11.3 Neural network models

Artificial neural networks are forecasting methods that are based on simple mathematical models of the brain. They allow complex nonlinear relationships between the response variable and its predictors.

Neural network architecture

A neural network can be thought of as a network of "neurons" which are organised in layers. The predictors (or inputs) form the bottom layer, and the forecasts (or outputs) form the top layer. There may also be intermediate layers containing "hidden neurons."

The simplest networks contain no hidden layers and are equivalent to linear regressions. Figure 11.11 shows the neural network version of a linear regression with four predictors. The coefficients attached to these predictors are called "weights." The forecasts are obtained by a linear combination of the inputs. The weights are selected in the neural network framework using a "learning algorithm" that minimises a "cost function" such as the MSE. Of course, in this simple example, we can use linear regression which is a much more efficient method of training the model.

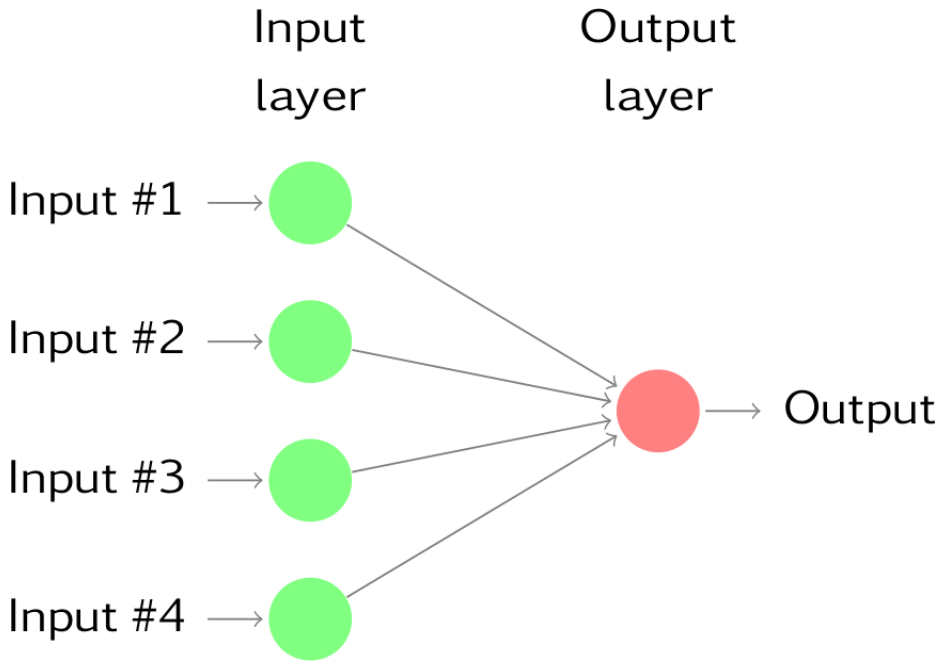

Figure 11.11: A simple neural network equivalent to a linear regression.

Once we add an intermediate layer with hidden neurons, the neural network becomes non-linear. A simple example is shown in Figure 11.12.

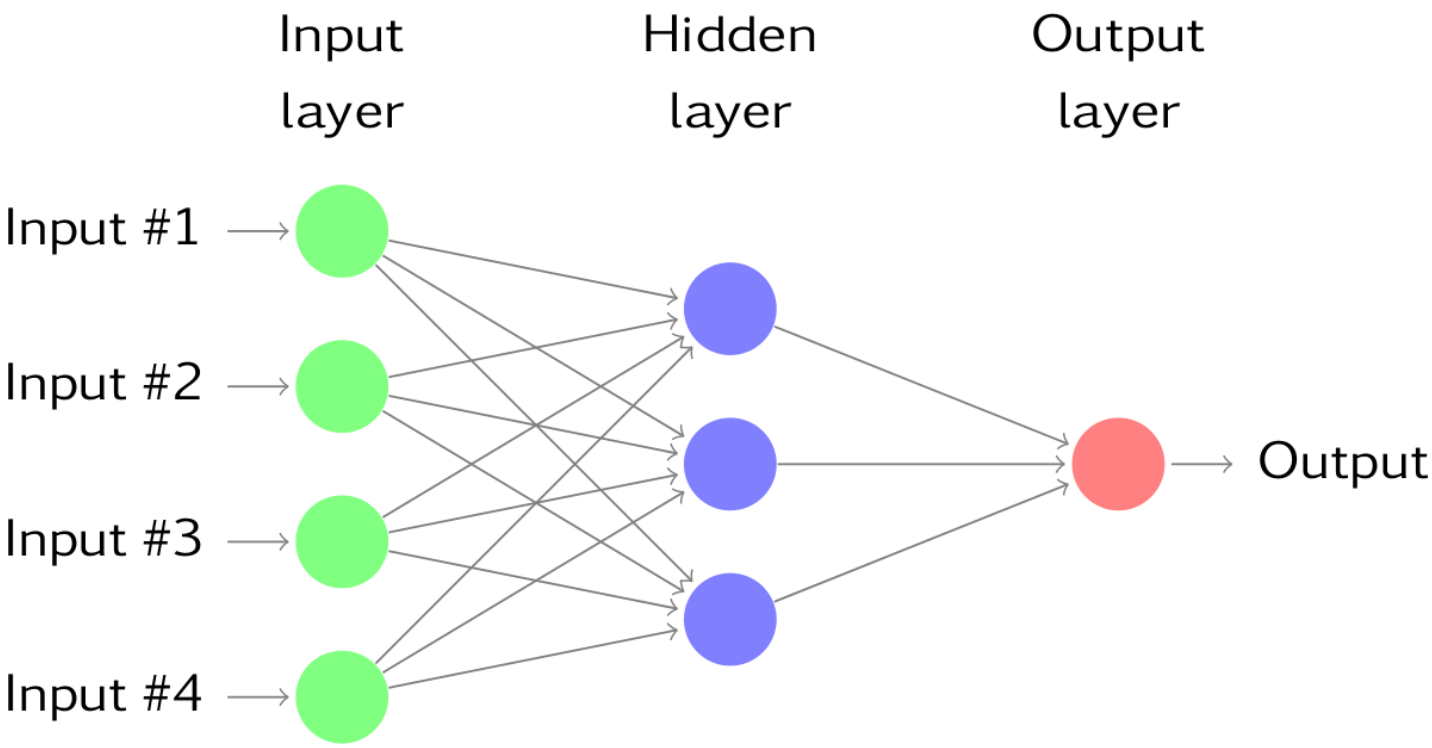

Figure 11.12: A neural network with four inputs and one hidden layer with three hidden neurons.

This is known as a *multilayer feed-forward network*, where each layer of nodes receives inputs from the previous layers. The outputs of the nodes in one layer are inputs to the next layer. The inputs to each node are combined using a weighted linear combination. The result is then modified by a nonlinear function before being output. For example, the inputs into hidden neuron j in Figure 11.12 are combined linearly to give

$$z_j = b_j + \sum_{i=1}^{4} w_{i,j} x_i.$$

In the hidden layer, this is then modified using a nonlinear function such as a sigmoid,

$$s(z) = \frac{1}{1+e^{-z}},$$

to give the input for the next layer. This tends to reduce the effect of extreme input values, thus making the network somewhat robust to outliers.

The parameters b_1, b_2, b_3 and $w_{1,1}, \dots, w_{4,3}$ are "learned" from the data. The values of the weights are often restricted to prevent them from becoming too large. The parameter that restricts the weights is known as the "decay parameter," and is often set to be equal to 0.1.

The weights take random values to begin with, and these are then updated using the observed data. Consequently, there is an element of randomness in the predictions produced by a neural network. Therefore, the network is usually trained several times using different random starting points, and the results are averaged.

The number of hidden layers, and the number of nodes in each hidden layer, must be specified in advance. We will consider how these can be chosen using cross-validation later in this chapter.

Neural network autoregression

With time series data, lagged values of the time series can be used as inputs to a neural network, just as we used lagged values in a linear autoregression model (Chapter 8). We call this a neural network autoregression or NNAR model.

In this book, we only consider feed-forward networks with one hidden layer, and we use the notation NNAR(p, k) to indicate there are p lagged inputs and k nodes in the hidden layer. For example, a NNAR(9,5) model is a neural network with the last nine observations $(y_{t-1}, y_{t-2}, \dots, y_{t-9})$ used as inputs for forecasting the output y_t, and with five neurons in the hidden layer. A NNAR(p, 0) model is equivalent to an ARIMA(p, 0, 0) model, but without the restrictions on the parameters to ensure stationarity.

With seasonal data, it is useful to also add the last observed values from the same season as inputs. For example, an NNAR(3,1,2)$_{12}$ model has inputs y_{t-1}, y_{t-2}, y_{t-3} and y_{t-12}, and two neurons in the hidden layer. More generally, an NNAR(p, P, k)$_m$ model has inputs $(y_{t-1}, y_{t-2}, \dots, y_{t-p}, y_{t-m}, y_{t-2m}, y_{t-Pm})$ and k neurons in the hidden layer. A NNAR(p, P, 0)$_m$ model is equivalent to an ARIMA(p, 0, 0)(P,0,0)$_m$ model but without the restrictions on the parameters that ensure stationarity.

The `nnetar()` function fits an NNAR(p, P, k)$_m$ model. If the values of p and P are not specified, they are selected automatically. For non-seasonal time series, the default is the optimal number of lags (according to the AIC) for a linear AR(p) model. For seasonal

time series, the default values are $P = 1$ and p is chosen from the optimal linear model fitted to the seasonally adjusted data. If k is not specified, it is set to $k = (p + P + 1)/2$ (rounded to the nearest integer).

When it comes to forecasting, the network is applied iteratively. For forecasting one step ahead, we simply use the available historical inputs. For forecasting two steps ahead, we use the one-step forecast as an input, along with the historical data. This process proceeds until we have computed all the required forecasts.

Example: sunspots

The surface of the sun contains magnetic regions that appear as dark spots. These affect the propagation of radio waves, and so telecommunication companies like to predict sunspot activity in order to plan for any future difficulties. Sunspots follow a cycle of length between 9 and 14 years. In Figure 11.13, forecasts from an NNAR(10,6) are shown for the next 30 years. We have set a Box-Cox transformation with `lambda=0` to ensure the forecasts stay positive.

```
fit <- nnetar(sunspotarea, lambda=0)
autoplot(forecast(fit,h=30))
```

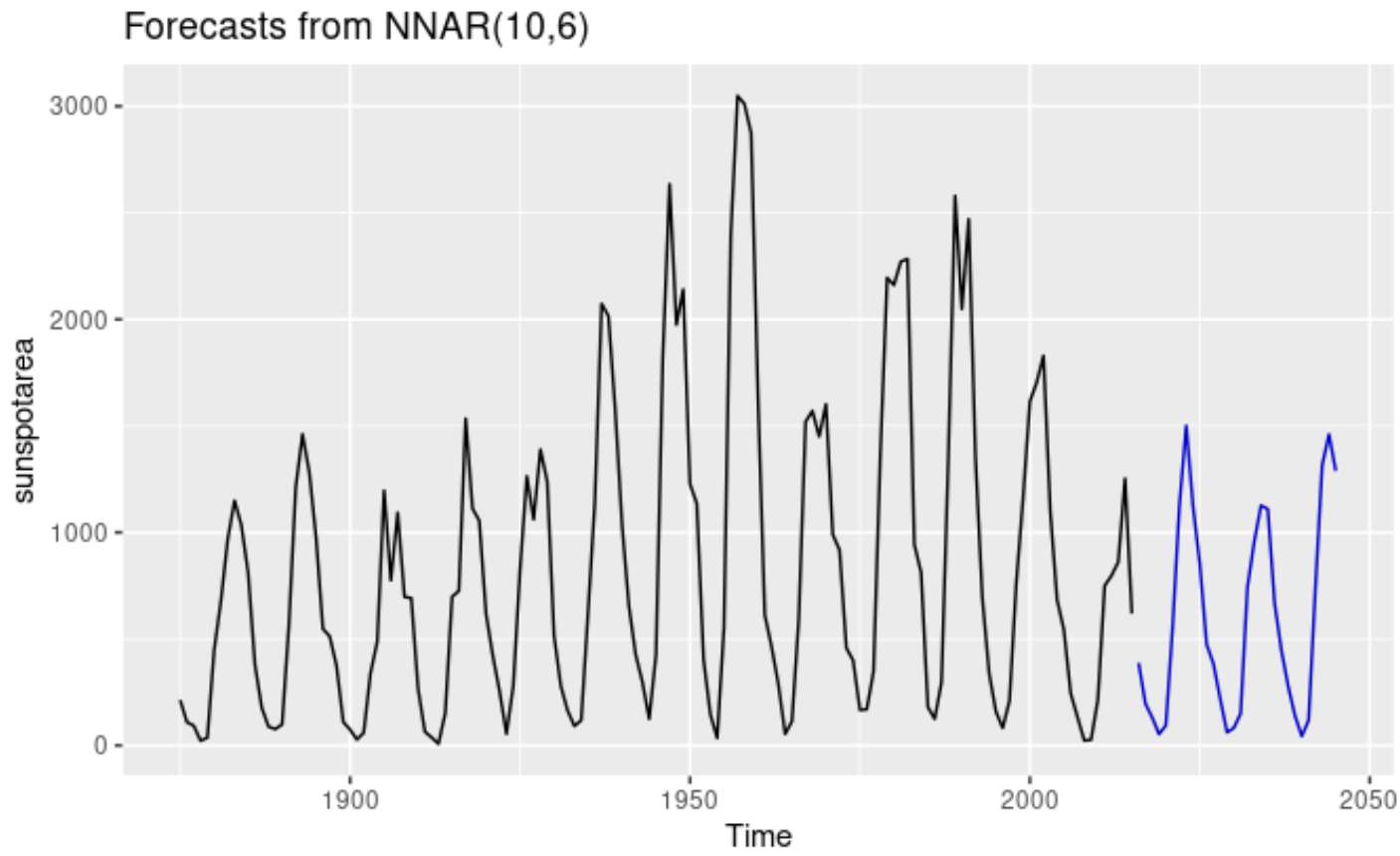

Figure 11.13: Forecasts from a neural network with ten lagged inputs and one hidden layer containing six neurons.

Here, the last 10 observations are used as predictors, and there are 6 neurons in the hidden layer. The cyclicity in the data has been modelled well. We can also see the asymmetry of the cycles has been captured by the model, where the increasing part of the cycle is steeper than the decreasing part of the cycle. This is one difference between a NNAR model and a linear AR model — while linear AR models can model cyclicity, the modelled cycles are always symmetric.

Prediction intervals

Unlike most of the methods considered in this book, neural networks are not based on a well-defined stochastic model, and so it is not straightforward to derive prediction intervals for the resultant forecasts. However, we can still compute prediction intervals using simulation where future sample paths are generated using bootstrapped residuals (as described in Section 3.5).

The neural network fitted to the sunspot data can be written as

$$y_t = f(\boldsymbol{y}_{t-1}) + \varepsilon_t$$

where $\boldsymbol{y}_{t-1} = (y_{t-1}, y_{t-2}, \dots, y_{t-10})'$ is a vector containing lagged values of the series, and f is a neural network with 6 hidden nodes in a single layer. The error series $\{\varepsilon_t\}$ is assumed to be homoscedastic (and possibly also normally distributed).

We can simulate future sample paths of this model iteratively, by randomly generating a value for ε_t, either from a normal distribution, or by resampling from the historical values. So if ε^*_{T+1} is a random draw from the distribution of errors at time $T+1$, then

$$y^*_{T+1} = f(\boldsymbol{y}_T) + \varepsilon^*_{T+1}$$

is one possible draw from the forecast distribution for y_{T+1}. Setting $\boldsymbol{y}^*_{T+1} = (y^*_{T+1}, y_T, \dots, y_{T-6})'$, we can then repeat the process to get

$$y^*_{T+2} = f(\boldsymbol{y}^*_{T+1}) + \varepsilon^*_{T+2}.$$

In this way, we can iteratively simulate a future sample path. By repeatedly simulating sample paths, we build up knowledge of the distribution for all future values based on the fitted neural network.

Here is a simulation of 9 possible future sample paths for the sunspot data. Each sample path covers the next 30 years after the observed data.

```
sim <- ts(matrix(0, nrow=30L, ncol=9L),
  start=end(sunspotarea)[1L]+1L)
for(i in seq(9))
  sim[,i] <- simulate(fit, nsim=30L)
autoplot(sunspotarea) + autolayer(sim)
```

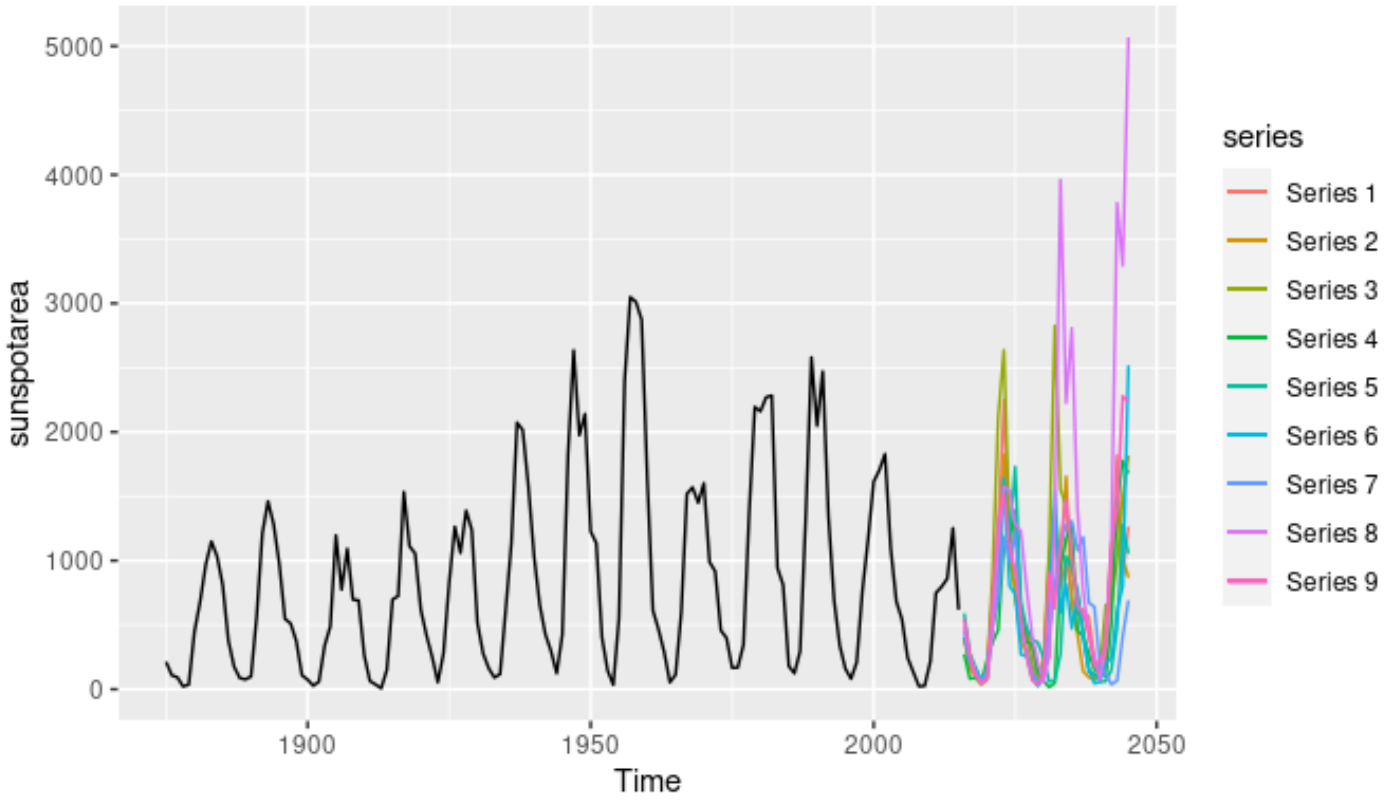

Figure 11.14: Future sample paths for the annual sunspot data.

If we do this a few hundred or thousand times, we can get a good picture of the forecast distributions. This is how the `forecast()` function produces prediction intervals for NNAR models:

```
fcast <- forecast(fit, PI=TRUE, h=30)
autoplot(fcast)
```

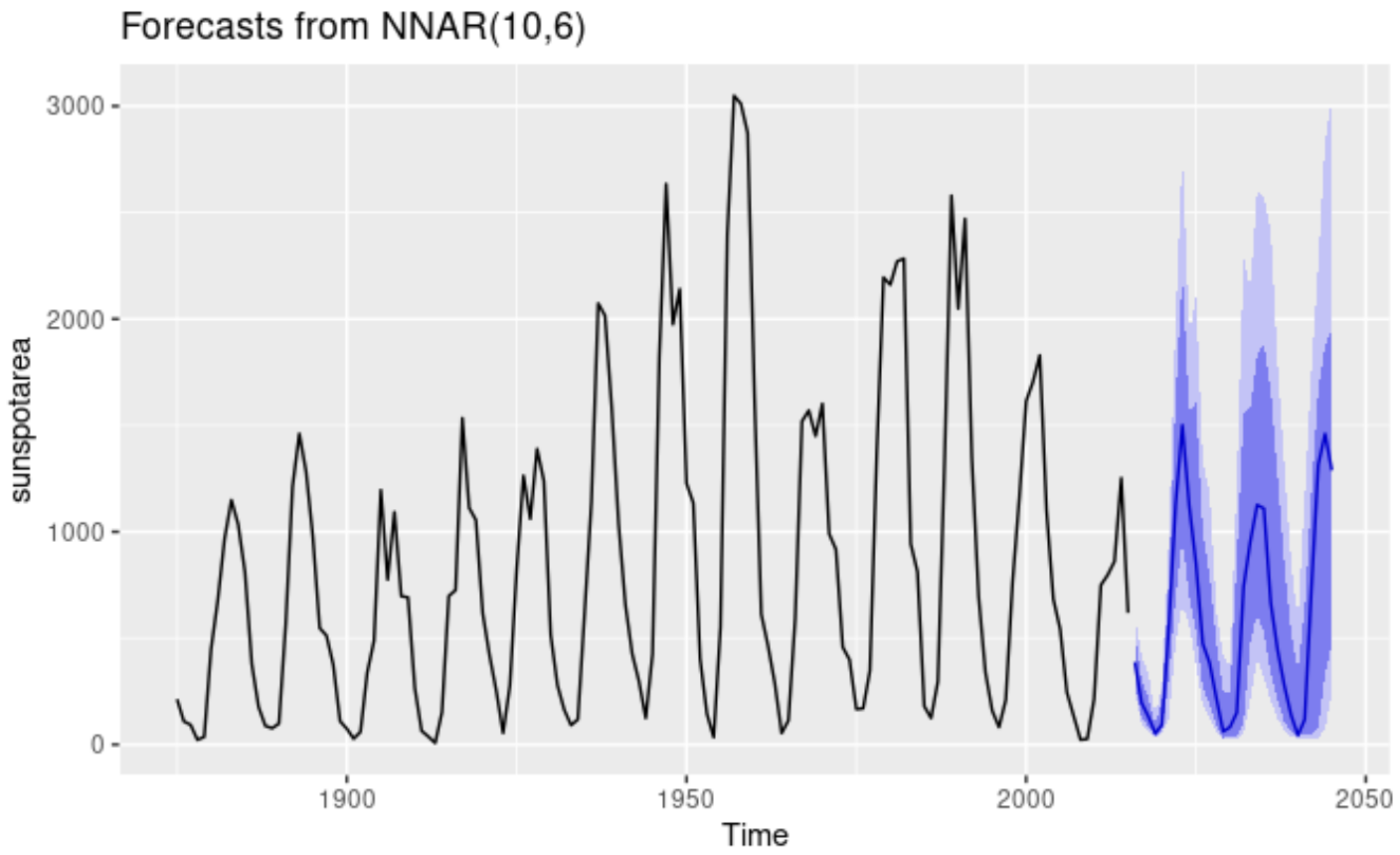

Figure 11.15: Forecasts with prediction intervals for the annual sunspot data. Prediction intervals are computed using simulated future sample paths.

Because it is a little slow, `PI=FALSE` is the default, so prediction intervals are not computed unless requested. The `npaths` argument in `forecast()` controls how many simulations are done (default 1000). By default, the errors are drawn from a normal distribution. The `bootstrap` argument allows the errors to be "bootstrapped" (i.e., randomly drawn from the historical errors).