British Actuarial Journal (2024), Vol. **29**, e20, pp. 1–21
doi:10.1017/S1357321724000217

SESSIONAL PAPER

Time series analysis of GSS bonds Part 2 – further univariate analysis of S&P Green Bond Index

[IFoA Data Science, Sustainability & Climate Change Working Party, August 2024]

Debashish Dey

Email: debsdey@hotmail.com

Abstract
The popularity of green, social and sustainability-linked bonds (GSS bonds) continues to rise, with circa US $939 billion of such bonds issued globally in 2023. Given the rising popularity of ESG-related investment solutions, their relatively recent emergence, and limited research in this field, continued investigation is essential. Extending non-traditional techniques such as neural networks to these fields creates a good blend of innovation and potential. This paper follows on from our initial publication, where we aim to replicate the *S&P Green Bond Index* (i.e. this is a time series problem) over a period using non-traditional techniques (neural networks) predicting 1 day ahead. We take a novel approach of applying an N-BEATS model architecture. N-BEATS is a complex feedforward neural network architecture, consisting of basic building blocks and stacks, introducing the novel *doubly residual stacking* of *backcasts* and *forecasts*. In this paper, we also revisit the neural network architectures from our initial publication, which include DNNs, CNNs, GRUs and LSTMs. We continue the univariate time series problem, increasing the data input window from 1 day to 2 and 5 days respectively, whilst still aiming to predict 1 day ahead.

Keywords: GSS Bonds; green bonds; time series; neural networks; N-BEATS; CNN; LSTM; GRU

Disclaimer

The views expressed in this publication are those of invited contributors and not necessarily those of the Institute and Faculty of Actuaries (IFoA).

The Institute and Faculty of Actuaries does not endorse any of the views stated, nor any claims or representations made in this publication and accept no responsibility or liability to any person for loss or damage suffered as a consequence of their placing reliance upon any view, claim or representation made in this publication. The information and expressions of opinion contained in this publication are not intended to be a comprehensive study, nor to provide actuarial advice or advice of any nature and should not be treated as a substitute for specific advice concerning individual situations. On no account may any part of this publication be reproduced without the written permission of the Institute and Faculty of Actuaries.

This paper expresses the views of the individual author and not necessarily those of their employer.

1. Executive Summary

We are pleased to publish our second paper as a Working Party using data science techniques to look at sustainability and climate change-related issues. In this paper, we summarise the second

© The Institute and Faculty of Actuaries, 2024. Published by Cambridge University Press on behalf of the Institute and Faculty of Actuaries. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted re-use, distribution, and reproduction in any medium, provided the original work is properly cited.

stage of our analysis, where we explore more complex data science techniques and architectures to continue our time series analysis of the *Standard & Poor's (S&P) Green Bond Index*. Details of the initial paper can be found in Dey (2024). The underlying data has been taken from https://www.spglobal.com/.

1.1. Scope of this Paper

This paper builds on the initial publication on 1 November 2023 via the IFoA blog site (Dey, 2023) and subsequent publication on 11 March 2024 via the British Actuarial Journal (Dey, 2024), where we extend the time series univariate analysis to a more complex neural network architecture. We have deliberately excluded traditional stationarity techniques such as ARIMA as well as restricted this paper to a univariate analysis, to help focus on the impact of using a pure neural network design and help with interpretability of the results. We may consider traditional stationarity techniques and expanding our analysis to multivariate in subsequent papers.

For the purposes of this paper, we have focussed the *S&P Green Bond Index* and performed various univariate time series analyses using a range of neural network architectures only. The early part of this paper, Section 3, mainly focusses on using a rolling window approach of one prior day's index value to predict today's index value. The latter part of this paper, Section 4, extends the input window to a longer input period but the output horizon remains as 1 day.

In particular, this paper discusses (arranged as per the following sections):

- 2. Introduction
  Recap on the prior analysis (Dey, 2024), the data used and how we will build on our analysis in this paper.
- 3. N-BEATS
  Introduction to N-BEATS (Neural Basis Expansion Analysis for interpretable Time Series forecasting) architecture and extending our time series analysis to incorporate N-BEATS.
- 4. Widening the window
  Extending our analysis to widen the input window into the original architectures from the first paper (Dey, 2024).
- 5. Conclusions and next steps
  Summary of conclusions from our analysis and potential areas of analysis for subsequent papers.

Please note that many of the techniques mentioned in this paper build on Dey (2024) and the detail is not duplicated here. For more underlying information on some of the architecture and modelling techniques, please refer to Dey (2024) – this paper will be referred to as "initial paper" throughout – and Appendix 3.

1.2. Summary of Analysis in this Paper

1.2.1. Aim of the analysis

This paper extends on the initial stages of our time series analysis on GSS (Green, Social and Sustainability) bonds in Dey (2024), specifically focussing on the daily values from the *Standard and Poor's (S&P) Green Bond Index* and whether or not we can create accurate prediction models using neural networks. This paper builds on the initial paper, where we hope to develop a model that can assist with GSS bond index prediction, which will have wider applications such as index price modelling and investment portfolio analyses for actuaries and non-actuaries alike. For the purposes of this paper, we continue to look at predicting a rolling 1-day value of the index, based on the prior x days index value over the period 2013 to 2023 inclusive.

We retain the same Baseline model i.e. today's value equals yesterday's value over the course of the full date range of 31 January 2013 to 17 February 2023. Similarly to the initial paper, we aim to see if we can accurately create a time series model with non-traditional methods such as neural networks and more complex neural network architectures, using N-BEATS. Analyses using traditional stationarity techniques such as ARIMA and multivariate techniques have been deferred to later papers.

Towards the end of this paper, we extend our analysis by looking at using the prior x index values (window) to predict the next y days in the future (horizon), rerunning the initial neural network architectures analysed in Dey (2024). We restrict the input window to 2 and 5 days, whilst retaining a future horizon to 1 day in this paper. In Section 3, for N-BEATS, we train the model with an input window of 1 day in addition.

1.2.2. Data and method

We have retained the same underlying data as per Dey (2024) – the *S&P Green Bond Index* values between 31 January 2013 and 17 February 2023, splitting the data using 70%/20%/10% ratios for training/validation/test data sets. Section 3 of this paper extends our analysis by introducing a more complex model architecture, the N-BEATS model architecture.

The latter part of this paper, Section 4, revisits the models analysed in our initial paper but extends the input window of data to 2 days and 5 days respectively. These models can be categorised into the following model categories: Deep Neural Network (DNN), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) neural network architectures. This is discussed in Section 4 and Appendix 3 in more detail.

The loss function used during training the models was set to Mean Absolute Error (MAE) for all models, with an Adam optimiser used to update the weights to minimise the loss function as part of the training process and L2 regularisation to reduce any overfitting during the training process. Hyperparameter tuning of all models was completed via the open-source library Optuna, using the Bayesian optimisation algorithm Tree-structure Parzen Estimator (TPE). These techniques are discussed in more detail in Dey (2024).

1.2.3. Results and conclusions

The results of our analysis for N-BEATS (Section 3) were inconclusive: the models from each category produced comparable results to the Baseline model with differences in Mean Absolute Percentage Error (MAPE) of up to approximately +0.04% and hence with no material outperformance.

Using similar neural network architectures as per the initial paper (Section 4), if we extend the input window to 2 and 5 days respectively, there was no improvement in model performance when compared with the Baseline, with differences in MAPE of up to around +0.3%. Widening the window from 2 days to 5 days did result in an improvement in some models, though again the overall model performance was still worse than the Baseline. Hence, the overall results were inconclusive.

We are potentially not sharing sufficient correlated information e.g. as per a multivariate analysis, for our models to learn underlying material information and patterns in the data to result in a model that materially outperforms the Baseline. We aim to address this in future papers (please see below and Section 5 for more details).

1.2.4. Next steps

For future papers, we will expand our analysis to include the following:

1. Expanding the analysis to general GSS bonds. The analysis in this paper is based on a single green bond index. We will look to expand our analysis to the wider GSS bond universe and

over differing date ranges for the data to see if there are general underlying conclusions across different data sets and GSS bond indices.
2. Expanding the analysis to include any potential relationships with the general market such as stock market and oil prices i.e. move to a multivariate analysis in subsequent papers. This is examined e.g. in Wang *et al.* (2021) which, when coupled with the CEEMDAN-LSTM model, seems to produce materially improved model predictions based on green bond time series data when compared to a baseline model.
3. Extend our analysis to alternative neural network architecture types, such as Graph Neural Networks (GNNs).

1.2.5. Additional disclaimers

Please note the following:

a. Information within this paper is **valid up to 31 May 2024**. Hence, there may be updates beyond this date that are not reflected in this paper e.g. changes to any legislation mentioned or updates to any open-source libraries used.
b. This paper is not intended to be a comprehensive audit of models. Neither is this paper recommending or promoting one approach over another, nor promoting any of the sources or references stated in this paper. Any user of this paper should still reference the underlying legislation, reference any standard mentioned in this paper, and should there be any conflict, the underlying information in the relevant standard, reference or legislation supersedes any information presented in this paper.
c. Though the work in this paper does not fall under the Financial Reporting Council's Technical Actuarial Standards, this paper has been reviewed both within the Working Party and by the Institute and Faculty of Actuaries' Data Science Practice Board.

2. Introduction

2.1. Recap on Prior Analysis

In our initial paper Dey (2024), we produced a time series analysis of the *S&P Green Bond Index* and aimed to replicate the index using various models over a certain time period. The data in our analysis was from the end of January 2013 to mid-February 2023.

Our initial stages of analysis focussed on a univariate time series model, where our model used the prior day's value to predict the index value one day ahead (i.e. today's value). The prediction models used were predominantly neural network architectures: deep neural networks (DNNs), Convolutional Neural Networks (CNNs), Long Short-Term Memory models (LSTMs) and Gated Recurrent Units (GRUs). Please note that for the purposes of this and the initial paper Dey (2024), DNNs included 1 to 3 hidden layers. Strictly, a DNN has 2 or more hidden layers. However, for ease of categorisation, we have extended this labelling to include 1 hidden layer as well. In our initial paper, we further extended the analysis to include the popular library XGBoost, which is a decision-tree model. We have not considered XGBoost further in this paper.

For the purposes of this paper, we will continue with our analysis with an overall aim to produce a sufficiently accurate predictive model, where the models will be based on a neural network architecture design. Data from 31 January 2013 to around mid-February 2022 will be used to train and validate the models. These models will then be used to predict daily index values on unseen data from mid-February 2022 to mid-February 2023 (i.e. the test data set). The difference in predicted values from our models with actual daily index figures will be used to gauge the accuracy of the proposed models.

2.2. Summary of Data Used

We have continued to use the same data set as per Dey (2024), downloading via a free subscription-based account from the main S&P website https://www.spglobal.com/ . For details of the data set and underlying testing of this data, please see Dey (2024).

For the purposes of this paper, we will analyse the *S&P Green Bond Index* (Total Performance, USD, from 31 January 2013 to 17 February 2023 inclusive). The splits are as per the initial paper and further shown in Figure 1 and Table 1. Given the nature of the time series analysis we have ordered the data and these splits chronologically, so that we can build models to infer some form of prior time-dependency based on the underlying data, and have not randomly allocated the data between these splits across the full data range.

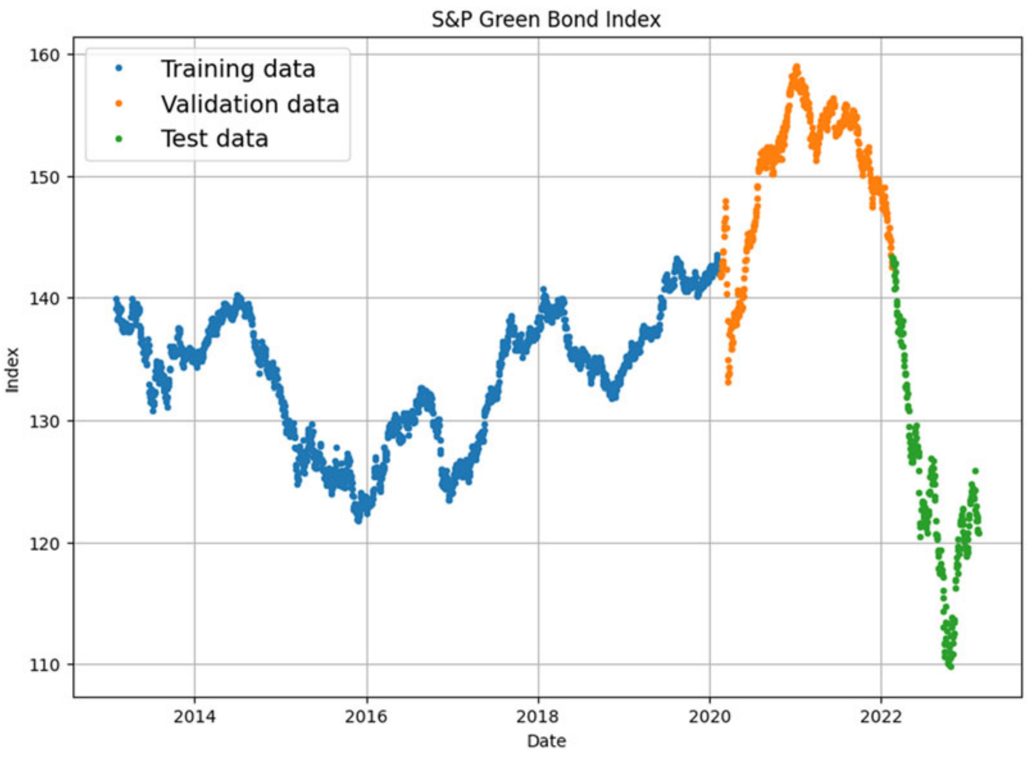

Figure 1. *S&P Green Bond Index* data with training/validation/test splits highlighted (output from Google Colab).

Table 1. Summary table of data used, split by training, validation and test data sets.

	Full Data	Training Data	Validation Data	Test Data
Start date	31 Jan 2013	31 Jan 2013	17 Feb 2020	17 Feb 2022
End date	17 Feb 2023	16 Feb 2020	16 Feb 2022	17 Feb 2023
Number of index entries	2,615	1,830	523	262
Index minimum	109.80	121.78	133.14	109.80
Index maximum	158.99	143.59	158.99	143.34
Index average (2 d.p.)	136.00	133.60	150.46	123.96
Index standard deviation (2 d.p.)	9.65	5.32	5.70	8.04

The splits are: to 16 February 2020, to 16 February 2022 and to 17 February 2023 inclusive. Table 1 details further each data split.

In summary, there is greater volatility in the test data set range when compared to the training and validation data sets. Hence, it will be interesting to see how our models cope given that they will be built on less volatile training and validation data.

Similar to Dey (2024), for the purposes of our analysis, we have not adjusted the data further i.e. no normalisation of the index (setting to a scale of 0 to 1, a technique typically used to result in a quicker convergence to a solution for a model) and no log transformation (which can be used to potentially dampen any impact of seasonality). Such techniques may be discussed in later papers.

3. N-BEATS

3.1. Introduction

We start our analysis with the N-BEATS (Neural Basis Expansion Analysis for interpretable Time Series forecasting) architecture. In this section, we give an overview of the model architecture, before explaining in more detail the generalised model architecture. We then explain how we trained this model architecture specifically on our data set and finally present our results from this analysis.

Originally submitted in 2019, with the publication revised in 2020, the N-BEATS time series model was specifically designed to tackle time series problems. We refer to this publication Oreshkin *et al.* (2019) as the “source paper”. The source paper states that N-BEATS demonstrates *state-of-the art performance, improving forecast accuracy by 11% over a statistical benchmark and by 3% over last year’s winner of the M4 competition* (Oreshkin *et al.*, 2019). The paper considers tackling a univariate discrete time series problem. Given this, using the model architecture for our forecasting problem seems appropriate.

Oreshkin *et al.* (2019) discusses how deep learning techniques have struggled to consistently outperform more traditional statistical techniques: *the rankings of the six “pure” ML methods submitted to M4 competition were 23, 37, 38, 48, 54, and 57 out of a total of 60 entries* (Oreshkin *et al.*, 2019). Please note however that we do not explore more traditional techniques in our paper and continue to focus on non-traditional techniques i.e. neural networks.

The Makridakis Competitions is an open time series competition (Makridakis *et al.*, 2020). The fourth iteration of the competition, M4 which is mentioned above, was held in 2020.

Beyond designing a model architecture that uses deep learning to tackle a time series problem, the aim of Oreshkin *et al.* (2019) was to also *design an architecture with interpretable outputs that can be used by practitioners in very much the same way as traditional decomposition techniques such as the “seasonality-trend-level” approach* (Oreshkin *et al.*, 2019).

Below are some key features of the N-BEATS model architecture:

1. It is *a deep neural architecture based on backward and forward residual links and a very deep stack of fully connected layers* (Oreshkin *et al.*, 2019). The model architecture is made up of stacks that are in turn made up of blocks. Each block produces a *backcast residual* and *partial forecast* with the aim of iteratively improving the prior input value and augmenting the future value respectively within the forecasting process. For more details of the underlying architecture, please see Section 3.2.
2. One of the aims of an N-BEATS model architecture is to ensure underlying patterns in the data are *interpretable* (i.e. understandable) for users, where the model can infer an underlying trend and seasonality pattern within any given time series data set.

For a high-level introductory summary of N-BEATS, please see for example an article published on the *Towards Data Science* website (Dancker, 2024).

3.2. Deeper dive into Generalised N-BEATS Model Architecture

Below we look at the generalised N-BEATS model architecture in more detail. We summarise some of the key points raised from Oreshkin *et al.* (2019) and use the same notation as per Oreshkin *et al.* (2019) when presenting formulae.

The generalised model architecture is based on the following principles (Oreshkin *et al.*, 2019):

1. It should be simple and generic, yet expressive (deep).
2. It should not rely on time series-specific feature engineering or input scaling.
3. The outputs should be extendable towards making its outputs human interpretable.

The first two principles allow a user to set up a time series forecasting model based on a pure deep learning architecture.

Figure 2 shows a generalised overview of the architecture design underlying N-BEATS model architecture. Figure 2 is taken directly from Oreshkin *et al.* (2019).

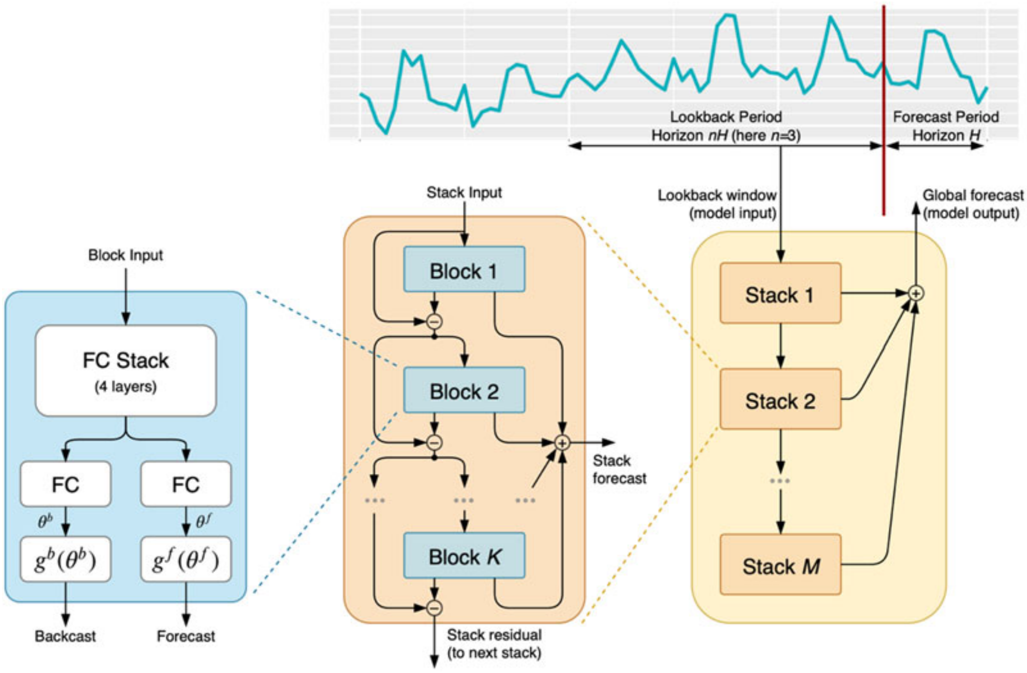

Figure 2. Generalised N-BEATS architecture. Source: Oreshkin *et al.* (2019).

Below is an overview of the N-BEATS architecture:

1. Basic building blocks consist of hidden layers each of several neurons, which are fully connected. The outputs from this structure can be further extended to incorporate basis functions which allow for time series features such as seasonality and trends. The basic building block is represented by the blue box on the left in Figure 2, which illustrates 4 hidden layers.
2. These basic building blocks in turn form a stack, which take the residual *backcasts* from the prior block and cumulate the partial trend *forecasts* outputs. This is represented by the middle box in Figure 2, which illustrates K basic building blocks.
3. In a similar fashion, the residual *backcasts* from the prior stack feed into later stacks, and then cumulate the partial trend *forecasts* to an overall *global forecast* i.e. the overall model output. This is represented by the box towards the right in Figure 2 earlier, which illustrates M stacks in total.

The above technique of residuals and partial forecasts form part of *doubly residual stacking*. This is discussed further in Section 3.2.2 below.

3.2.1. Basic building block

The basic building blocks generate a *backcast* and *forecast* as previously mentioned. The *backcast* can be viewed as iterative adjustments to the input vectors via residuals, whilst the *forecast* from each basic building block can be viewed as a partial *forecast* which is cumulated and form part of the overall *global forecast*.

Focusing on the l^{th} basic building block, as per Oreshkin *et al.* (2019): each basic building block has a fork architecture which takes an input vector (or residual outputs from a prior basic building block) $\boldsymbol{x}_l$ and produces two outputs: $\widehat{\boldsymbol{x}}_l$ i.e. the block's best estimate of $\boldsymbol{x}_l$ and the block's *forward forecast* $\widehat{\boldsymbol{y}}_l$. The input of the very first block in the model is the original data input. The remaining basic building block inputs are residual outputs from a prior basic building block, and *can be thought of as running a sequential analysis of the input signal* (Oreshkin *et al.*, 2019). See Section 3.2.2 for more details.

The internal structure of a basic building block broadly consists of two sections: a fully connected section and a basis layer section.

The fully connected layer produces backward (*backcast*) and forward (*forecast*) predictors, θ_l^b and θ_l^f respectively, of *expansion coefficients*. The predictors determine how much each basis function contributes to the overall approximation of the function. Please see Section 3.2.1.2 for more details on basis function.

The above two sections of the internal structure of a basic building block are discussed in more detail immediately below in Sections 3.2.1.1 and 3.2.1.2.

3.2.1.1. Fully connected section.

Using the same formulae notation and architecture discussed in Oreshkin *et al.* (2019), which has four hidden layers, the equations within the l^{th} basic building block are as follows:

- $\boldsymbol{h}_{l,1} = \mathrm{FC}_{l,1}(\boldsymbol{x}_l)$, $\boldsymbol{h}_{l,2} = \mathrm{FC}_{l,2}(\boldsymbol{h}_{l,1})$, $\boldsymbol{h}_{l,3} = \mathrm{FC}_{l,3}(\boldsymbol{h}_{l,2})$, $\boldsymbol{h}_{l,4} = \mathrm{FC}_{l,4}(\boldsymbol{h}_{l,3})$, where $\boldsymbol{h}_{l,i}$ represents the ith hidden layer in the lth block, $\mathrm{FC}_{l,i}$ represents the ith fully connected layer in the lth block, and x_l is the input vector for the lth block.
- $\theta_l^b = LINEAR_l^b(\boldsymbol{h}_{l,4})$ and $\theta_l^f = LINEAR_l^f(\boldsymbol{h}_{l,4})$, where $\boldsymbol{h}_{l,i}$, θ_l^b and θ_l^f are as per earlier. LINEAR is a linear projection layer i.e. $\theta_l^b = W_l^b(\boldsymbol{h}_{l,4})$ and $\theta_l^f = W_l^f(\boldsymbol{h}_{l,4})$ with the subscript l representing the lth block, the superscripts b and f representing backcast and forecast respectively, and W is a weight matrix active on a hidden layer.

For the purposes of Oreshkin *et al.* (2019), *FC* is a standard fully connected layer with a *relu activation function* i.e. $\boldsymbol{h}_{l,1} = RELU\ (W_{l,1}\boldsymbol{x}_l + \boldsymbol{b}_{l,1})$, where $W_{l,1}$ represents the weight vector connecting the input x_l to the first hidden layer in the l^{th} block and $\boldsymbol{b}_{l,1}$ is the bias term for the first hidden layer in the l^{th} block.

3.2.1.2. Basis layer section.

The basis layer maps the *backcast and forecast expansion coefficients* θ_l^b and θ_l^f to outputs, where the partial output $\widehat{\boldsymbol{y}}_l = g_l^f\left(\theta_l^f\right)$ and the residual output $\widehat{\boldsymbol{x}}_l = g_l^b\left(\theta_l^b\right)$. The basis functions g_l^b and g_l^f are for *backcast* and *forecast* respectively.

More generally, *expansion coefficients* are parameters that are learned during the training process. When combined with basis functions, they help to capture any underlying patterns or features in a time series data set such as a trend, an anomaly or seasonal patterns. Examples of basis functions include polynomial, Fourier, wavelet and Gaussian functions.

3.2.2. Doubly residual stacking

A unique feature of the N-BEATS model architecture is the concept of *doubly residual stacking*. Instead of a single residual branch of outputs, the architecture proposes two branches as mentioned: one for a *backcast residual* and one for a *partial forecast*. Mathematically, we represent this as:

$$\boldsymbol{x}_l = \boldsymbol{x}_{l-1} - \widehat{\boldsymbol{x}_{l-1}}, \quad \widehat{\boldsymbol{y}} = \sum\nolimits_l \widehat{\boldsymbol{y}}_l,$$

where $\boldsymbol{x}_l$ and $\widehat{\boldsymbol{x}_{l-1}}$ are defined earlier in Section 3.2.1, $\widehat{\boldsymbol{y}}$ is the *global forecast* i.e. overall model output which is the sum of individual $\widehat{\boldsymbol{y}}_l$. Each $\widehat{\boldsymbol{y}}_l$ can be viewed as a *partial forecast*.

3.2.3. Interpretability

There are 2 types of configurations or design patterns for an N-BEATS architecture:

1. *Generic architecture*, which does not assume any specific underlying time series model and relies on a generalised deep learning approach.
2. *Interpretable architecture*, which is an augmented model that often includes a *trend* and *seasonality model* to the generalised N-BEATS architecture discussed earlier for a time series forecasting problem.

For the purposes of this paper, we have focussed on the first configuration i.e. *generic architecture* as the aim of this paper is to seek a model architecture that accurately predicts the time series future values without any further input by the modeller or prior knowledge of any underlying time series features from the modeller. For more details on the above, and other considerations, please see Oreshkin *et al.* (2019).

3.2.4. Comparison with other model architectures

In Table 2, we provide a high-level comparison of the N-BEATS architecture versus DNNs, and LSTMs/GRUs. Please note that these are generalised comments based on experience and the comparison table below may differ for different data sets, problems and scenarios.

Table 2. Comparison of N-BEATS model architecture.

Model Architecture	DNN	LSTM, GRU	N-BEATS
Neural network type	Feedforward	Recurrent	Feedforward
Handles long-term memory	No	Yes	Yes – via *backcasting* and *forecasting*
Complexity	Low	Medium	High
Computational resources	Low	Medium	High
Indicative time taken to train model in Google Colab based on data set and methodology discussed in this paper (epochs and Optuna trials may vary)	Up to circa 15 mins	Up to circa 30 mins	circa 3 to 4 hours
Performance	Varies though better on short sequences	Varies though better on long sequences	Excellent for time series
Interpretability	Low	Medium	High

(*Continued*)

Table 2. (*Continued*)

Model Architecture	DNN	LSTM, GRU	N-BEATS
Typical usage	Wide range of problems	Sequential data e.g. Sentiment analysis, Language modelling, Speech recognition	Time series forecasting
Practical considerations	Easy to implement	Better for tasks requiring understanding of long-term dependencies	Highly effective for complex time series forecasting tasks where interpretability and performance are crucial

3.3. Model Approach Used and Training the Model

Below we give a brief overview of our code implementation of the N-BEATS model.

3.3.1. Code background

The underlying basis of our code for the N-BEATS model is taken from *Zero to Mastery TensorFlow for Deep Learning Book* (Bourke, 2023) and has been adapted for our analysis, where we look at a general N-BEATS architecture. We vary the number of layers within each block and the number of blocks per stack but set the total number of stacks equal to 1 (i.e. M is 1 in Figure 2). This is in part to simplify the trained model architecture given that the time taken to train such a model was close to 4 hours. Further, in doing so, we do not believe that taking such an approach would materially impact any overall conclusion, as effectively adding another stack would repeat a similar iterative fine-tuning process and potentially the hyperparameter optimisation could compensate by increasing the number of blocks in the single stack.

The original authors of the N-BEATS model also extend their analysis to allow for an ensemble technique (using multiple different loss functions and multiple different lookback periods in Figure 2) to make predictions when testing on M4 dataset. To retain consistency with the approach taken with other models within our initial paper, we have not extended our analysis to allow for such ensemble techniques.

We have assumed the same underlying architecture per block within each stack e.g. the same number of hidden layers within each block. Similarly, we have assumed the same underlying architecture for each hidden layer within each block e.g. number of neurons, activation function and L2 loss regularisation parameter. The output layer from the block is a generalised dense layer, each with the same number of neurons in effect as the total combined length of the input window and output horizon. Similarly, we have assumed that all output layers have the same activation function, though this may differ to the hidden layers above.

Similar to our initial paper, we have used the same underlying data and training/validation/test splits within the data to train our model. We have run our code in Google Colab, using similar libraries including Tensorfow and Keras as per our initial paper. For hyperparameter optimisation, we have again used Optuna. For an in-depth discussion, please see Dey (2024).

3.3.2. Hyperparameter optimisation

We have used Optuna for hyperparameter optimisation. Below we list some key search spaces used within our hyperparameter optimisation.

1. The number of epochs was set to 400.
2. The number of trials within Optuna was set to 30.

3. The batch size for the input data was set to 128.
4. The *activation list* was as follows: *elu*, *gelu*, *linear*, *relu*, *sigmoid*, *swish* and *tanh*.
5. The specific search space for each N-BEATS block includes:
   a. Number of neurons in each hidden layer from 4 to 512 at steps of 4.
   b. Number of hidden layers from 1 to 4 at steps of 1. As mentioned earlier, we have assumed all hidden layers have the same underlying architecture e.g. the same number of neurons, same activation function and same L2 regularisation.
   c. L2 regularisation with a search space of 1e-5 to 1e-1.
   d. Output *theta layer* or *basis layer* is a generalised deep layer.
      i. The number of neurons is set equal to the *theta size* – in effect the combined length is the window size and horizon size for this analysis.
      ii. The *activation function* is from the *activation list* above. No L2 regularisation function was applied to this layer.
      iii. No further adjustments were made for seasonality and trend i.e. via basis functions, as previously mentioned.
6. Specific search space for number of N-BEATS blocks per stack from 2 to 30 at steps of 2.
7. Similar to Dey (2024), the model was compiled on a loss function of Mean Absolute Error (MAE) using an Adam optimiser with a learning rate search space of 1e-5 to 1e-1.

3.3.3. Training the model

Below is a loss history curve based on training one of the models, using the search space, number of epochs and number of trials in Optuna as mentioned in Section 3.3.2 (Figure 3).

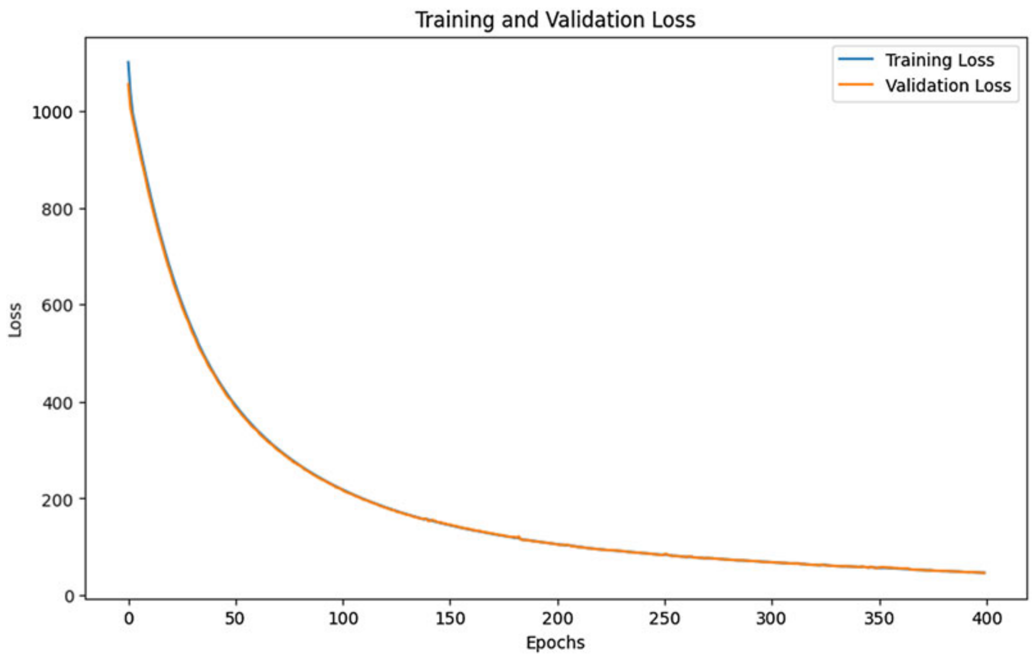

Figure 3. Training and validation loss curves (output from Google Colab).

As we would expect, the model loss decreases as the number of epochs increases, suggesting that the model is able to converge to a solution.

Similarly, we have used the open source *Keras visualizer* library to demonstrate the underlying architecture of the trained model in Figure 4. Please note that we have taken a snippet of the first few blocks and initial stacks, given the size of the overall final trained model.

The visualisation tool in general gives an alternative simplified representation of any final neural network model architecture. Further, given that we have generalised our code and set the overall number of stacks equal to 1 as mentioned earlier, there are cosmetic differences in Figure 4 when compared with Figure 2.

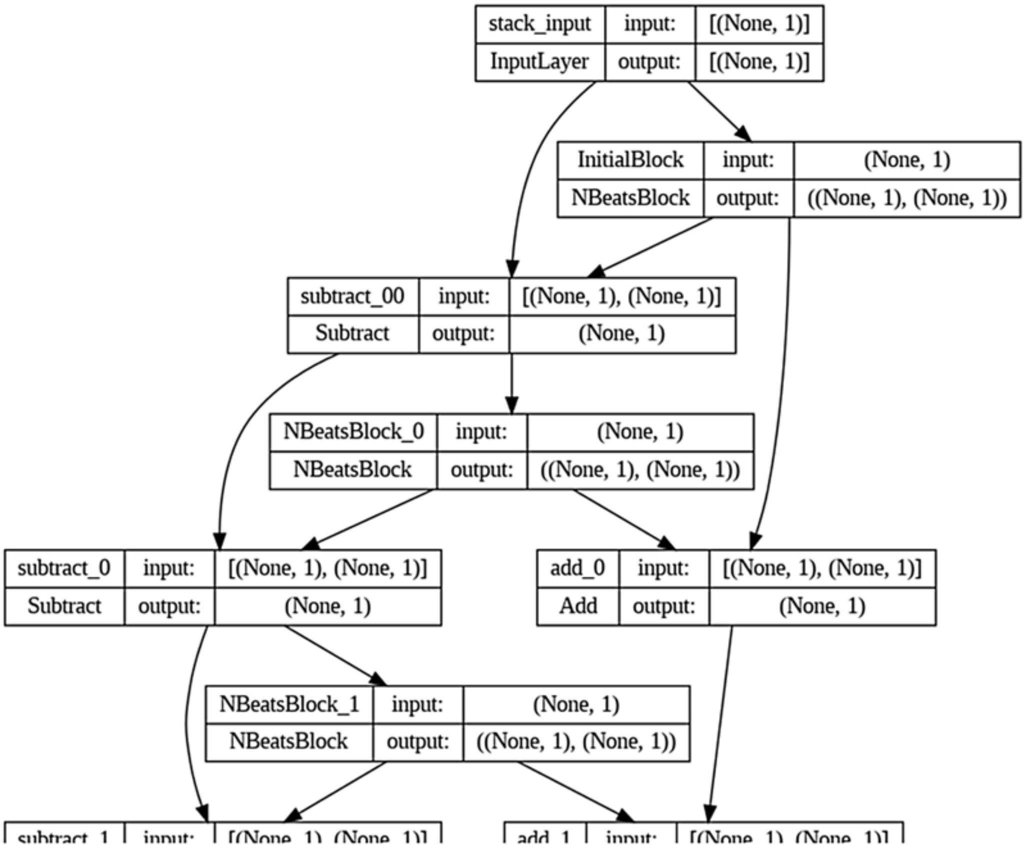
stack_input | input: | [(None, 1)]
InputLayer | output: | [(None, 1)]
InitialBlock | input: | (None, 1)
NBeatsBlock | output: | ((None, 1), (None, 1))
subtract_00 | input: | [(None, 1), (None, 1)]
Subtract | output: | (None, 1)
NBeatsBlock_0 | input: | (None, 1)
NBeatsBlock | output: | ((None, 1), (None, 1))
subtract_0 | input: | [(None, 1), (None, 1)]
Subtract | output: | (None, 1)
add_0 | input: | [(None, 1), (None, 1)]
Add | output: | (None, 1)
NBeatsBlock_1 | input: | (None, 1)
NBeatsBlock | output: | ((None, 1), (None, 1))
subtract_1 | input: | [(None, 1), (None, 1)]
add_1 | input: | [(None, 1), (None, 1)]

Figure 4. Snippet of the final trained N-BEATS model architecture using *Keras visualizer* (output from Google Colab).

In Figure 4, the Keras visualisation tool helps visualise basic building blocks (represented by "NBeatsBlock") showing:

1. Subtraction (represented by "Subtract" in Figure 4) to represent the process of residuals or *backcasts*.
2. Additions (represented by "Add" in Figure 4) to represent the process of *partial forecasts*.

The final trained model (with an input window of 1 day) has over 4 million trainable parameters (i.e. weights and biases). As a comparison, Model 1 from Appendix 3 in Dey (2024), which is a DNN model, has 625 trainable parameters. Similarly, Model 6 from Appendix 3 in Dey (2024), which is an LSTM model, has 47,629 trainable parameters.

3.3.4. Model outputs and observations

The paper Oreshkin *et al.* (2019) mentions that the input length of the window is typically set to a multiple of the length of the output horizon H, between 2H and 7H. For the purpose of this paper, we have trained the model setting this input length to a multiple of one (1 times) to align with Dey (2024), two (2 times) and five (5 times) in order to adhere to the recommendation above and in Oreshkin *et al.* (2019), as well as our analysis in Section 4. The output horizon is as per the initial paper i.e. 1 day.

In Table 3, we compare the performance of the trained N-BEATS models, which have been trained on the same data but on input windows of 1, 2 and 5 days, and with the same output horizon of 1 day.

Table 3. Comparison of performance measure, between the baseline model and N-BEATS, based on MAE and MAPE, varying the input window between 1, 2 and 5 days.

Performance Measure	Baseline (Window = 1 Day) (3 d.p.)	N-BEATS (Window = 1 Day) (3 d.p.)	N-BEATS (Window = 2 Days) (3 d.p.)	N-BEATS (Window = 5 Days) (3 d.p.)
MAE	0.610	0.620	0.662	0.657
Difference from Baseline MAE	–	+0.010	+0.052	+0.047
MAPE	0.497%	0.505%	0.539%	0.536%
Difference from Baseline MAPE	–	+0.008%	+0.042%	+0.039%

Though the N-BEATS models performed relatively accurately, based on the data set, split of data between training/validation/test, and hyperparameter optimisation, there is no improvement to the Baseline for each of our N-BEATS models based on either MAE or MAPE measures on the test data set. The model trained on a 5-day input window performs marginally better than the model trained on a 2-day input window. However, the model trained on a 1-day input window outperforms the N-BEATS models trained on a 2-day and 5-day input window. All trained models perform marginally worse than the Baseline model. Hence, the overall results are inconclusive.

One potential area of future study is to incorporate ensemble techniques as per the original authors of N-BEATS as mentioned in Section 3.3.1, as well as allow for the number of stacks to vary during hyperparameter optimisation when training the model.

4. Widening the Window

4.1. Introduction

We now change the focus of our analysis and widen the input window of data going into our models when training, validating and testing, with the aim of hopefully improving the model accuracy, as we are looking to provide more information to the models to recognise any underlying potential patterns and dependencies within the data. Please note that we revisit the majority of the original neural network architectures discussed in Dey (2024), though have excluded *return sequence true* models as we are predicting 1 day into the future in this paper. We have also excluded XGBoost in our analysis in Section 4, given that this is a decision-tree library and not based on any underlying neural network architecture. We have used various chart functions from the Python-based, open source *statsmodels* library. For more details on this library, please see: https://www.statsmodels.org/.

4.2. Autocorrelation (ACF) and Partial Autocorrelation (PACF) Plots

We used the *seasonal_decompose* function from the *statsmodels* library to separate potential trend lines, seasonality and residual plots across the full data set discussed in Section 2. These can be viewed further in Figure 5. We analysed the daily log returns of the underlying data set to infer potential lags in the data and correlations, and hence to help determine alternative window sizes.

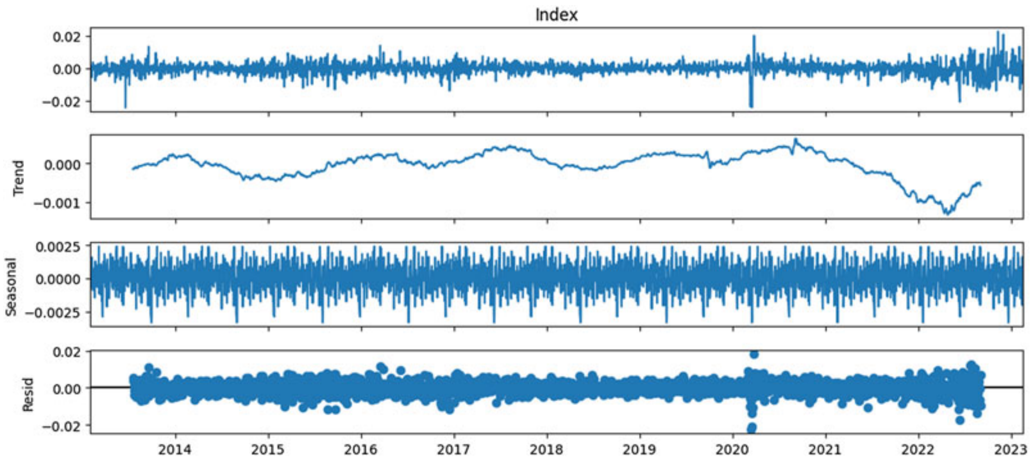

Figure 5. Seasonal, trend and residual plot outputs from the data set (output from Google Colab).

Within Figure 5, “index” represents the daily log returns of the data set, “trend” represents any underlying trend, “seasonal” represents any potential seasonal element and “resid” represents any unaccountable residual (after the trend and seasonal components have been accounted for).

Using the built-in *plot_acf* and *plot_pacf* functions from the *statsmodels* library, we have created an autocorrelation (ACF) plot in Figure 6 and partial autocorrelation plot in Figure 7 respectively.

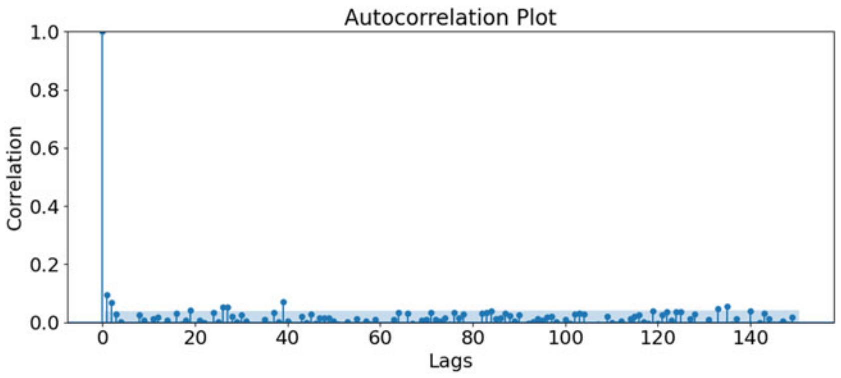

Figure 6. Autocorrelation plot for data set mentioned in Section 2 (output from Google Colab).

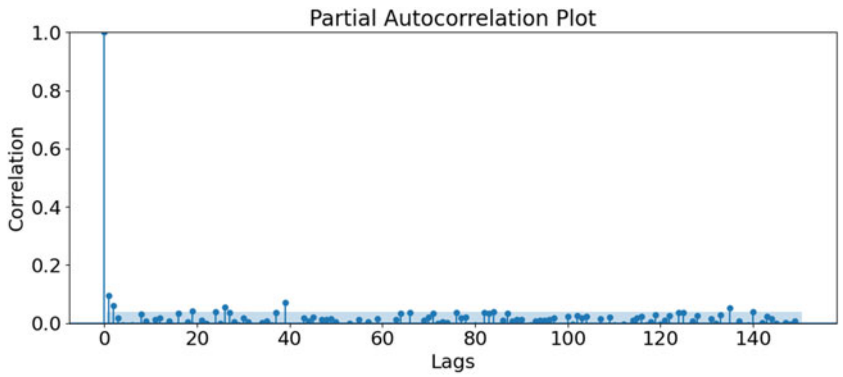

Figure 7. Partial autocorrelation plot for data set mentioned in Section 2 (output from Google Colab).

Both plots can be used to infer a correlation between different lags. For more general background to ACF and PACF plots, please see for example Monigatti (2022).

Though there is a certain amount of subjectivity involved in interpreting Figures 6 and 7, we could expect a 2-day historic lag in data to be representative of any potential underlying feature.

4.3. Window Length to Choose

Based on the analysis in Section 4.2, the implied input window is 2 days. Similarly, the window lag implied by Peters *et al.* (2022) is 5 days. Strictly speaking, these lags are based on the daily log returns. Reversing the log transformation implies a lag of around 7 working days and around 148 working days respectively. The latter time period of 148 days is equivalent to around 6 to 7 months, and may indicate that the prior 2 quarters of annual data incorporate longer term market trend information, and hence may useful when producing better future predictions.

Rerunning all models within this paper based on 7-day and a sample on 148-day input windows resulted in worse performing models, using the same underlying conditions and predicting 1 day ahead, than if the input window was restricted to 5 days. Hence, we have retained input windows of 2 and 5 days, and not extended beyond 5 days. Given that the data is per working day, intuitively 5 days may seem more appropriate. The results from model outputs based on an input window of 7 and 148 days have not been included in this paper given the above.

4.4. Training the Models

We have retrained the neural network models from our initial paper Dey (2024) based on a 2-day input window and 5-day input window. The approach taken is as per the initial paper.

4.5. Results from Analysis of 2-day and 5-day Input Window

We have summarised the MAE and MAPE results from different model runs for some of the model architectures originally analysed. We have maintained the same labelling e.g. model number and abbreviated name. For a full list of these models, please see Appendix 3.

Table 4 compares the performance based on MAE and MAPE for these different models, based on input windows of 1, 2 and 5 days.

Table 4. Comparison performance of neural network models with 1, 2 and 5 days of input window information.

			Data Input					
			Window = 1 Day		Window = 2 Days		Window = 5 Days	
Model Number	Abbreviated Name	Category	MAE (3 d.p.)	MAPE (3 d.p.)	MAE (3 d.p.)	MAPE (3 d.p.)	MAE (3 d.p.)	MAPE (3 d.p.)
0	**Baseline**	**Baseline**	**0.610**	**0.497%**	–	–	–	–
1	DNN0	DNN	0.607	0.495%	0.739	0.602%	0.933	0.763%
2	DNN1		0.627	0.511%	0.706	0.576%	0.948	0.773%
3	DNN2		0.620	0.505%	1.320	1.085%	0.798	0.651%
4	CNN0	CNN	0.619	0.504%	0.686	0.560%	0.992	0.811%
5	CNN1		0.611	0.498%	0.705	0.575%	0.959	0.783%
6	LSTM_0HL_F	LSTM	0.609	0.497%	0.769	0.627%	0.719	0.585%
8	LSTM_1HL_F		0.617	0.503%	0.655	0.534%	0.667	0.545%
10	GRU_0HL_F	GRU	0.632	0.514%	0.680	0.554%	0.647	0.528%
12	GRU_1HL_F		0.638	0.519%	0.623	0.507%	0.643	0.524%

We have retained the original Baseline, which has been trained on an input window of 1 day across all comparison scenarios, given the existing level of accuracy/relative difficulty for other neural networks architecture models to materially outperform this. As can be seen in Table 4, generally the GRU and LSTM models outperform CNN and DNN models, based on both a 2- and 5-day input window.

Table 5 repeats similar information but compares the relative performance of a 2-day and 5-day input window against a 1-day input window for the same model i.e. the comparison is within the same row.

Table 5. Comparison performance 2-day and 5-day input window against window of 1 day for the same model.

			Data input					
			Window = 1 day		Window = 2 days		Window = 5 days	
Model number	Abbreviated name	Category	MAE (3 d.p.)	MAPE (3 d.p.)	MAE (3 d.p.)	MAPE (3 d.p.)	MAE (3 d.p.)	MAPE (3 d.p.)
0	**Baseline**	**Baseline**	**0.610**	**0.497%**	–	–	–	–
1	DNN0	DNN	0.607	0.495%	+0.132	+0.107%	+0.326	+0.268%
2	DNN1		0.627	0.511%	+0.079	+0.065%	+0.321	+0.262%
3	DNN2		0.620	0.505%	+0.700	+0.580%	+0.178	+0.146%
4	CNN0	CNN	0.619	0.504%	+0.067	+0.056%	+0.373	+0.307%
5	CNN1		0.611	0.498%	+0.094	+0.077%	+0.348	+0.285%
6	LSTM_0HL_F	LSTM	0.609	0.497%	+0.160	+0.130%	+0.110	+0.088%
8	LSTM_1HL_F		0.617	0.503%	+0.038	+0.031%	+0.050	+0.042%
10	GRU_0HL_F	GRU	0.632	0.514%	+0.048	+0.040%	+0.015	+0.014%
12	GRU_1HL_F		0.638	0.519%	–0.015	–0.012%	+0.005	+0.005%

The pink colour boxes in Table 5 indicate that the model has performed worse than the corresponding model trained on a window of 1-day input. The yellow colour boxes in Table 5 indicate that the model has performed worse than the corresponding same model architecture trained on a window of 1-day input, but better than the corresponding model architecture trained on a window of 2-day input. The green colour boxes in Table 5 indicate that the model has performed better than the corresponding same model architecture trained on a window of 1-day input.

As can be seen in Table 5, model 12 (a GRU model with 1 hidden layer) outperforms with a 2-day input window compared to 1-day input window. The remaining models perform worse if the input window is expanded from 1 day to 2 or 5 days.

Models 3, 6 and 10 performed better with a 5-day input window versus a 2-day input window. However, as per earlier, the differences are relatively similar and hence the overall results are inconclusive.

Table 6 repeats similar information but compares the relative performance of increasing the data input window to 2 and 5 days respectively, against the Baseline across all scenarios (which was trained with an input window of 1 day).

Table 6. Comparison performance of each neural network model against baseline.

			Datalnput					
			Window = 1 Day		Window = 2 Days		Window = 5 Days	
Model Number	Abbreviated Name	Category	MAE (3 d.p.)	MAPE (3 d.p.)	MAE (3 d.p.)	MAPE (3 d.p.)	MAE (3 d.p.)	MAPE (3 d.p.)
0	**Baseline**	**Baseline**	**0.610**	**0.497%**	–	–	–	–
1	DNN0	DNN	0.607	0.495%	+0.129	+0.105%	+0.323	+0.266%
2	DNN1		0.627	0.511%	+0.096	+0.079%	+0.338	+0.276%
3	DNN2		0.620	0.505%	+0.710	+0.588%	+0.188	+0.154%
4	CNN0	CNN	0.619	0.504%	+0.076	+0.063%	+0.382	+0.314%
5	CNN1		0.611	0.498%	+0.095	+0.078%	+0.349	+0.286%
6	LSTM_0HL_F	LSTM	0.609	0.497%	+0.159	+0.130%	+0.109	+0.088%
8	LSTM_1HL_F		0.617	0.503%	+0.045	+0.037%	+0.057	+0.048%
10	GRU_0HL_F	GRU	0.632	0.514%	+0.070	+0.057%	+0.037	+0.031%
12	GRU_1HL_F		0.638	0.519%	+0.013	+0.010%	+0.033	+0.027%

The colour coding within Table 6 is as per Table 5 but comparing relative performance against the Baseline performance across all scenarios.

Models 3, 6 and 10 performed better with a 5-day input window versus a 2-day input window. However, as per earlier, the differences are relatively similar with all models performing worse than the Baseline. Hence the overall results are inconclusive.

4.6. Conclusions

By increasing the input window from 1 day to 2 and 5 days, on the whole, there were no material improvements to the results. The differences in model performance (based on a MAPE measure) are relatively small and hence the overall results are inconclusive.

The performance of increasing from 1 to 2 or 5 days did however produce relatively accurate models, with a MAPE of circa 0.5% to circa 1.1% across all models which were investigated.

5. Conclusions and Next Steps

5.1. Conclusions

Based on our analysis, given the data range and training/validation/test splits of the *S&P Green Bond Index* and modelling approach discussed earlier, we can draw the following conclusions.

5.1.1. N-BEATS

As observed, we trained a model based on the N-BEATS architecture in Section 3. The outputs of each model (trained on an input window of 1, 2 and 5 days) did produce output forecasts for a 1-day horizon (i.e. the next day) which were relatively accurate, with the MAPE differing by approximately +0.5% over the test range data set.

However, the trained models did not outperform the Baseline model over the test data set. The trained N-BEATS models differed from the Baseline results by up to about +0.04% i.e. our findings are inconclusive. This may be indicative of the fact that the current Baseline is actually a good predictor in the first place, given that the relatively low coefficient of variation of 7.1% for the index across the full data set. Similarly, it may be indicative of the fact that the problem we are facing with the *S&P Green Bond Index* is as per any other stock price movement i.e. a random walk, where it is difficult to continually outperform *any* baseline model (Bourke, 2023).

5.1.2. 2-day lag and 5-day lag

Expanding the input window from 1 day to 2 and 5 days respectively for the neural network architectures explored in Dey (2024) did not result in an improvement, with the performance relatively similar to the Baseline i.e. our findings are still inconclusive.

5.2. Next Steps

The analysis to date has been based on a univariate analysis and simpler neural network architectures – such as DNNs, CNNs, LSTMs and GRUs, as discussed in Dey (2024). We have further expanded this to a complex and cutting-edge architecture with N-BEATS.

For future papers, other potential areas we may explore include:

1. Explore other complex neural network architectures based on a univariate analysis e.g. Google’s TFT (Temporal Fusion Transformer) and temporal GNNs (Graph Neural Networks).
2. Expanding our analysis to a multivariate analysis.
3. Broaden the projection horizon from 1 day to 1 week or possibly 1 month into the future.

References

Bourke, D. (2023). Welcome to the zero to mastery TensorFlow for deep learning book, available at https://dev.mrdbourke.com/tensorflow-deep-learning/ (accessed 14 August 2024).

Dancker, J. (2024). N-BEATS – the first interpretable deep learning model that worked for time series forecasting, available at https://towardsdatascience.com/n-beats-the-first-interpretable-deep-learning-model-that-worked-for-time-series-forecasting-06920daadac2 (accessed 14 August 2024).

Dey, D. (2023). Time series analysis of S&P green bond index, available at https://blog.actuaries.org.uk/time-series-analysis-of-s-p-green-bond-index/ (accessed 14 August 2024).
Dey, D. (2024). Time series analysis of GSS bonds. *British Actuarial Journal*, **29**(1), https://doi.org/10.1017/S1357321724000011 (accessed 14 August 2024).
Makridakis, S., Spiliotis, E. & Assimakopoulos, V. (2020). The M4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, **36**(2020), 54–74, https://doi.org/10.1016/j.ijforecast.2019.04.014 (accessed 14 August 2024).
Monigatti, L. (2022). Interpreting ACF and PACF plots for time series forecasting, available at https://towardsdatascience.com/interpreting-acf-and-pacf-plots-for-time-series-forecasting-af0d6db4061c (accessed 14 August 2024).
Oreshkin, B.N., Carpov, D., Chapados, N. & Bengio, Y. (2019). N-BEATS: neural basis expansion analysis for interpretable time series forecasting. *arXiv preprint*, arXiv:1905.10437, available at https://arxiv.org/pdf/1905.10437 (accessed 14 August 2024).
Peters, G., Zhu, R., Tzougas, G., Rabitti, G. & Yusuf, I. (2022). The role and significance of green bonds in funding transition to a low carbon economy: a case study forecasting portfolios of green bond instrument returns. *SSRN Electronic Journal*, 4299196, available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id = 4299196 (accessed 14 August 2024).
Wang, J., Tang, J. & Guo, K. (2021). Green bond index prediction based on CEEMDAN-LSTM. *Frontiers in Energy Research*, **9**(1), https://doi.org/10.3389/fenrg.2021.793413 (accessed 14 August 2024).

Appendix 1 – Working Party Members and Acknowledgements

Working Party members

The Working Party is made up of two sub-groups: practitioners and academics.

Below is a list of current Working Party members for the Practitioner Group, along with details of their position within the Working Party and corresponding LinkedIn link.

Name	Position	LinkedIn
Debashish Dey	Chair	https://url.avanan.click/v2/___https:/uk.linkedin.com/in/debashish-dey-669025a3___.YXAxZTpjYW1icmlkZ2Vvcmc6YTpvOjhiM2VjMTNmMjZlYTM3YmZiY2ExYWE3OTJlNWQ0OGQ4OjY6OGVkNTo2ZWFjYmRkYzA2MjI1ZGMzNmM3NWJkYzk4NTliNDgzNzU3OWFiNjJiNjU5YmFmMGRkNDE1ZmJiNTdjMTBmZjJmOnA6VDpG
Cem Öztürk	Member	https://url.avanan.click/v2/___http:/linkedin.com/in/ozturkcemm___.YXAxZTpjYW1icmlkZ2Vvcmc6YTpvOjhiM2VjMTNmMjZlYTM3YmZiY2ExYWE3OTJlNWQ0OGQ4OjY6M2NjODo3N2QyYmRmZjAzNGY3Nzg4NTZiMDMwMjYyYmEwYTdkZTFlOGRkZjk3NmQwYzFlODU1OGI2Yjk2OWEzNzIzYjdmOnA6VDpG
Shubham Mehta	Member	https://url.avanan.click/v2/___http:/linkedin.com/in/mehta-shubham___.YXAxZTpjYW1icmlkZ2Vvcmc6YTpvOjhiM2VjMTNmMjZlYTM3YmZiY2ExYWE3OTJlNWQ0OGQ4OjY6M2RkNDplYjVlMTUzMjhjMTc2YTMxOTEyOGNlNjgwZTExZDg4ZTUyNGIxZDg0MjliOWI1YjgwNjNkOWMwZjIzMjAyMWM3OnA6VDpG

Please note that this Working Party sits within the Lifelong Learning pillar of the IFoA. For further details of the Data Science section of the IFoA, please see https://actuaries.org.uk/learn/lifelong-learning.

Acknowledgements

Please see below acknowledgements for the review of the code and paper:

- Code review by Cem Öztürk [https://url.avanan.click/v2/___http:/linkedin.com/in/ozturkcemm___.YXAxZTpjYW1icmlkZ2Vvcmc6YTpvOjhiM2VjMTNmMjZlYTM3YmZiY2ExYWE3OTJlNWQ0OGQ4OjY6M2NjODo3N2QyYmRmZjAzNGY3Nzg4NTZiMDMwMjYyYmEwYRkZjk3NmQwYzFTdkZTFlOGlODU1OGI2Yjk2OWEzNzIzYjdmOnA6VDpG].
- Technical review of paper by:
  - Dr Alexey Mashechkin [https://url.avanan.click/v2/___https:/www.linkedin.com/in/alexey-mashechkin/___.YXAxZTpjYW1icmlkZ2Vvcmc6YTpvOjhiM2VjMTNmMjZlYTM3YmZiY2ExYWE3OTJlNWQ0OGQ4OjY6ZjQ0NTplYTBmMWJkNGQ4OTdhM2VkMjMzZjRmOWRiYjM2ZjdiMTZjZDEyYTNhNTM1OTc3NTU5YWY2OThiYjUyYmEwZDAzOnA6VDpG].

  - Assoc. Prof. George Tzougas [https://url.avanan.click/v2/___https:/www.linkedin.com/in/george-tzougas-648711294/___.YXAxZTpjYW1icmlkZ2Vvcmc6YTpvOjhiM2VjMTNmMjZlYTM3YmZiY2ExYWE3OTJlNWQ0OGQ4OjY6YzkxNDpkZDU5NmU0NWFjZTNhMDI4MjQ1ZWM3MTRlMmJmYzFhM2QzZWQ4ZGI0ZDBmM2M2MzMxNWEyMmE0NDU4MGM4MTFkOnA6VDpG].
- Review of graphs and tables within the paper by Shubham Mehta [https://url.avanan.click/v2/___http:/linkedin.com/in/mehta-shubham___.YXAxZTpjYW1icmlkZ2Vvcmc6YTpvOjhiM2VjMTNmMjZlYTM3YmZiY2ExYWE3OTJlNWQ0OGQ4OjY6M2RkNDplYjVlMTUzMjhjMTc2YTMxOTEyOGNlNjgwZTExZDg4ZTUyNGIxZDg0MjliOWI1YjgwNjNkOWMwZjIzMjAyMWM3OnA6VDpG].

Appendix 2 – List of Abbreviations

Below is a list of abbreviations used within this paper.

Abbreviation	Explanation
ARIMA	AutoRegressive Integrated Moving Average
CEEMDAN	Complete Ensemble Empirical Mode Decomposition with Adaptive Noise
CNN	Convolutional Neural Network
DNN	Deep Neural Network
GNN	Graph Neural Network
GRU	Gated Recurrent Unit
GSS bonds	Green, Social and Sustainability bonds
IFoA	Institute and Faculty of Actuaries
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MSCI	Morgan Stanley Capital International
N-BEATS	Neural Basis Expansion Analysis for interpretable Time Series
RNN	Recurrent Neural Network
S&P	Standard and Poor’s
TFT	Temporal Fusion Transformer
TPE	Tree-structure Parzen Estimator

Appendix 3 – Summary of Models Analysed in the Initial Paper

Below is a summary of the models analysed in the initial paper Dey (2024), with a brief description of the underlying model architecture for each.

Model	**Abbreviated name**	**Category**	**Description of architecture**
0	Baseline	Baseline	Baseline model which assumes today’s value is the same as yesterday’s value.
1	DNN0	DNN	Feedforward artificial neural network with one hidden dense layer.
2	DNN1		Feedforward artificial neural network with two hidden dense layers.
3	DNN2		Feedforward artificial neural network with three hidden dense layers.
4	CNN0	CNN	Convolutional neural network, with one Conv1D layer and no additional hidden layers.
5	CNN1		Convolutional neural network, with one Conv1D layer and one hidden dense layer.

6	LSTM_0HL_F	LSTM	LSTM neural network, with one LSTM layer, return sequence set to false, and no additional hidden layers.
7	LSTM_0HL_T		LSTM neural network, with one LSTM layer, return sequence set to true, and no additional hidden layers.
8	LSTM_1HL_F		LSTM neural network, with one LSTM layer, return sequence set to false, and one additional hidden dense layer.
9	LSTM_1HL_T		LSTM neural network, with one LSTM layer, return sequence set to true, and one additional hidden dense layer.
10	GRU_0HL_F	GRU	GRU neural network, with one GRU layer, return sequence set to false, and no additional hidden layers.
11	GRU_0HL_T		GRU neural network, with one GRU layer, return sequence set to true, and no additional hidden dense layers.
12	GRU_1HL_F		GRU neural network, with one GRU layer, return sequence set to false, and one additional hidden dense layer.
13	GRU_1HL_T		GRU neural network, with one GRU layer, return sequence set to true, and one additional hidden dense layer.
14	XGBoost	XGBoost	XGBoost model. Hyperparameters analysed are described earlier in this paper.

Appendix 4 – Useful links

- IBM's introductory series on neural networks: https://www.ibm.com/topics/neural-networks
- IFoA's Certificate in Data Science programme: https://www.actuaries.org.uk/news-and-insights/news/data-science-credential
- IFoA's Data Science Lifelong Learning page: https://actuaries.org.uk/learn/lifelong-learning/data-science/
- Google Tensorflow's tutorial on time series: https://www.tensorflow.org/tutorials/structured_data/time_series

Cite this article: Dey D. (2024). Time series analysis of GSS bonds Part 2 – further univariate analysis of S&P Green Bond Index. *British Actuarial Journal*. https://doi.org/10.1017/S1357321724000217