FORECASTING SUNSPOT NUMBER TIME SERIES WITH AUTOREGRESSIVE MODELS

A Thesis

by

JACINTO DE LA CRUZ HERNANDEZ

Submitted to Texas A&M International University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

December 2019

Major Subject: Mathematics

FORECASTING SUNSPOT NUMBER TIME SERIES WITH AUTOREGRESSIVE MODELS

A Thesis

by

JACINTO DE LA CRUZ HERNANDEZ

Submitted to Texas A&M International University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Chair of Committee,	Runchang Lin, Ph.D.
Committee Members,	Saqib Hussain, Ph.D.
	Tariq H. Tashtoush, Ph.D.
	Rohitha Goonatilake, Ph.D.
Chair of Department,	Rohitha Goonatilake, Ph.D.

December 2019

Major Subject: Mathematics

ABSTRACT

Forecasting Sunspot Number Time Series with Autoregressive Models (December, 2019)

Jacinto de la Cruz Hernandez, B.S., Texas A&M International University;

Chair of Committee: Runchang Lin, Ph.D.

Researchers in many fields share a deep interest in the sunspot activity of the Sun. This kind solar activity has many consequences for human interests, and thus, it is important to study the Sun's behavior and be able to predict future sunspot appearances. This task is a problem of time series forecasting. Some of the most popular methods used to forecast time series data are autoregressive (AR) models that predict future data points using a linear combination of previous values. Then, there are neural networks, a popular new tool for regression that can perform with outstanding results. These machine learning models have been shown to forecast time series successfully. In this thesis we use a variety of neural network architectures based on the classic AR models to predict future values of sunspot activity.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

1 INTRODUCTION [1]

A sunspot is a dark region on the Sun's surface. These sunspots tend to appear in groups along with the presence of strong magnetic fields. These fields can produce solar flares, creating what is called "space weather." Sunspots can then be used to predict the level of solar activity and space weather which can pose a hazard to some technologies. This makes sunspots of key importance to researchers, and when we track the number of sunspots that appear every year, these sunspots become part of a time series (Boteler, 2013).

Time series are collections of observations made over time, and they are crucial to understand as they occur in all kinds of fields. Economists encounter time series when they examine a nation's GDP per year, and political scientists do too when they study the voting patterns of citizens by election cycle. Time series are studied in virtually every field, and as such, their analysis has not been immune to the popularity burst of machine learning methods (Peña et al., 2001; Gupta and Smith, 2003).

The incorporation of more and more fields under the umbrella of machine learning has made a myriad of new methods available to the time series forecaster. Perhaps most popular among these are neural network models like the multilayer perceptron (MLP). These are so called due to their structure being based on a simplistic model of how a brain works. Artificial "neurons", or perceptrons, take inputs scaled by some weights and pass them through an activation function that stands in for a brain neuron's activation. Stacking layers of these perceptrons on top of each other simulates the connections between neurons in the brain. While incredibly simple in comparison to a natural brain's functioning, the predictive power of these models seems to be rivaled only by their popularity. MLPs are now found and successfully used throughout all fields of research, including time series analysis.

1. The journal model used is from the *Journal of Machine Learning Research*

In this thesis, we will explore the history of the models used attempting to forecast the sunspot number time series. Then, we will apply a variety of neural network architectures to compare how well they forecast time series data in comparison to classic autoregressive (AR) models. In order to get an interesting comparison, the networks will be autoregressive, meaning that they will forecast future values of the time series given a set of past values as an input. Section 2 of this thesis, Theoretical Framework, will discuss these models and their development in more depth. This section will also include previous literature on the subject used to guide our evaluations. Section 3, Methodology, will detail how the exploration of the topic will take place and what exactly is hoped we will achieve. Here, we discuss what measures we will use to evaluate models, what models will be evaluated, and the data set to be modelled. Finally, Section 4, Findings, discusses the results of the experiment.

2 THEORETICAL FRAMEWORK

In this section, we introduce some of the concepts and models necessary to engage with the intended research. Then, we will discuss the history of sunspot series forecasting and some of the models used by other researchers.

2.1 Time Series

A time series is a sequence of observations recorded over time, $\{y_t\}$, for time $t \in T$, where T is the period of time we are interested in (Peña et al., 2001). Here, time can be measured in any of a variety of units, such as seconds, hours, days, weeks, years, decades, etc. Also, we note that while a time series can record qualitative observations, such as whether a light bulb is on or off at certain hours of the day, we will deal only with quantitative data (i.e. when the observations y_t are real numbers).Now, time series may often posses some characteristics which will be important for us to note when forecasting. These are:

- *Trend:* Here loosely defined as "long term change in the mean" (Chatfield, 1975).

- *Seasonal effect and other cyclic changes:* This encompasses the variation over fixed periods that a time series might exhibit. Likewise, oscillations without a fixed period but that are reasonably predictable will fall under this category. (Chatfield, 1975).

These categories are used to define a *stationary* time series, which is, broadly speaking, a time series with no trend and no strictly periodic variations. For the purposes of forecasting, many models require stationary time series; however, it is usually sufficient to achieve *second-order stationarity*, which means that the variance and mean of a time series are constant. We will preprocess our data accordingly when dealing with *nonstationary* time series (Chatfield, 1975; Gupta and Smith, 2003). Lastly, for the purposes of this thesis, we will deal only with univariate time series, that is time series that consist of observations of a single variable.

2.2 Autoregressive Integrated Moving Average Models

The goal of time series forecasting is to predict future values in the sequence using the observed values. The forecaster identifies a pattern in the data and extrapolates it to the future. The family of ARIMA forecasting models, or *autoregressive integrated moving average* models, are some of the most popular forecasting models available for time series. These are combinations of other more fundamental types of models. There exist various libraries of code that can implement this family of models already. Given that these models are only of interest to us as a benchmark to compare our neural network models with, only a brief description of the AR models within this family will be presented here.

2.2.1 Autoregressive Models

Autoregressive models, also known as AR models, assume that terms in a time series are autoregressive, that is, that future terms in the sequence can be predicted by a linear regression of previous terms (Peña et al., 2001). An autoregressive process of order p follows the following form:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + a_t, \quad t > p. \tag{1}$$

Here, the t^{th} term in the sequence is being determined by some linear combination of the previous p terms and a random process a_t assumed to have mean zero. As mentioned above, we will preprocess our data before fitting this model.

Assuming a time series behaves as in (1), we call this the autoregressive model of order p, or AR(p). ARIMA models of order $(p, 0, 0)$ correspond to the AR(p) model. These models will help us develop a neural network architecture fit to forecast time series.

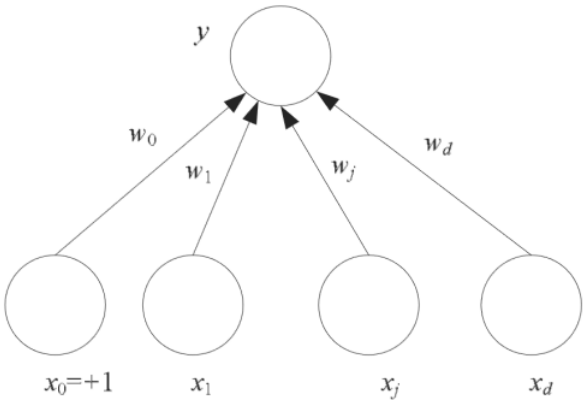


Figure 1: Perceptron with 3 inputs. Reprinted from *Introduction to Machine Learning* (p 237), by Ethem Alpaydin, 2010, Cambridge, MA: The MIT Press

2.3 Artificial Neural Networks

Artificial neural networks (ANNs) are so called because they are built to emulate the way neurons connect and behave in the human brain. Like the human neuron is to the human brain, the artificial neuron, or *perceptron*, is the basic building block to the ANN.

The perceptron is a processing unit, generally of the form:

$$\mathbf{x} \mapsto \sigma(\mathbf{w}^T\mathbf{x}), \tag{2}$$

where $\mathbf{x}$ is an input vector in $\mathbb{R}^{d+1}$, $\mathbf{w}$ is a weight vector in $\mathbb{R}^{d+1}$, σ is one of a family of activation functions used to emulate the activation of a neuron, and the T superscript denotes the transpose. Figure 1 illustrates the mapping described by (2), where each connection from an input to the perceptron represents the multiplication by a weight. Then, the perceptron adds all the products and applies and activation function σ to emulate the activation of a human neuron. For d inputs, we let the input vector to be in $\mathbb{R}^{d+1}$ by defining a 1 as its $d+1$ coordinate. This allows us to represent the multiplication of each *weight*, w_i for $i > 0$, to each input plus the *bias*, w_0, as a single inner product (Alpaydin, 2010; Peña et al., 2001). Common activation functions, σ, used include the sigmoid, or logistic, function, the hyperbolic tanget, and the ReLU function (or rectified linear unit).

2.3.1 Multilayer Perceptron

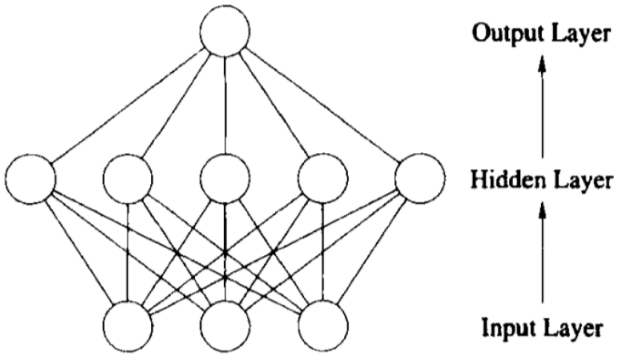


Figure 2: Multilayer perceptron with three inputs, one hidden layer of 5 nodes, and one output: This representation does not include an explicit node to represent the bias as an input as in Figure.1. Reprinted from *A Course in Time Series Analysis* (p 351), by Daniel Peña et al., 2001, New York, NY: John Wiley & Sons, Inc.

The most basic form of ANN is the multilayer perceptron, or MLP. Its most basic form is the single hidden layer MLP. In this form, a layer of h perceptrons, what we call a hidden layer, is formed between the input layer and the output layer. Each node in the hidden layer computes the same kind of transformation as in (2). The output layer may compute only a linear combination of the hidden layer outputs, or it may pass this through a final activation function, depending on whether the task is classification or regression. The MLP in Figure 2 could then be represented as:

$$z = \sigma_2 \left(\sum_{j=1}^{5} \beta_j \sigma_1 \left(\sum_{i=i}^{3} w_{ji} x_i + w_{j0} \right) + \gamma_0 \right), \tag{3}$$

where each w_{ji} is the weight of input i corresponding to node j of the hidden layer and w_{j0} is the bias for node j, each β_j is the weight corresponding to each node of the hidden layer connecting to the output layer, and γ_0 is the bias of the output layer. In general, then, each output of an MLP with one hidden layer of h nodes, d inputs, and n outputs, can be

represented as:

$$z_k = f_k(\mathbf{x}, W) = \sigma_2 \left(\sum_{j=1}^{h} \beta_{kj} \sigma_1 \left(\sum_{i=i}^{d} w_{ji} x_i + w_{j0} \right) + \gamma_{k0} \right), \quad 1 \leq k \leq n. \tag{4}$$

Hence, the MLP is just the function $f = (f_1, \ldots, f_n)$ where $\mathbf{x}$ is our input vector, and W is our collection of parameters. We call this function f, the network's *architecture*, and it defines the structure of the network.

To train the network's parameters, we make use of *gradient descent*. To do this, a *loss function* needs to be defined. There are various loss functions available, but the appropriate loss function must be chosen according to the task the network must perform. In the case of regression, for example, the mean square error (MSE) works best as a loss function. Once the parameters have been initialized randomly, and the inputs have been passed through the network, we can calculate the loss:

$$L\left(f(\mathbf{x}, W)\right) = \frac{1}{n} \sum_{i=i}^{n} (y_i - z_i)^2 = \frac{1}{n} \sum_{i=i}^{n} (y_i - f(\mathbf{x}, W))^2. \tag{5}$$

Since the loss function quantifies how far a prediction is from the true value, the goal is then to minimize the loss based on the parameters in W. The most common way of doing this is through gradient descent. The negative gradient of a function points in the direction of steepest descent. This means that the component of the gradient corresponding to each parameter, tells us how much to change our parameter's value by, in order to reduce the value of the loss function. this leads to the following parameter update iteration for a generic parameter ω:

$$\omega^{t+1} = \omega^t - \alpha \frac{\partial}{\partial \omega} L(f(\mathbf{x}, W)). \tag{6}$$

Here, α is called the *learning rate*, which is chosen in order to guarantee proper convergence of the algorithm. If the learning rate is too large, the updates might overshoot the minimum and the process will diverge. If it is too small, the gradient descent algorithm will

update very slowly (Alpaydin, 2010). Once the parameters are all updated, they are fed back into the network along with the inputs and the whole process is repeated, until our loss converges to some local minimum. It is worth noting that convergence to a global minimum is not guaranteed, and it is very much dependent on the initialization of the parameters.

2.3.2 Recurrent Neural Networks

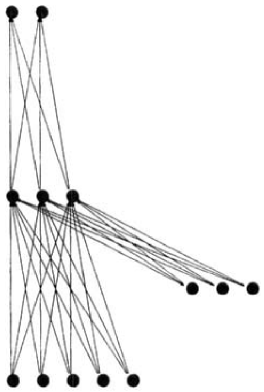


Figure 3: Elman type Recurrent Neural Network. Reprinted from *A Course in Time Series Analysis* (p 358), by Daniel Peña et al., 2001, New York, NY: John Wiley & Sons, Inc.

Recurrent nerual networks (RNNs) are ANNs with specialized nodes, or perceptrons. An RNN's hidden layer takes its previous outputs, and passes them through the layer again. This can be expressed mathematically if we let the MLP layer's output, as defined before, be $\mathbf{u} = (u_1, \ldots, u_h)$ and each $u_j = \sigma\left(\sum_{i=i}^{d} w_{ji}x_i + w_{j0}\right)$ is the output of node j. Then, a recurrent network's node output at time t is:

$$u_j^t = \sigma\left(\mathbf{w}^T\mathbf{x}^t + \mathbf{v}^T\mathbf{u}^{t-1}\right). \tag{7}$$

Here, the parameter and input vectors are defined as in (2) for ease of reading and writing. While $\mathbf{v}$ is a new parameter vector being applied to the hidden layer's output from the previous time. RNNs are likewise trained by gradient descent.

2.3.3 Autoregressive Neural Networks

The family of ANNs called autoregressive neural networks (NNARs) are structured to act similarly to a classic AR model. These networks can be given a variety of architectures, such as that of a basic feed-forward MLP or an RNN. What distinguishes this family of networks is that they take the AR model's characteristic of outputing a value for some time t based on p previous inputs. This means that, besides the number of hidden layers, hidden units, and their connections, the number of lags, p, used to predict a future value determine the architecture of the network. In general, these networks have the form:

$$y_t = f(y_{t-1}, \ldots, y_{t-p}, W). \tag{8}$$

As with AR models NNARs with one hidden layer can be specified with a number of lags, p, along with a number of hidden nodes, k, by NNAR(p,k). Such that an autoregressive neural network with one hidden layer, nine lags, and six nodes in its hidden layer is denoted by NNAR(9,6).

2.4 Activation Functions

There is a large variety of activation functions available to be used in neural network architectures. The original motivation for activation functions was to emulate the activation of a brain neuron. Organic neurons communicate through electrical signals. Now, these signals must pass a certain threshold to activate the neuron to perform its task. Because of this, the fundamental activation function is a threshold function defined as:

$$s(x) = \begin{cases} 1 & , \text{x} > 0 \\ 0 & , \text{otherwise} . \end{cases}$$

However, this activation function is outdated. There are many smooth functions that can

perform approximately the same goal. Depending on the task at hand, different activation functions may perform better than others.

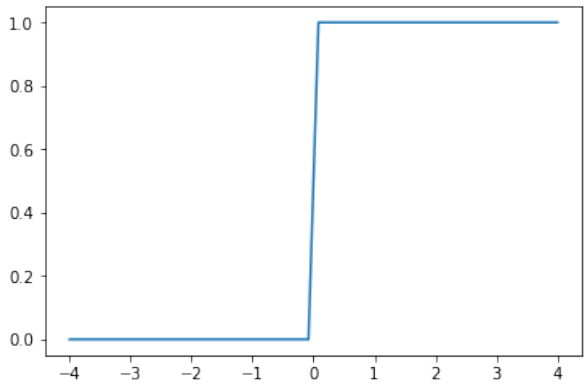


Figure 4: The Threshold function

2.4.1 Sigmoid

The sigmoid activation function is one of the most common activation functions used in ANNs. It is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{9}$$

Since the range for this function is the interval $(0, 1)$, sigmoid is particularly useful when the output needed is a probability. Its primary advantage is the introduction of nonlinearities into the neural network. Without an activation function, a neural network would be a series of linear transformations and thus would not have the same approximation power that neural networks with nonlinearities posses.

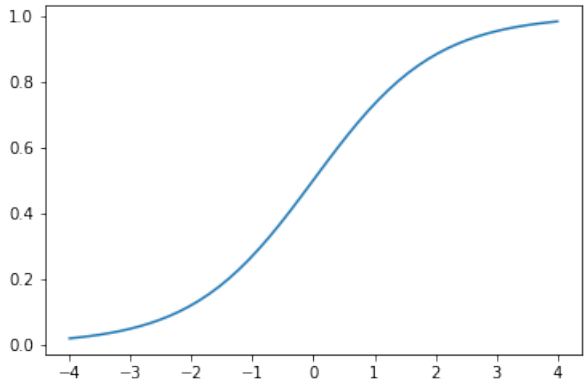


Figure 5: The Sigmoid function: It closely resembles a smooth version of the threshold function

2.4.2 ReLU

Currently, the most common activation function of all is the Rectified Linear Unit, or ReLU. This is defined as:

$$ReLU(x) = max(0, x). \tag{10}$$

This has certain advantages over the sigmoid function when computing the gradient descent algorithm for deep networks; however, for our purposes, its main advantage comes in the form of its range. By including values greater than 1, this activation function performs better at tasks of regression. Nonetheless, its nonlinear structure still gives the same advantage that the sigmoid did by introducing nonlinearities into the network. Its primary disadvantage is that negative values all get mapped to zero. This hurts our regression model if we are trying to forecast values below zero.

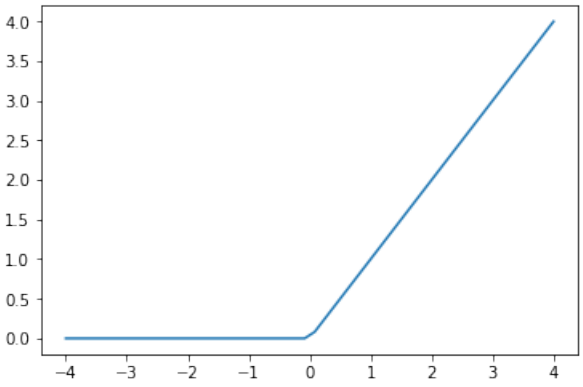


Figure 6: The ReLU function: It sends all negative values to zero

2.4.3 Leaky ReLUs

A common modification to circumvent the ReLU function's issues with negative values is to allow some of the negative value through the function. This gives us what is called a Leaky ReLU, defined as:

$$LeakyReLU(x) = max(0, x) + \alpha * min(0, x), \quad \alpha \in \mathbb{R}. \tag{11}$$

Three different activation functions of this form are defined based on the value of alpha. A true Leaky ReLU will have the standard fixed value of $\alpha = 0.01$. A Randomized ReLU, or RReLU, will have a fixed α selected at random between some chosen upper and lower bounds. Lastly, a Parametrized ReLU, or PReLU, starts with an initial value for α, either chosen or randomized, and learns the optimal value through gradient descent. The same way the gradient for the selected loss function is used to calculate the optimal value for the network's weights, the gradient of the loss function is calculated in terms of the parameter α.

Because of its range, ReLU has trouble predicting negative values for regression problems, but the Leaky ReLUs fix this issue by allowing some portion of negative values through.

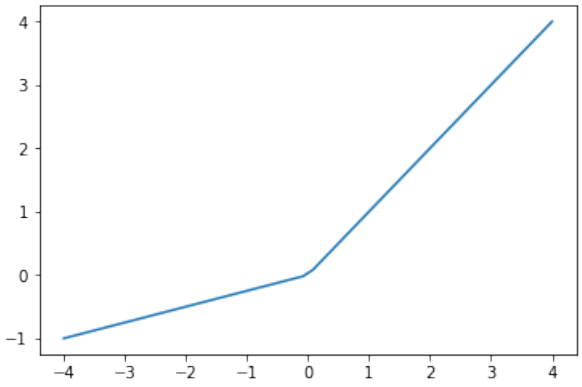


Figure 7: The Leaky ReLU function with $\alpha = .25$

2.5 "Forecasting Sunspot Numbers Using Neural Networks"

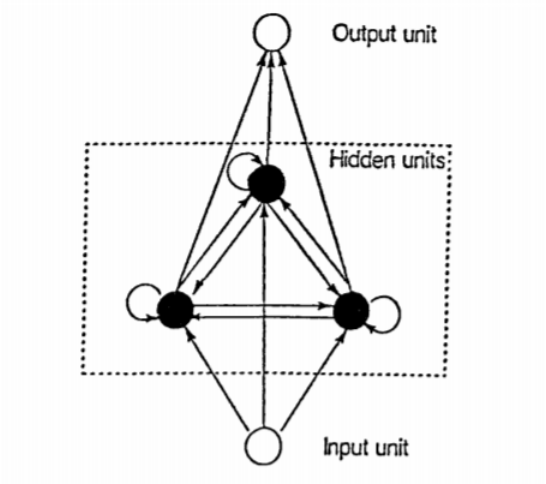


Figure 8: Fully connected MLP: In a fully connected hidden layer, each node is connected laterally to each other and to themselves. Reprinted from "Forecasting Sunspot Number Using Neural Networks" by Li et al., 1990, Syracuse, NY: Syracuse University

In "Forecasting Sunspot Number Using Neural Networks," published in 1990, Li et al. forecast future values in the sunspot time series using a feed-forward autoregressive neural network. Their network architecture features one fully connected hidden layer, such as the one seen in Figure 8. Their test assessed networks employing between two to eighteen hidden nodes, with eleven nodes performing best. As a baseline, Li et al. use four models from the previous literature to compare their results. These are all AR models that use varying

lag values, some going as far back as 18 periods.

Li et al. evaluated their models using the MSE as their measure. Four tests were carried out to evaluate performance. The first measured the models' fit on the training data, with the network achieving as low as a 79.73 MSE and the best of the baseline models coming in at 229. The second evaluates the models' forecasts one year in the future carried out over a period of 12 years. This test always uses observed data as inputs for the next year's prediction. The network's best result was an MSE of 120.98 for this challenge. The third test has the models forecasting 12 years into the future using only the training data. This means that when forecasting past the first year after the data, the models use their previous outputs as inputs. As expected, this yields higher errors, but it provides a good measure of what the performance of a model would be like in actual applications. This test yields a low MSE of 342.6 for the neural network. Finally, test four produces forecasts for 55 years into the future. Here, the network performed best with an MSE of 343.27.

NNARs performed more favorably than AR models consistently over all of the tests. Thus, the authors were successful in demonstrating that neural networks can produce quite satisfactory results in the forecasting of time series values.

2.6 "Forecasting Sunspot Time Series Using Deep Learning Methods"

Similarly to Li et al., Atici and Pala set out to evaluate the performance of deep-learning (DL) models at the task of forecasting time series and compare them with more classic methods such as the ARIMA models among others. In "Forecasting Sunspot Time Series Using Deep Learning Models," Atici and Pala forecast the sunspot time series using DL algorithms such as the long short-term memory (LSTM) and NNAR models.

The LSTM family of machine learning algorithms is a subset of recurrent neural networks. These models use the recurrent connections in their structure to memorize sequences

of data. LSTMs can retain data over longer periods of time than usual RNNs (Atici and Pala, 2019). This allows LSTMs too outperform standard feed-forward networks, such as NNARs, in complex tasks with temporal effects (Gers, Eck, and Schmidhuber, 2002).

Model performance was measured in this article by the root mean square error (RMSE), which is the square root of the MSE value. Two LSTM models were tested, along with an NNAR, an ARIMA, a Mean method, a Naïve method, a Seasonal Naïve method, and a Drift method model. The LSTM models outperformed all others with RMSEs of 35.9 and 36.9, with the NNAR coming in second with an RMSE of 42.41. The ARIMA model achieved an RMSE of 45.60 with all other methods falling behind. These results further illustrate the competence that machine learning models in general, and NNARs in particular, display in time series forecasting.

3 METHODOLOGY

The data set to be used in this experiment is the annual sunspot number time series. Predicting some future value of sunspot appearances is crucial in order to protect sensitive technologies from space weather. Furthermore, due to its availability and popularity, this data set provides a variety of literature to reference on the subject (Li et al., 1990; Peña et al., 2001; Werner, 2012). This data set also has idiosyncrasies that make it an excellent yardstick for new statistical modeling and forecasting techniques (Li et al., 1990).

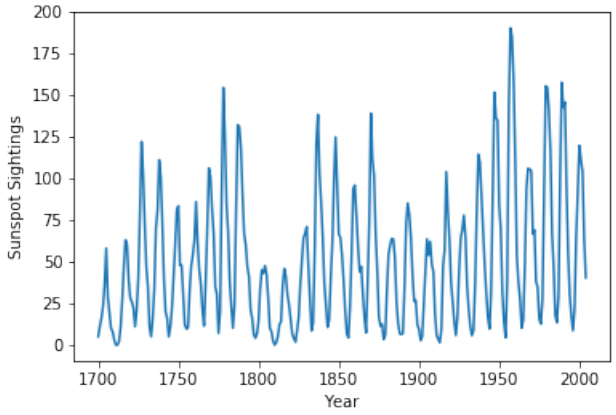
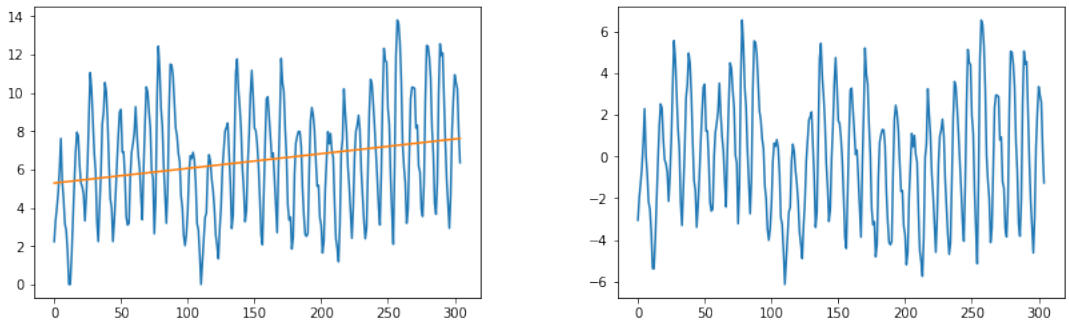


Figure 9: Annual sunspot data since 1700

The literature suggests that the best fit AR model for this series is the 9-lag model (Li et al., 1990; Peña et al., 2001; Werner, 2012). However, because of their fairly low cost of calculation, we will test models with lags varying from 1 to 12. Similarly, we will use the same range of lags for the ARNN models to be tested.

The method used to fit the AR models will be as suggested in Section 2. First, the data set will be detrended, and a square root transformation will be applied in order to achieve a second-order stationarity for the time series. This is following the procedure in (Werner, 2012). The data points, 305 in total, will be split into a training set and test set of 240 and 65 points respectively. The model will then be fit to the training data using



(a) (b)

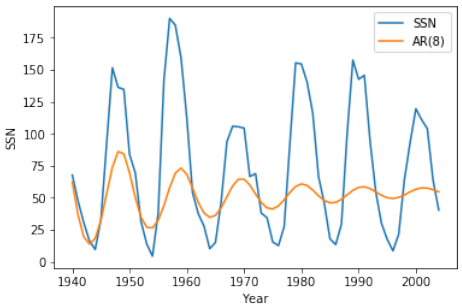
Figure 10: (a) Square root and (b) detrended Sunspot Data

the statsmodels python module's ARMA function. A simple loop will allow us to swiftly fit AR models for lags from 1 to 12. At the same time the loop will calculate the MSE for the test set only. The model will predict the values for the test based on its own outputs. This is similar to the evaluation carried out in (Li et al., 1990). In their third test, they predict the values for the test set by passing the training set values through the model until and subsequently using the outputs of the model as inputs. When forecasting actual future values, researchers won't have observed values for times further along than the present. In practice, forecasting future points will require using forecasted outputs to predict even more values. Thus, we will use this method to forecast values in the test set. This will give us a clearer idea of how far into the future can we reliably forecast values for this data set. The models with lowest MSEs will be selected for closer examination. At this point we will look at the percent error of individual outputs to get a closer look at the accuracy of predictions. Since MSE is aggregated over all the test set, we may observe a model with lower MSE across the entire set but higher percent errors for the first and most important forecasts.

The neural network architectures to be tested will be single hidden layer NNARs with simple feed-forward connections. The number of hidden units will vary between four, six, eight, and eleven. The eleven is chosen due to (Li et al., 1990). Li et al. found that they achieved their best results with 11 hidden units in their fully connected RNN hidden layer.

Because ANN activation functions have values centered around 0, it is common practice to standardize the data according to its mean and standard deviation. The standardized data will be split between training and test set in the same manner, and the training set will be used to fit the NNAR models. We will again be using the outputs of the model to predict the values of test set. Like with the AR models, a simple loop will allow us to test these NNAR architectures with lag values ranging from 1 to 12. We will use the PReLU activation function in our networks and a stochastic gradient descent algorithm to update our weights. A sequence of tests led to the best results with 300 training epochs. Less epochs led to insufficient training, while more epochs led to overfitting to the training data. Both cases led to worse performance with the test data. Like with the AR models, the lowest MSE networks will be selected for closer examination, comparing individual percent errors to those of the other networks and the AR models.

4 FINDINGS



(a)

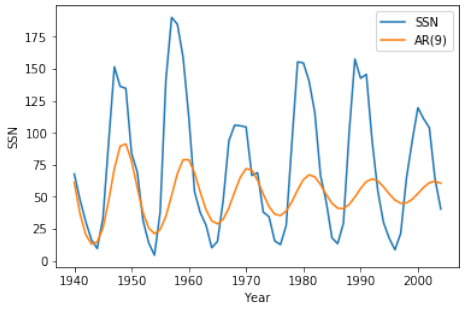
	year	SSN	AR(8)	Percent Error
240	1940-01-01	67.8	62.243114	8.195997
241	1941-01-01	47.5	36.594332	22.959300
242	1942-01-01	30.6	19.432348	36.495593
243	1943-01-01	16.3	13.867203	14.925135
244	1944-01-01	9.6	18.307532	-90.703460
245	1945-01-01	33.2	32.396075	2.421459
246	1946-01-01	92.6	53.234520	42.511317
247	1947-01-01	151.6	73.887384	51.261620
248	1948-01-01	136.3	86.123783	36.813072
249	1949-01-01	134.7	84.279912	37.431394

(b)

Figure 11: (a) AR(8) Forecast (b) and percent error on test set per value

The testing with the AR models showed an unexpected, but uninteresting, result. The best performing AR model was the one with 8 lags with an MSE of 2320.73. All of the literature suggested that the best model would be the one with 9 lags, and we can even see that our AR(9) model fit close values for most of the significant lag coefficients as the model in (Werner, 2012). Nonetheless, neither the MSE or the percent errors are significantly different to the AR(9) model to warrant a revising of the literature. Our AR(9) model scored an MSE of 2352.82, just below the AR(8). The discrepancy is most likely due to the large size of the testing set, and it is not significant enough to claim one model performs much better than the other. They seem to be quite interchangeable up to the fifth forecast where the AR(9) significantly outperforms the AR(8) model; however, they both have an unacceptably high error.

In both cases, the only truly acceptable forecast seems to be the first value. Trying to forecast more than a year into the future leads to errors of 20% of the actual value and more.



(a)

	year	SSN	AR(9)	Percent Error
240	1940-01-01	67.8	61.258289	9.648541
241	1941-01-01	47.5	36.590520	22.967326
242	1942-01-01	30.6	20.546750	32.853759
243	1943-01-01	16.3	13.042989	19.981660
244	1944-01-01	9.6	14.740687	-53.548821
245	1945-01-01	33.2	25.389242	23.526378
246	1946-01-01	92.6	47.004894	49.238775
247	1947-01-01	151.6	72.025256	52.489936
248	1948-01-01	136.3	89.476270	34.353434
249	1949-01-01	134.7	91.247534	32.258698

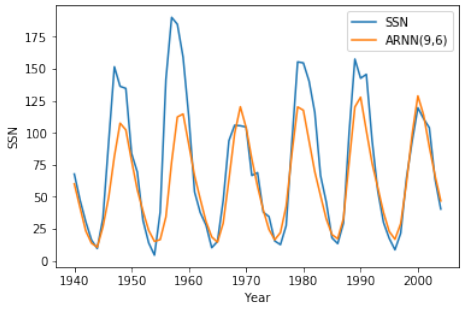
(b)

Figure 12: (a) AR(9) Forecast (b) and percent error on test set per value

Table 1: Coefficients of lag terms for AR(9) models

Lag Term	Results	Werner, 2012
1	1.23	1.22
2	-0.47	-0.50
3	-0.13	-0.10
4	0.20	0.20
5	-0.17	-0.22
6	-0.06	0.06
7	0.20	0.08
8	-0.18	0.16
9	.23	0.27

Now, for our neural networks, it seems that the two best fits are the 9 lag model with 6 hidden nodes and the 9 lag model with 11 hidden nodes. Their MSEs were of 873.37 and 1083.86 respectively. It is interesting that it was the models with 9 lags that performed best for our neural networks. Li et al. only tested models with 9 lags while they varied the number of hidden nodes in their network, while Peña et al. tested various lags and arrived at their best scores with 6 and 7 lags. Our results would suggest that there truly is a strong autocorrelation between a term of this series and the 9 terms that precede it.

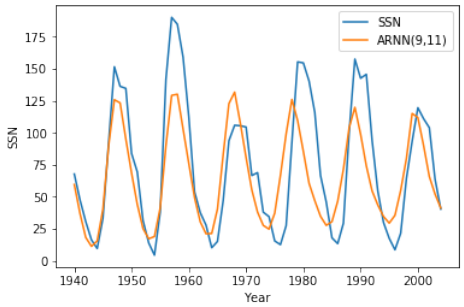


(a)

	SSN	ARNN(9,6)	Percent Error
0	67.800003	60.153023	11.278731
1	47.500000	41.571739	12.480549
2	30.600000	23.382069	23.588013
3	16.299999	13.798351	15.347534
4	9.600000	10.615711	-10.580320
5	33.200001	26.533676	20.079290
6	92.599998	49.542816	46.498035
7	151.600006	81.635773	46.150547
8	136.300003	107.524017	21.112242
9	134.699997	102.279465	24.068695

(b)

Figure 13: (a) NNAR(9,6) Forecast (b) and percent error on test set per value



(a)

	SSN	ARNN(9,11)	Percent Error
0	67.800003	59.571278	12.136763
1	47.500000	36.856457	22.407459
2	30.600000	18.243887	40.379456
3	16.299999	11.263214	30.900522
4	9.600000	15.529682	-61.767517
5	33.200001	40.259186	-21.262606
6	92.599998	89.957283	2.853904
7	151.600006	125.851028	16.984814
8	136.300003	123.338951	9.509209
9	134.699997	95.130295	29.376169

(b)

Figure 14: (a) NNAR(9,11) Forecast (b) and percent error on test set per value

Upon the closer inspection, when comparing the percent error for each term, we can see that initially the NNAR(9,6) model is most reliable. It outperforms every other model in that it results in a percent error below 25% for the first six forecasts. Compare that to the two predictions we get from both of the AR models. Now, while the NNAR(9,11) model produces poorer results for the first six values, we can see that it aligns much more closely to the observed values starting with the seventh prediction. This is the reason we get such a close MSE value for the two models, even though the NNAR(9,6) seems to perform much better at first.

5 Conclusion

Both, the literature and our results, show the significant advantage in using NNARs over more classic AR models to predict time series in general, and the sunspot series in particular. Our best performing NNAR(9,6) was able to produce reliable forecasts up to six years into the future, compare this to the AR(9) model in (Werner, 2012) where only two to three future values had an acceptable error of below 30%. These models will be of great importance in anticipating the way space weather may come to affect our technologies in the future.

REFERENCES

Alpaydin, Ethem. *Introduction to Machine Learning.* The MIT Press, 2010.

Atici, R. and Pala, Z. "Forecasting Sunspot Time Series Using Deep Learning Methods." Solar Physics. Springer, 2019, https://doi.org/10.1007/s11207-019-1434-6

Boteler D.H. "Sunspots." Encyclopedia of Natural Hazards. Encyclopedia of Earth Sciences Series. Springer, 2013, https://doi-org.tamiu.idm.oclc.org/10.1007/978-1-4020-4399-4_337

Chatfield, C. *The Analysis of Time Series: Theory and Practice.* Chapman and Hall, 1975.

Gers F.A., Eck D., Schmidhuber J. "Applying LSTM to Time Series Predictable Through Time-Window Approaches." In: Tagliaferri R., Marinaro M. (eds) *Neural Nets WIRN Vietri-01.* Springer, 2002.

He, Kiaming, et al. "Deep Residual Learning for Image Recognition." ArXiv, 2015, arxiv.org/abs/1512.03385.

Li, Ming, Mehrotra, Kishan, Mohan, Chilukuri K., and Ranka, Sanjay. "Forecasting Sunsport Numbers Using Neural Networks." Syracuse University, 1990, surface.syr.edu/cgi/viewcontent.cgi?article=1056context=eecs_techreports&fbclid=IwAR1_Bdk6sPR2cDnvVlZs2IlH0yO18W7GSphBUZFIQoSBmuDpH7CG82gt8l0

Peña Daniel, et al. *A Course in Time Series Analysis.* John Wiley & Sons, 2001.

Smith, Kate A., and Jatinder N. D. Gupta. *Neural Networks in Business: Techniques and Applications.* IRM Press, 2003.

Werner, R. "Sunspot Number Prediction by an Autoregressive Model." Sun and Geosphere, 2012.

APPENDIX A

NNAR(p,6) CODE

```
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

data=pd.read_csv("D:\\CODING\Practice\Sunspots\sunspots.dat", sep='\s+', header=None)

x=torch.tensor(data[:][0], dtype=torch.float)
y=torch.tensor(data[:][1], dtype=torch.float)

means=y.mean()
stds=y.std()
nrml_y=(y-means)/stds



x_train=x[:240]
y_train=nrml_y[:240]
x_test=x[240:]
y_test=y[240:]



losses=[]
m=nn.PReLU()
for p in range(12):
```

```
class Model(nn.Module):
    def __init__(self, input_size, H1, output_size):
        super().__init__()
        self.linear = nn.Linear(input_size, H1)
        self.linear2 = nn.Linear(H1, output_size)
    def forward(self, x):
        xout=x[:p+1]
        for j in range(240-p-1):
            xtemp=x[j:j+p+1]
            xtemp = m(self.linear(xtemp))
            xtemp = m(self.linear2(xtemp))
            xout=torch.cat((xout,xtemp))
        for i in range(65):
            xtemp=xout[240-p-1+i:240+p+i]
            xtemp = m(self.linear(xtemp))
            xtemp = m(self.linear2(xtemp))
            xout=torch.cat((xout,xtemp))
        return xout


torch.manual_seed(1)
model = Model(p+1, 11, 1)
criterion=nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.1)
epochs = 300
for i in range(epochs):
    y_pred = model.forward(y_train)
    loss = criterion(y_pred[p+1:240], y_train[p+1:])
    optimizer.zero_grad()
```

```
    loss.backward()
    optimizer.step()

y_test_pred=y_pred[240:]*stds+means
loss=criterion(y_test_pred,y_test)
losses.append(loss.item())
print("For p =",p+1,"Test set MSE =",loss.item())
print(y_test_pred[-5:])
```

APPENDIX B

AR(p) CODE

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from statsmodels.tsa.arima_model import ARMA
from sklearn.linear_model import LinearRegression

data=pd.read_csv("D:\\CODING\Practice\Sunspots\sunspots.dat", sep='\s+', header=None)

data.columns=['Year','Sunspots']

y=data['Sunspots']
year=pd.to_datetime(data['Year'], format="%Y")

yeartrain=year[:240]
yeartest=year[240:]
sqrt=np.sqrt(y)
ytest=y[240:]

n=np.linspace(1,305,305)
n=n.reshape(-1,1)
model = LinearRegression()
model.fit(n, sqrt)

trend = model.predict(n)
```

```
detrended=sqrt-trend


mse_store=np.zeros(12)
for i in range(12):
mod = ARMA(detrended[:240], order=(i+1,0),dates=yeartrain, freq='AS-JAN')
res=mod.fit()
fore=np.square(mod.predict(res.params, start='1940-01-01', end='2004-01-01',
dynamic=True)+trend[240:])
mse=np.sum(np.square(ytest-fore))/65
mse_store[i]=mse
print('AR(',i+1,') model:', 'Test set MSE =',mse)
```