus, we were able to validate **Hypothesis 2**, by offering a method that had state-of-the-art results, comparable to those of the original N-BEATS, with a rather drastic time reduction, as measured by GPU-days and the sheer number of models.

Finally, as measured by the Goals and Success Criteria established in Chapter 1, this research has at least partially achieved its objectives of performance and cost-efficiency improvements. Most essentially, this research has presented a more computationally efficient method that could more favorably democratize "pure" DL solutions for the univariate forcasting problem.

Publications

The following publication is a result from this work:

- SBRANA, A.; NALDI, M. C.; ROSSI, A. L. D. N-BEATS-RNN: deep learning for time series forecasting, *19th IEEE International Conference On Machine Learning And Applications, ICMLA 2020*. Miami, USA. 2020.

Bibliography

ADAM, G.; LORRAINE, J. Understanding Neural Architecture Search Techniques. *Computing Research Repository*, arXiv:1904.00438, p. 1–7, 2019. Disponível em: <http://arxiv.org/abs/1904.00438>. Citado na página 23.

AFOLABI, D. et al. Hierarchical meta-learning in time series forecasting for improved interference-less machine learning. *Symmetry*, v. 9, n. 11, p. 283, nov 2017. ISSN 20738994. Disponível em: <http://www.mdpi.com/2073-8994/9/11/283>. Citado na página 14.

ASSIMAKOPOULOS, V.; NIKOLOPOULOS, K. The theta model: A decomposition approach to forecasting. *International Journal of Forecasting*, v. 16, n. 4, p. 521–530, 2000. ISSN 01692070. Citado 2 vezes nas páginas 14 and 20.

BAKER, B. et al. Designing neural network architectures using reinforcement learning. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, Toulon, France, p. 1–18, 2017. Citado na página 22.

BROWN, P. F. et al. An estimate of an upper bound for the entropy of english. *Comput. Linguistics*, v. 18, n. 1, p. 31–40, 1992. Citado na página 23.

ELSKEN, T.; METZEN, J. H.; HUTTER, F. Neural Architecture Search: A Survery. *Journal of Machine Learning Research*, v. 20, p. 63–77, 2019. Disponível em: <http://jmlr.org/papers/v20/18-598.html.> Citado 3 vezes nas páginas 16, 21, and 22.

GARDNER, E. S. Exponential smoothing: The state of the art—part ii. *International Journal of Forecasting*, v. 22, n. 4, p. 637 – 666, 2006. ISSN 0169-2070. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0169207006000392>. Citado na página 14.

GILLILAND, M. The value added by machine learning approaches in forecasting. *International Journal of Forecasting*, Elsevier B.V., v. 36, n. 1, p. 161–166, 2019. ISSN 01692070. Disponível em: <https://doi.org/10.1016/j.ijforecast.2019.04.016>. Citado na página 14.

He, K. et al. Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: [s.n.], 2016. p. 770–778. Citado na página 25.

HE, X.; ZHAO, K.; CHU, X. AutoML: A Survey of the State-of-the-Art. *Computing Research Repository*, arXiv:1908.00709, p. 1–17, aug 2019. Disponível em: <http://arxiv.org/abs/1908.00709>. Citado 4 vezes nas páginas 15, 16, 21, and 22.

HUI, K. Y.; GRATZL, M. Anomalies of convolutional smoothing and differentiation. *Analytical Chemistry*, v. 68, n. 6, p. 1054–1057, 1996. ISSN 00032700. Citado na página 40.

HYNDMAN, R. J. A brief history of forecasting competitions. *International Journal of Forecasting*, Elsevier B.V., v. 36, n. 1, p. 7–14, 2019. ISSN 01692070. Disponível em: <https://doi.org/10.1016/j.ijforecast.2019.03.015>. Citado 2 vezes nas páginas 19 and 27.

LI, L.; TALWALKAR, A. Random Search and Reproducibility for Neural Architecture Search. arXiv:1902.07638, p. 1–20, 2019. Disponível em: <http://arxiv.org/abs/1902.07638>. Citado 4 vezes nas páginas 23, 27, 28, and 35.

LIU, H. et al. Hierarchical Representations for Efficient Architecture Search. *Computing Research Repository*, arXiv:1711.00436, p. 1–13, 2017. Disponível em: <http://arxiv.org/abs/1711.00436>. Citado na página 23.

LIU, H.; SIMONYAN, K.; YANG, Y. DARTS: Differentiable architecture search. *7th International Conference on Learning Representations, ICLR 2019*, 2019. Disponível em: <https://github.com/quark0/dartshttp://arxiv.org/abs/1806.09055>. Citado 4 vezes nas páginas 22, 23, 27, and 28.

LUO, R. et al. Neural architecture optimization. In: *Advances in Neural Information Processing Systems*. Montreal, Canada: [s.n.], 2018. v. 2018-Decem, p. 7816–7827. ISSN 10495258. Disponível em: <https://github.com/renqianluo/NAO.> Citado 2 vezes nas páginas 23 and 35.

LUONG, M. T. et al. Multi-task sequence to sequence learning. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, n. c, p. 1–10, 2016. Citado na página 22.

M4-TEAM. *M4 competitor's guide: prizes and rules*. 2018. URL www.m4.unic.ac.cy/wp-content/uploads/2018/03/M4-CompetitorsGuide.pdf. Citado 2 vezes nas páginas 30 and 31.

M4-TEAM. *M4 dataset*. 2018. URL https://github.com/M4Competition/M4-methods/tree/master/Dataset. Citado na página 30.

MAKRIDAKIS, S.; PETROPOULOS, F. The M4 competition: Conclusions. *International Journal of Forecasting*, v. 36, p. 224–227, 2019. ISSN 01692070. Citado na página 20.

MAKRIDAKIS, S.; SPILIOTIS, E.; ASSIMAKOPOULOS, V. The M4 Competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, Elsevier B.V., v. 34, n. 4, p. 802–808, 2018. ISSN 01692070. Disponível em: <https://doi.org/10.1016/j.ijforecast.2018.06.001>. Citado na página 19.

MAKRIDAKIS, S.; SPILIOTIS, E.; ASSIMAKOPOULOS, V. Predicting/hypothesizing the findings of the M4 Competition. *International Journal of Forecasting*, Elsevier B.V., v. 36, n. 1, p. 29–36, 2019. ISSN 01692070. Disponível em: <https://doi.org/10.1016/j.ijforecast.2019.02.012>. Citado na página 21.

MAKRIDAKIS, S.; SPILIOTIS, E.; ASSIMAKOPOULOS, V. The M4 Competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, Elsevier B.V., v. 36, n. 1, p. 54–74, 2019. ISSN 01692070. Disponível em: <https://doi.org/10.1016/j.ijforecast.2019.04.014>. Citado 5 vezes nas páginas 10, 14, 19, 20, and 27.

ORESHKIN, B. N. et al. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. arXiv:1905.10437, p. 1–12, 2019. Disponível em: <http://arxiv.org/abs/1905.10437>. Citado 7 vezes nas páginas 15, 16, 21, 25, 27, 33, and 40.

PHAM, H. et al. Efficient Neural Architecture Search via parameter Sharing. In: *35th International Conference on Machine Learning, ICML 2018*. Stockholm, Sweden: [s.n.], 2018. v. 9, p. 6522–6531. ISBN 9781510867963. Disponível em: <https://arxiv.org/abs/1802.03268>. Citado 4 vezes nas páginas 22, 23, 27, and 28.

SMYL, S. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, Elsevier B.V., v. 36, n. 1, p. 75–85, 2019. ISSN 01692070. Disponível em: <https://doi.org/10.1016/j.ijforecast.2019.03.017>. Citado 7 vezes nas páginas 14, 15, 16, 21, 25, 26, and 27.

SPILIOTIS, E. et al. Are forecasting competitions data representative of the reality? *International Journal of Forecasting*, Elsevier B.V., v. 36, n. 1, p. 37–53, 2019. ISSN 01692070. Disponível em: <https://doi.org/10.1016/j.ijforecast.2018.12.007>. Citado na página 14.

WERON, R. Electricity price forecasting: A review of the state-of-the-art with a look into the future. *International Journal of Forecasting*, v. 30, 10 2014. Citado na página 15.

WISTUBA, M.; RAWAT, A.; PEDAPATI, T. A Survey on Neural Architecture Search. *Computing Research Repository*, arXiv:1905.01392, p. 1–53, 2019. Disponível em: <http://arxiv.org/abs/1905.01392>. Citado 3 vezes nas páginas 16, 21, and 22.

YU, K. et al. Evaluating the Search Phase of Neural Architecture Search. p. 1–16, 2019. Disponível em: <http://arxiv.org/abs/1902.08142>. Citado na página 23.

ZAREMBA, W. et al. Recurrent Neural Network Regularization. n. 2013, p. 1–8, 2014. Disponível em: <http://arxiv.org/abs/1409.2329>. Citado na página 23.

ZOPH, B.; LE, Q. V. Neural architecture search with reinforcement learning. In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. Toulon, France: [s.n.], 2017. p. 1–16. ISBN 1611.01578v2. Citado 2 vezes nas páginas 22 and 23.

ZOPH, B. et al. Transfer learning for low-resource neural machine translation. *EMNLP 2016 - Conference on Empirical Methods in Natural Language Processing, Proceedings*, Austin, TX, USA, p. 1568–1575, 2016. Citado na página 22.