Neural Network for Forecasting

# The Neural Network Model an Example

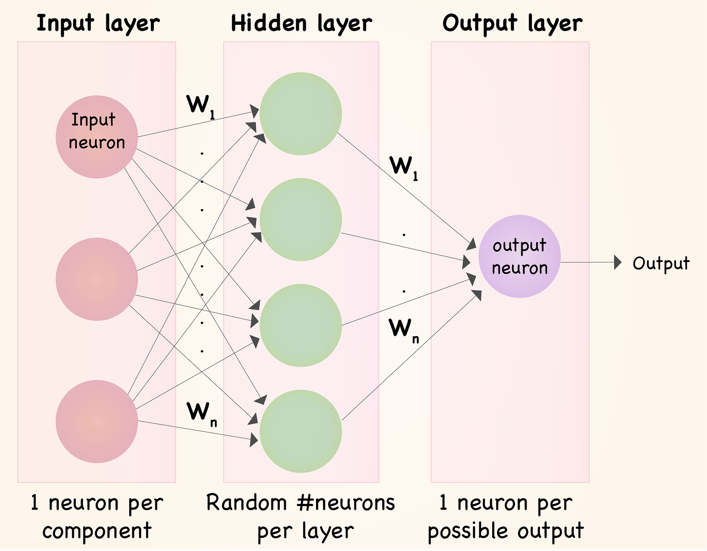
We have seen that linear regression can be used for modeling non-linear relationships between the predictors and the outcome variable by using transformations such as log(y) or creating derived variables from the predictor variables such as t2 (part of parabolicity of the trend) or (as a part of the seasonality function).

Neural networks offer a further extension of linear regression, by creating multiple layers of derived variables. A derived variable is a transformation of the original variables. Examples include dummy variables created from a categorical variable and transformations such as log(y).

A neural network links the predictors and the outcome through a sequence of layers. In each layer, some operation is performed on the input information to produce an output (thereby creating a derived variable). The output of one layer is the input into the next layer. A neural network consists of three types of "layers", as illustrated in Figure 1. The figure shows only one node of the middle or Hidden layer.

1. An input layer accepts the input values of the predictors (numerical or binary)
2. One or more hidden layers create derived variables. A hidden layer receives inputs from the previous layer, a bias or weight and performs a calculation on those inputs and generates an output.
3. An output layer receives inputs from the last hidden layer and produces predicted values (numerical or binary).

Each layer contains nodes (neuron), denoted by circles or other shapes such as one in Figure 1. The input layer nodes are the original variable, predictor, or component variables. For example, in Figure 1 nodes #4 as input has the values of predictors x1, x2, …,and xn respectively. Their output is identical to their input.



1

2

3

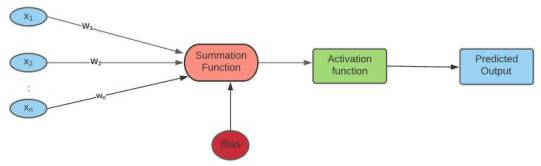
4

5

n

Figure 1: A General View of NN components

Hidden layer nodes are derived variables (Figure 2). In particular, they are a weighted sum of the inputs to which some monotone function, called an activation function, is applied. Common functions are linear, exponential, and s-shaped functions such as the logit and hyperbolic tangent.



Hidden Layer Node 1

Figure 2: Hidden Node Detail

The computation performed in a node could be shown in the following example. This example shows the summation, considering weighted predictors and the bias coefficient.

Let’s assume we have a set of input predictors the output of the a given node in the hidden layer say *j*, could be as weighted sum below:

Where are weights and are initially set randomly (in general but not always) and then adjusted as the network learns. The constant controls the level of the contribution of node *j* .

An activation function of choice (for example, g(s) ) is then applied to the weighted sum to produce the derived variable (output). That is:

Linear regression, logistic regression and autoregressive models can be written as neural networks that lack hidden layers. This is important to notice, because in practice the best performing neural network is often the simplest one, that is, one with no hidden layers.

Figure 3, displays a neural network diagram for a multi-linear or a logistic regression with 4 predictor. The activation function for these two models is different. As you remember for regression the summation itself is the activation but for logistic regression the summation is used in the logit function. Then I use the textbook example to show same approach but using logit function. As you see the bias

A diagram of mathematical equations

Description automatically generated

Figure 3: NN without Hidden Layer

Textbook Example: Consider a neural network diagram for a linear or logistic regression with three predictors: a linear trend and two dummies for capturing an additive weekday/

Saturday/Sunday seasonality pattern. Note that:

* in the following activation functions the predictors are replace with t (trend) and seasonality (Weekday, and Saturday) you understand why Sunday is missing.
* a linear activation function corresponds to the linear regression model
* an exponential activation function (g(s) = exp(s)) corresponds to the linear regression model
* a logit activation function corresponds to the logistic regression model

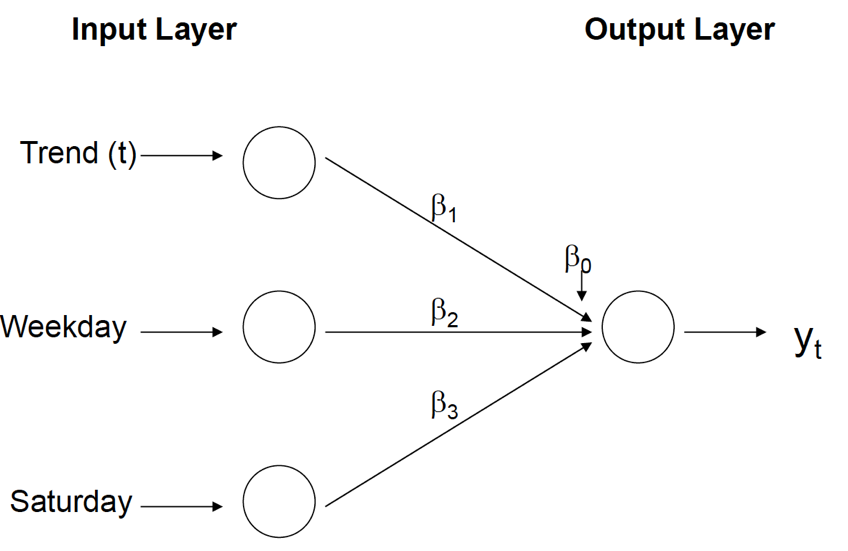


Figure 4: NN without Hidden Layer regression

A diagram of a diagram

Description automatically generatedFigure 5 shows a neural network schematic for an AR(3) model (using a linear activation function).

Figure 5: NN without Hidden Layer for AR(3)

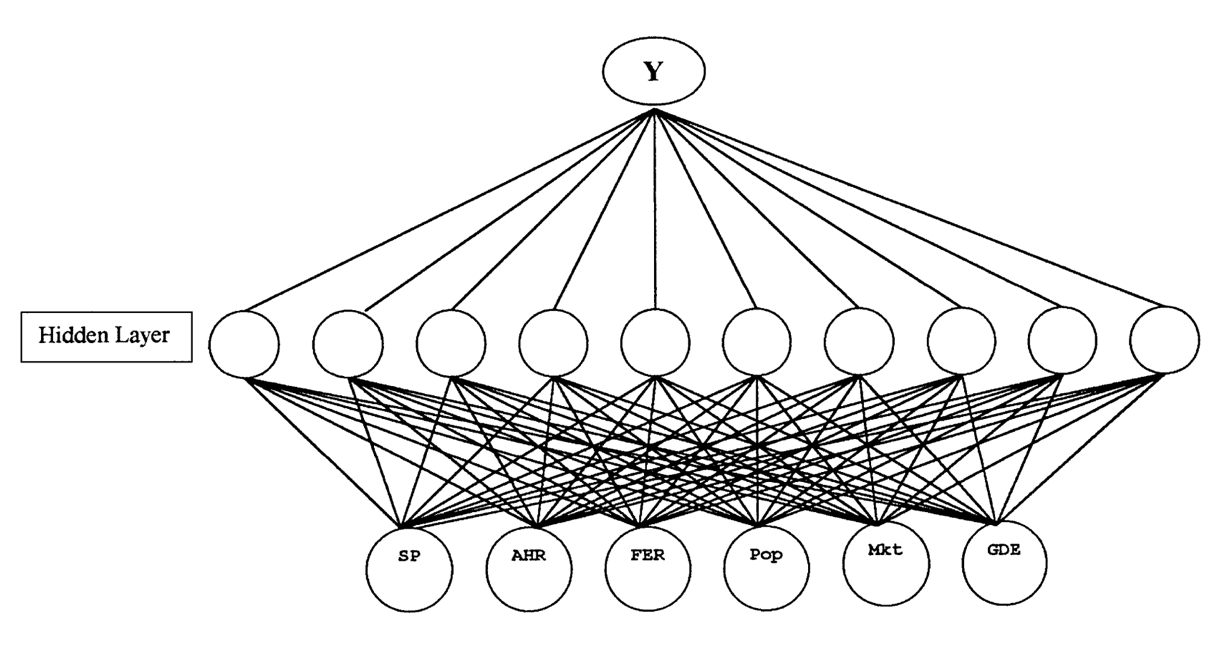
A general schematic for a neural network used for forecasting a time series is shown in Figure 6. Note that a neural net can include multiple output nodes. In the context of time series forecasting, we use this to produce forecasts of different horizons. The schematic in Figure 5 is based on roll-forward forecasting,

where forecasts for times t + 1, t + 2, . . . are based on the series "moving forward" one period at a time. Additional hidden layers can be added in Figure 6 to further increase the complexity of the relationship between the inputs and output. The diagram can also be expanded to include external information by specifying additional input nodes. In fact, successful applications of neural networks for time series forecasting are often based on including external information rather than extrapolation. For example, Law and Au found that for forecasting annual Japanese tourist arrivals in Hong Kong, a neural network outperformed naïve forecasts, linear regression, moving average, and exponential smoothing. The input layer of their neural network model included six nodes: Service Price, Average Hotel Rate, Foreign Exchange Rate, Population, Marketing Expenses, and Gross Domestic Expenditure. All of these capture external information. A sample of dataset for this case is shown in Table 1 and The NN diagram is shown in Figure 6. Input are six predictors and output is the forecasted Number of visitors.

Table 1: Japanese tourist arrivals sample dataset

A table with numbers and text

Description automatically generated



Output: Forecasted Number of visitors

Input: Six predictors

Figure 6: A demand neural network model for travel by Japanese tourists to Hong Kong.

# Pre-Processing

Four pre-processing issues should be considered prior to running a neural network:

* *Creating Derived Predictors*: As in regression models, we can create predictors such as lagged versions of the time series, lagged versions of external information, seasonal dummies, and a time index to capture trend. These are then used as inputs to the neural network. In R, a neural network with generic predictors can be run using the avNNet( ) function in the caret package. When the predictors are lagged versions of the time series, however, working with this function can be tedious. Fortunately, the forecast package offers a function that does all this work for us. The function is called nnetar( ), which stands for neural network autoregression. Throughout the rest of this notes, we will focus on neural network autoregressions. To learn more about how to fit neural networks with generic predictors using avNNet in R, see textbook Chapter 9 of the online textbook Forecasting: Principles and Practice by Hyndman and Athanasopoulos at <https://otexts.com/fpp3/advanced.html>
* *Removing rows with missing values:* When creating lagged variables that will be included as predictors, rows with missing values must be removed. Again, in R, nnetar() will do this work for you.
* *Scaling:* Neural networks perform best when the predictors and response are on a scale of [0,1] or [-1,1]. Hence, in the time series context, the series itself as well as all predictors should be scaled before running the neural network. In R, the forecast package’s nnetar function automatically scales the variables. When the predictors are lagged versions of the time series, scaling to [-1,1] is achieved by dividing by the maximum of the absolute values of the time series.
* *Removing trend and seasonality*: There have been mixed reports in the literature regarding the need for removing trend and seasonality prior to using neural networks for forecasting time series. While some researchers claim that neural networks can automatically account for trend and seasonal patterns, substantial empirical evidence has shown that deseasonalization and detrending improve the performance of neural networks.8 To de-seasonalize and/or de-trend a series, use any of the methods described in previous chapters (regression, exponential smoothing, etc.) with the exception of differencing, which has been shown to lead to inferior performance.9

# User Input

To run a neural network, users must determine the number of input nodes, the number of hidden layers, the number of nodes per hidden layer, the activation function, and the number of output nodes. Let us examine each choice separately:

* *The number of input nodes* is equal to the number of predictors and is therefore problem-dependent. Generally, the more input nodes, the less time periods for the network to train on. Zhang and Kline conducted a large study on neural network forecasting and reached the conclusion that the number of predictors is a critical choice:

*[. . . ] different combinations of input variables can have significant impact on the model performance. Therefore, in applying [neural networks], it is critical to identify a set of important input variables to be included in the modeling process.*

* *The number of hidden layers* affects the level of complexity of the relationship between the inputs and outputs. Typically, 1-2 layers are sufficient. Using too many layers can cause overfitting, while too few leads to under-fitting.
* *The number of nodes per hidden layer* requires trial and error. Using too few nodes will lead to under-fitting, while too many nodes will lead to overfitting. Comparing the training and validation performance can help in determining an adequate number.
* *Choice of activation function*: The most common activation functions are s-shaped functions.
* *The number of output nodes* depends on the number of forecasts that we want to generate. Some software packages are limited to a single output node.

# Forecasting with Neural Nets in R

In R, a generic neural network—a network with predictors of any kind—are typically run with the function avNNet from the caret package. The avNNet function actually fits several neural networks and averages their outputs in order to come up with a forecast. Consequently, avNNet produces an ensemble forecast. When working with time series, the pre-processing work that avNNet requires can be somewhat tedious. First, you need to set up lagged versions of the series as predictors and normalize them to [0,1] or [-1,1]. Once you have fit the model in the training period, you need to set up the predictors for the validation period. Here is where it can get tricky. When you want to forecast more than one-step ahead, some of the predictors in validation period will need to be forecasted values. For instance, suppose you want to forecast two-months ahead in a validation period and you have included the last two months’ values as predictors in the training period. Your predictors in the validation period for the two-steps-ahead forecast will then need to include the last value in the training period and the one-step

ahead forecast.

Fortunately, there is a function in R that will do all this preprocessing work for you when working with a time series. The function nnetar, which stands for neural network autoregression model, automatically pre-processes your time series before calling the avNNet function to fit and average several neural networks. Note that nnetar applies to time series where the only predictors are lagged versions of the series itself. To include other predictors, such as seasonal dummies, in a neural network, you will want to use avNNet. The function nnetar has four main arguments: repeats, p, P, and size. The argument repeats controls the number of neural networks fit; the default is 20. If not specified, nnetar chooses the number of non-seasonal lags p based on the best-fit AR(p) model. For instance, a neural network autoregression with p = 2 would include the lag 1 and lag 2 of the series as inputs. The number of seasonal lags P is set to 1 as its default. A seasonal lag is the last observed value from the same season. For example, with monthly data, a neural network autoregression with p = 2 and P = 1 would include lag 1, lag 2, and lag 12 of the series as inputs. Because the function nnetar will only fit a neural network with a single hidden layer, size refers to the number of nodes in the hidden layer. Its default setting is the number of inputs nodes plus one (p + P + 1) divided by two (and rounded to the nearest integer). We denote a neural network autoregression model by NNAR(p,P,size). R’s nnetar uses the logit activation function to map the input value into the hidden node’s value. To go from the hidden nodes’ values to the output value, nnetar uses either linear or logit activation function. When the series takes values on a continuous range, you will want to use the linear activation function by setting linout = TRUE as an argument in nnetar. For series with binary outcomes (rain or not), you set linout = FALSE to use the logit activation function when going from the hidden nodes’ values to the output value. Using the logit activation function in this last step will ensure the output is a probability.

# Example: Forecasting Amtrak Ridership

To illustrate the use of neural networks for forecasting a time series, we return to the Amtrak ridership example, using nnetar. Suppose that we are interested again in producing one-monthahead, two-month-ahead,...,36-month-ahead forecasts.

*Preparing the Data*

The annual seasonality that characterizes the monthly ridership series can be addressed in several ways in the neural network:

1. include 11 non-seasonal lags and 1 seasonal lag as predictors
2. include 11 dummy variables
3. deseasonalize the series using some method

While each avenue can be investigated, we illustrate the results for the first option, which is a common choice in neural network forecasting. As usual, we partition the data into training and validation periods, such that the last 36 months are the validation period. Note that there are 159 months in the ridership time series. After dropping the first 12 months due to the lag variables and the 36 months for the validation period, the training period contains 111 months of data. Neural Network Output The output from running a neural network autoregression for forecasting monthly ridership is shown in Figure 9.5. The code for creating this figure is given below it. The NNAR(11,1,7) fitted is a neural network with the last 12 months as predictors (11 from the non-seasonal lags and 1 from the seasonal lag) and 7 hidden nodes. Because each neural network’s parameters are initialized randomly before they are optimized, we set our seed at 201 so that we can replicate our results. Otherwise, the neural

networks would be different each time we run nnetar. The summary line in the code below will print out the weights in a neural network, which we do not show here. For instance, by changing the number inside the double brackets to 2, you can see the weights for the second of 20 fitted neural networks. The training and validation performance measures can be compared to the exponential smoothing models in Chapter 5 of the textbook. In terms of performance measures, the neural network results are inferior to the Holt-Winter’s method (Figure 5.6 and Table 5.5) in both the training and validation periods. In the training period, the neural network autoregression has an RMSE of 4.6, compared to Holt-Winter’s 56.2. The neural network autoregression is significantly worse in the validation period. Its RMSE is 121.6 versus Holt-Winter’s 82.1. Visually we can notice that the neural network is overfit to the training set and does poorly on the validation set. Unfortunately, even when we let nnetar choose the optimal neural network autoregression (using the call nnetar(train.ts)), we find that the RMSE in the validation period goes up to 130.9.

A graph of a graph showing a number of data

Description automatically generated with medium confidence

# Forecasting Amtrak Ridership R Codes

The following R code are for building ts model with nnetar( ) function and forecasting validation for accuracy and then, forecasting in the future year. As you see I do not follow the given recommendation but still my model is close to what is give by the text Book

path = "Z:/CSDA 5410 Time Series/CSDA 5410 Fall 2023/Chapter 9 plus Problem 9-1"

setwd(path)

Amtrak.data <- read.csv("Amtrak data.csv")

library(forecast)

library(caret)

library(zoo)

library(knitr)

Amtrack.Date <- Amtrak.data$Month

'## creating a date column. with alink for transforming and generating all date format

## This assignment didn''t need this transformation

Amtrak.data$Date <-as.Date(paste("01-",Amtrack.Date, sep = ""), format = "%d-%b-%Y")

#https://shanghai.hosting.nyu.edu/data/r/dates-and-times.html

##'

ridership.ts <- ts(Amtrak.data$Ridership, start = c(1991, 1), end = c(2004, 3), freq = 12)

stepsAhead <- 36

nTrain <- length(ridership.ts) - stepsAhead

train.ts <- window(ridership.ts, start = c(1991, 1), end = c(1991, nTrain))

valid.ts <- window(ridership.ts, start = c(1991, nTrain + 1), end = c(1991, nTrain + stepsAhead))

set.seed(201)

ridership.nnetar <- nnetar(train.ts, repeats = 20, p = 11, P = 1, size = 7)

summary(ridership.nnetar$model[[1]])

ridership.nnetar.pred <- forecast(ridership.nnetar, h = stepsAhead)

kable(accuracy(ridership.nnetar.pred, valid.ts))

'#

A blue background with white text

Description automatically generated

#Forecast in the future for one year (12 months)

one.year <- 12

#building model on entire dataset

ridership.nnetar.Future.model <- nnetar(ridership.ts, repeats = 20, p = 11, P = 1, size = 7)

ridership.nnetar.pred.Future <- forecast(ridership.nnetar.Future.model, h = one.year)

plot(train.ts, ylim = c(1300, 2900), ylab = "ridership", xlab = "Time",

bty = "l", xaxt = "n", xlim = c(1991,2006.25), lty = 1, main = "Forecasing Validation with NNETAR() model")

axis(1, at = seq(1991, 2006, 1), labels = format(seq(1991, 2006, 1)))

lines(ridership.nnetar.pred$fitted, lwd = 2, col = "blue")

lines(ridership.nnetar.pred$mean, lwd = 1, col = "red")

lines(ridership.nnetar.pred.Future$mean, lwd = 1, col = "darkgreen")

lines(valid.ts, lwd = 1, col = "darkgreen" )

lines(c(2004.25 - 3, 2004.25 - 3), c(0, 3500))

lines(c(2004.25, 2004.25), c(0, 3500))

text(1996.25, 2500, "Training")

text(2002.75, 2500, "Validation")

text(2005.25, 2500, "Future")

arrows(2004 - 3, 2450, 1991.25, 2450, code = 3, length = 0.1, lwd = 1,angle = 30)

arrows(2004.5 - 3, 2450, 2004, 2450, code = 3, length = 0.1, lwd = 1,angle = 30)

arrows(2004.5, 2450, 2006, 2450, code = 3, length = 0.1, lwd = 1, angle = 30)