Santa Clara University

# Forecasting Annual Temperature Changes Using Neural Network Time Series Analysis

Peter Sushko and Wesley Denison

Economics 174: Time Series Analysis

Professor Schlapfer

December 2018

**Abstract**

Artificial neural networks have gained widespread attention over recent years due to their application in advancing the fields of deep learning and artificial intelligence, yet their usefulness extends into a myriad of fields including time-series analysis. This paper will explore the basics of how artificial neural networks function, their mathematical foundations, and their ability to assist in time series analysis by running an example neural network forecast on a time series containing global temperature records of the last 130 years. The results showed that while the framework can be a powerful analytical tool and its use has great potential to continue to grow in time series analysis, its specific applications and limitations need to be thoroughly studied and appreciated to use this powerful tool effectively.

**Introduction**

Neural networks are computing systems inspired by the model of an animal brain. Themselves, neural networks aren't algorithms but instead frameworks for data processing and algorithm modeling[1]. Systems “learn” to create forecasts and provide solutions based on the inputs they receive. For example, they can learn to differentiate an image between two sets based on previous data received. In data forecasting, neural networks can spot seasonality and adjust the forecast based on its findings.

A neural network is based on a collection of artificial neurons. Each of these neurons is a mathematical function that takes numerical data as an input and provides a single number as an output. The output of every neuron is called its activation, and in most cases, it is a value in the [0,1] range. Neurons are packed together in layers which communicate with each other. The activations in one layer determine the activations in the next layer. The first layer is always an input layer which receives data about the task and passes it down to the next layers where this data is processed. The last layer is the output layer where the network presents its answer. The layers in between are called hidden layers, and they are responsible for doing calculations and actually solving the task.

[1] "Artificial Neural Networks as Models of Neural Information Processing | Frontiers Research Topic"

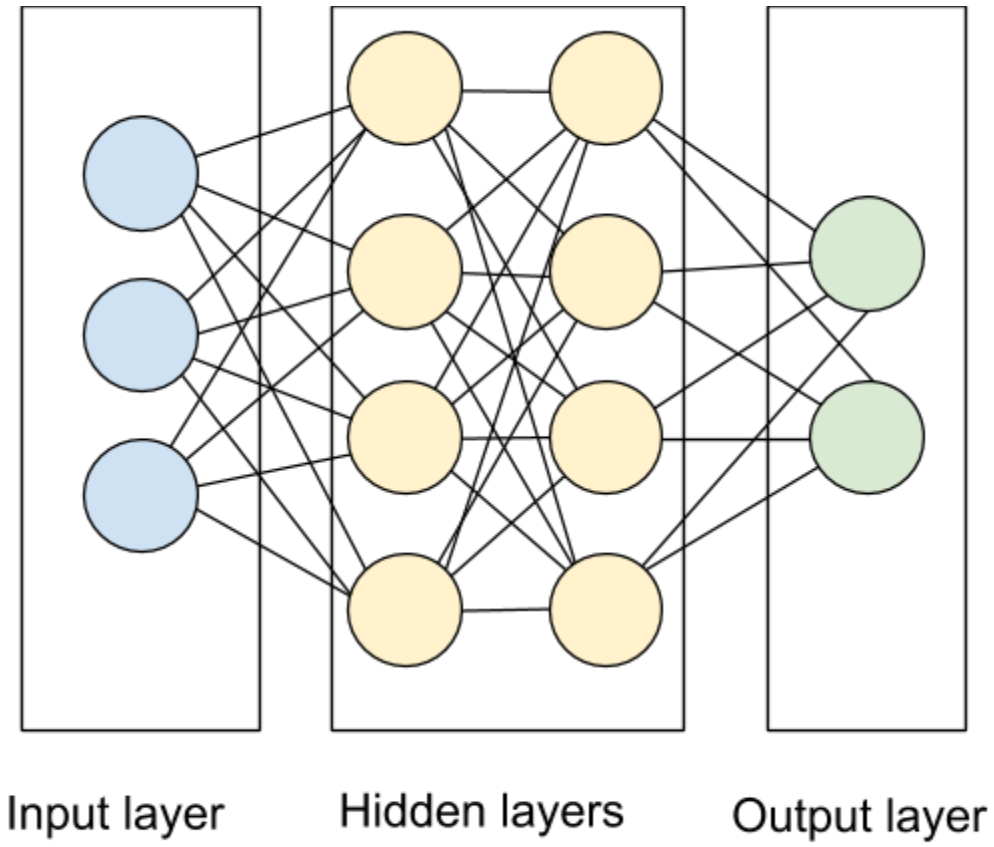



*Figure 1*

**Workings**

Here is why it is reasonable to have this layered structure. When humans recognize simple patterns we piece together various components. Recognizing the capital letter A, for example, consists of multiple parts: the letter is made out of three straight lines, these lines intersect and they make a triangle in the middle. The letter B has two areas outlined by curved lines and a vertical line on the left. Other letters have their own unique combination of components that define what the letter is. We hope that neurons in the second to last layer correspond to these components. Every time there is a loop in a letter, there will be a neuron which understands and saves this information. Then going from the second to last layer to the very last layer will require putting all the information together to understand which letter is written, similar to how humans do it. In order to achieve it, earlier hidden layers will need to recognize these components which will require learning how these components are created and checking for these factors.

Recognizing these subcomponents is hard, as is learning what the subcomponents should be. Recognizing the letter A will require recognizing a triangle in the middle which itself is based on three intersecting lines which are determined by the previous layer. Recognizing a letter, therefore, breaks down into smaller tasks such as understanding lines, shapes they outline, and what combination of these shapes and lines correspond to each letter. However, more complicated tasks might require recognizing more complicated components that can be far harder to spot using different frameworks. This is the goal of a layered structure. There is a lot of complicated tasks we can accomplish by breaking them down into smaller abstract operations.

**Mathematical Derivations**

Going from layer to layer, the neural network passes down parameters that contain information. The way the network does this is by creating a formula which determines the value of each neuron, based on the value of the previous neurons. Each neuron gets a weight which determines the importance of the information passed to the next layer[2]. The values of the neurons in the next layer are calculated by taking the weighted sum of all neurons in the previous layer. Here is the preliminary formula used for calculating the weighted sum:

$$\beta_1 = w_1 * \alpha_1 + w_2 * \alpha_2 + ... + w_i * \alpha_i + ...w_n * \alpha_n$$

[2] Nielsen, and Michael A. “Neural Networks and Deep Learning.” Neural Networks and Deep Learning, Determination Press, 1 Jan. 1970, neuralnetworksanddeeplearning.com/index.html.

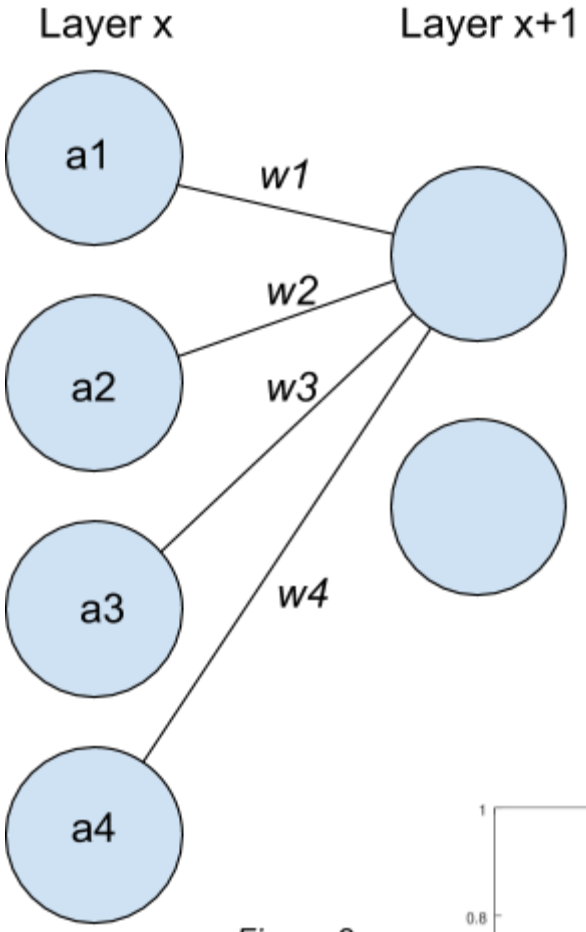
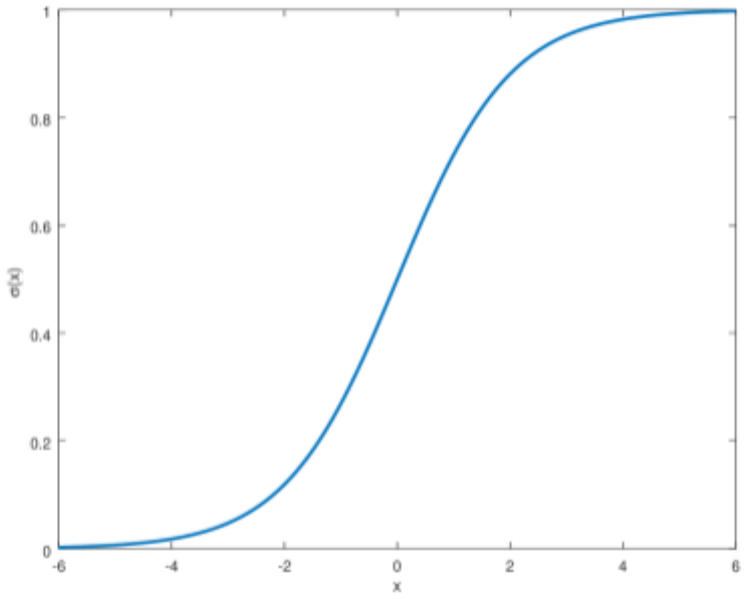



*Figure 2*

This weighted sum can take any value and, therefore, we need to limit which values are significant and characterize their significance levels. To be easily able to determine the value of this weight, we use a function that returns an output between 0 and 1 which tells us the significance. The function that's used in this case in a sigmoid that has the following formula:

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




*Graph of the sigmoid function*

We can also add a bias to the function. For example, if we need the sum of neurons to be larger than some value, assume 200, we can add 200 the weighted equation $((\sum_{i}^{n} x_i * a_i)$ - 200). In this case, the bias tells us how high the sum needs to be before the neuron becomes meaningfully active.

An easy mathematical way to represent these connections is to use linear algebra. To do this, first, we organize all the values of one layer into a column as a vector. Then we organize weights as a matrix where each row corresponds to connections between one layer and a particular neuron in the next layer. The weighted sum of the activations in the first layer corresponds to one of the terms in the matrix-vector product. Adding biases can be done by creating a vector that contains the biases and adding it to the matrix-vector product.

$$\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,0} & w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ w_{m,0} & w_{m,1} & w_{m,2} & w_{m,3} & \cdots & w_{m,n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \cdots \\ b_m \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \cdots \\ x_m \end{bmatrix}$$

*$w_{i,j}$ = weight of j's neuron in the previous layer with respect to i's neuron in the next layer*
*$a_i$ = i's neuron in the previous layer*
*$b_j$ = bias of the j's neuron in the next layer*
*$x_i$ = preliminary value of the j's neuron in the next layer before the sigmoid transformation*

A sigmoid of the sum will contain the information on which neurons will activate in the next layer given a previous layer. This method of using linear algebra makes writing code much easier. Thus each neuron is a function that takes values from the previous neurons and outputs a value between zero and one. Really, the neural network itself is a complicated function which takes the input and determines the output. For letter recognition, it takes a picture of a letter and outputs one of the 26 possible outcomes.

The sum of connection and weights in a simple network can easily be in the thousands, while more complicated networks can store up to billions of these connections. Each of these connections can be tweaked and adjusted to make the networks behave in different ways. Neural networks "learn" the optimal adjustments by themselves through trial and error. It this case learning means getting the computer to find a valid setting for all of these many connections to solve the given problem. Usually, each layer represents something. In the case of letter recognition, layers are

often loosely connected to recognizing lines, then shapes, and finally letters. For very complicated tasks, neural networks contain a lot of parameters. Its complexity assures that it can tackle complicated tasks.

**Example: Global Temperature Time Series**

Because of neural network analysis' ability to analyze complex non-linear data sets with many variables, and ability to theoretically do this without previous knowledge about these variables affecting it, it seems to be well suited (and therefore is often used) to analyze and predict scientific measurements. Since it is a hotly discussed and contested topic, we began to look at data and predictions regarding climate change. Perhaps the most (in)famous time series data regarding climate change are so-called "Hockey-Stick Graphs," which present global mean average temperature records of the past few hundred to thousand-plus years. These graphs show global temperatures taking a sharp upward turn in the early 20th century and shooting upward seemingly exponentially. These graphs have been heavily published and pointed to as evidence for global warming as well as used in attempts to predict global temperatures in the future. We decided to apply the "nnetar" function which uses "Feed-forward neural networks with a single hidden layer and lagged inputs for forecasting univariate time series."[3]

After considering various sources for our global temperature time series, we decided on a package from datahub.io[4] that contains both monthly and annual temperature readings from 1880 to 2006 from Global component of Climate at a Glance (GCAG)[5] and Global Land-Ocean Temperature Index (GISTEMP)[6]. GCAG comes from the NOAA National Climatic Data Center and combines the Global

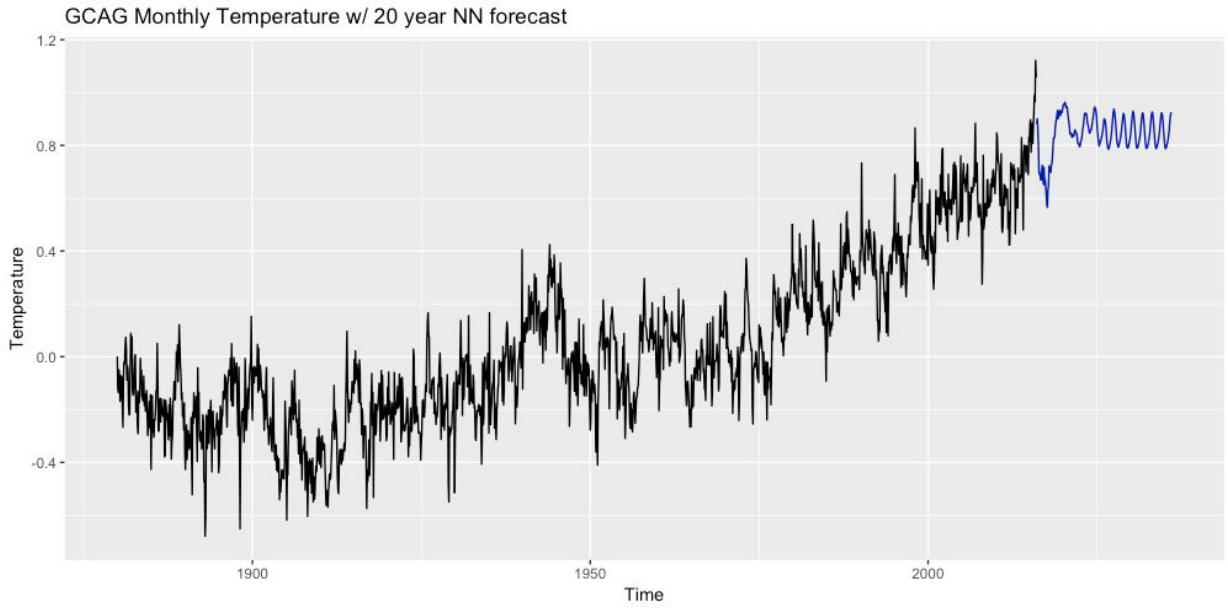
---

[3] Hyndman, Rob. "Neural Network Time Series Forecasts." Function | R Documentation, www.rdocumentation.org/packages/forecast/versions/8.4/topics/nnetar.
[4] Datopian. "Global Temperature Time Series." *DataHub*, datahub.io/core/global-temp#sources.
[5] NOAA National Climatic Data Center (NCDC), global component of Climate at a Glance (GCAG).
[6] GISTEMP: NASA Goddard Institute for Space Studies (GISS) Surface Temperature Analysis, Global Land-Ocean Temperature Index.

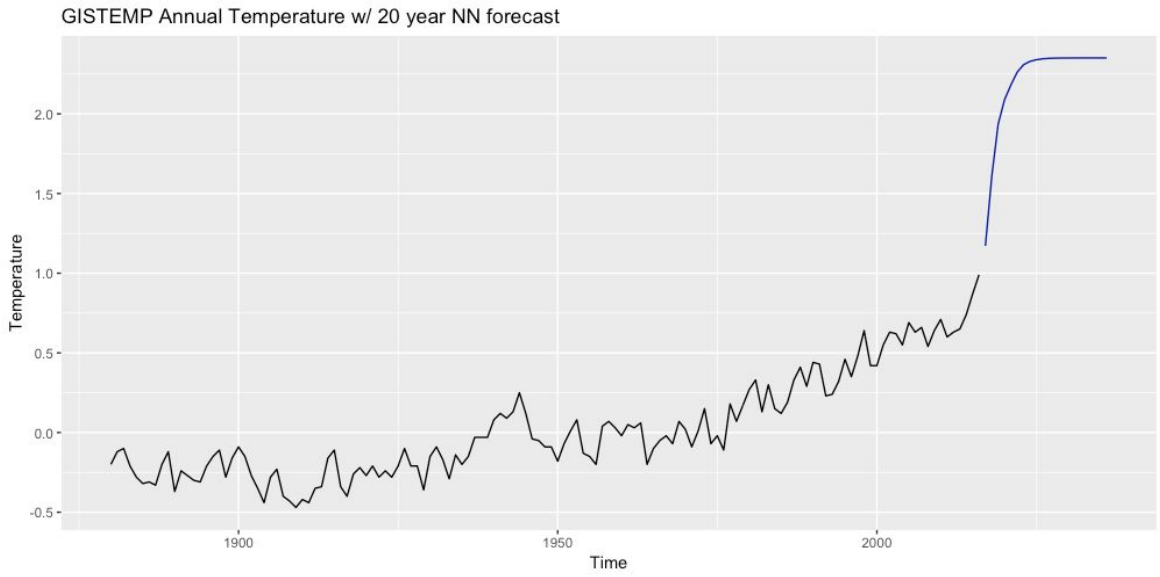
Historical Climatology Network-Monthly dataset the and International Comprehensive Ocean-Atmosphere Dataset to estimate global land and ocean temperature anomalies. [7] GISTEMP is an estimation of global temperature change created and maintained by NASA that uses data from NOAA GHCN v3 (meteorological stations), ERSST v5 (ocean areas), and SCAR (Antarctic stations).[8]

We chose this particular package because we felt confident about the validity of the data, the inclusion of both monthly and annual datasets allowed us to explore how our neural network would react with two very different time-series representing the same thing (large vs small number of data points, seasonal vs non-seasonal data), and the two separate sources allowed us to compare results and check for anomalies.

After splitting up the time series into the two separate sources and adjusting the format to be easily recognized by R, each .csv file was imported and declared as a time series. We then use the "nnetar" command to run the feed-forward neural network analysis on the time series and plot the results with a 20-year prediction using the autoplot command. We have set lambda = "auto" which uses BoxCox.lambda to select a value. Here are the two sample results from this initial run.




[7] NCDC Data, ncdc.noaa.gov/cag/global/data-info
[8] NASA Goddard Institute for Space Studies data.giss.nasa.gov/gistemp/



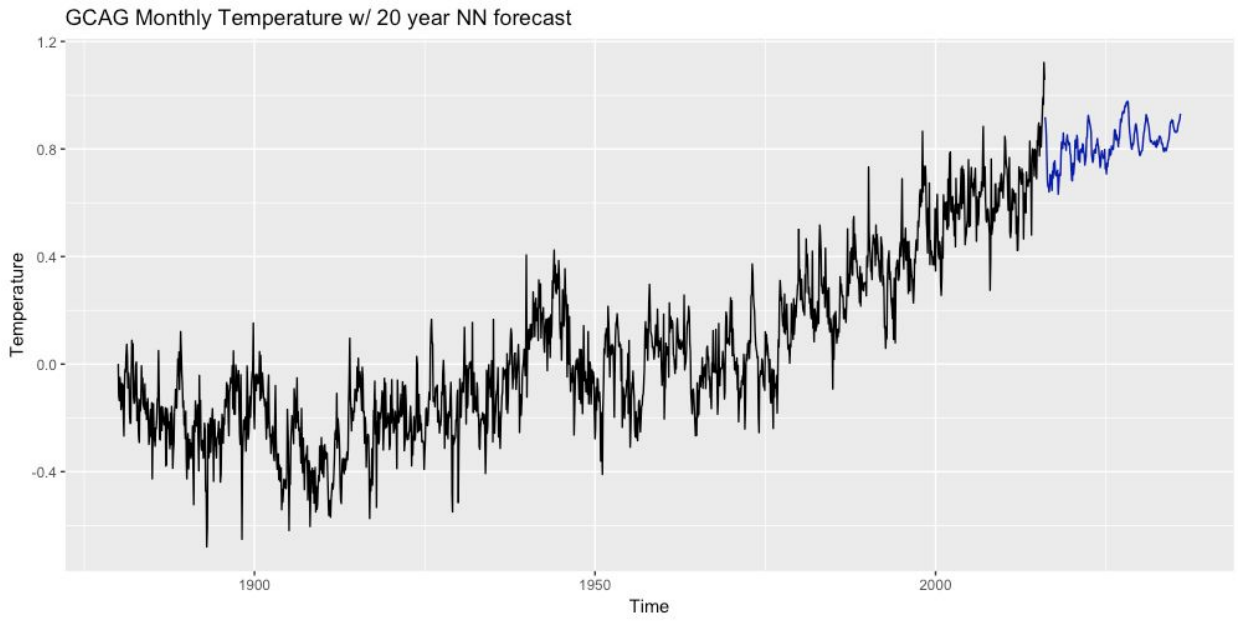

While unexpected and certainly far from perfect, these results are actually a great demonstration of some of the challenges associated with neural networks.

Why did the prediction for monthly temperature ignore the overall trend of the data and settle into a repetitive oscillating pattern? And why did the annual prediction start incredibly steep and then level off to perfectly flat after a decade? Attempting to answer these questions introduced us to one of the major challenges of neural networks: as neural networks become more complex it is nearly impossible to understand the logic or workings of the network to deduce “why” it produces the outcomes it does. This makes troubleshooting unexpected behavior incredibly difficult. For this reason, artificial neural networks and other forms of deep learning algorithms are often referred to as “black boxes” where inputs are entered and outputs are received but how these outputs were calculated remains a mystery.

Despite this, we did have some immediate theories as to why the neural network had predicted monthly temperatures in this way. Without removing the extremely significant seasonality between summer and winter, and huge variances between months found in this time series, the neural network was perhaps only able to model these oscillating patterns and not the overall trend. This issue was most likely exacerbated by the fact that the nnetar function utilizes a limited number of neurons and only one hidden layer to analyze time series, perhaps limiting the network’s ability to internalize other aspects of the time series outside of its seasonality.

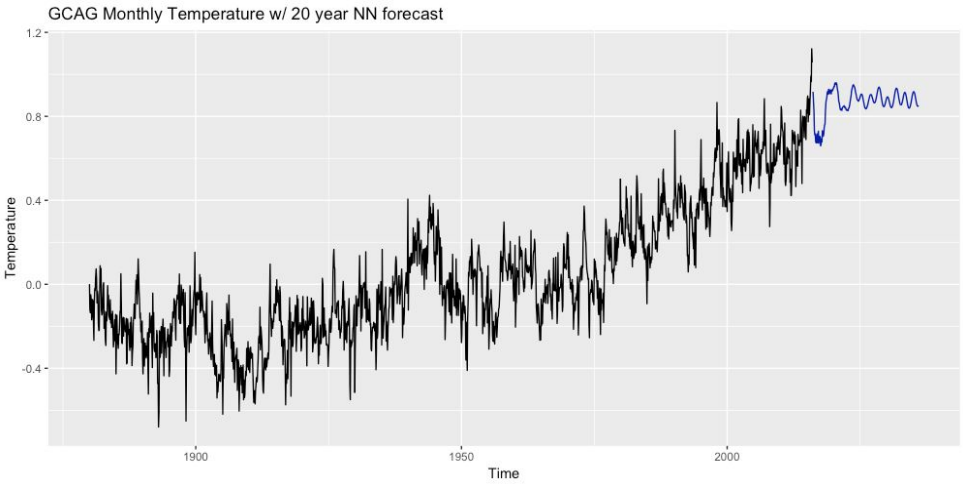
To combat this issue, nnetar has an argument “P” that adjusts the seasonal lags used as inputs to assist the neural network in its ability to handle seasonal data. By

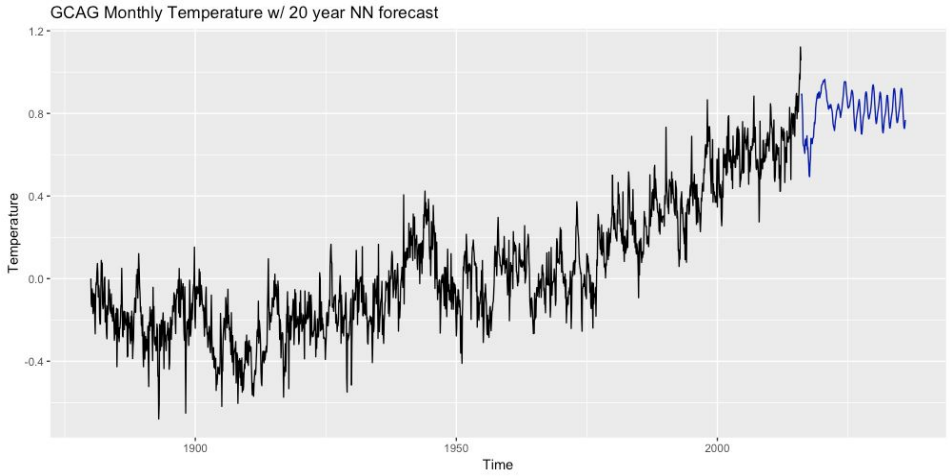
setting P=12 we were able to produce a prediction that seems much more in line with what we expect.



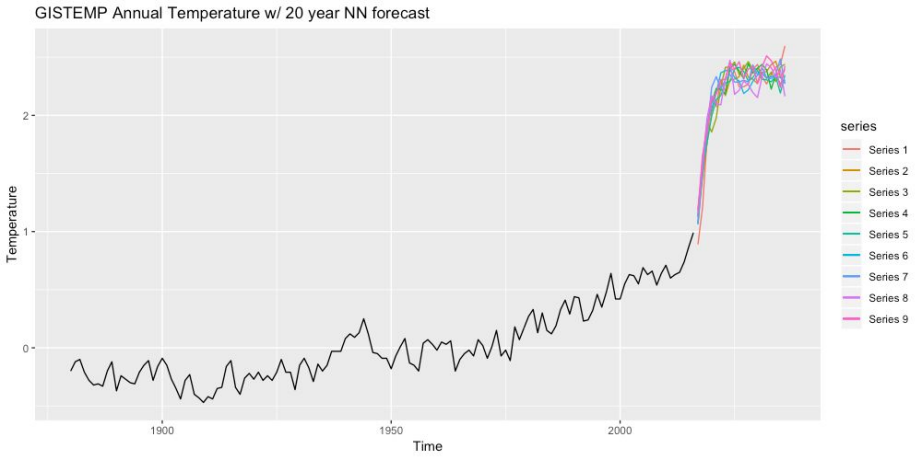

The unusual annual prediction is harder to explain, however, it is worth noting that neural networks are known for being extremely data hungry, often producing widely inaccurate analyses when trained with limited data sets. The annual time series' comparatively minuscule number of data points might have resulted in a much lower quality prediction and is the best explanation we could think of to account for its behavior.

Our analysis of temperature changes also reveals another one of the unique aspects of neural network analysis. Because weights are randomly assigned at the beginning of each iteration, without running the neural network a massive number of times there are clear variances between predictions, despite these predictions themselves being the product of hundreds to thousands of neural network iterations. As a demonstration, these predictions of the GCAG Monthly Temperature time series were run back to back with no code modifications, yet have produced significantly different predictions:



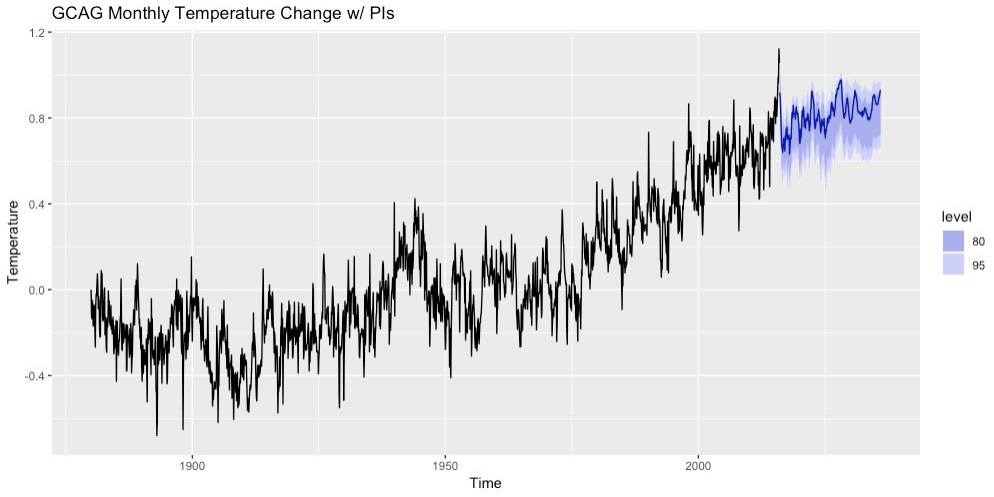

We can also see this randomness demonstrated in the variance between individual networks with varied random starting weights. The graph below displays these individual network's outputs before they are averaged together to produce the forecast.

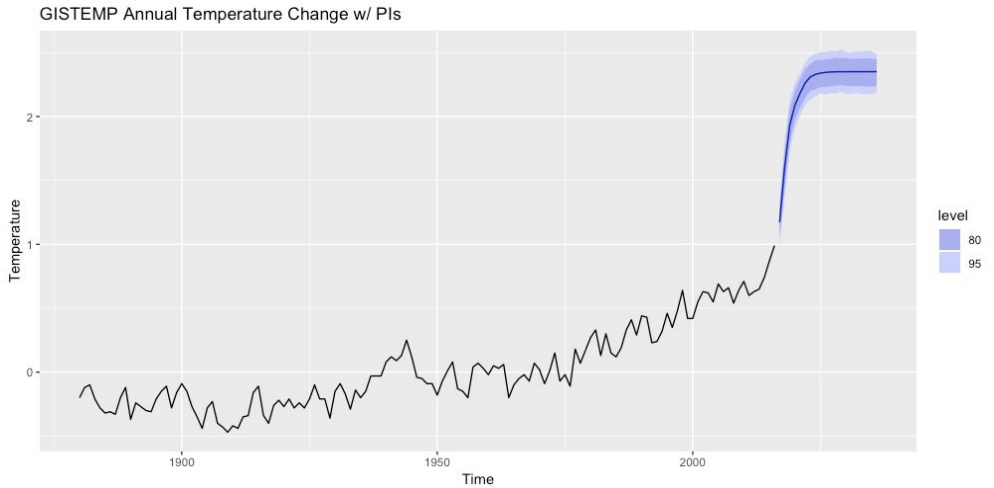


**Prediction Intervals**

Because neural networks adjust their parameters throughout the process, there is no stochastic model that defines prediction intervals like it is the case with ARIMA forecasting. Because of that, there are no mathematical derivations to find the prediction intervals. There is a way to estimate them using the normal distribution of errors in the derivations of consecutive iterations. For more precision, we ran the neural networks for a few thousand times and took the 80 and 95th percentile of the distribution of the forecasts. While the empirical method provides extremely exact values for the prediction intervals, it has its downsides. Mainly, it takes a lot of time to calculate and uses a large amount of memory which on a large scale results in high power consumption.

Here are the results of forecasting prediction intervals based on GCAG monthly and annual data respectively.



## Conclusion

Artificial neural network analysis is a powerful tool that has great potential to be used to tackle complex time series that other methods struggle to forecast. However, it is far from a catch-all solution: neural networks have numerous parameters and when they aren't well understood and constructed, the frameworks can produce unreliable forecasts. Much like other methods, it comes with its own unique challenges and limitations that need to be understood and accounted for to be used effectively. As the framework continues to mature and its application within the field of time series analysis is further studied, it is realistic that it will become a major tool in data analysis studies.

## References


Datopian. “Global Temperature Time Series.” *DataHub*, datahub.io/core/global-temp#sources.

GISTEMP: NASA Goddard Institute for Space Studies (GISS) Surface Temperature Analysis, Global Land-Ocean Temperature Index.

NOAA National Climatic Data Center (NCDC), global component of Climate at a Glance (GCAG).

NCDC Data, ncdc.noaa.gov/cag/global/data-info

NASA Goddard Institute for Space Studies data.giss.nasa.gov/gistemp/

Gerven, Marcel Van, and Sander Bohte. “Artificial Neural Networks as Models of Neural Information Processing.” Frontiers, Frontiers, 19 Dec. 2017, www.frontiersin.org/research-topics/4817/artificial-neural-networks-as-models-of-neural-information-processing.

Nielsen, and Michael A. “Neural Networks and Deep Learning.” Neural Networks and Deep Learning, Determination Press, 1 Jan. 1970, neuralnetworksanddeeplearning.com/index.html.

Hyndman, Rob. “Neural Network Time Series Forecasts.” Function | R Documentation, www.rdocumentation.org/packages/forecast/versions/8.4/topics/nnetar.

Hyndman, Rob J, and George Athanasopoulos. “Forecasting: Principles and Practice.” Forecasting: Principles and Practice, Monash University, Australia, Apr. 2018, otexts.org/fpp2/.