ISSN 1870-4069

Data Augmentation vs Regularization for Time Series Forecasting

Juan J. Flores, Miguel A. Reynoso, Josué D. González, Felix Calderon

Universidad Michoacana de San Nicolás de Hidalgo,
Facultad de Ingeniería Eléctrica,
División de Estudios de Posgrado,
Mexico

{juan.flores, 0850750b, jdgonzalez, felix.calderon}@umich.mx

Abstract. This article presents a study on the introduction of noise to the training data of an Artificial Neural Network that models the forecasting process for time series. The introduction of noise can act as a form of regularization, improving the network capacity to generalize on test data. When using an Artificial Neural Network to perform time series forecasting, we need to find out the best way to introduce noise to the data, preserving its temporal relation, while being able to populate the training set with a large enough number of noisy data points as to achieve the desired regularization effect. First we convert the time series to a design matrix, by determining the reconstruction dimension of the underlying process that produced it (m), together with a subsampling period (τ); sweeping the time series forms all possible delay vectors. The resulting design matrix maps the forecasting problem into a regression problem. Noise is introduced to the design matrix, populating it with noisy delay vectors; noise is generated following a normal distribution $N(0, \sigma)$. Neural Networks are designed to produce regressors that solve the forecasting problem for a set of time series. The forecasting accuracy of the noise-regularized models is compared against models regularized by Early Stopping, second order Bayesian Regularization and both. Noise regularization produced better results in the vast majority of the cases.

Keywords: Time series, forecasting, regularization, data augmentation, noise.

1 Introduction

A time series is a time-ordered sequence of observations values of a variable made at equally spaced time intervals, represented as a set of discrete values [1]. Time series are found in many fields, such as economics, sociology, meteorology, medicine, seismology, oceanography, geomorphology, astronomy, etc. [2]. Time series analysis helps to detect regularities in the observations of a variable, detect regularities in data, determine a suitable model, and/or exploit all information included in this variable to better predict future developments [3].

Time series forecasting is currently a very important research area, due to the importance of prediction in many fields. Applications range from natural (wind speed, ambient temperature, solar irradiance, etc.) to anthropic phenomena (stock price, electric energy consumption, etc.). Time series forecasting is an area in which past observations of the same variable are collected and analyzed to develop a model that describes the underlying relationship. The model is then used to extrapolate the time series into future scenarios [4].

Over the past several decades, many studies have been conducted to develop innovative forecasting approaches and improve their accuracy. In general, these models can be categorized into three types: statistical models, artificial intelligence models and hybrid models [5].

Statisticians and econometricians tend to rely on autoregressive integrated moving average (ARIMA) and derived or related models, while the artificial intelligence community mainly looks at neural networks, either using multilayer perceptrons or recurrent networks [6].

After fitting a time series model, one can evaluate it with forecast fit measures. The researcher may subtract the forecast value from the observed value of the data at that point in time and obtain a measure of error or bias. The statistics used to describe this error are similar to the univariate statistics just mentioned, except that the forecast is often substituted for the average value of the series. The chosen fitness function to evaluate the amount of this forecast error is the symmetric mean absolute percentage error (SMAPE), defined as shown by equation (1). This version of SMAPE ranges from 0 to 100%:

$$SMAPE = \frac{100}{n} \sum_{t=1}^{n} \frac{|y_t - \hat{y}_t|}{y_t + \hat{y}_t}. \tag{1}$$

In equation (1), n is the length of the time series, y_t represents the real value, and $\hat{y}_t$ the forecast value.

We will address several research questions in this article: what is the relation between regularization and the introduction of noise in the training process of an ANN?, how to introduce noise to a time series, while preserving the relation of its measurements with time?, how much noise and how many noisy points must be introduced in the ANN training set?, and finally, is the introduction of noise to the training data any better or at least comparable to the process of Bayesian Regularization?

2 Forecasting as Regression

When using artificial neural networks (ANN) to solve the problem of time series forecasting there are several possibilities. One of them is to use Recurrent Neural Networks, which maintain a state of the sequence presented to the network and takes it into consideration for the following computation. GRU and LSTM are examples of such networks [7]. Another approach is the Dynamic Multi-layer Perceptrons [8], which use tapped delay lines to constantly provide part of the

time series history to an MLP. Another resource and the one we are using in this implementation is to convert the time series to a database or design matrix, that constitutes the examples to feed the ANN (training, validation, and test sets).

One problem present with any of these approaches is to determine how much of the history of the time series can be fed as input to the ANN. Let us say that we determined that a certain interval of the past is important to the ANN to produce an accurate forecast. The following problem to solve is if all of those measurements are important, or only a subset of them. Flores et al. [9] solved this problem using Evolutionary Computation.

Kantz and Screiber [10] solve the first part of this problem using a method called false nearest neighbors. This method allows determines the dimension, m, of the underlying dynamic system that produced the time series data. Another method based on mutual information determines a subsampling period, τ, such that the amount of information provided to the ANN is maximized.

Let $\mathbb{S} = [s_1, s_2, \ldots, s_t, \ldots, s_N]$ be a time series, where s_t is the value of variable s at time t. It is desired to obtain the forecast of the point at $t + \Delta$ as a function of the observations available in $\mathbb{S}$.

By using a τ delay and an embedding dimension m, it is possible to build delay vectors of the form $[s_{t-(m-1)\tau}, s_{t-(m-2)\tau}, \ldots, s_{t-\tau}, s_t]$, which constitute a reconstruction of the phase portrait of the underlying dynamic system. We then append the value to be forecasted, $s_{t+\Delta}$ to each of those vectors. The resulting matrix is the design matrix, which represents the set of examples to be presented to the ANN for training, validating, and testing.

This transformation, from the time series to a design matrix, maps the forecasting problem to a regression problem. The first part of each example, $[s_{t-(m-1)\tau}, s_{t-(m-2)\tau}, \ldots, s_{t-\tau}, s_t]$, represents the independent variables and the second part, $s_{t+\Delta}$, the dependent variable in a regression problem. We already know how to solve this kind of problem using an MLP.

3 Neural Networks

An artificial neural networks is essentially a collection of nonlinear transfer functions that relate some output variable(s) of interest to some input variables, which may themselves be functions of even deeper explanatory variables [11].

The ANNs approach has been suggested as an alternative technique to time series forecasting [12, 13]; this approach has gained immense popularity in the last few years. ANNs try to recognize regularities and patterns in the input data, learn from experience, and then provide generalized results based on previous knowledge.

The main ANN used in forecasting problems is Multi-Layer Perceptrons (MLP). This model is characterized by a network of three types of layers (input, hidden, and output). These networks consist of neurons arranged in layers in which every neuron is connected to all neurons of the next layer by introducing the input in a feed-forward manner, which propagates through the hidden layer and the output layer [14].

The signals propagate from node to node and are modified by weights associated with each connection. The receiving node sums the weighted inputs from all of the nodes connected to it from the previous layer. The output of this node is then computed as the function of its input called the "activation function". The data moves forward from node to node with multiple weighted summations occurring before reaching the output layer [15].

Neural networks have been mathematically shown to be universal approximators of functions thus are inherently nonlinear and estimate well non-linear functions [16]. The universal approximator property suggests that neural networks can effectively model seasonality [17].

4 Generalization Problems with ANN

Overfitting is produced when a model fits the data too closely, so it is unlikely to generate good classification or prediction on previously unseen patterns. When that happens, the model's loss in training data may become very small, but the model loses its generalization capabilities. In the extreme, the loss tends to zero, in which case we can say the model memorizes the data. This is a problem found in most of the machine learning techniques [18].

Neural networks use a great number of parameters (weights and biases) to model phenomena, that is what makes neural networks so easy to overfit. An example of overfitting is shown in figure 1, the black curve follows perfectly all the data (red dots), but it is unlikely to have a good performance on new data. On the other hand, the blue curve is the best model to fit the data, since it is not too dependent on the data.

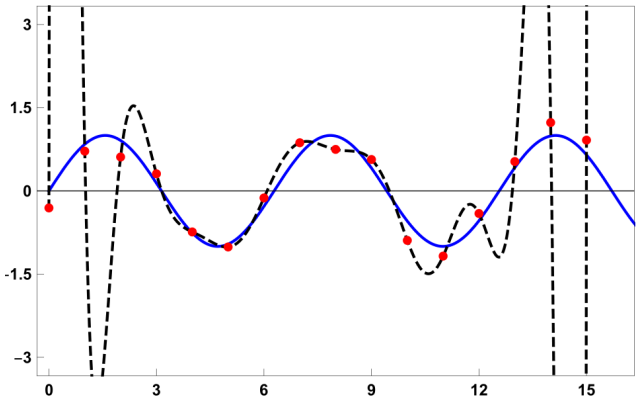


Fig. 1. Overfitting

Contrary to what happens with overfitting, underfitting is present when the model is incapable of capturing the underlying dynamics of the data.

Underfitting can happen for many reasons, one reason is when there is not enough data to build an accurate model, another reason is when models are not complex enough to reproduce the dynamics exhibited by the data.

Figure 2 shows the data to be fit by the model. The black curve is unable to follow the data, and, again, it is unlikely to have a good performance on new data.

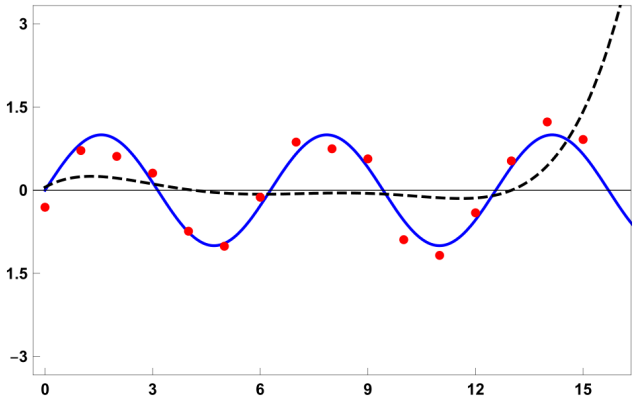


Fig. 2. Underfitting.

There are several strategies to avoid overfitting. One of them is to stop the learning process when the generalization starts to decrease, *early stopping*. Another strategy is to penalize complex model, *regularization*. One third strategy is to obtain more training data, *data augmentation (DA)*.

Early Stopping When the model performance, as measured by the loss function on the validation set, deteriorates in an arbitrary number of iterations, we can infer that the model's generalization capability is decreasing (Figure 3). That is, the model is being to overtrain; the training algorithm recognizes that condition and the training process stops. The number of iterations the network "waits" for the loss function to recover and continue improving is called **patience**.

Regularization. Since a larger number of parameters cause overfitting, a natural approach is to constrain the model to use fewer parameters, the fewer degrees of freedom it has, the harder it will be for it to overfit the data [19].

To create less complex models when the number of features in the data set is large, it is helpful to use some of the regularization techniques to avoid overfitting. One technique is called Ridge Regression; this technique regularizes the model by adding a *regularization term* equal to $\lambda \sum_{i=1}^{n} \beta_i^2$ to the cost function. This forces the learning algorithm to fit the data and keep the model's

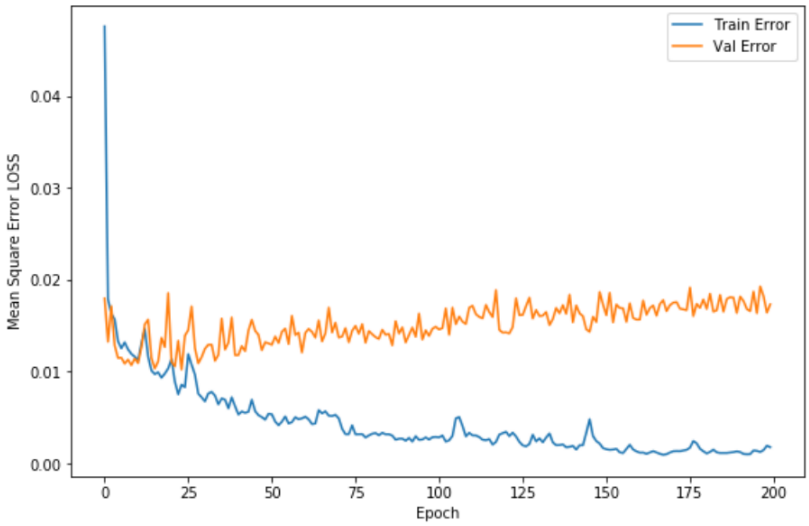


Fig. 3. Overfitting detection: while error in training set tends to decrease, it tends to increase in validation set.

weights as small as possible. The parameter λ controls the entropy of the model distribution. As $\lambda \to \infty$, the model distribution tends a uniform distribution and all weights end up very close to zero and the prediction is a line going through the data's mean [20].

Data augmentation. is a set of techniques whose objective is to acquire synthetic samples from the training set [21]. DA is used for two reasons, first of all, in many cases the training set is limited, and it is difficult to obtain more samples; using DA we can produce additional synthetic samples. The second is to gain better generalization properties of the model. For example on computer vision the samples of the training set are images, we can obtain synthetic samples from the training set by applying transformations to the images, e.g., rotations, translations, etc. [3]. This technique provides a larger data set and makes the vision system more robust to changes in images.

There are other methods to provide additional samples to the training set. Examples of those techniques are adding noise to the samples, randomly erasing parts of a data sample, etc. Currently, generative adversarial networks (GANs) are attracting attention from research [21, 22].

In GANs, a generator captures the data probability distribution to randomly generate new samples with the same distribution; finally a discriminator tries to learn which samples are real and which are fake.

The DA techniques that give good results in computer vision, give poor results (or are not suitable) for time series analysis. Therefore, great care must

be taken when choosing the DA technique; it is important to have a large number of samples as the generate are of good forecasting accuracy [23].

The data augmentation technique used in this paper is noise injection. This technique consists on generating synthetic samples by taking each sample of the training set and adding Gaussian noise [24]. From each sample of the training set, S synthetic samples will be created. To generate a synthetic sample, a certain amount of Gaussian noise is added to the original sample. Therefore, the training set has $N \cdot S$ synthetic samples

The noise to be injected into a sample (a delay vector) must be normally distributed. The amount of noise added when creating the synthetic samples is given by an SNR value, given in decibels [25]. A standard deviation, σ, must be calculated from this value [26]; this standard deviation is used to generate the Gaussian noise that is added to the original samples while creating the synthetic samples.

To determine the value of σ, for a given SNR, we first compute the power of the time series:

$$E_{ts} = \frac{1}{L} \sum_{i=0}^{L-1} |s_i|^2 , \tag{2}$$

where L is the length of the time series. Expressing the decibel SNR to a linear scale:

$$SNR_{lin} = 10^{SNR_{dB}/10}, \tag{3}$$

allows us to finally determine σ:

$$\sigma = \sqrt{\frac{E_{ts}}{SNR_{lin}}}. \tag{4}$$

A sample delay vector is shown in Figure 4, plotted in red, as well as four synthetic samples in black. The synthetic samples were created from the original sample contained in the training set. We can observe that although all the samples are different, they maintain the same qualitative form.

5 Tests Description

Seven-time series were used in the experiments, each of them was used to train a neural network. For each time series, we created one multilayer perceptron as forecasting tool, figure 5 shows one example of MLP. We considered in their design their modeling capabilities; i.e., they do not underfit the data.

For each time series, four tests were performed using the original training set:

Test 1 Train a neural network without regularization nor early-stopping (this test serves as a reference point for the following).
Test 2 Train the neuronal networks using early-stopping.
Test 3 Training the neuronal networks using regularization L2.

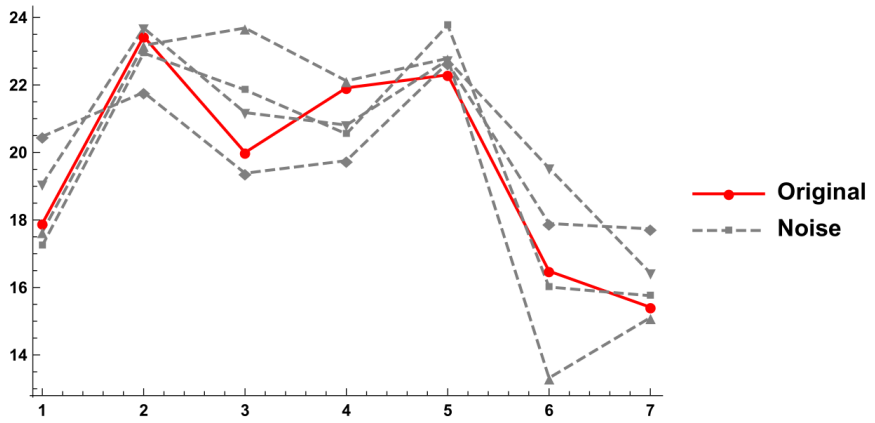


Fig. 4. Data augmentation.

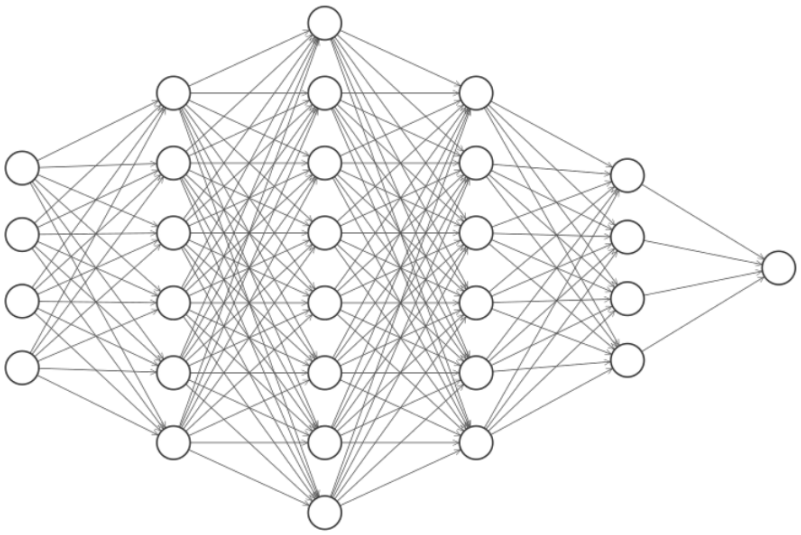


Fig. 5. Five hidden layers MLP.

Test 4 Train the neuronal network using early-stopping and regularization L2.

We repeated those four tests augmenting the training set with synthetic samples.

In the comparison of the results, we used the SMAPE of the test set against the prediction of the neural network. This error function ranges from 0 to 100%, providing a way to compare results in a time series of different scales. The difference between the test lies in the way of training the neural networks:

Eight-time series were used in the experiments. Those time series are:

1. Ambient temperature: This dataset provides samples every 10 minutes from Morelia, Michoacán. This time series is stationary. The current day temperature is quite similar to the previous day, thus exhibiting daily seasonality.
2. Ambient temperature: subsampled hourly.
3. Air passengers: The air passengers dataset provides a monthly total of US airline passengers from 1949 to 1960.
4. Solar irradiance: Just like the temperature dataset, solar irradiance provides samples every 10 minutes and also shows daily seasonality and stationary.
5. Solar irradiance: subsampled hourly.
6. Sunspots: The sunspots dataset provides a monthly mean of the seen sunspots from 1749 to 2017.
7. Electric power distribution: Electric power distribution dataset provides samples every hour.

Figure 6 shows the time series used for the experiments.

6 Results

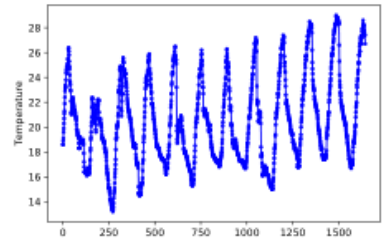
Table 1 shows the SMAPE values obtained from the tests. The first column shows the time series, the second column shows the results using the original data, and the third column shows the results using the original data plus the synthetic samples.

In addition, the second and third columns are subdivided into four columns: simple (results from the test without early-stopping nor regularization), early stopping (ES), regularization (λ_2), early stopping, and regularization (ES+λ_2). The results marked in red are the lowest error we find, and the green ones are the lowest error on the other category.

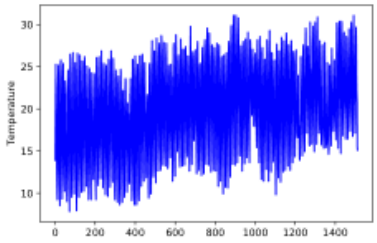
Table 1. Summary using SMAPE.

Time Series	Original				Data Augmentation			
	Simple	ES	λ_2	ES+λ_2	Simple	ES	λ_2	ES+λ_2
Temperature	1.16	1.16	0.81	0.99	0.69	0.71	0.67	0.61
Temperature (subsampled)	2.21	2.46	2.22	2.23	2.20	2.24	2.30	2.21
Airpassengers	0.45	0.45	0.40	0.45	0.33	0.39	0.32	0.32
Sunspots	23.24	23.19	23.15	23.58	22.68	22.67	22.79	23.80
Solar irradiance	57.65	59.23	57.60	59.17	56.89	59.40	57.13	57.34
Solar irradiance (subsampled)	59.60	58.78	59.42	59.51	58.85	58.354	58.75	58.48
Distribution data	12.09	12.67	12.07	11.99	11.94	11.90	11.98	12.42

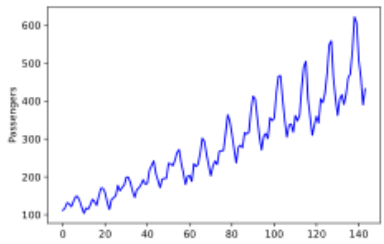
By observing Table 1, it can be noted that better results are obtained in most cases by adding synthetic samples to the training set. However, some considerations should be taken into account when using this Data Augmentation technique, such as the fact that two additional hyper-parameters are added to the



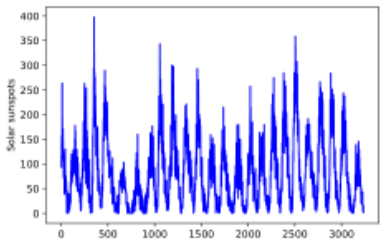
(a) Temperature.



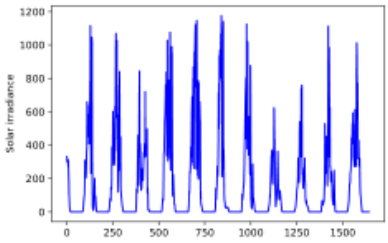
(b) Temperature subsampled.



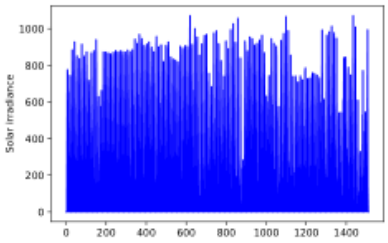
(c) Air passengers.



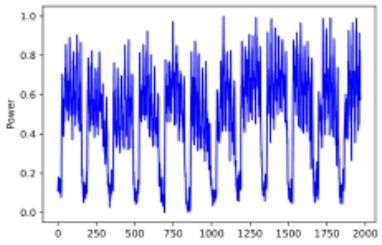
(d) Solar sunspots.



(e) Solar irradiance.

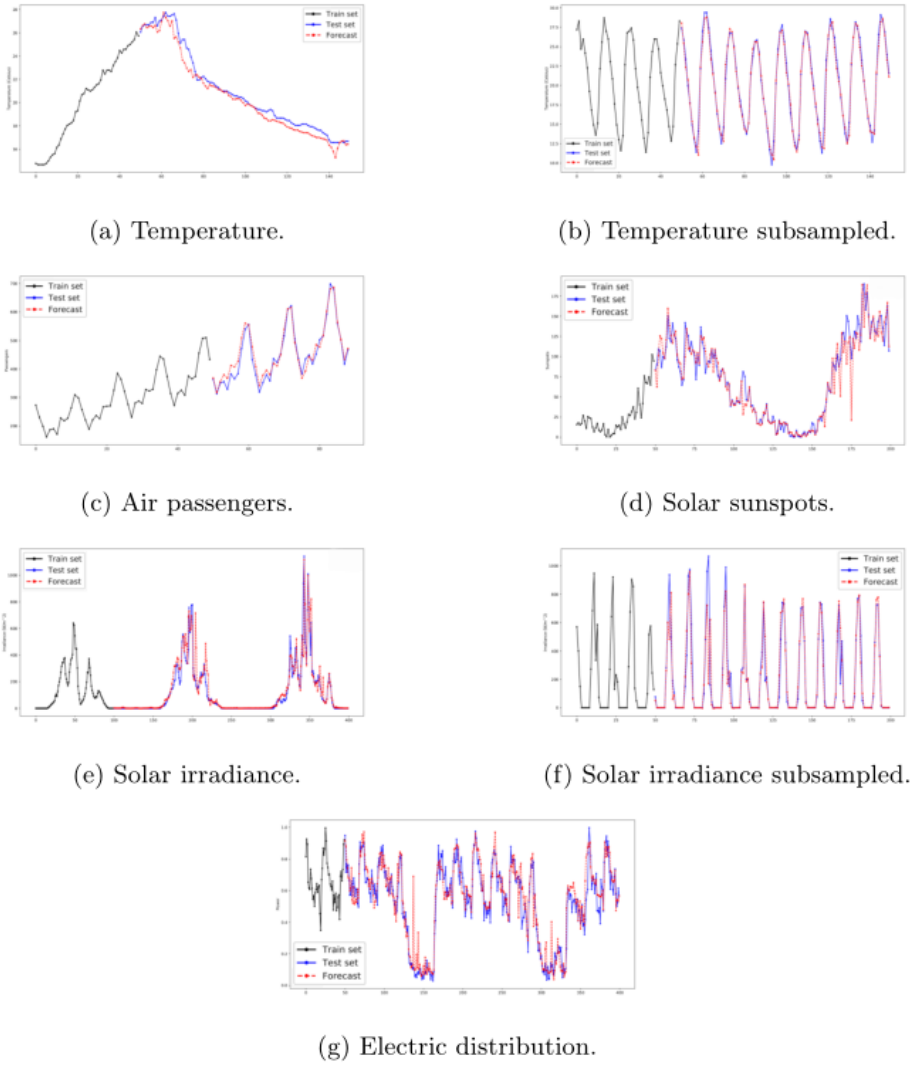


(f) Solar irradiance subsampled.



(g) Electric distribution.

Fig. 6. Plots of the time series used for testing.



(a) Temperature.

(b) Temperature subsampled.

(c) Air passengers.

(d) Solar sunspots.

(e) Solar irradiance.

(f) Solar irradiance subsampled.

(g) Electric distribution.

Fig. 7. Forecast plots. The black line is the train set, the blue curve is the test set and the red-dashed line is the forecast.

training process. These two hyper-parameters are the magnitude of the noise to be added and the number of noisy samples used in the creation of the augmented data sets.

The value of SNR varies depending on the time series. When considering the number of synthetic samples, something similar happens since each time series yields good results with a different number of samples. Besides, the extra

computational effort that is made by increasing the number of samples in the training set must be considered.

7 Conclusions

Regularization is a filter used to reduce signal noise; with the appropriate parameters, it is possible to have a noise-free signal with entropy reduction. In this case, a regularized model reduces the entropy of the model distribution, thus reducing overfitting.

We propose one method to introduce noise to the forecaster ANN training set. First, we convert the time series to a design matrix, by determining the reconstruction dimension of the underlying the process that produced it (m), together with a subsampling period (τ); sweeping the time series forms all possible delay vectors. The resulting design matrix maps the forecasting problem into a regression problem. Noise is then introduced to the design matrix, populating it with noisy delay vectors; noise is generated following a normal distribution $N(0, \sigma)$.

How much noise and how many noisy points we need to introduce to the training set are questions that were solved empirically. On one extreme, too little a σ introduces no noise; on the other end, too large a σ destroys the information the training set conveys; we performed several experiments and determined that a good SNR for the introduced noise was 30 dB. From there, we determined the appropriate value for σ. About the second question, if we introduce 0 noisy points, the training set remains unchanged.

Adding more noisy points improves the network generalization capabilities up to a certain point, where it stabilizes asymptotically to an error limit. Increasing the size of the training set is not free, it consumes time and memory resources in the training process. Executing different experiments, we determined that adding 3 noisy points to the training set achieves a good equilibrium between generalization and training time.

Finally, Table 1 shows that the introduction of noise in the training set outperforms other regularization techniques. A set of experiments were designed to compare the performance of DA against models regularized by Early Stopping, second-order Bayesian Regularization, and both. Noise regularization produced better results in the vast majority of the cases.

References

1. Palit, A., Popovic, D.: Computational intelligence in time series forecasting: Theory and engineering applications. Advances in Industrial Control, Springer London (2006)
2. Granger, C., Newbold, P., Shell, K.: Forecasting economic time series. Elsevier Science (2014)
3. Kirchgässner, G.,Wolters, J.: Introduction to modern time series analysis. Springer (2008)

4. Zhang, G.P.: Time series forecasting using a hybrid arima and neural network model. Neurocomputing, 50, pp. 159–175 (2003)
5. Xu Shuojiang, C.H.K., Tiantian, Z.: Forecasting the demand of the aviation industry using hybrid time series sarima-svr approach. Transportation Research Part E: Logistics and Transportation Review, 122(2) (2019)
6. Lemke, C., Gabrys, B.: Meta-learning for time series forecasting and forecast combination. Neurocomputing, 73(10-12), pp. 2006–2016 (2010)
7. Ayoubi, M.S.M., Sinsel, S.: Dynamic neural units for nonlinear dynamic systems identification. Lecture Notes in Computer Science, 930, pp. 1045–1051 (1995)
8. Brezak, D., Bacek, T., Majetic, D., Kazac, J., Novakovic, B.: A comparison of feed-forward and recurrent neural networks in time series forecasting. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1–6 (2012)
9. Flores, J., Graff, M., Rodriguez, H.: Evolutive design of arma and ann models for time series forecasting. Renewable Energy, 44, pp. 225–230 (2012)
10. Kantz, H., Schreiber, T.: Nonlinear time series analysis. 7, Cambridge University Press (2004)
11. Kamstra, M., Donaldson, R.G.: Forecast combining with neural networks. Journal of Forecasting, 15 (1996)
12. Khashei, M., Reza-Hejazi,S., Bijari, M.: A new hybrid artificial neural networks and fuzzy regression model for time series forecasting. Fuzzy Sets and Systems, 159 (2008)
13. de Oliveira, K.A., Vannucci, A., da Silva, E.C.: Using artificial neural networks to forecast chaotic time series. Physica A: Statistical Mechanics and its Applications, 284 (2000)
14. Shirvany, Y., Hayati, M., Moradianc, R.: Multilayer perceptron neural networks with novel unsupervised training method for numerical solution of the partial differential equations. Applied Soft Computing, 9 (2009)
15. Pijanowski, B.C., Brown, D.G., Shellito, B.A., Manik, G.A.: Using neural networks and gis to forecast land use changes: A land transformation model. Computers, Environment and Urban Systems, 26 (2002)
16. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems (MCSS), 2 (1989)
17. Nelson, M., Hill, T., Remus, W., O'Connor, M.: Time series forecasting using neural networks: Should the data be deseasonalized first?. Journal of forecasting, 18 (1999)
18. Berzal, F.: Redes neuronales and deep learning. Fernando Berzal (2018)
19. Aggarwal, C.: Neural networks and deep learning: A textbook. Springer International Publishing (2018)
20. Géron, A.: Hands-on machine learning with scikit-learn and tensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media (2017)
21. Perez, L., Wang, J.: The effectiveness of data augmentation in image classification using deep learning (2017)
22. Zhang, X., Wang, Z., Liu, D., Ling, Q.: Dada: Deep adversarial data augmentation for extremely low data regime classification. In: ICASSP IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2807–2811 (2019)
23. Hassan, I.F., Germain, F., Jonathan, W., Lhassane, I., Pierre-A., M.: Data augmentation using synthetic data for timeseries classification with deep residual networks (2018)

24. Brown, W.M., Gedeon, T.D., Groves, D.I.: Use of noise to augment training data: A neural network method of mineral–potential mapping in regions of limited known deposit examples. Natural Resources Research, 12(2), pp. 141–152 (2003)
25. Velázquez,V.M.T.: Modelado del error de predicción en series de tiempo basado en la calidad de sus datos. Master's thesis, Universidad Michoacana de San Nicolás de Hidalgo, 8 (2019)
26. Viswanathan, M.: How to generate awgn noise in matlab/octave (without using in-built awgn function) (2015)