R offre librerie meno complicate e più accessibili, come nnet (https://cran.r-project.org/web/packages/nnet/), AMORE (https://cran.r-project.org/web/packages/AMORE/) e neuralnet (https://cran.r-project.org/web/packages/neuralnet/). Questi brevi esempi in R mostrano come addestrare una rete di classificazione (sul dataset Iris) e una rete di regressione (sul dataset Boston). Partendo dal problema di classificazione, il codice seguente carica il dataset e lo suddivide in test di addestramento e set di test:

```
library(MASS)
library("neuralnet")

target <- model.matrix( ~ Species - 1, data=iris )
colnames(target) <- c("setosa", "versicolor", "virginica")

set.seed(101)
index <- sample(1:nrow(iris), 100)

train_predictors <- iris[index, 1:4]
test_predictors  <- iris[-index, 1:4]
```

Poiché le reti neurali sono basate sulla discesa del gradiente, è necessario standardizzare (ricordiamo, sottrarre la media e dividere per la deviazione standard) o normalizzare gli input (ovvero riportare i valori di input entro uno specifico intervallo). È meglio normalizzare, perché in questo modo per ogni feature il minimo diventa 0 e il massimo è 1. Naturalmente, si deve prestare attenzione a effettuare queste conversioni numeriche utilizzando come riferimento il set di addestramento al fine di evitare qualsiasi possibilità di utilizzare informazioni dal test out-of-sample (il problema dello snooping di cui abbiamo parlato nei capitoli precedenti).

```
min_vector   <- apply(train_predictors, 2, min)
range_vector <- apply(train_predictors, 2, max) -
  apply(train_predictors, 2, min)

train_scaled <- cbind(scale(train_predictors,
                            min_vector, range_vector),
                            target[index,])
test_scaled  <- cbind(scale(test_predictors,
                            min_vector, range_vector),
                            target[-index,])

summary(train_scaled)
```

Quando il set di addestramento è pronto, si può iniziare ad addestrare il modello per giungere alla previsione dei tre output binari, ciascuno dei quali rappresenta una classe. L'output risultante è un valore per ciascuna classe proporzionale alla probabilità che sia la classe reale. Le previsioni vengono scelte prendendo il valore più elevato. È anche possibile visualizzare la rete utilizzando la funzionalità grafica interna e vedendo quindi l'architettura interna della rete neurale e i pesi assegnati, come mostrato nella Figura 12.5.

```
set.seed(102)
nn_iris <- neuralnet(setosa + versicolor + virginica ~
                                Sepal.Length + Sepal.Width
                                + Petal.Length + Petal.Width,
                                data=train_scaled, hidden=c(2),
                                 linear.output=F)
plot(nn_iris)
predictions <- compute(nn_iris, test_scaled[,1:4])
y_predicted <- apply(predictions$net.result,1,which.max)
y_true <- apply(test_scaled[,5:7],1,which.max)
confusion_matrix <- table(y_true, y_predicted)
accuracy <- sum(diag(confusion_matrix)) /
                                        sum(confusion_matrix)
print (confusion_matrix)
print (paste("Accuracy:",accuracy))
```

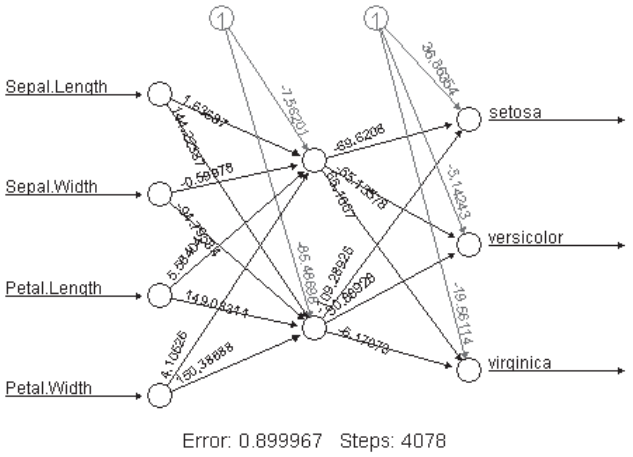

FIGURA 12.5 Si può tracciare il grafico di una rete neurale addestrata.

L'esempio riportato di seguito mostra come prevedere i valori delle case a Boston, utilizzando il dataset Boston. La procedura è la medesima utilizzata poc'anzi per la classificazione, ma in questo caso c'è un'unica unità di output. Il codice traccia anche i risultati previsti del set di test rispetto ai valori reali, per verificare se il modello è corretto:

```
no_examples <- nrow(Boston)
features <- colnames(Boston)

set.seed(101)
index <- sample(1:no_examples, 400)

train <- Boston[index,]
test  <- Boston[-index,]
```