Chapter 5
Artificial Neural Networks

5.1 Multilayer perceptron

In the early days of artificial intelligence (AI), artificial neural networks (ANNs) were considered a promising approach to find good learning algorithms to solve practical application problems [189]. Perhaps, a certain unjustified hype was associated to their use, since, nowadays, ANNs seem to have less appeal for researchers. In fact, they are not considered to be among the top 10 data mining techniques [237]. Moreover, publications using ANNs are found not to be backed by a sound statistical analysis [75] and that statistical evaluation of ANNs experiments is a necessity [74]. There are, however, applications in which ANNs have been successfully used. Among such applications, there are the applications in the agricultural-related areas which are discussed in Section 5.4 of this chapter. Therefore, even though they may not be so appealing for some researchers anymore, we decided to dedicate this chapter to ANNs.

ANNs can be used as data mining techniques for classification. They are inspired by biological systems, and particularly by research on the human brain. ANNs are developed and organized in such a way that they are able to learn and generalize from data and experience [99]. Despite their origin related to brain studies, the networks discussed in this chapter have little to do with biology.

In general, ANNs are used for modeling functions having an unknown mathematical expression. In Chapter 2 we showed that, given a set of independent variables (inputs) and corresponding dependent variables (outputs), interpolation and regression techniques can be used for modeling such data. As already discussed, when interpolating polynomials are used, one problem is that their degree grows with the dimension of the set of data. This problem is avoided when splines or regression approaches are used. However, there are reasons that brought researchers to use ANNs instead of interpolation and regression models. First of all, ANNs do not become more complex if the set of data used is larger. Moreover, ANNs can model very complex functions without the need of finding their (complex) mathematical expressions.

A. Mucherino et al., *Data Mining in Agriculture*, Springer Optimization and Its Applications 34, 107
DOI: 10.1007/978-0-387-88615-2_5,

According to [180], ANNs consist in a number of independent and simple processors: the neurons. The network is formed by neurons, which are connected and usually organized in layers. The human brain contains tens of billions of neurons and tens of trillions of such connections. Each neuron is characterized by an activity level and by its input and output connections. The activity level represents the state of polarization of a neuron, the input connections feed the neuron with signals, whereas the output connections broadcast the neuron signal to others. All these neuron properties are represented mathematically by real numbers. Each link or connection between neurons has an associated weight, which determines the effect of the incoming input on the activation level of the neuron. The weights can be positive or negative. If a connection has a positive weight, its effect on the signal passing through is excitement, whereas effect is inhibitory if the weight is negative. In other words, if the weight sign is positive, it raises the activation; if the sign is negative, it lowers the activation. ANNs differ from each other by the way in which the neurons are connected, by the way each neuron processes its input, and by the learning method used. Usually, the network structure is defined a priori, and must be tailored to the process that must be modeled. During the learning phase, only the connection weights are optimized in a way that the network can respond with the given outputs when it has certain inputs.

The multilayer perceptron is the kind of ANNs that are the focus of this chapter. The multilayer perceptron has the neurons organized in layers, one input layer, one or multiple hidden layers and one output layer. In some applications there are only one or just two hidden layers, but it is more convenient to have more than two layers in some other applications. Figure 5.1 shows an example of a multilayer perceptron. The input data are provided to the network through the input layer, which sends this information to the hidden layers. The data are processed by the hidden layers and the output layer. Each neuron receives output signals from the neurons in the previous layer and sends its output to the neurons in the successive layer. The last layer, the output one, receives the inputs from the neurons in the last hidden layer, and its neurons provide the output values. The neurons of the input layer do not perform any computation, since they are just allowed to receive the data that they send to the first hidden layer. Layer by layer, then, the neurons communicate among them and process the data they receive. The network is able to provide the output values after the inputs have propagated from the input layer to the output layer through the entire network.

As already mentioned, initial research on ANNs presented them as a very promising approach for learning from data. For instance, many benefits in using ANNs have been discussed in [99]. In this paper, besides presenting neural networks as a good alternative to polynomial interpolation or regression, other advantages are discussed. ANNs are for instance said to be able to handle imperfect and incomplete data. Therefore they may be useful when working with data from the real world, which are noisy and imprecise. Moreover, data from the real world are often complex, and since multilayer perceptron is nonlinear, it can capture complex iterations among the input variables of the system. Finally, ANNs are highly parallel, so that

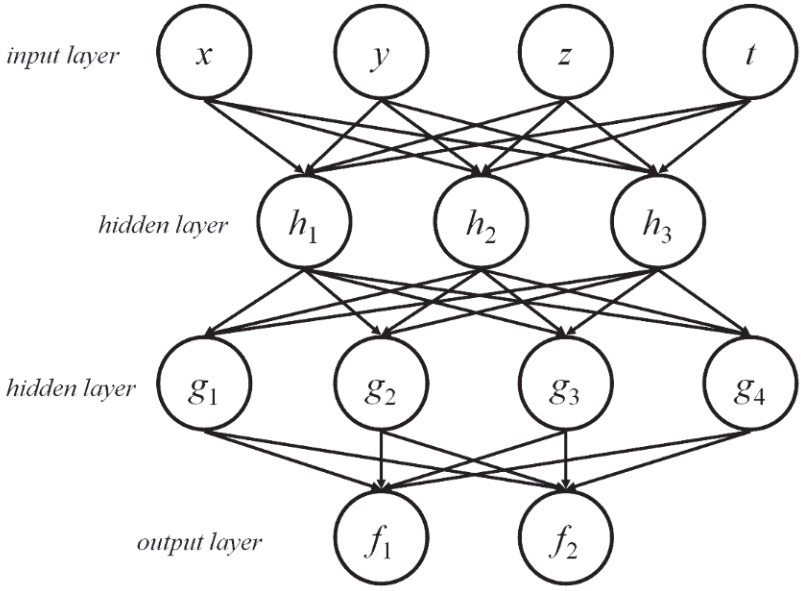

Fig. 5.1 Multilayer perceptron general scheme.

they can naturally be developed in a parallel environment. In fact, an implementation for parallel computing of ANNs is provided in Section 9.3.3.

As already pointed out, ANNs can be used for mathematically modeling a certain unknown process. A network having n neurons in the input layer and m neurons in the output layer can be used for describing a function having n independent variables and m dependent variables. Using a mathematical language, ANNs can model functions defined in $\Re^n$ and having values in $\Re^m$ (where $\Re$ is the set of real numbers), if the network has n input neurons and m output neurons.

Each neuron receives as input the outputs from the neurons in the previous layer. Passing through the connections, these outputs are lowered or raised, depending on the connection weights. All these values are assumed to sum linearly yielding an activation value for the current neuron. If j is one of the neurons in the current layer, and L is the number of neurons in the previous layer connected to j, then the function

$$\text{net}_j = \sum_i^L w_{ij} o_i$$

computes the activation value for j, where w_{ij} is the weight associated to the link between the neuron i of the previous layer and the neuron j, and where o_i is the output provided by neuron i. The obtained value is then processed by neuron j in the current layer, by computing its output

$$o_j = O_j(\text{net}_j).$$

The function O_j is fixed for each neuron and it is normally a nonlinear function of its activation value. Usually it is chosen to be a smooth function, and the default choice is the standard sigmoid function:

$$O_j = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}.$$

Other functions are also used, as for instance the one used in the application described below in Section 5.4.2, which is the logistic function:

$$O_j = \text{logistic}(x) = \frac{1}{1 + e^{\frac{-x}{T}}}. \tag{5.1}$$

In the formula, the parameter T of the logistic function yields functions of different slopes. Section 5.4.1, instead, is focused on an application in which the logistic function is used only for the neurons on the output layer, while the function O_j corresponds to

$$O_j = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{5.2}$$

if the neurons on the hidden layers are considered. Function (5.2) is called hyperbolic tangent. Functions O_j, in general, are usually predefined a priori and they are not modified during the learning process.

The simplest neural network whose neurons are organized in layers is the one having one input layer, one output layer, but no hidden layers. This kind of network is actually called *single* perceptron and was presented in [198] in 1958. In this case, the input variables are processed only by the neurons in the output layer: the output variables are computed by functions O_j which have as input a linear combination of the input variables. This kind of network may be useful for its simplicity, but it cannot solve some types of problems, in particular when the function to model is not linearly separable. In other words, if the function to model cannot be written as a linear combination of its inputs, then the single perceptron cannot model it, just because net_j is a linear combination. The hidden layers have been introduced for overcoming this problem: a multilayer perceptron having just one hidden layer can model nonlinear functions.

ANNs are commonly used as classification techniques. They can be used for supervised learning, since the network parameters (the neuron weights) are computed by computational procedures based on a certain training set of data. The hope is that the network so designed is able to generalize, i.e., to correctly classify data that are not present in the training set. As explained in [215], generalization is usually affected by three factors. The first one consists of the size and efficiency of the training set, since small sets of data cannot contain information enough for generalization, and, even when they are larger, they may not be efficient. The training set may, for instance, contain data which are representative for some classes and not for some others, providing in this way incomplete information. Another important factor is the complexity of the network. The number of hidden layers can impact the accuracy of the system: a system with a large number of hidden layers has better chances

to provide better accuracy. However, if the complexity grows, the training and the normal use of the neural network may become too computationally demanding, and therefore a good trade-off must be found. Finally, a crucial factor is the complexity of the process which needs to be modeled. After the network size is determined, including the number of hidden layers and the number of neurons for each of them, the network must be trained. An important issue is to select a good algorithm for this purpose. In Section 5.2 we will overview some of the commonly used training algorithms for ANNs. After the learning phase, the network is able to use what it did learn from the data. Evaluating and using these trained networks can be computationally expensive, and some redundant links and useless neurons may be removed to make the network more efficient. This phase is called *pruning* of the network, and these issues are discussed in Section 5.3.

5.2 Training a neural network

The problem of learning in neural networks is the problem of finding a set of connection weights which allows the network to carry out the desired computations. During the learning process, the neural network must learn how to model the data. The most used method is the back-propagation method.

The basic idea of the back-propagation method is as follows. It is supposed that a set of input data and a set of corresponding output data are available. It is required that the network is able to provide the correct output when a certain input is provided. In other words, the network has to deliver certain output results $\{o_1, o_2, \ldots, o_m\}$ when it receives certain input variables $\{i_1, i_2, \ldots, i_n\}$. The back-propagation method works on the weights associated to each link between neurons. Predefined weights can be used at the start of the algorithm. In the case predefined weights are not available they can be randomly generated. The method starts feeding the network with inputs i_k and allows these signals to propagate through the network layer by layer. Every time a neuron receives inputs from the neurons in the previous layer, it computes a weighted sum of them and sends its output to the neurons in the successive layer. When the signal arrives to the output layer, its neurons compute the outputs. Let us denote the generic output obtained with the symbols co_k, meaning "current output." At this point, these current co_k outputs and the outputs o_k the network should learn to provide can be compared. The difference between o_k and co_k can be defined as the current error e_k which is present in the output neuron k. The error values are then passed back to the last hidden layer using the same weights. This backward propagation gives the name to the algorithm. By computing the weighted sums of the received errors, each neuron is able to compute its contribution to the output error, and adjust its weight for reducing the output error. The back-propagation method is iterative and it stops when the network can process the input with sufficient accuracy. The final weights represent what the network has learned.

The difficult task in this back-propagation process is to find out the connections between the neurons that are not performing correctly or that are performing worse

than the others. This is a nontrivial problem, especially if the network has hidden layers. A possible way for facing the problem is to avoid finding a single connection or a set of connections to blame for the network error and considering a measure of the overall performance of the system. The performance of the network can be defined as follows:

$$E = \sum_{\xi,k} \left(co_k^{\xi} - o_k^{\xi} \right)^2, \tag{5.3}$$

where o_k represents the k^{th} expected output, and where co_k is the current output provided by the network. The superscripts ξ correspond to the sets of input/output to be learned. E represents the total error of the network. Therefore the task of learning from a given training set can be seen as an optimization problem, where E must be minimized. The problem is unconstrained, and it may be solved by using one of the optimization methods discussed in Section 1.4.

Many approaches have been proposed for the learning phase of a neural network. In [129], for instance, genetic algorithms (GAs) are used [88]. GAs are meta-heuristic methods for global optimization and they use simple operators in order to simulate evolution according to Darwinian theory. GAs are among the meta-heuristic methods for global optimization listed in Section 1.4. In this case, GAs work with a population of networks, which are randomly generated when the algorithm starts. The main operator in the search is the crossover, which generates new network children starting from network parents. In these studies, a network (or individual or chromosome) is represented as a square matrix such that each single element in row i and column j has value $\eta_{ij} = 0$ if there are no connections between neurons i and j, and value $\eta_{ij} \neq 0$ if there is a connection. This matrix can contain all the information regarding a neural network, such as connectivity and weights. Two special crossover operators are used, which are tailored to the matrix representation of the network. The row-wise crossover is performed generating two children by exchanging two random rows between two parents. In the same way, the column-wise crossover is performed by exchanging two random columns between two parents. GAs have also been used in other studies with the aim of training a network as fast and efficiently as possible. In [113], for instance, GAs have been used coupled with a BFGS (Broyden-Fletcher-Goldfarb-Shano) method [204] for improving the training performance.

One problem that may occur during the learning process is overfitting. At some point, in later stages of the learning process, the network may start to fit the data in the training set very well. In the meantime, though, it may start to lose generalization. In other words, the network begins to be very good at reproducing the data on which it is trained, whereas it may be completely wrong on any other kind of data. For avoiding overfitting, the generalization ability of the network during training can be checked and the learning process can be stopped when this ability begins to decrease. The simplest method is to divide the data into a training set and a validation set. The training set can then be used during the learning process, whereas the validation set can be used to estimate the generalization ability. The learning process must therefore be stopped when the error on the validation set begins to increase. This technique can

work very well for avoiding overfitting, but it may not be practical when only a small amount of data is available, since the validation data cannot be used for training.

After a network has been trained, it is expected to be able to classify samples using the parameters established during the training phase. It is desirable that the classification is as fast as possible. In order to improve the performance of a neural network, the network can be pruned. During the pruning process, all the redundant and useless connections that affect the performance of the network can be removed. In Section 5.3 we will discuss pruning strategies for neural networks.

5.3 The pruning process

As discussed in Section 5.2, ANNs can generalize well from the training set if the network does not overfit during the training process. A way for avoiding this phenomenon could be to use the smallest network able to model a certain problem [193]. However, it is not easy to determine the optimal network size for a particular problem. One possible approach is to train successively smaller networks until the smallest one is found that is able to learn from the data. This process can work but it can be time consuming. Therefore, other strategies have been proposed over time in order to improve the ability of the neural network to generalize.

Training many networks having a decreasing number of neurons and choosing the smallest one able to generalize from the data can be computationally demanding. The alternative is to try training a network with a number of neurons unnecessarily large. Training a large network can be expensive, but not as expensive as training many networks. The problem is that this large network can also be very expensive to use, and for this reason it needs to be pruned after the training process. The initial large network size allows learning reasonably quickly and the network can then work efficiently when the unnecessary neurons and connections have been removed.

A brute-force pruning method is as follows. After the network has been trained, all its weights can be considered one per time, and set to zero. The total error provided by the network can then be checked on the training set. If the error increases too much, it means that the link corresponding to the weight set to zero is indispensable and cannot be removed. Otherwise, if the total error is acceptable, the link can be eliminated from the network. If all the connections related to one neuron are removed, the neuron itself can be eliminated from the network. The brute-force method can be quite expensive. If W is the number of weights contained in the network and M the number of input/output couples from the training set, the computational cost is about MW^2, because, every time the method tries to delete one of the W weights, it has to check M errors over $W - 1$ connection.

Other methods, even more sophisticated, have been proposed over the time for pruning a neural network, and they can be divided into two main groups. One group contains methods that estimate the sensitivity of the error function to the removal of a neuron, and the ones with the least effect are then removed. The second group

contains methods that add terms to the objective function that reward the network for choosing solutions in which the weights are smaller. For instance, a term proportional to the sum of all weight magnitudes favors solutions with small weights. The ones that are nearly zero are not likely to influence the output much and hence they can be eliminated. There is some overlap in these two groups, because the term added to the objective function can include sensitivity terms.

Many pruning tests have been proposed in the literature. In [37], for instance, the pruning problem is formulated in terms of solving a system of linear equations. The basic idea is to iteratively eliminate neurons and adjust the remaining weights in such a way that the network performance does not worsen over the entire training set. In [78], instead of pruning the network as a whole, it is pruned layer by layer with the use of a pruning decision based on local parameters. Other recent works on pruning algorithms can be found in [179, 238, 249].

5.4 Applications

ANNs have been used experimentally for decades in practical applications. An interesting work is for instance the one presented in [200] for detecting frontal views of faces in gray-scale images. In this approach, more than one neural network is used and each of them is trained to output the presence or the absence of a face in an image. This is a very difficult detection task. Unlike face recognition, in which the classes to be discriminated represent different kind of faces, the two classes to be discriminated in face detection are "images containing faces" and "images not containing faces." Obtaining a representative sample representing images without faces is the most difficult task. Experiments presented in [200] showed that neural network can handle this kind of problem, and one of the experiments is presented in Figure 5.2. Other general applications of neural networks include the classification of recorded musical instrument sounds [62], the development of decision making tools in the field of cancer [155], and the classification of events during high-energy physics experiments at the Super Proton Synchrotron at CERN in Geneva, Switzerland [224].

Neural networks have been successfully applied in agriculture and related fields. For instance, a neural network approach has been proposed in [135] for evaluating sugar and acid contents of a variety of oranges by a machine vision system. Machine vision can replace human visual judgment by providing a more consistent and reliable system. The measurement of the sugar or acid contents of an orange fruit is, however, a difficult task, because its skin is thick and usually light cannot penetrate the skin effectively. In the approach proposed in [135], images of the oranges have been taken and the sugar and acid contents have been measured by the standard equipment. The neural network has been used for finding the relationships between the orange aspect and the acid and sugar contents. The used three-layer network has been able to predict that reddish, low height, medium size and glossy orange fruits are relatively sweet.

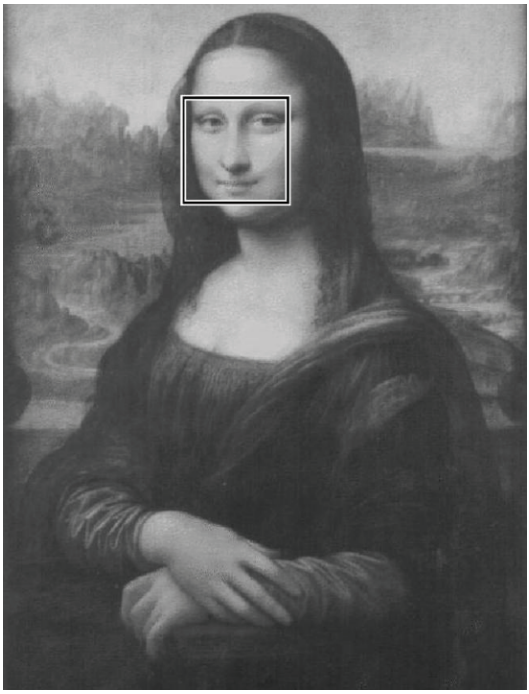

Fig. 5.2 The face and the smile of Mona Lisa recognized by a neural network system. Image from [200].

However, the network could not provide a clear indication of the level of sugar content, but the feasibility to evaluate inside quality of fruits by neural networks and machine vision has been anyway demonstrated.

Other applications of neural networks in the field of agriculture are for instance:

- Classification of fertile and infertile eggs by machine vision [53];
- Prediction of flowering and maturity dates of soybean [67];
- Detection of cracks in eggs using computer vision [185];
- Forecasting water resources variables [160];
- Detection of pig coughs in farms by recorded sounds [45];
- Detection of watercores in apples by X-ray images [210];
- Wine classifications by taste sensors made from ultra-thin films [196];
- Modeling of sediment transport [22].

In the following we will focus on the problem of detecting pig coughs with the aim of identifying diseases in farms (Section 5.4.1) and on the problem of detecting watercore inside apples for a good selection of fruits for the market (Section 5.4.2).

5.4.1 Pig cough recognition

Coughing, in human and animals, is associated with the sudden expulsion of air. This is a defense mechanism of the body, against the possible entry of materials into the respiratory system. Coughing is typically accompanied by a sound, whose changes may reflect the presence of diseases affecting the airways or the lungs or of early symptoms of diseases. If someone is coughing, it is easy to say if he or she has a bad or normal cough from the sound produced. In the same way, the sound provided by pig coughing can be used for monitoring possible health problems. An expert could say if the cough of a pig signals the presence of a potential disease, and eventually check the health of the pig. Nowadays, however, human attention is not so present anymore, because big farms have a large quantity of animals and, moreover, the environment can be very harmful for the presence of contagious diseases [3].

Systems for the automatic control of the pig houses are useful. Their use can prevent the transmission of diseases from pigs to humans, and at the same time guarantee a constant control on pig health conditions. Therefore, considerable efforts have been undertaken for the development and application of sensors and sensing techniques for diagnosis in pig farms. Besides the advantages farmers can have, such as improving the health of the pigs and avoiding contaminations, the final consumer also can benefit from these techniques. The early detection of an animal disease can bring to the consumer's table better meat, by reducing, for instance, the residuals of antibiotics. The different techniques developed for cough detection have the common characteristic of being based on supervised learning methods. As a consequence, the failure or success of a technique depends highly on the quality of the training set of data. The training set is obtained by experimental observations, where the sounds produced by pigs are recorded and where each record is labeled by an expert in different ways. An expert farmer is indeed able to distinguish among coughs and other sounds pigs can issue.

We will focus in this section on the studies presented in [45, 170, 171] where neural networks have been used as a supervised learning technique. There is also a similar example in the literature that uses a fuzzy c-means algorithm [231]. In the neural network approach, a metal chanmber has been built in order to perform the experiments (see Figure 5.3). It is covered with transparent plastic for controlling the environment around the animal, and its dimensions are 2 m long, 0.80 m wide and 0.95 m high. The pigs are invited to enter in the metal chamber and sound measurements by a microphone are recorded. During this process, the environment inside the chamber is controlled by checking the temperature, the dust and the NH_3 concentration, and other variables. A full description of the experiment set up is presented in [169]. We just point out that the microphone is placed in the chamber from 0.4 m to 1.0 m from the pig, and it is positioned through an aperture into the plastic cover. The sample rate chosen is 22,050 Hz, because the frequencies of a typical cough are below 10,000 Hz. After the pig is invited to enter the chamber, normal pig sounds are recorded, such as grunting and other sounds due to respiration. Other sounds from the surrounding environment are also recorded. Animal movements can cause metal clanging, because the construction used in the experiment is metallic. Moreover, the

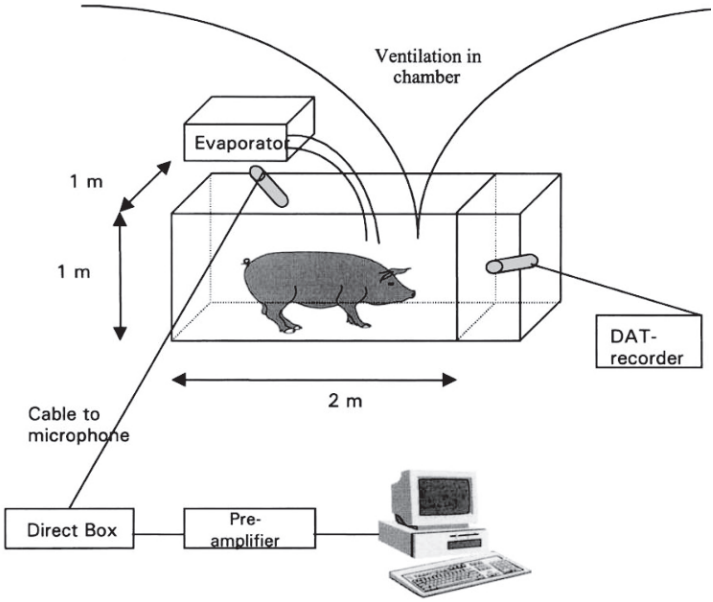

Fig. 5.3 A schematic representation of the test procedure for recording the sounds issued by pigs. Image from [45].

controlled environment that needs ventilation and the presence of researchers may cause other noises. When all these sounds are recorded, pigs are finally induced to cough for recording cough sounds.

A neural network is trained using the sounds obtained during these experiments. The training set contained 354 sounds: 212 samples are records of coughs from different pigs, 50 samples represent metal clanging, 23 samples grunts, and 69 samples background noise. Each sound is analyzed by a human expert to determine whether it is a cough or not. All these samples are then divided into two sets, the training and the testing set. The sounds have been equally distributed between the two sets, except for the sounds of coughs that are used more in the testing set, because it is important to check if the recognition of the coughs is correct. Figure 5.4 shows the time signal of a pig cough. The amplitude for the cough in all the samples recorded is 0.5 ± 0.09. The grunts have a larger duration and variability; among all the samples the duration is 1.2 ± 0.15. The time signal of these sounds is analyzed mathematically and transformed in a vector formed by 64 real numbers. For further details about this process, the reader may refer to [45] and the citations therein. This transformation is very useful, because it allows one to work on vectors and not on signals. In the following, then, two sounds are compared by comparing the components of two real vectors. They are normalized before use, because their components can vary significantly even when comparing two vectors from the same class. These variations are mainly due to the distance and direction between the pigs and the microphone.

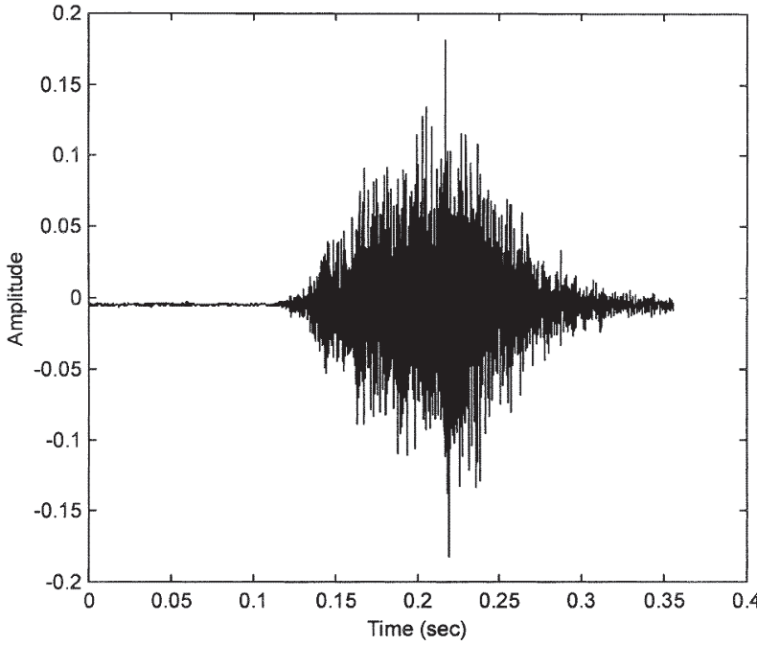

Fig. 5.4 The time signal of a pig cough. Image from [45].

Such variations do not negatively affect the quality of the sound because of the low environmental noise.

The network is trained using a BFGS optimization procedure [204]. The network is a multilayer perceptron with one or two hidden layers of hyperbolic tangent neurons (see equation (5.2)), while the output layer consists in logistic neurons (see equation (5.1)). The multilayer perceptrons with two hidden layers did not provide any improvement on the correct classification percentage. Once the network is trained to discriminate between coughs and metal clanging, it is able to reach percentages of correct recognition greater than 90%. This is a very difficult task, because these two sounds have a similar frequency range. Then the network is trained to distinguish among four sounds: coughs, metal clanging, grunting and background noise. The confusion matrix shown in Figure 5.5 describes how many of the sounds, whose correct class appears in the first column, are misclassified. The recognition accuracy remains high, as the figure shows.

5.4.2 Sorting apples by watercore

Grading fruits before marketing is a very important process that can increase the profits, since quality defects decrease the marketability of the fruit. In this section, we

Sound	Coughs	Metal Clanging	Grunting	Noise
Coughs	69.5	21.7	8.7	0.0
Metal clanging	4.3	82.6	0.0	0.0
Grunting	0.0	0.0	91.3	8.7
Noise	0.0	0.0	8.7	91.3

Fig. 5.5 The confusion matrix for a 4-class multilayer perceptron trained for recognizing pig sounds.

will focus on grading procedures of apples. Some defects, such as discoloration, poor shape, external damage, and bruising in light colored apples are visible externally and apples containing such defects are commonly removed at sorting tables. A recognition system based on the k-means algorithm and based on the external appearance of the fruit is discussed in Section 3.5.2. Unfortunately, other defects are internal. Such defects are particularly harmful to consumer acceptance since they are typically recognized after purchase. Internal defects include internal browning, internal small black regions of unknown origin, core and other rot, watercore and insect damage. Bruises are generally referred to as external defects. Codling moth problems in exported apples can be expected to increase with the phase-out of methyl bromide fumigation, resulting in more sustained insect damage.

We will focus in this section on watercore. Watercore is an internal apple disorder, found in most apple varieties, that adversely affects the longevity of the fruit. Apples with slight or mild watercore are sweeter, and this may be considered a good feature of the apple. Unfortunately, apples with moderate to severe degree of watercore cannot be stored for any length of time. Moreover, internal tissue breakdown of a few fruits during storage may damage the whole batch. For this reason, apples with a sufficient percentage of watercore need to be detected and separated from the batch. Non-destructive methods such as X-ray imaging have shown promising results for detecting internal quality defects in various horticultural products. X-ray is a radioactive method which can penetrate into the apple without serious surface reflection. In particular, radiographic imaging, which is sensitive to density differences, is a good candidate for detecting the internal defects so far neglected as well as for detecting watercore and bruises. The fact that this technology is also quite inexpensive makes the X-ray method the best choice for detecting internal disorders in apples. The major challenge in this field is thus to develop adequate image analysis and classification schemes that can successfully classify products using X-ray image data.

A normal fruit has 20–35% of the total tissue volume occupied by the intercellular air space, whereas in apples with watercore this large air space is filled with a liquid. These changes in density and water content of fruit can be exploited for watercore detection by non-destructive techniques based on X-ray. In [203] watercores in apples have been detected with an accuracy of more than 90% by using still X-ray images. In this approach, apples have been scanned by X-ray and successively sliced and photographed (see Figure 5.6). The obtained images, both normal and X-ray images, have then been used to characterize them as defective or not. In this phase, both kinds of images are inspected and evaluated by human experts. In order to create an automatic classifier, computational procedures are needed for performing some

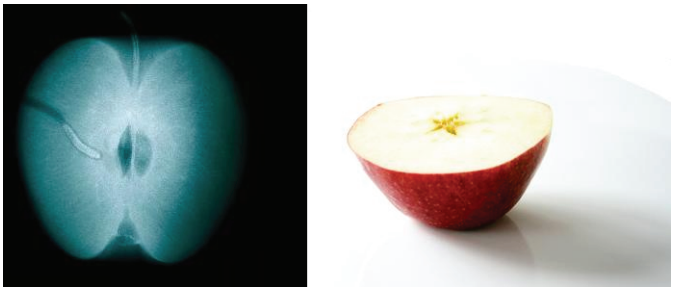

Fig. 5.6 X-ray and classic view of an apple. X-ray can be useful for detecting internal defects without slicing the fruit.

of these tasks on a computer. The inspection of the X-ray images can be carried out by a computer, which needs though to learn how to inspect such images before. Therefore, classifiers such as neural networks can be useful in these studies. In fact a method based on ANNs has been proposed in [210] for detecting watercores in apples by X-ray.

In this work, line scan images of 240 Red Delicious apples with varying degree of watercore have been acquired and three features of the images, considered good indicators of watercore, have been extracted from the images. Details about this process can be found in [209]. After scanning, the apples are cut and opened in order to check the presence of watercores from a human expert. Each fruit is scored on a scale from 0 to 2 based on watercore severity. Apples labeled with 0 do not have a watercore or they have a mild watercore, whereas apples labeled with 1 have a moderate watercore and the ones labeled with 2 a severe watercore. The final set of data obtained includes the three X-ray features and the corresponding scores. The aim is to teach a neural network to predict the score when it is fed by the three features of the images.

The set of data is randomly divided into two subsets. The first one includes 150 samples (55 having score 0, 46 having the score 1 and 49 having the score 2) and is used as training set. The second one includes 90 samples (58 having the score 0, 14 having the score 1 and 18 having the score 2) and is used as testing set. The employed network is a multilayer perceptron having three layers in total, and hence only one hidden layer. The output function O_j is the logistic function (see equation (5.1)).

Usually the number of neurons in the input layer equals the dimension of the input vector, and therefore the considered network has three neurons on the input layer, each one related to one of the features extracted from the X-ray images. There are actually two choices for selecting the number of neurons in the output layer, depending on the nature of the problem at hand. In classification problems where the network is trained to recognize well-defined classes, the number of output nodes usually equals the number of classes. A sample is recognized as belonging to a certain class when the output corresponding to this class is higher in value. However, there may be problems in using this strategy. When the classes are not well-defined by the network, the network may give two similar outputs and there may be uncertainty in

assigning a sample to a class or to another. For this reason, a single output neuron is used in these studies with a continuous value coupled with two threshold levels. If the output is lower than the first threshold, the sample is considered to have mild watercore. Instead, it is considered to have severe watercore if the output value is larger than the second threshold. Samples are considered to have moderate watercore when the output value is between the two thresholds.

The number of neurons in the hidden layer is often determined either by trial and error or by ad hoc schemes. Several networks with different numbers of hidden neurons (from 2 to 10) have been evaluated to determine an optimal structure for achieving a good generalization. The network with the maximum classification accuracy is considered the optimal classifier for sorting apples. The optimal network found in this application has 4 hidden neurons. The method used for training the network is the standard back-propagation method described in Section 5.2. The 150 samples contained in the training set are divided in two other subsets. The first one, containing 105 samples, is used for normal training, while the second one having just 45 samples is used for validating the network during the training process. This strategy is used for avoiding the overfitting of the network.

The best classification accuracy has been obtained using a neural network having four neurons on the hidden layer and by using as thresholds 0.35 and 0.60. The neural classifier achieved an overall accuracy of 88% with the losses and false positives as low as 5%. The overall accuracy approached the target of 90% whereas the losses and false positives were well below the target limits of 10%. In [210] the classification of the apples has also been carried out using a fuzzy c-means method. The experiments showed that the neural network classifier performs better.

5.5 Software for neural networks

Instead of presenting experiments in MATLAB® with the technique discussed in this chapter, we just provide here a list of available software for neural networks. The main reason is that the training and use of the simplest neural network would require the need of developing relatively long codes in MATLAB. Since there is various software available for training and using neural networks, we decided it was not worthwhile to devote a section to possible implementations of the data mining technique in MATLAB. The list below includes the most popular software currently on the Internet. The reader can extend the list with a simple search with Google.

- **NeuroSolutions**, `http://www.nd.com/`
  It is advertised as the most powerful and flexible neural network modeling software currently available. NeuroSolutions is also available for MATLAB and Excel. It has an icon-based network design interface with an implementation of advanced learning procedures, such as conjugate gradients and backpropagation through time.
- **EasyNN**, `http://www.easynn.com/`
  Quoting the Web site, complex data analysis with EasyNN is fast and simple.

Prediction, forecasting, classification and time series projection is easy. Moreover, EasyNN allows one to train, validate and query ANNs with just a few button pushes.

- **MATLAB toolbox**, `http://www.mathworks.com/products/neuralnet/`
  Neural Network Toolbox extends the MATLAB environment with tools for designing, implementing, visualizing, and simulating neural networks. This software provides comprehensive support for many proved network paradigms, as well as graphical user interfaces (GUIs) that enable one to design and manage the networks.

5.6 Exercises

Exercises related to ANNs follow.

1. Consider a multilayer perceptron having one input neuron, two hidden neurons on only one hidden layer and one output neuron. The function O_j related to all the active neurons is just the identity function. Train the network so that it is able to model the equation:
$$y = 2x.$$
2. Prove that the network used in the previous exercise cannot model exactly the equation:
$$y = 2x + 1.$$
3. Train a multilayer perceptron having one hidden layer with 2 neurons for the AND classification problem. The network has 2 input neurons, 2 hidden neurons and only one output neuron. Suppose that the function O_j is not preassigned and choose it so that the network can perform the AND operator.
4. Consider a network with the same structure of the one in the previous exercise and with the sigmoid function (O_j) associated to the only output neuron. Suppose that all the weights have unitary value. Feed the network with the points (6, 1) and $(-1, -1)$.
5. Keep working on the same network as the one in the previous exercise, but have all the weights equal to 2 and the logistic function (with $T = 2$) associated to the output neuron, and feed the network with the points (1, 1) and (0, 2).
6. Consider the two networks used in Exercises 4 and 5. State which of them can have the hidden layer deleted without changing the output of the network.
7. Consider the network with 2 input neurons, 3 hidden neurons on only one hidden layer and one output neuron. Suppose that the weights are equal to 0.1 if they are related to links with the input layer, and that they are equal to 0.3 if related to links with the output neuron. Suppose that the identity function is associated to all the neurons. Remove a link that caused the inactivation of one neuron.
8. Design a network having the same organization in layer and the same number of neurons as in the previous exercise, but having all the neurons from a layer connected to the following layer.