# GAS-Norm: Score-Driven Adaptive Normalization for Non-Stationary Time Series Forecasting in Deep Learning

Edoardo Urettini*
University of Pisa
Scuola Normale Superiore
Pisa, Italy
edoardo.urettini@sns.it

Daniele Atzeni*
University of Pisa
Pisa, Italy
daniele.atzeni@phd.unipi.it

Reshawn J. Ramjattan*
University of Pisa
Pisa, Italy
reshawn.ramjattan@phd.unipi.it

Antonio Carta*
University of Pisa
Pisa, Italy
antonio.carta@unipi.it

## Abstract

Despite their popularity, deep neural networks (DNNs) applied to time series forecasting often fail to beat simpler statistical models. One of the main causes of this suboptimal performance is the data non-stationarity present in many processes. In particular, changes in the mean and variance of the input data can disrupt the predictive capability of a DNN. In this paper, we first show how DNN forecasting models fail in simple non-stationary settings. We then introduce GAS-Norm, a novel methodology for adaptive time series normalization and forecasting based on the combination of a Generalized Autoregressive Score (GAS) model and a Deep Neural Network. The GAS approach encompasses a score-driven family of models that estimate the mean and variance at each new observation, providing updated statistics to normalize the input data of the deep model. The output of the DNN is eventually denormalized using the statistics forecasted by the GAS model, resulting in a hybrid approach that leverages the strengths of both statistical modeling and deep learning. The adaptive normalization improves the performance of the model in non-stationary settings. The proposed approach is model-agnostic and can be applied to any DNN forecasting model. To empirically validate our proposal, we first compare GAS-Norm with other state-of-the-art normalization methods. We then combine it with state-of-the-art DNN forecasting models and test them on real-world datasets from the Monash open-access forecasting repository. Results show that deep forecasting models improve their performance in 21 out of 25 settings when combined with GAS-Norm compared to other normalization methods.

## CCS Concepts

• **Computing methodologies → Machine learning**.

## Keywords

Time series Forecasting, Input Normalization, Deep Learning

**ACM Reference Format:**
Edoardo Urettini, Daniele Atzeni, Reshawn J. Ramjattan, and Antonio Carta. 2024. GAS-Norm: Score-Driven Adaptive Normalization for Non-Stationary Time Series Forecasting in Deep Learning. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24), October 21–25, 2024, Boise, ID, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3627673.3679822

## 1 Introduction

Time series forecasting has played a crucial role in decision-making and planning, becoming prevalent in a variety of real-world scenarios, such as economics [35], health care [26], and energy consumption planning [9]. Following the exciting results in natural language processing [28] and computer vision [13], nowadays deep neural networks (DNNs) have been applied to forecasting problems [34].

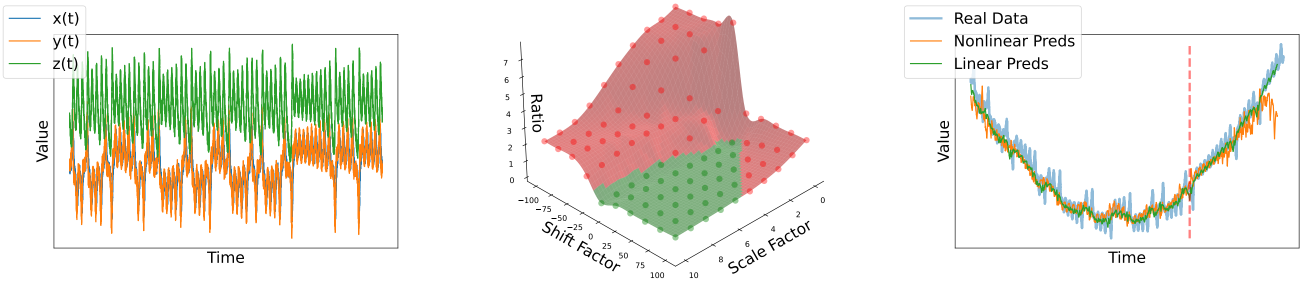
However, contrary to what occurs in other domains, DNNs do not seem to excel in time series forecasting. In this area, they achieve forecasting capabilities that are often comparable to classic statistical models [12]. Among the most accredited explanations of this behavior, researchers identified signal-to-noise ratio and non-stationarity of the input data as likely causes [16, 20]. Indeed, out-of-distribution data can severely impact the outputs of strongly nonlinear machine learning models, which are easily subject to overfitting and performance degradation when input data changes in scale [30].

To avoid this, input normalization became a standard practice in deep learning. It also proved to be useful in boosting the optimization and robustness of the models by having all input features on the same scale [37]. Common estimation of input distribution parameters (e.g., mean and variance) on the training set is enough to ensure normalization if the input is stationary. However, for most time series scenarios, data usually comes from a non-stationary environment. In that case, the input distribution can change over time, requiring a different normalization approach. Given this distribution change, the normalization of the input has the additional duty of generalizing the model knowledge to the new input distribution [19].

In this work, we propose GAS-Norm, a novel normalizing approach that combines DNNs and Generalized Autoregressive Score

[*] All authors contributed equally to the paper

*CIKM '24, October 21–25, 2024, Boise, ID, USA*



(a)

(b)

(c)

**Figure 1: (a) The evolving coordinates of a 3D Lorenz attractor. (b) The ratio between the MSE of the nonlinear MLP and the MSE of the linear model. The green area is where the nonlinear model is better. (c) Prediction of the 500-steps ahead value done by the linear and the nonlinear model. The vertical line shows the end of the training data.**

(GAS) models, a class of statistical autoregressive models developed to handle time series data with time-varying parameters [8]. Thanks to GAS models' autoregressive nature and a training phase independent of the deep model, our normalizing module can be used in combination with a wide variety of deep forecasting architectures, whether they use autoregressive prediction or not, and with input and output time series of any length. To control the normalization strength we extend the GAS formulation with an additional parameter that allows us to set the update speed of the model. Thus, we can control how much variability is present in the data the deep model must handle during forecasting. The final output of the model combines the DNN outputs and the statistics predicted by the GAS model with a denormalization step.

We test our method against state-of-the-art dynamic normalization techniques and in combination with state-of-the-art forecasting models. The testing is done using 7 synthetic and real-world benchmarks. Using GAS-Norm improves the results of deep forecasting methods across 21 out of 25 configurations of datasets and deep learning architectures. The code for this implementation and evaluation is public and open source[1].

## 2 Related Works

A variety of deep neural network architectures have been adapted to solve forecasting problems, from recurrent autoregressive networks like DeepAR [32], to convolutional models such as WaveNet [39]. More recently, transformers have also been used for forecasting [43]. However, results in the literature including the Monash repository [12], suggest that DNNs can still lag behind classic methods in simple forecasting benchmarks. We hypothesize that this gap can be reduced by improving the DNNs' input data distribution stability.

To boost performance on time series data, multiple attempts combined classic methods and neural networks, trying to overcome the weaknesses of both approaches. In [42], authors propose a hybrid model made of an ARIMA and a neural network component. The M4 competition winner, the ES-RNN [36], combines exponential smoothing and recurrent networks. Similarly, GAS-Norm can be seen as a hybrid model that combines the GAS approach with a neural forecasting one.

Among the proposed statistical models for non-stationary data, Generalized Autoregressive Score models [8] are a class used for filtering time-varying parameters that update the estimation with the observed score. Their Information-Theoretic optimality has been shown with respect to the Kullback-Leibler divergence between the true and the estimated density function [5]. One example of a GAS model is the GARCH model, widely used to filter time-varying variance in time series data [6].

Recently, different studies have tried to address challenges caused by non-stationary time series combining neural forecasting models and dynamic normalization procedures. Initial works focused only on the input normalization [27]. [23] proposes a batch normalization method for domain adaptation. DAIN [29] learns the normalization with a nonlinear network, while [10] normalizes the input both in time and frequency. Unlike our proposal, these methods ignore non-stationarity over time within the input time series. RevIN [19] introduced a denormalization step to restore the statistics removed during the normalization step. Similarly, [11] adopted a normalization methodology combined with a denormalization step. Despite this work considering also intra-space shift, i.e., non-stationarity between input and output time series, it adopts fixed statistics for the forecast. Finally, SAN [24] proposed a dynamic normalization approach that splits both input and output into shorter temporal slices, in which non-stationarity can be less impactful, and uses them to estimate means and variances. Unlike SAN, our method adapts the statistics online and avoids possible problems caused by slices that are too long, like non-stationarity, or too short, such as overly noisy estimations.

## 3 Proposed Method

### 3.1 Why Adaptive Normalization for a DNN?

DNNs are a composition of many nonlinear functions, where the output of a layer becomes the input of the next one. Given the complex nonlinear structure of these models, the importance of the stability of the input distribution of each layer has been extensively explored in the literature [18, 21], showing improvements in training speed, stability, and generalization of DNNs.

[1] https://github.com/edo-urettini/GAS_Norm
https://github.com/daniele-atzeni/GAS-Norm-gluonTS

In this paper, we focus on improving the generalization of the nonlinear forecasting model in non-stationary settings. While DNNs can approximate any function [25] (with some regularity conditions), their general definition also creates the risk of overfitting the observed data without learning the real generating function, particularly if the training data is noisy or does not fully represent the possible input space. Addressing this shortcoming is what our focus on generalization refers to, in the context of this paper.

Using a simple example, we demonstrate that a complex nonlinear model can be less robust to changes in the input distribution than a linear model when applied to time series data. We generate a 3D Lorenz attractor as the solution of a Lorenz system with specified initial conditions, resulting in three chaotic time series representing the Cartesian coordinates of the system's evolution over time. Subsequently, we introduce a minimal amount of noise to these time series (see Figure 1a).

We split the data into training, validation, and test sets, and train a simple 3-layer ReLU MLP to predict the future evolution of one coordinate based on the recent past of all three coordinates. This MLP is then compared to the same MLP using only linear activations, which is a linear model. Both models are tested on the same data shifted and rescaled with an affine transformation. Figure 1b shows the ratio between the test MSE of the nonlinear MLP and the MSE of the linear model. The plot reveals a "generalization area" (green area) where the nonlinear model outperforms the linear one. In this limited region, the nonlinear model remains superior even when the input has been shifted and rescaled. Outside of this region, the nonlinear model's performance rapidly degrades compared to the linear model (red area). For instance, when the data is shifted downward, the linear model significantly outperforms the ReLU MLP in predicting the chaotic time series.

In time series, changes in the mean and variance of the process are common and can be either predictable (like a simple linear trend) or unpredictable (like random regime changes)[14]. In both cases, many approaches have been proposed, given the frequency of these kinds of problems. While our previous example shows the difficulties of a nonlinear model in case of an unexpected change in the location and scale of the data, limitations in learning can also arise due to predictable and deterministic trends. To illustrate this, we try to solve the same problem as in the previous example, but this time we add a quadratic trend to our time series.

In this case, the nonlinear model is not able to learn the deterministic trend, resulting in an error higher than that of the linear model when the input data exits from the known distribution (Figure 1c).

To solve these problems we need an adaptive and flexible way to filter the location and scale parameters of our data online. This would allow us to normalize the data even in the presence of deterministic or unpredictable non-stationarity.

## 3.2 Parameter Filtering for Non-stationary Time Series

In this section, we define the theoretical framework of our filtering method. We set the forecasting problem as a conditional expectation problem where we predict, at time $t$, the expectation of a random variable $Y_{t+h}$ given the input $\{x_t, ..., x_{t-l}\}$, the ordered realizations of the random variables $\{X_t, ..., X_{t-l}\}$. All random variables can be multivariate. In common use, $Y_{t+h}$ and $\{X_t, ..., X_{t-l}\}$ are instances of the same stochastic process, where the first represents the future of the series and the second the past. It is also possible to use realizations from a process different from what we are predicting as input. The target of the forecasting problem is then:

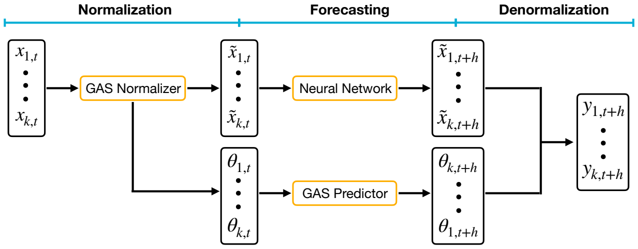


**Figure 2: GAS-Norm procedure for a single time step input. Statistical parameters $\theta$ in this case are the means and variances.**

$$E[Y_{t+h}|X_t, ..., X_{t-l}; w] = f_w(X_t, ..., X_{t-l})$$

The conditional expectation itself is a random variable that depends on the realizations of $\{X_t, ..., X_{t-l}\}$. The function $f_w$ is an arbitrary function parametrized by $w$. In deep learning, we try to approximate this function with a neural network $\hat{f}_w$, a strongly nonlinear model that is generally able to approximate any function [15].

The input distribution at time $t$ of a $k$-dimensional input vector is an unknown joint distribution of all input features, denoted as $P_{X_t}(x_t) = P_{X_{1,t},X_{2,t},...X_{k,t}}(x_{1,t}, x_{2,t}, ...x_{k,t})$, with mean vector $\mu_t = [\mu_{1,t}, \mu_{2,t}, ...\mu_{k,t}]$ and covariance matrix:

$$\Sigma_t = \begin{bmatrix} \sigma^2_{1,t} & Cov(x_{1,t}, x_{2,t}) & ... \\ Cov(x_{1,t}, x_{2,t}) & \sigma^2_{2,t} & ... \\ ... & ... & \sigma^2_{k,t} \end{bmatrix}$$

The first index is the feature index. In the stationary case, the mean and the variance of the input distribution would be constant. In our non-stationary setting, they are both allowed to change in time.

While keeping the input joint distribution unknown, we assume to know the type of parametrized density function of the marginal distributions of each input feature, conditional to the past observations of that feature itself. As an example, we can assume that the marginal distribution of each input feature is a Gaussian distribution with time-varying mean and variance:

$$P_{X_{i,t}|\mathcal{F}_{i,t}}(x_{i,t}) = \mathcal{N}(\mu_{i,t}, \sigma^2_{i,t})$$

where $\mathcal{F}_{i,t}$ is the collection of observations of the feature $x_i$ that we have at time $t$.

To normalize each feature we need to filter the time-varying mean and variance. Given our parametric choice, we set our problem with an observation-driven state-space representation where the realizations of our input feature $x_{i,t}$ are given by:

$$x_{i,t} = \mu_{i,t} + \sigma_{i,t}\epsilon_{i,t}$$
$$\mu_{i,t} = g(\mu_{i,t-1}, x_{i,t-1})$$
$$\sigma^2_{i,t} = g'(\sigma^2_{i,t-1}, x_{i,t-1}),$$

meaning that our input feature at time $t$ is generated by a random process with time-varying mean and variance and a random noise $\epsilon_{i,t}$ sampled from our assumed distribution. The value of the current states depends on their previous value and on the previous observation of the series. To ease the notation, we use $\theta_t = [\mu_t, \sigma_t^2]$ as our time-varying parameter vector and we drop the feature index $i$. To solve the filtering problem, we need to find the best update for the parameter that maximizes the likelihood of the realizations, while penalizing changes that are too quick with respect to past predictions (as done in the Kalman Filter [4]). In other words, we need a balance between the stability of the parameters and the update speed. As we will show, this is the problem GAS models try to solve. We modify the GAS formulation in [22] by adding a new hyperparameter $\gamma \in [0, 1)$ to control how much importance is given to maximizing the likelihood and how much is given to keeping the parameter stable. This hyperparameter explicitly controls the update speed of the normalization parameters, controlling the normalization strength of our method. Low values of $\gamma$ will result in a slower adaptation, leaving the normalized input more similar to the original one. At the extreme, $\gamma = 0$ is equivalent to a static normalization. The problem is written as:

$$\max_{\theta} \ \gamma \ \log p(x_t|\theta) - \frac{1-\gamma}{2}||\theta - \theta_{t|t-1}||^2_{P_t}$$

where $P_t$ is a penalization matrix and $\theta_{t|t-1}$ is the prediction of the current parameter done at the previous time step. To compute explicitly the gradient, we approximate the problem with first-order Taylor expansion:

$$\log p(x_t|\theta) = \log p(x_t|\theta_{t|t-1}) + (\theta - \theta_{t|t-1})\nabla_\theta(x_t|\theta_{t|t-1}).$$

The first-order condition to solve the maximization problem is:

$$\theta_{t|t} = \theta_{t|t-1} + \frac{\gamma}{1-\gamma} P_t^{-1} \ \nabla_\theta(x_t|\theta_{t|t-1}).$$

The penalization matrix proposed by [8] is the Fisher Information Matrix (FIM), rescaled by a vector $\alpha^{-1}$ that works as a learning rate. The FIM is defined as the variance of the score and represents the expected curvature of the log-likelihood with respect to each parameter. By regularizing the score with the FIM, we obtain an online learning algorithm, where the parameter is updated at each new observation following the direction of the natural gradient $\tilde{\nabla}$. The natural gradient [2] is the score adapted to the expected curvature of the log-likelihood. The final update step at each new observation is:

$$\theta_{t|t} = \theta_{t|t-1} + \frac{\gamma}{1-\gamma} \alpha \ \tilde{\nabla}_\theta(x_t|\theta_{t|t-1}) \tag{1}$$

Finally, we add a simple linear prediction step guided by two parameters: $\omega$, which represents the unconditional mean of the parameters, and $\beta$, which controls the mean-reversion:

$$\theta_{t+1|t} = \omega + \beta \ \theta_{t|t}.$$

The whole statistics update process extends that of [22] by including an additional hyperparameter $\gamma$. $\gamma$ can be selected with hyperparameters tuning techniques while the other parameters ($\alpha$, $\beta$, $\omega$) are directly optimized on the training data. To optimize these static parameters, we proceed with the prediction error decomposition where the time series of the single feature can be represented as:

$$p(x_1, x_2, ..., x_T) = p(x_T|x_{T-1})...p(x_2|x_1)p(x_1).$$

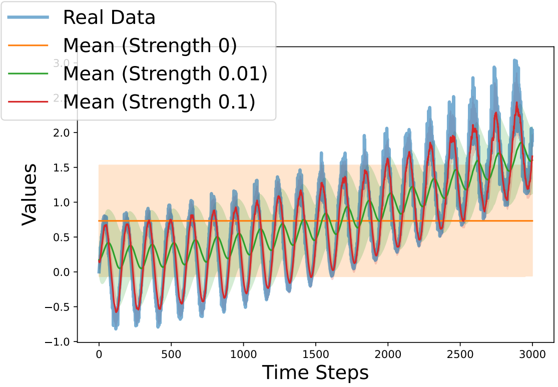


**Figure 3: Comparison of the filtering process of GAS-Norm with different normalization strengths ($\gamma$). The shaded area is the mean ± standard deviation.**

Maximizing the logarithm of Eq. 3.2 and adding a penalization term, we obtain a new optimization problem as a function of the static parameters:

$$\begin{aligned} max_{\alpha,\omega,\beta} \ \gamma \ \log p(x_1) - \frac{1-\gamma}{2}||\theta_1 - \theta_0||^2_{P_t} + \\ \sum_{t=2}^{T} \gamma \ \log p(x_t|x_{t-1}) - \frac{1-\gamma}{2}||\theta_t - \theta_{t|t-1}||^2_{P_t}. \end{aligned} \tag{2}$$

Once we solve the optimization problem for each feature and we obtain the optimal $\alpha, \beta, \gamma$, we can use them to filter the mean and variance that will be used to normalize the marginal distribution of each feature at each time step. If the parameters are correctly filtered, after the normalization each marginal distribution has zero mean and unitary variance, and the joint input distribution becomes:

$$P_{\tilde{X}_{1,t}, \tilde{X}_{2,t}, ...\tilde{X}_{k,t}}(\tilde{x}_{1,t}, \tilde{x}_{2,t}, ...\tilde{x}_{k,t}) =$$
$$p([0, 0, ...0], \begin{bmatrix} 1 & Corr(x_{1,t}, x_{2,t}) & ... \\ Corr(x_{1,t}, x_{2,t}) & 1 & ... \\ ... & ... & 1 \end{bmatrix})$$

an unknown distribution $p$ with mean zero and a covariance matrix equal to the correlation matrix of the original joint distribution. The *tilde* symbol is used to denote the normalized feature. The normalized features can be used as input for the DNN model.

## 3.3 GAS-Norm

The GAS-Norm procedure proposed in this work is summarized in Figure 2, where different blocks show the normalization, forecasting, and denormalization phases, with input and output data for each of them. Formally, our forecasting model of the feature $y_{i,t+h}$ at a future time $t + h$ can be described as:

$$\begin{aligned} y_{i,t+h} &= \mu_{i,t+h} + \sigma_{i,t+h} e_{i,t+h} \\ \mu_{i,t+h} &= g(\mu_{i,t+h-1}) \\ \sigma^2_{i,t+h} &= g'(\sigma^2_{i,t+h-1}) \\ e_{i,t+h} &= f_w(\tilde{X}_t, ..., \tilde{X}_{t-l}) + \epsilon_{i,t+h}, \end{aligned} \tag{3}$$

where $e_{i,t+h}$ is now a deterministic residual and $\epsilon_{i,t+h}$ is random noise of unknown distribution[2]. The target of our forecasting problem, i.e., the expected value $E[Y_{t+h}|X_t, ..., X_{t-l}; w]$, is now the result of combining the independent prediction of the mean and variance process by GAS model, and the prediction done by the deep model using the normalized residuals. In this way, the DNN has the role of learning what the simple mean and variance predictions cannot forecast, effectively resulting in a **residual learning** procedure. Details of each step of our procedure are described in the following.

**Normalization.** The normalization step is performed online on the input data (the portion of the input sequence used by the encoder). The GAS model can be adapted to different distributional assumptions. We use a Student's t-distribution for our experiments. According to this choice, equations for mean and variance estimation (see the Appendix for the derivation) are computed as:

$$\begin{aligned}\mu_{t+1} =&\omega_\mu + \beta_\mu [\frac{\gamma}{1-\gamma}\alpha_\mu \frac{y_t - \mu_t}{1+\frac{(y_t-\mu_t)^2}{\nu\sigma_t^2}} + \mu_t] \\ \sigma_{t+1}^2 =&\omega_\sigma + \beta_\mu [\frac{\gamma}{1-\gamma}\alpha_\sigma (\frac{(\nu+1)(y_t-\mu_t)^2}{\nu + \frac{(y_t-\mu_t)^2}{\sigma_t^2}} - \sigma_t^2) + \sigma_t^2]\end{aligned} \tag{4}$$

where, again, we removed the subscript $i$ from each variable to ease the notation. As introduced in section 3.2, the $\alpha, \beta$ and $\omega$ parameters result from the feature-wise optimization problem described in equation 2. The best values of these parameters are found for each feature of each time series of the training dataset, then they are left frozen during the training phase of the DNN. Parameter $\nu$, which reflects the fatness of the tails of the Student's t-distribution, can be either optimized as the others or fixed. For large values of $\nu$, the Student's t converges to a Gaussian distribution. Finally, $\gamma$ is a hyperparameter selected with a validation set that controls the stability of the means and variances. It controls the amount of information about the original mean and variance the filtering process removes from the observed input sequence. In the rest of the paper, we will refer to this parameter also as normalization strength. The effect of different values of $\gamma$ can be seen in Figure 3. Notice how, when setting the value to 0, our method collapses into a simple static normalization procedure, where the initial mean and variance can be set as the unconditional mean and variance of the whole training set.

This procedure for the online update of means and variances strives towards making the method more model-agnostic, while still providing stationary input to the deep model.

**Forecasting and Denormalization.** The normalized input time series is used by the deep model to make the forecast, leveraging the unfiltered linear and nonlinear correlations. At the same time, the GAS model is used to forecast future means and variances. In this case, the time series predicted by the models need to be part of the input features (the GAS model can only predict the same series it has filtered). We obtain the per-feature forecasting equation by modifying equations 4 substituting the observation with the last prediction of the mean and the variance:

$$\begin{aligned}\mu_{t+1} &= \omega_\mu + \beta_\mu \ \mu_t \\ \sigma_{t+1}^2 &= \omega_\sigma + \beta_\sigma \sigma_t^2.\end{aligned}$$

This is a kind of autoregressive approach: the prediction is assumed as real observation to predict the next step. Finally, the forecast made by the deep model is combined with the means and variances predicted by the GAS, hence re-introducing the information removed during the normalization procedure as in equation 3. During the training phase of the deep model, the final output is then used to compute the loss and update the DNN's parameters.

## 4 Experiments with Normalization Methods

In this section, we will focus on the comparison between the GAS-Norm and other normalization methods with data of different characteristics. We will begin with a qualitative comparison, followed by a quantitative experiment.

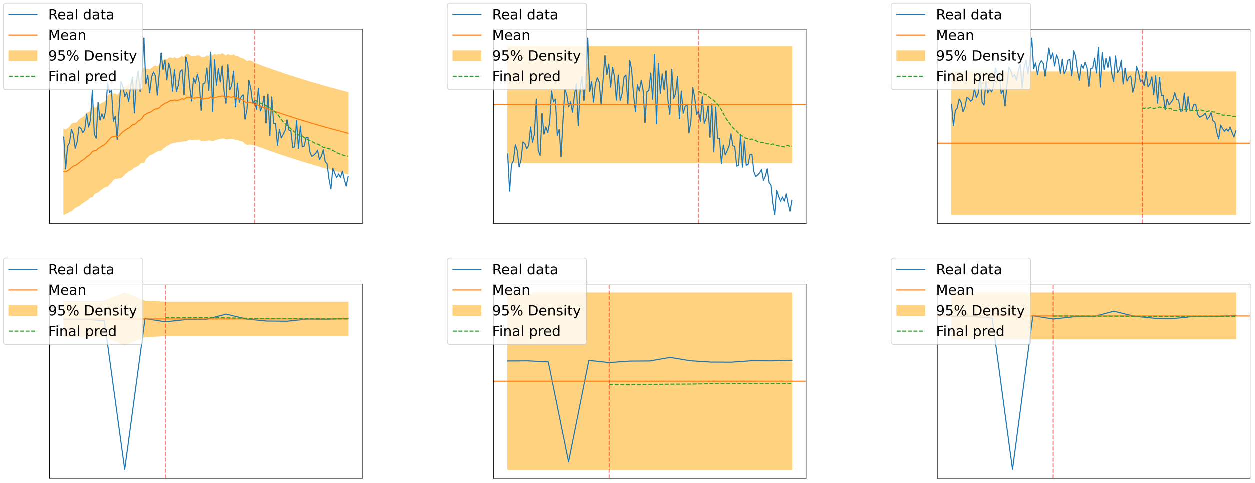
Unlike other methods such as RevIN [19], the EncoderNormalizer of the Pytorch-Forecasting library[3], or Batch normalization [18], our method is not dependent on the length of the encoder sequence or the batch size. The EncoderNormalizer (we will call it "Local Norm" as it is used also on GluonTS with another name) normalizes each input sequence using the mean and the variance of the single input itself, similarly to Instance normalization [38], by taking the mean and the variance from the single context length (the observed part of the series used as input for the encoder). RevIN extends this concept by adding a learnable affine transformation to the data. Both methods reverse their normalization for the output sequence with an inverse transformation (denormalization). Batch normalization instead, when applied to the input data, extracts the statistics from the batch dimension of the input and applies an affine transformation to the normalized data. These methods all depend on the particular training settings, losing their impact when the input sequence is too short (or the batch size too small) or too long (or too large). GAS-Norm, being directly fitted on the time series, does not suffer any of these limitations: by updating the mean and the variance used for normalizing the data at each timestep, our approach is truly adaptive and independent of the network architecture or training procedure.

A more traditional approach consists of normalizing the data with the mean and the variance of the entire training set (we will call this method Global Norm). Another shortcoming of all these approaches is that, by normalizing a sequence with a single mean and variance, it will remain non-stationary. In contrast, GAS-Norm enables a high level of flexibility, partially removing the non-stationary behavior of the encoder sequence and allowing the use of a dynamic mean and variance to denormalize the final prediction of the deep model.

Figure 4 illustrates some examples of how GAS-Norm, Local Norm, and Global Norm will act on an encoder/decoder sequence. The first row shows an example of AR data (described below). It is clearly visible how GAS-Norm is able to follow the data, while also helping the deep model by providing a prediction for the future behavior of the sequence. The Global Norm, by using a single

[2] The distributional assumptions depend on the data characteristics.

[3] https://pytorch-forecasting.readthedocs.io/en/stable/



**(a) GAS-Norm**

**(b) Local Norm**

**(c) Global Norm**

**Figure 4: Examples of input/output sequences with different methods' filtered and predicted statistics together with the denormalized prediction of the DNN. The top row shows a sequence from AR data, while the bottom row shows from the VIX data with an outlier. The red vertical dashed line splits the context part of the sequence from the part to be predicted.**

**Table 1: Normalization methods comparison. Mean (standard error) of MASE on the test set.**

| Data | Enc length | GAS-Norm(100) | GAS-Norm(20) | Local Norm | Global Norm | BatchNorm | RevIN |
|---|---|---|---|---|---|---|---|
| AR | 25 | 2.49 (0.459) | **2.0597 (0.3818)** | 2.0818 (0.1493) | 2.7052 (0.3712) | 4.9385 (0.4268) | 2.6416 (0.3685) |
| | 100 | **2.0038 (0.4591)** | 2.3813 (0.3722) | 2.3615 (0.289) | 2.5001 (0.2555) | 3.8886 (0.1987) | 3.4488 (0.0754) |
| | 200 | **1.9345 (0.3078)** | 2.1518 (0.1018) | 2.1728 (0.2807) | 2.3558 (0.2278) | 8.2509 (0.5197) | 5.3887 (0.4934) |
| VIX | 25 | 0.6841 (0.0002) | **0.6839 (0.0001)** | 0.6957 (0.0002) | 0.6843 (0.0001) | 0.6843 (0.0002) | 0.6958 (0.0002) |
| | 100 | 0.6734 (0.0002) | **0.6733 (0.0001)** | 0.6758 (0.0002) | 0.6736 (0.0001) | 0.6735 (0.0002) | 0.6761 (0.0005) |
| | 200 | 0.6696 (0.0002) | **0.6695 (0.0002)** | 0.6699 (0.0003) | 0.6697 (0.0002) | 0.6696 (0.0001) | 0.6698 (0.0003) |
| ECL | 25 | 2.2159 (0.0377) | **2.093 (0.0319)** | 2.1269 (0.0168) | 2.4096 (0.2796) | 4.5395 (0.0512) | 2.1266 (0.0156) |
| | 100 | 2.0942 (0.018) | 1.9876 (0.0181) | **1.9434 (0.0126)** | 2.311 (0.1835) | 4.4926 (0.0444) | 1.9437 (0.01) |
| | 200 | 2.2395 (0.1238) | **1.9616 (0.0096)** | 2.0133 (0.0172) | 2.278 (0.2978) | 4.4318 (0.0422) | 2.0133 (0.0162) |

global statistic for all the data, is visibly using a wrong variance to standardize this sequence.

In the second row, the same comparison is done with an example of VIX data[4] (described below) with the addition of an outlier. In this case, the Global Norm's robustness is playing in its favor. A single outlier will not change the overall mean and variance computed on the entire training set. Conversely, the Local Norm will be highly affected by an outlier (as would RevIN), particularly in the case of shorter input sequences. GAS-Norm, instead, can assume any distribution, becoming robust to outliers. In this case, a Student's t-distribution with 20 degrees of freedom is assumed, thus mitigating the large changes encouraged by the outliers.

## 4.1 Quantitative Evaluation

To compare the different normalization methods, we test them in three widely different univariate datasets. In our experiments, we compare GAS-Norm to standard normalization approaches like Global Norm and batch normalization [18] and to time series-specific approaches like Local Norm and the well-known state-of-the-art RevIN [19]. The experiments are performed in three datasets with different characteristics:

(1) AR: The AR dataset is a synthetic dataset generated by the built-in function of the pytorch-forecasting library "generate-ar-data". It is the realization of an autoregressive process to which a sinusoidal seasonality and a trend are added. This data is particularly difficult to learn due to the constantly moving mean of the input data.
(2) VIX: The Chicago Board Options Exchange's Volatility Index is a fundamental financial index measuring the expected volatility of the stock market, implied in the S&P 500 index options. We used the daily data from January 1990 to May 2024 downloaded from Yahoo Finance. As is frequently done in econometrics [7], we took the first difference of the data thus removing the non-stationarity of the mean. The resulting series is highly heteroscedastic, with an increase in the

[4] www.cboe.com/tradable_products/vix/

**Table 2: MASE obtained with multiple architectures and normalization strategies on different datasets.**

| Dataset | Model | Default | Local Norm | Global Norm | Mean Scaling | GAS-Norm |
|---|---|---|---|---|---|---|
| NN5-8 Weekly | FFNN | **0.868 (0.041)** | 0.886 (0.019) | 0.899 (0.033) | 0.867 (0.032) | 0.881 (0.030) |
| | Transformer | 1.138 (0.201) | 1.589 (0.526) | 0.895 (0.049) | 0.952 (0.089) | **0.828 (0.032)** |
| | DeepAR | 1.043 (0.238) | - | 0.868 (0.033) | 1.048 (0.373) | **0.822 (0.031)** |
| | MQCNN | 1.042 (0.054) | 0.931 (0.027) | 0.921 (0.008) | 0.896 (0.069) | **0.890 (0.013)** |
| NN5-35 Weekly | FFNN | 6.109 (7.914) | 1.389 (0.031) | 1.409 (0.013) | 4.774 (0.976) | **1.274 (0.014)** |
| | Transformer | 1.753 (0.456) | 1.697 (0.384) | 1.349 (0.052) | 1.638 (0.518) | **1.309 (0.089)** |
| | DeepAR | 1.663 (0.400) | - | 1.369 (0.020) | 1.837 (0.594) | **1.206 (0.018)** |
| | MQCNN | 5.192 (3.462) | 1.332 (0.053) | 1.434 (0.031) | 11.141 (4.630) | **1.291 (0.066)** |
| M4 Weekly | FFNN | 0.603 (0.012) | 0.582 (0.008) | 0.591 (0.017) | 0.620 (0.007) | **0.572 (0.009)** |
| | Transformer | 3.674 (0.190) | 1.689 (0.046) | 0.952 (0.142) | 0.872 (0.046) | **0.685 (0.059)** |
| | DeepAR | 1.528 (0.119) | - | **0.603 (0.036)** | 1.643 (0.105) | 0.606 (0.018) |
| | MQCNN | 0.751 (0.051) | 0.683 (0.033) | 0.778 (0.041) | 0.809 (0.037) | **0.673 (0.026)** |
| Fred MD | FFNN | 0.765 (0.054) | 0.657 (0.048) | 0.593(0.024) | 0.741 (0.034) | **0.593 (0.030)** |
| | Transformer | 8.411 (0.339) | 2.360 (0.028) | 0.733 (0.053) | 1.459 (0.496) | **0.719 (0.112)** |
| | DeepAR | 5.540 (0.091) | - | **0.629 (0.065)** | 6.480 (0.130) | 0.773 (0.193) |
| | MQCNN | 0.776 (0.076) | 0.813 (0.010) | 0.768 (0.080) | 0.821 (0.046) | **0.662 (0.050)** |

variance during financial crises and a decrease in variance during stable times.

(3) ECL: The Electricity Consuming Load[5] contains the hourly kWh electricity consumption of multiple clients. We took a subset of it (the first 10 thousand hours of the first client). This data has strong recurrent patterns due to habit repetition and sudden jumps between high and low consumption.

We apply the different normalization methods to a two-layer LSTM, always predicting the next 50 steps in time but using different input sequence lengths. GAS-Norm is used assuming a Student's t-distribution experimented with 20 and 100 degrees of freedom. Hyperparameter optimization is performed for the learning rate of each method and the normalization strength of GAS-Norm. Each configuration is trained 10 times with different random seeds. The training is executed on a Tesla V100 16GB. We report the Mean Absolute Scaled Error (MASE) for all our experiments. The fundamental work of Hyndman, et al. [17] suggests this as the preferable metric to compare forecasting models on different series. Note that a value less than 1 is acceptable in multistep forecasting due to the additional complexity of predicting a distant future compared to the easier naive approach with access to the last time step. The means and the standard deviations of the MASE in the test set are presented in Table 1. In our tests, GAS-Norm showed remarkable robustness to the different data characteristics. AR data, since it is synthetic, has no outliers, resulting in the GAS-Norm with 100 degrees of freedom winning the comparison with both the Local Norm and the Global Norm performing similarly well. Notice also how the longer the input sequence, the better the Global Norm performs. VIX data is very heteroscedastic with many outliers. As expected, the robust GAS-Norm performs better. With the ECL data, both RevIN and the Local Norm obtain good results compared to the other methods (this is one of the datasets used in RevIN paper [19]). Still, GAS-Norm is the best method in 2 out of 3 settings. Since ECL shows strong seasonality with sudden jumps, GAS-Norm results could be greatly improved by using a seasonal GAS, which we leave as future work.

## 5 Experiments with SOTA Forecasting Models

In addition to the above comparisons, we also evaluate our method using state-of-the-art forecasting models on more extended real-world data. For these experiments, we rely on the Monash Forecasting repository [12] and their model evaluation procedure. Datasets in the repository are collections of several time series. Each time series shares the same context length and prediction length. The former refers to the length of the portion of the time series to be used as input (sometimes known as the lag length). The latter is the forecasting horizon. Datasets are then used to train a global probabilistic forecasting model, i.e., a single model trained across all series to predict the parameters of a pre-defined output distribution (Student's t in our case). To evaluate the models, a portion of elements from the end of each time series are used as the test set, and the length of that portion is equal to the prediction length.

We selected three different datasets, Fred MD, NN5 weekly, and M4 weekly, which present different non-stationary natures and cover a variety of domains. Table 4 shows some statistics for each dataset, including context and prediction lengths used. We use the same lengths for these datasets as the Monash repository curators. In the case of NN5 weekly, we also test our procedure with a prediction length equal to 35, the longest possible length for this dataset. The GluonTS package for time series modeling [1] is used as the foundation for the deep learning models. Among these are the feedforward neural network [31], transformer [40], DeepAR [33], and MQCNN [41] models. These models are chosen as they represent various complexities and are based on the most common deep learning architectures.

We extended GluonTS to support different normalization methods. Training is again done using a Tesla V100 with 16GB of RAM. Each model is trained and evaluated with GAS-Norm, global and local normalization settings from section 4 and two additional procedures:

[5] archive.ics.uci.edu/dataset/321

- Default state: The out-of-the-box state of the model's implementation. For DeepAR, this state includes local normalization. For the other models, it includes no normalization.
- Mean scaling: dividing values in the time series using the average value of the previous context length values.

Since the model forecasts are distributional, the output is taken to be the mean of 100 forecast samples. For each approach, we perform the same hyperparameter tuning on each model. The hyperparameters were tuned over 10 trials, selecting the best configuration on the validation set for the final evaluation. The tuning search spaces are shown in Table 3.

In addition to searching the model hyperparameters, we also perform a simple search for the GAS-Norm hyperparameter, i.e., the normalization strength. The search space is (0, 0.001, 0.01, 0.1, 0.5). For each strength value, we repeat the model hyperparameter tuning as described above, thus resulting in the selection of the normalization strength, along with the corresponding model hyperparameters.

After selecting the best values, they are used to train and evaluate each model five times, recording the mean absolute scaled error results.

### 5.1 Results

The mean and standard deviation of these results are presented in Table 2. In 13 out of 16 configurations, the GAS-Norm approach gives the best result. Global normalization is the most effective in 2 occurrences, but it is not meaningfully ahead of GAS-Norm. In one unusual case of the default setups, for NN5-8 weekly with FFN, there is an occurrence where it had the best result so any added normalization worsens performance. However, considering the small differences across settings for that model and dataset, the occurrence could be due to the limited number of trials and tuning. Furthermore, when the default setups perform poorly they show results notably worse than the rest. Mean scaling, on the other hand, shows somewhat similar behavior as the default setups. That is, it suffered in the same cases as the defaults, but without excelling in any configurations. For local normalization, DeepAR was excluded from this evaluation since its default state already includes it. The remainder of configurations for this approach did not stand out in performance but contained no unusually poor instances either. An important caveat to note with these experimental results is that this GAS-Norm implementation is only a starting point for the method's capabilities. It is probably possible to get additional improvements by leveraging time series seasonality or using tailored distributional assumption.

Comparing the different forecasting methods, the results show that DeepAR and FFN tend to outperform the rest on two datasets each. On the other hand, the worst results for each dataset came from the Transformer on three benchmarks and MQCNN on one. We hypothesize that internal normalization methods in the Transformer and MQCNN (LayerNorm or BatchNorm layers), which do not include a denormalization step, may be partially responsible for the underperformance.

The GAS-Norm's normalization phase added a negligible overhead. More specifically, with an Intel Xeon Gold 6140M CPU, the normalization phases for NN5 weekly, M4 weekly and Fred MD took 4.08s, 28.46s and 8.37s respectively at a mean and variance strength of 0. The strength parameter affects the runtime of the normalization phase, where they increase in tandem. So the same respective datasets with a strength of 0.5 yielded runtimes of 14.55s, 61.08s and 307.02s.

## 6 Conclusion

In this paper, we show that non-stationarity is one of the main challenges when training nonlinear deep networks for forecasting. We focus on non-stationarity in the mean and variance, showing that they disrupt the predictions of nonlinear deep networks even in simple settings. To mitigate this issue, we propose GAS-Norm, a novel normalization method. The GAS model works as a filter that updates the mean and variance online. As a result, the normalized input passed to the deep network is more stable even in non-stationary settings. The output of the deep network is denormalized using the GAS predictions, obtaining the final forecasting. We evaluate GAS-Norm on a wide array of diverse datasets and models, which encompass the most popular modeling choices. We show that this method is more robust than popular normalization methods both on synthetic data and in real benchmarks. The results also show that most forecasting methods improve when combined with GAS-Norm.

GAS-Norm provides a general framework that can be easily extended in future works with different distributional assumptions and update dynamics, such as counting processes or time series with explicit seasonality. Additionally, it could also be investigated as a way to mitigate non-stationarity of the hidden activations of the network.

## A Appendix

### A.1 Data Statistics and Hyperparameters

**Table 3: Hyperparameter tuning space per model.**

| Model | Hyperparameter | Searched Values |
|---|---|---|
| Feed Forward | Number of Layers | [1,5] |
| | Hidden Dimensions | [10,100] |
| | Learning Rate | [1e-3,1e-6] |
| | Number of Epochs | [10,100] |
| Transformer, DeepAR, MQCNN | Learning Rate | [1e-3,1e-6] |
| | Number of Epochs | [10,100] |

**Table 4: Dataset statistics.**

| Dataset | # Time Series | Avg TS Length | Context Length | Prediction Length |
|---|---|---|---|---|
| Fred MD | 107 | 728 | 15 | 12 |
| NN5 Weekly | 111 | 113 | 65 | 8, 35 |
| M4 Weekly | 359 | 1035 | 26 | 13 |

## A.2 Gaussian Score Driven Adaptive Normalization

In this section, we show the Gaussian version of the GAS normalization procedure.

Assume each feature to be a time series with a Gaussian conditional distribution:

$$y_{i,t} \sim \mathcal{N}(\mu_{i,t}, \sigma^2_{i,t})$$

Where $i$ is the feature index that we can now drop, considering each feature separately with the same procedure. The update for the two parameters of the distribution becomes:

$$\begin{pmatrix}\mu_{t+1}\\ \sigma^2_{t+1}\end{pmatrix} = \begin{pmatrix}\omega_\mu\\ \omega_\sigma\end{pmatrix} + \begin{pmatrix}\frac{\gamma}{1-\gamma}\alpha_\mu & 0\\ 0 & \frac{\gamma}{1-\gamma}\alpha_\sigma\end{pmatrix} S_t \begin{pmatrix}\nabla_\mu\\ \nabla_\sigma\end{pmatrix} + \begin{pmatrix}\beta_\mu & 0\\ 0 & \beta_\sigma\end{pmatrix}\begin{pmatrix}\mu_t\\ \sigma^2_t\end{pmatrix}$$

where $S_t$ is a $2 \times 2$ scaling matrix (we use the Fisher Information Matrix as suggested by Creal et al. [8] and to obtain a natural gradient [2] adapted to the data geometry). We now compute the scores for the mean, the variance and the component of the FIM:

$$log\mathcal{N}(y_t|\mu_t, \sigma^2_t) = -\frac{1}{2}log(2\pi) - \frac{1}{2}log\sigma^2_t - \frac{1}{2\sigma^2_t}(y_t - \mu_t)^2$$

$$\frac{\partial log\mathcal{N}(y_t|\mu_t, \sigma^2_t)}{\partial \mu_t} = \frac{y_t - \mu_t}{\sigma^2_t}$$

$$\frac{\partial log\mathcal{N}(y_t|\mu_t, \sigma^2_t)}{\partial \sigma^2_t} = \frac{1}{2}(\frac{(y_t - \mu_t)^2}{\sigma^4} - \frac{1}{\sigma^2_t})$$

$$\mathbf{V}[\frac{\partial log\mathcal{N}(y_t|\mu_t, \sigma^2_t)}{\partial \mu_t}] = \frac{1}{\sigma^2}$$

$$\mathbf{V}[\frac{\partial log\mathcal{N}(y_t|\mu_t, \sigma^2_t)}{\partial \sigma^2_t}] = \frac{1}{2\sigma^4}$$

The covariance between the two gradients is 0. Given our parameterization of the update, this results in two independent update functions. Using the Fisher Information as a scaling matrix we obtain:

$$\mu_{t+1|t} = \omega_\mu + \beta_\mu[\frac{\gamma}{1-\gamma}\alpha_\mu(y_t - \mu_t) + \mu_t]$$

$$\sigma^2_{t+1|t} = \omega_\sigma + \beta_\sigma[\frac{\gamma}{1-\gamma}\alpha_\sigma((y_t - \mu_t)^2 - \sigma^2_t) + \sigma^2_t].$$

## A.3 Student's t Score Driven Adaptive Normalization

Student's t-distribution has a subexponential decay rate, allowing for fat tails. This means that this distribution can be used when we expect to have outliers in our feature time series. The degrees of freedom control the shape of the distribution, with fatter tails with lower degrees of freedom.

Again we standardize the features independently from the others. We assume a feature $i$ to be generated as:

$$y_t = \mu_t + \sigma_t \epsilon_t \qquad \epsilon_t \sim t(0, 1, \nu)$$

where $\nu$ represents the degrees of freedom.

The resulting likelihood is

$$t(y_t|\mu_t, \sigma^2_t, \nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\pi\nu\sigma^2}}(1 + \frac{(y_t - \mu_t)^2}{\nu\sigma^2})^{-(\nu+1)/2},$$

where $\Gamma$ is the Gamma function. The corresponding log-likelihood becomes

$$logt(y_t|\mu_t, \sigma^2_t, \nu) = log(\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\pi\nu}}) - \frac{1}{2}log(\sigma^2) - - \frac{\nu+1}{2}log(1 + \frac{(y_t - \mu_t)^2}{\nu\sigma^2}).$$

We can now compute the gradients that compose the score and the variances of the scores:

$$\frac{\partial t(y_t|\mu_t, \sigma^2_t, \nu)}{\partial \mu_t} = \frac{(\nu+1)(y_t - \mu_t)}{\nu\sigma^2 + (y_t - \mu_t)^2}$$

$$\frac{\partial t(y_t|\mu_t, \sigma^2_t, \nu)}{\partial \sigma^2_t} = \frac{1}{2}(\frac{(\nu+1)(y_t - \mu_t)^2}{\nu\sigma^4 + \sigma^2(y_t - \mu_t)^2} - \frac{1}{\sigma^2})$$

$$V[\frac{\partial t(y_t|\mu_t, \sigma^2_t, \nu)}{\partial \mu_t}] = \frac{\nu+1}{(\nu+3)\sigma^2_t}$$

$$V[\frac{\partial t(y_t|\mu_t, \sigma^2_t, \nu)}{\partial \sigma^2_t}] = \frac{\nu}{2(\nu+3)\sigma^4_t}.$$

Again the correlation between the two gradients is 0. The degrees of freedom are chosen as a static parameter. The dynamic version is possible but the optimization is much more difficult. As suggested by Artemova et al. [3], we regularize scores with a scaling proportional to the inverse Fisher information: $S_{\mu,t} = \frac{\nu\sigma^2_t}{1+\nu}$ for the mean and $S_{\mu,t} = 2\sigma^4$ for the variance:

The update we obtain is:

$$\mu_{t+1} = \omega_\mu + \beta_\mu[\frac{\gamma}{1-\gamma}\alpha_\mu\frac{y_t - \mu_t}{1 + \frac{(y_t-\mu_t)^2}{\nu\sigma^2_t}} + \mu_t]$$

$$\sigma^2_{t+1} = \omega_\sigma + \beta_\mu[\frac{\gamma}{1-\gamma}\alpha_\sigma(\frac{(\nu+1)(y_t - \mu_t)^2}{\nu + \frac{(y_t-\mu_t)^2}{\sigma^2_t}} - \sigma^2_t) + \sigma^2_t]$$

## Acknowledgments

Antonio Carta acknowledges support from the Ministry of University and Research (MUR) as part of the FSE REACT-EU - PON 2014-2020 "Research and Innovation" DM MUR 1062/2021. Reshawn J. Ramjattan acknowledges support from the Ministry of University and Research (MUR) as part of the PON 2014–2020 "Research and Innovation" resources—Green/Innovation Action—DM MUR 1061/2022.

## References

[1] Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C Maddix, Syama Rangapuram, David Salinas, Jasper Schulz, et al. 2020. Gluonts: Probabilistic and neural time series modeling in python. *Journal of Machine Learning Research* 21, 116 (2020), 1–6.

[2] Shun-ichi Amari. 2016. *Information geometry and its applications*. Vol. 194. Springer.

[3] Mariia Artemova, Francisco Blasques, Janneke van Brummelen, and Siem Jan Koopman. 2022. Score-driven models: Methodology and theory. In *Oxford Research Encyclopedia of Economics and Finance*.

[4] Christopher M Bishop. 2006. Pattern recognition and machine learning. *Springer google schola* 2 (2006), 1122–1128.

[5] Francisco Blasques, Siem Jan Koopman, and Andre Lucas. 2015. Information-theoretic optimality of observation-driven time series models for continuous responses. *Biometrika* 102, 2 (2015), 325–343.

[6] Tim Bollerslev. 1986. Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics* 31, 3 (1986), 307–327.

[7] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.

[8] Drew Creal, Siem Jan Koopman, and André Lucas. 2013. Generalized Autoregressive Score Models with Applications. *Journal of Applied Econometrics* 28, 5 (2013), 777–795. https://doi.org/10.1002/jae.1279

[9] Chirag Deb, Fan Zhang, Junjing Yang, Siew Eang Lee, and Kwok Wei Shah. 2017. A review on time series forecasting techniques for building energy consumption. *Renewable and Sustainable Energy Reviews* 74 (2017), 902–924.

[10] Jinliang Deng, Xiusi Chen, Renhe Jiang, Xuan Song, and Ivor W Tsang. 2021. St-norm: Spatial and temporal normalization for multi-variate time series forecasting. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 269–278.

[11] Wei Fan, Pengyang Wang, Dongkun Wang, Dongjie Wang, Yuanchun Zhou, and Yanjie Fu. 2023. Dish-ts: a general paradigm for alleviating distribution shift in time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 7522–7529.

[12] Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I. Webb, Rob J. Hyndman, and Pablo Montero-Manso. 2021. Monash Time Series Forecasting Archive. arXiv:2105.06643 [cs, stat]

[13] Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S Lew. 2016. Deep learning for visual understanding: A review. *Neurocomputing* 187 (2016), 27–48.

[14] JD Hamilton. 1994. FTime Series Analysis.

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[16] Abir Jaafar Hussain, Adam Knowles, Paulo JG Lisboa, and Wael El-Deredy. 2008. Financial time series prediction using polynomial pipelined neural networks. *Expert Systems with Applications* 35, 3 (2008), 1186–1199.

[17] Rob J Hyndman and Anne B Koehler. 2006. Another look at measures of forecast accuracy. *International journal of forecasting* 22, 4 (2006), 679–688.

[18] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. pmlr, 448–456.

[19] Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. 2021. Reversible Instance Normalization for Accurate Time-Series Forecasting against Distribution Shift. In *International Conference on Learning Representations*.

[20] Tae Yoon Kim, Kyong Joo Oh, Chiho Kim, and Jong Doo Do. 2004. Artificial neural networks for non-stationary time series. *Neurocomputing* 61 (2004), 439–447.

[21] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-normalizing neural networks. *Advances in neural information processing systems* 30 (2017).

[22] Rutger-Jan Lange, Bram van Os, and Dick JC van Dijk. 2022. Robust Observation-Driven Models Using Proximal-Parameter Updates. *Available at SSRN 4227958* (2022).

[23] Yanghao Li, Naiyan Wang, Jianping Shi, Xiaodi Hou, and Jiaying Liu. 2018. Adaptive Batch Normalization for Practical Domain Adaptation. *Pattern Recognition* 80 (Aug. 2018), 109–117. https://doi.org/10.1016/j.patcog.2018.03.005

[24] Zhiding Liu, Mingyue Cheng, Zhi Li, Zhenya Huang, Qi Liu, Yanhu Xie, and Enhong Chen. 2024. Adaptive normalization for non-stationary time series forecasting: A temporal slice perspective. *Advances in Neural Information Processing Systems* 36 (2024).

[25] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. 2017. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems* 30 (2017).

[26] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. 2018. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics* 19, 6 (2018), 1236–1246.

[27] Eduardo Ogasawara, Leonardo C Martinez, Daniel De Oliveira, Geraldo Zimbrão, Gisele L Pappa, and Marta Mattoso. 2010. Adaptive normalization: A novel data normalization approach for non-stationary time series. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[28] Daniel W Otter, Julian R Medina, and Jugal K Kalita. 2020. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems* 32, 2 (2020), 604–624.

[29] Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. 2019. Deep Adaptive Input Normalization for Time Series Forecasting. https://doi.org/10.48550/arXiv.1902.07892 arXiv:1902.07892 [cs, q-fin]

[30] Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. 2022. *Dataset shift in machine learning*. Mit Press.

[31] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.

[32] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2020. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *International Journal of Forecasting* 36, 3 (July 2020), 1181–1191. https://doi.org/10.1016/j.ijforecast.2019.07.001

[33] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2020. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International journal of forecasting* 36, 3 (2020), 1181–1191.

[34] Saptarshi Sengupta, Sanchita Basak, Pallabi Saikia, Sayak Paul, Vasilios Tsalavoutis, Frederick Atiah, Vadlamani Ravi, and Alan Peters. 2020. A review of deep learning with special emphasis on architectures, applications and recent trends. *Knowledge-Based Systems* 194 (2020), 105596.

[35] Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. 2020. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied soft computing* 90 (2020), 106181.

[36] Slawek Smyl. 2020. A Hybrid Method of Exponential Smoothing and Recurrent Neural Networks for Time Series Forecasting. *International Journal of Forecasting* 36, 1 (Jan. 2020), 75–85. https://doi.org/10.1016/j.ijforecast.2019.03.017

[37] Jorge Sola and Joaquin Sevilla. 1997. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on nuclear science* 44, 3 (1997), 1464–1468.

[38] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. 2016. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022* (2016).

[39] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A Generative Model for Raw Audio. *arXiv:1609.03499 [cs]* (Sept. 2016). arXiv:1609.03499 [cs]

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[41] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. 2017. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053* (2017).

[42] G. Peter Zhang. 2003. Time Series Forecasting Using a Hybrid ARIMA and Neural Network Model. *Neurocomputing* 50 (Jan. 2003), 159–175. https://doi.org/10.1016/S0925-2312(01)00702-0

[43] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 12 (May 2021), 11106–11115. https://doi.org/10.1609/aaai.v35i12.17325