9.3 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" organised in layers. The predictors (or inputs) form the bottom layer, and the forecasts (or outputs) form the top layer. There may be intermediate layers containing "hidden neurons".

The very simplest networks contain no hidden layers and are equivalent to linear regression. Figure 9.9 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 MSE. Of course, in this simple example, we can use linear regression which is a much more efficient method for training the model.

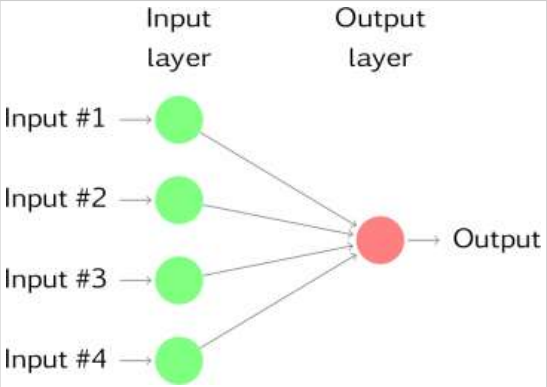

Figure 9.9: A simple neural network equivalent to a linear regression.

Once we add an intermediate layer with hidden neurons, the neural network becomes non-linear. A simple example is shown in Figure 9.10.

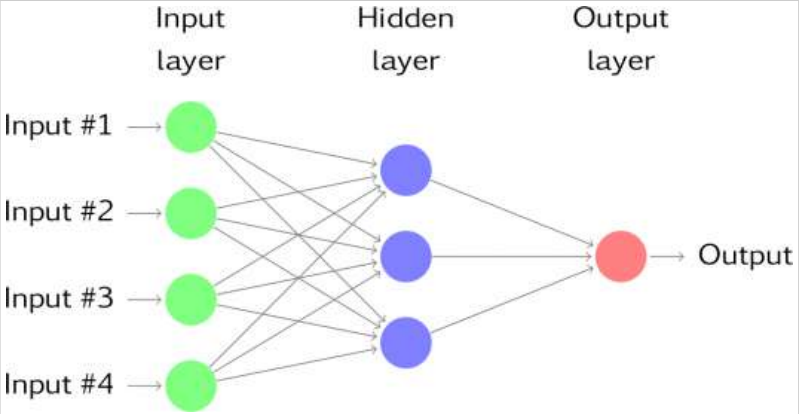

Figure 9.10: A neural network with four inputs and one hidden layer with three hidden neurons.

This is known as a *multilayer feed-forward network* where each layer of nodes receives inputs from the previous layers. The outputs of 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 9.10 are linearly combined 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 $w_{1,1}, \ldots, w_{4,3}$ are "learned" from the data. The values of the weights are often restricted to prevent them 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, which 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.

Example 9.5 Credit scoring

To illustrate neural network forecasting, we will use the credit scoring example that was discussed in Chapter 5. There we fitted the following linear regression model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + e,$$

where

$$
\begin{aligned}
y &= \text{credit score},\\
x_1 &= \log \text{savings},\\
x_2 &= \log \text{income},\\
x_3 &= \log \text{time at current address},\\
x_4 &= \log \text{time in current job},\\
e &= \text{error}.
\end{aligned}
$$

Here "log" means the transformation $\log(x+1)$. This could be represented by the network shown in Figure 9.9 where the inputs are $x_1, \dots, x_4$ and the output is y. The more sophisticated neural network shown in Figure 9.10 could be fitted as follows.

R code

```
library(caret)
creditlog  <- data.frame(score=credit$score,
 log.savings=log(credit$savings+1),
 log.income=log(credit$income+1),
 log.address=log(credit$time.address+1),
 log.employed=log(credit$time.employed+1),
 fte=credit$fte, single=credit$single)
fit  <- avNNet(score ~ log.savings + log.income + log.address +
 log.employed, data=creditlog, repeats=25, size=3, decay=0.1,
 linout=TRUE)
```

The `avNNet` function from the **caret** package fits a feed-forward neural network with one hidden layer. The network specified here contains three nodes (`size=3`) in the hidden layer. The decay parameter has been set to 0.1. The argument `repeats=25` indicates that 25 networks were trained and their predictions are to be averaged. The argument `linout=TRUE` indicates that the output is obtained using a linear function. In this book, we will always specify `linout=TRUE`.

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 can use lagged values in a neural network autoregression.

In this book, we only consider feed-forward networks with one hidden layer, and 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 to forecast 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 to 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 automatically selected. 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).

Example 9.6: 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 9.11, forecasts from an NNAR(9,5) are shown for the next 20 years.

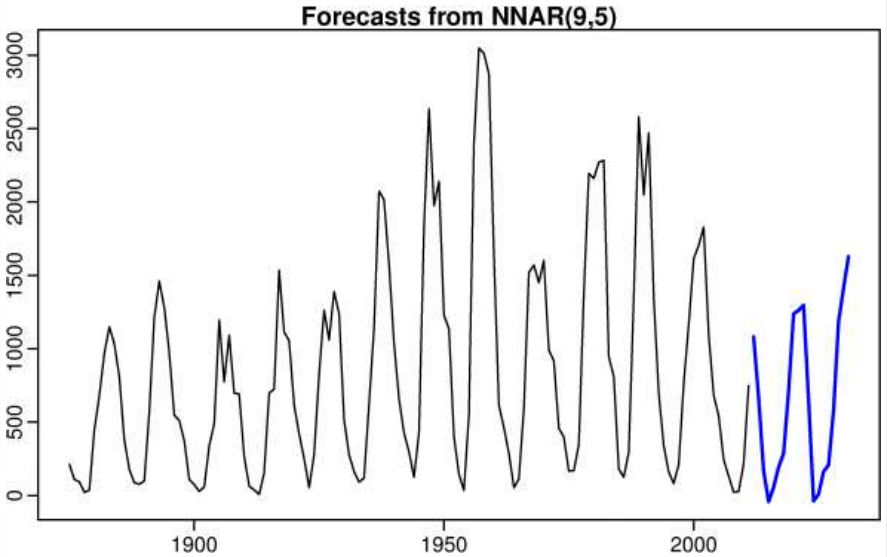

Figure 9.11: Forecasts from a neural network with nine lagged inputs and one hidden layer containing five neurons.

R code

```
fit <- nnetar(sunspotarea)
plot(forecast(fit,h=20))
```

The forecasts actually go slightly negative, which is of course impossible. If we wanted to restrict the forecasts to remain positive, we could use a log transformation (specified by the Box-Cox parameter $\lambda = 0$):

R code

```
fit <- nnetar(sunspotarea,lambda=0)
plot(forecast(fit,h=20))
```

‹ 9.2 Vector autoregressions up 9.4 Forecasting hierarchical or grouped time series ›