6.2.8 Neural networks for data mining

Walter Kosters[1]

In many application areas neural networks are known to be valuable tools. This also holds for data mining. In this chapter we discuss the use of neural networks, we shall give an informal description of the way they work internally (just for one distinctive member of the large family of neural networks), and we finally focus on their usefulness for data mining. In particular we shall not deal with biological or psychological backgrounds. We only notice that the idea of neural networks originates from the physiology of the human brain.

General background

Neural networks are powerful general purpose learning devices. This sentence shows the strength (the general purpose character), which is also its weakness: its generality. Another weakness is the complex internal structure, which — if one finally understands the algorithms involved — still shows black box behavior: it is very hard to get an idea of the meaning of the internal computations.
Neural networks perform well in pattern recognition tasks, such as recognition of handwritten characters or spoken text. It should be noted that the way these networks learn (their 'training') is often supervised: the user should provide as many positive examples as possible; negative examples are also helpful. So for classification and clustering it is necessary to have classified or clustered input available. Note, however, that special neural networks are available that can cope with unsupervised situations.
It is easy to build a neural network that tries to solve a given problem. In fact, many software packages contain plug and play neural networks. But still many parameters need to be set, the training stage may take a while, and the tuning therefore can be awkward. State of the art hardware is a prerogative.
Another feature of neural networks is their random behavior. This does not mean that they act or react randomly, but that their training process contains random elements. For instance, in the beginning the network is carefully initialized with random numbers. When this is repeated, the same input set may yield very different networks. Sometimes they differ in performance, one showing good behavior, while others behave badly. Note that it is perfectly possible for a neural network to get stuck in some suboptimal situation; this unfortunate situation can sometimes be avoided through sophisticated techniques.
Neural networks have many parameters, such as learning rate, number of layers, number of neurons, and so on. It always pays off to use different settings for a given problem, thereby trying to find an — at least for the time being — optimal setting in an empiric way. It is not necessarily true that larger networks outperform smaller ones. Not only will the training process take longer, it might

1 Dr W.A. Kosters, kosters@liacs.nl, Leiden University, LIACS, Leiden, The Netherlands

also be the case that the smaller network is more capable of catching the problem at hand.
One difficult problem is to decide whether or not the training has finished. In fact, training can go on as long as one likes, but this sometimes leads to a phenomenon known as ‘overfitting’: the network gets better and better on the examples it uses during training, but looses its generalizing power. Careful schemata using independent validation sets can avoid these pitfalls. If done carefully, neural networks are perfectly capable of dealing with noisy data, but the danger of overfitting is always present.

Working principles

In this section we describe a simple neural network, officially called a multilayer feed forward neural network. The presentation is kept simple in order to achieve enough understanding of the matter. We focus on what is known as Backpropagation, the most common neural network algorithm. In the next section we shall discuss other ‘architectures’.

Network

In our simple network we have one real variable *x* which is our input. We try to learn *f*(*x*), where the function *f* is unknown to us, but correct pairs (*x*,*f*(*x*)) are available. This function (or the value *f*(*x*)) is called the target. Often *x* and *f*(*x*) are normalized between 0 and 1. For instance, *x* may be a time variable and *f*(*x*) may be the height of the water in a harbor, or *x* may indicate the distance from the wall and *f*(*x*) may be the desired speed of a robot, or *x* is a zip code and *f*(*x*) is the mean income in the corresponding area.
The network has to be trained in such a way that, given input *x*, it delivers output *O* with the property that *O* and *f*(*x*) differ as little as possible. The difference *f*(*x*)–*O* between target and output is called the error. Note that the error depends on the particular choice of *x*.

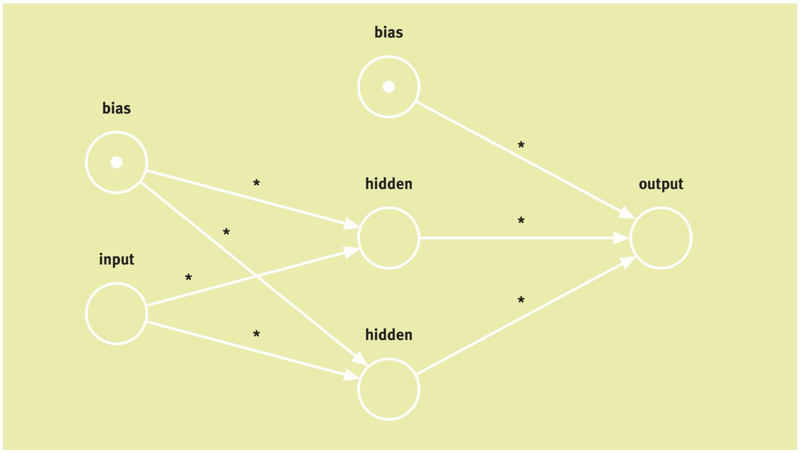

Figure 1
*An example neural network with one input neuron, two hidden neurons and one output neuron. Note the two special bias neurons. Every directed connection has an associated weight, denoted by a *.*

The network itself consists of several so-called neurons. These can be considered as very simple input-output devices. A very simple neural network (see Figure 1) may have one input neuron, one output neuron, and two other neurons: the so-called hidden ones. The input neuron receives the input *x* and hands it to the two hidden neurons. It does so by multiplying *x* by some 'weight', one for every connection. A weight can be viewed as being attached to the directed connection between two neurons. A hidden neuron receives its input, and delivers its output to the output neuron, again multiplying it by some weight. The internal function of a neuron is usually very simple: if its input is low, its output will be low (near zero), and if its input is high, its output will be high (near one). This transfer function is governed by some parameters, one being the 'threshold' or 'bias', which is often implemented by means of extra neurons, one for every layer. Note that in this example network we have four independent weights, or seven if the bias neurons are added. In Figure 1 they are indicated by means of *'s.

Training

The purpose of the training stage is to update the weights in such a way that errors approach, if possible. The change of weights is directed by the errors. Larger errors will lead to larger changes, where the weights that contribute the most are heavily adapted. The so-called learning rate determines the relative change in this process. The algorithm that is used here is called Backpropagation: it propagates the error back through the network, from output to input.

The training usually takes place in the following way. The network is presented with a (random) series of correct input-output pairs. For every pair Backpropagation is used to update (and improve) the weights. After some fixed period, or if the errors are small enough, the training is stopped. The network found can be judged by giving it fresh input-output pairs: the so-called test set. It should come as no surprise that on these pairs the performance of the network will be inferior to that on the training examples. Nevertheless, it gives a good measure of the quality of the network. The performance on a special validation set may be used for the decision to stop training or not: if the error on the validation set starts to increase, this may indicate over fitting.
Many variations are possible. Let us briefly describe some possibilities:

- Training and testing requires a large quantity of examples. In some cases, not enough examples are available. Methods like bootstrapping can be used to artificially increase the number of input-output pairs.
- The behavior of the network is that of a black box. In some cases it is possible to manipulate the internal structure to match the problem at hand. In particular cases this might be extremely difficult, and it is sometimes preferable

to restrict oneself to more complex and hard to interpret networks that give adequate output.
- In line with Occam's razor, which says that in case of several acceptable solutions the simplest one should be preferred, neural network researchers developed all sorts of schemata to decrease network complexity. This results in more complex learning rules, that for instance cause weights to be zero (corresponding to the elimination of weights).
- It might be useful to train several networks at the same time, giving an ensemble of networks. Their independent results can then be combined to obtain a better joined output. In this case statistical techniques can improve the outcome.
- The training can be adapted in many ways. It can or cannot keep track of infeasible solutions. The price paid is that it can become harder to find the proper training algorithm and the proper parameters. But there are many other possibilities. For instance, instead of just using the current input-output pair one can also use information on the previous pair(s) in the form of 'momentum'. Instead of separate incremental training for every example, it is also possible to combine several examples in so-called batch mode. The learning rate can also be adapted during the training process.

In our example we dealt with a situation in which there was only one input variable and one output variable. By adding adequate neurons it is easy to generalize to more complex situations. It is easy to add hidden neurons, which can also be 'layered'. The 'layered' neurons sum all their incoming signals. In this more general setting the error measure is a sum of the squares error. Note that this generalization makes the network much more complex. It is even possible to let the number of neurons grow or shrink during training.
A small problem occurs if one or more variables are not real-valued. As an example, think of a situation where an output variable should contain the day of the week in the form of an integer between 1 and 7. The network may provide 0.314 as output, which can be interpreted as day 3. It is also possible to represent the output by means of seven neurons, each one corresponding to a certain day of the week. Hopefully the network will produce situations where only one of the seven outputs has a value near one, indicating that this day is the proper output.

Data mining with neural networks

In this section we briefly describe some other 'architectures', especially suited to data mining purposes. For more details and yet even more possibilities the interested reader is referred to texts like [Bishop, 1995] and [Silipo, 1999]. These works cover Hopfield nets, Boltzmann machines, specialized networks for time series analysis, and so on.

The Radial Basis Function

The Radial Basis Function (RBF) architecture is especially suited for clustering. Special units try to catch prototypes for each cluster. Variants like probabilistic neural networks are meant for classification tasks. For all these nets a training set of labeled data is necessary. If not, other methods are needed.

Unsupervised learning

In unsupervised learning the network itself tries to find patterns in the data.

Competitive learning

In competitive learning, the neurons compete for an unclassified input example; during the training process their weights gradually converge to some sort of equilibrium, providing an adequate description of the clustering that is hopefully learned.

Self-Organizing Map

One particular successful technique is Kohonen's Self-Organizing Map (SOM, see Section 6.3.2), which can be viewed as a non-linear projection: the number of dimensions of the data diminishes, showing underlying structure. This so-called feature map has the property that inputs which are close together activate neurons that are close together in the network. Other variants pursue Principal Components Analysis (see Section 6.2.4).

Rule extraction

There are several ways to extract rules from neural networks. In a local approach one tries to understand the local behavior of (part of) the network, in a global approach the overall behavior is examined. As an example of the latter, in sensitivity analysis the effect of changes in one variable are studied: how does the output vary?

Conclusion

We may conclude that the family of neural network techniques contains a large number of data mining tools, especially suited for clustering, classification and prediction. See also the CD-rom: tutorial [Andina, 2001]

Literature

- Andina de la Fuente, D. (2001). Artificial Neural Networks. Universidad Politecnica de Madrid, Spain
- Bishop, C.M. (1995). Neural Networks for Pattern Recognition. Oxford University Press
- Silipo, R. (1999). Neural Networks. In: M. Berthold, D.J. Hand. (eds.). Intelligent Data Analysis. Chapter 7. Springer Verlag