Neural Networks: Applications & Processes

Deepak Garg, Lalit Garg, Kant Rajni
Computer Science & Engineering Department,
Thapar Institute of Engineering & technology
{dgarg,lgarg } @mail.tiet.ac.in

Abstract

Neural networks have seen an explosion of interest over the last few years, and are being successfully applied across an extraordinary range of problem domains, in areas as diverse as finance, medicine, engineering, geology and physics. Indeed, anywhere that there are problems of prediction, classification or control, neural networks are being introduced.

The sweeping success of neural networks can be attributed to a few key factors:

- **Power.** Neural networks are very sophisticated modeling techniques capable of modeling extremely complex functions. In particular, neural networks are *nonlinear*. For many years linear modeling has been the commonly used technique in most modeling domains since linear models have well-known optimization strategies. Where the linear approximation was not valid (which was frequently the case) the models suffered accordingly. Neural networks also keep in check the *curse of dimensionality* problem that bedevils attempts to model nonlinear functions with large numbers of variables.

- **Ease of use.** Neural networks *learn by example*. The neural network user gathers representative data, and then invokes *training algorithms* to automatically learn the structure of the data. Although the user does need to have some heuristic knowledge of how to select and prepare data, how to select an appropriate neural network, and how to interpret the results, the level of user knowledge needed to successfully apply neural networks is much lower than would be the case using (for example) some more traditional nonlinear statistical methods.

Neural networks are also intuitively appealing, based as they are on a crude low-level model of biological neural systems. In the future, the development of this neurobiological modeling may lead to genuinely intelligent computers.

1. Applications for Neural Networks

Neural networks are applicable in virtually every situation in which a relationship between the predictor variables (independents, inputs) and predicted variables (dependents, outputs) exists, even when that relationship is very complex and not easy to articulate in the usual terms of "correlations" or "differences between groups." A few representative examples of problems to which neural network analysis has been applied successfully are:

- **Detection of medical phenomena.** A variety of health-related indices (e.g., a combination of heart rate, levels of various substances in the blood, respiration rate) can be monitored. The onset of a particular medical condition could be associated with a very complex (e.g., nonlinear and interactive) combination of changes on a subset of the variables being monitored. Neural networks have been used to recognize this predictive pattern so that the appropriate treatment can be prescribed.

- **Stock market prediction.** Fluctuations of stock prices and stock indices are another example of a complex, multidimensional, but in some circumstances at least partially-deterministic phenomenon. Neural networks are being used by many technical analysts to make predictions about stock prices based upon a large number of factors such as past performance of other stocks and various economic indicators.
- **Credit assignment.** A variety of pieces of information are usually known about an applicant for a loan. For instance, the applicant's age, education, occupation, and many other facts may be available. After training a neural network on historical data, neural network analysis can identify the most relevant characteristics and use those to classify applicants as good or bad credit risks.
- **Monitoring the condition of machinery.** Neural networks can be instrumental in cutting costs by bringing additional expertise to scheduling the preventive maintenance of machines. A neural network can be trained to distinguish between the sounds a machine makes when it is running normally ("false alarms")

versus when it is on the verge of a problem. After this training period, the expertise of the network can be used to warn a technician of an upcoming breakdown, before it occurs and causes costly unforeseen "downtime."

- **Engine management.** Neural networks have been used to analyze the input of sensors from an engine. The neural network controls the various parameters within which the engine functions, in order to achieve a particular goal, such as minimizing fuel consumption.

2. The Biological Inspiration

Neural networks grew out of research in Artificial Intelligence; specifically, attempts to mimic the fault-tolerance and capacity to learn of biological neural systems by modeling the low-level structure of the brain. The main branch of Artificial Intelligence research in the 1960s - 1980s produced Expert Systems. These are based upon a high-level model of reasoning processes (specifically, the concept that our reasoning processes are built upon manipulation of symbols). It became rapidly apparent that these systems, although very useful in some domains, failed to capture certain key aspects of human intelligence. According to one line of speculation, this was due to their failure to mimic the underlying structure of the brain. In order to reproduce intelligence, it would be necessary to build systems with a similar architecture.

The brain is principally composed of a very large number (circa 10,000,000,000) of *neurons*, massively interconnected (with an average of several thousand interconnects per neuron, although this varies enormously). Each neuron is a specialized cell which can propagate an electrochemical signal. The neuron has a branching input structure (the dendrites), a cell body, and a branching output structure (the axon). The axons of one cell connect to the dendrites of another via a synapse. When a neuron is activated, it *fires* an electrochemical signal along the axon. This signal crosses the synapses to other neurons, which may in turn fire. A neuron fires only if the total signal received at the cell body from the dendrites exceeds a certain level (the firing threshold).

The strength of the signal received by a neuron (and therefore its chances of firing) critically depends on the efficacy of the synapses. Each synapse actually contains a gap, with neurotransmitter chemicals poised to transmit a signal across the gap. One of the most influential researchers into neurological systems (Donald Hebb) postulated that learning consisted principally in altering the "strength" of synaptic connections. For example, in the classic Pavlovian conditioning experiment, where a bell is rung just before dinner is delivered to a dog, the dog rapidly learns to associate the ringing of a bell with the eating of food. The synaptic connections between the appropriate part of the auditory cortex and the salivation glands are strengthened, so that when the auditory cortex is stimulated by the sound of the bell the dog starts to salivate. Recent research in cognitive science, in particular in the area of nonconscious information processing, have further demonstrated the enormous capacity of the human mind to infer ("learn") simple input-output co variations from extremely complex stimuli.

Thus, from a very large number of extremely simple processing units (each performing a weighted sum of its inputs, and then firing a binary signal if the total input exceeds a certain level) the brain manages to perform extremely complex tasks. Of course, there is a great deal of complexity in the brain which has not been discussed here, but it is interesting that artificial neural networks can achieve some remarkable results using a model not much more complex than this.

3. The Basic Artificial Model

To capture the essence of biological neural systems, an artificial *neuron* is defined as follows:

- It receives a number of inputs (either from original data, or from the output of other neurons in the neural network). Each input comes via a connection that has a strength (or *weight*); these weights correspond to synaptic efficacy in a biological neuron. Each neuron also has a single threshold value. The weighted sum of the inputs is formed, and the threshold subtracted, to compose the *activation* of the neuron (also known as the post-synaptic potential, or PSP, of the neuron).

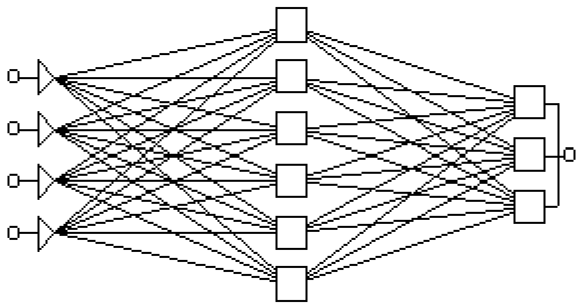
- The activation signal is passed through an activation function (also known as a transfer function) to produce the output of the neuron.

If the step activation function is used (i.e., the neuron's output is 0 if the input is less than zero, and 1 if the input is greater than or equal to 0) then the neuron acts just like the biological neuron described earlier (subtracting the threshold from the weighted sum and comparing with zero is equivalent to comparing the weighted sum to the threshold). Actually, the step function is rarely used in artificial neural networks, as will be discussed. Note also that weights can be negative, which implies that the synapse has an inhibitory rather than excitatory effect on the neuron: inhibitory neurons are found in the brain.

This describes an individual neuron. The next question is: how should neurons be connected together? If a network is to be of any use, there must be inputs (which carry the values of variables of interest in the outside world) and outputs (which form predictions, or control signals). Inputs and outputs correspond to sensory and motor nerves such as those coming from the eyes and leading to the hands. However, there also can be hidden neurons that play an internal role in the network. The input, hidden and output neurons need to be connected together.

The key issue here is *feedback* (Haykin, 1994). A simple network has a *feedforward* structure: signals flow from inputs, forwards through any hidden units, eventually reaching the output units. Such a structure has stable behavior. However, if the network is *recurrent* (contains connections back from later to earlier neurons) it can be unstable, and has very complex dynamics. Recurrent networks are very interesting to researchers in neural networks, but so far it is the feedforward structures that have proved most useful in solving real problems.

A typical feedforward network has neurons arranged in a distinct layered topology. The input layer is not really neural at all: these units simply serve to introduce the values of the input variables. The hidden and output layer neurons are each connected to all of the units in the preceding layer. Again, it is possible to define networks that are partially-connected to only some units in the preceding layer; however, for most applications fully-connected networks are better.



When the network is executed (used), the input variable values are placed in the input units, and then the hidden and output layer units are progressively executed. Each of them calculates its activation value by taking the weighted sum of the outputs of the units in the preceding layer, and subtracting the threshold. The activation value is passed through the activation function to produce the output of the neuron. When the entire network has been executed, the outputs of the output layer act as the output of the entire network.

4. Using a Neural Network

The previous section describes in simplified terms how a neural network turns inputs into outputs. The next important question is: how do you apply a neural network to solve a problem?

The type of problem amenable to solution by a neural network is defined by the way they *work* and the way they are *trained*. Neural networks work by feeding in some input variables, and producing some output variables. They can therefore be used where you have some known information, and would like to infer some unknown information. Some examples are:

Stock market prediction. You know last week's stock prices and today's DOW, NASDAQ, or FTSE index; you want to know tomorrow's stock prices.

Credit assignment. You want to know whether an applicant for a loan is a good or bad credit risk. You usually know applicants' income, previous credit history, etc. (because you ask them these things).

Control. You want to know whether a robot should turn left, turn right, or move forwards in order to reach a target; you know the scene that the robot's camera is currently observing.

Needless to say, not every problem can be solved by a neural network. You may wish to know next week's lottery result, and know your shoe size, but there is no relationship between the two. Indeed, if the lottery is being run correctly, there is no fact you could possibly know that would allow you to infer next week's result. Many financial institutions use, or have experimented with, neural networks for stock market prediction, so it is likely that any trends predictable by neural techniques are already discounted by the market, and (unfortunately), unless you have a sophisticated understanding of that problem domain, you are unlikely to have any success there either!

Therefore, another important requirement for the use of a neural network therefore is that you know (or at least strongly suspect) that there is a relationship between the proposed known inputs and unknown outputs. This relationship may be noisy (you certainly would not expect that the factors given in the stock market prediction example above could give an exact prediction, as prices are clearly influenced by other factors not represented in the input set, and there may be an element of pure randomness) but it must exist.

In general, if you use a neural network, you won't know the exact nature of the relationship between inputs and outputs - if you knew the relationship, you would model it directly. The other key feature of neural networks is that they learn the input/output relationship through training. There are two types of training used in neural networks, with different types of networks using different types of training. These are supervised and unsupervised training, of which supervised is the most common and will be discussed in this section (unsupervised learning is described in a later section).

In supervised learning, the network user assembles a set of *training data*. The training data contains examples of inputs together with the corresponding outputs, and the network learns to infer the relationship between the two. Training data is usually taken from historical records. In the above examples, this might include previous stock prices and DOW, NASDAQ, or FTSE indices, records of previous successful loan applicants, including questionnaires and a record of whether they defaulted or not, or sample robot positions and the correct reaction.

The neural network is then trained using one of the supervised learning algorithms (of which the best known example is *back propagation*, devised by Rumelhart et. al., 1986), which uses the data to adjust the network's weights and thresholds so as to minimize the error in its predictions on the training set. If the network is properly trained, it has then learned to model the (unknown) function that relates the input variables to the output variables, and can subsequently be used to make predictions where the output is *not* known.

5. Gathering Data for Neural Networks

Once you have decided on a problem to solve using neural networks, you will need to gather data for training purposes. The training data set includes a number of *cases*, each containing values for a range of input and output *variables*. The first decisions you will need to make are: which variables to use, and how many (and which) cases to gather.

The choice of variables (at least initially) is guided by intuition. Your own expertise in the problem domain will give you some idea of which input variables are likely to be influential. As a first pass, you should include any variables that you think could have an influence - part of the design process will be to whittle this set down.

Neural networks process numeric data in a fairly limited range. This presents a problem if data is in an unusual range, if there is missing data, or if data is non-numeric. Fortunately, there are methods to deal with each of these problems. Numeric data is scaled into an appropriate range for the network, and missing values can be substituted for using the mean value (or other statistic) of that variable across the other available training cases.

Handling non-numeric data is more difficult. The most common form of non-numeric data consists of nominal-value variables such as *Gender*={*Male*, *Female*}. Nominal-valued variables can be represented numerically. However, neural networks do not tend to perform well with nominal variables that have a large number of possible values.

For example, consider a neural network being trained to estimate the value of houses. The price of houses depends critically on the area of a city in which they are located. A particular city might be subdivided into dozens of named locations, and so it might seem natural to use a nominal-valued variable representing these locations. Unfortunately, it would be very difficult to train a neural network under these circumstances, and a more credible approach would be to assign ratings (based on expert knowledge) to each area; for example, you might assign ratings for the quality of local schools, convenient access to leisure facilities, etc.

Other kinds of non-numeric data must either be converted to numeric form, or discarded. Dates and times, if important, can be converted to an offset value from a starting date/time. Currency values can easily be converted. Unconstrained text fields (such as names) cannot be handled and should be discarded.

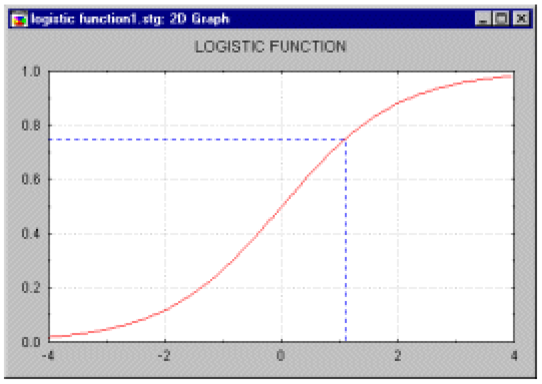
The number of cases required for neural network training frequently presents difficulties. There are some heuristic guidelines, which relate the number of cases needed to the size of the network (the simplest of these says that there should be ten times as many cases as connections in the network). Actually, the number needed is also related to the (unknown) complexity of the underlying function which the network is trying to model, and to the variance of the additive noise. As the number of variables increases, the number of cases required increases nonlinearly, so that with even a fairly small number of variables (perhaps fifty or less) a huge number of cases are required. This problem is known as "the curse of dimensionality," and is discussed further later in this chapter.

For most practical problem domains, the number of cases required will be hundreds or thousands. For very complex problems more may be required, but it would be a rare (even trivial) problem which required less than a hundred cases. If your data is sparser than this, you really don't have enough information to train a network, and the best you can do is probably to fit a linear model. If you have a larger, but still restricted, data set, you can compensate to some extent by forming an ensemble of networks, each trained using a different resampling of the available data, and then average across the predictions of the networks in the ensemble.

Many practical problems suffer from data that is unreliable: some variables may be corrupted by noise, or values may be missing altogether. Neural networks are also noise tolerant. However, there is a limit to this tolerance; if there are occasional outliers far outside the range of normal values for a variable, they may bias the training. The best approach to such outliers is to identify and remove them (either discarding the case, or converting the outlier into a missing value). If outliers are difficult to detect, a city block error function may be used, but this outlier-tolerant training is generally less effective than the standard approach.

Pre- and Post-processing

All neural networks take numeric input and produce numeric output. The transfer function of a unit is typically chosen so that it can accept *input in any* range, and produces *output in a strictly limited* range (it has a squashing effect). Although the input can be in any range, there is a saturation effect so that the unit is only sensitive to inputs within a fairly limited range. The illustration below shows one of the most common transfer functions, the logistic function (also sometimes referred to as the sigmoid function, although strictly speaking it is only one example of a sigmoid - S-shaped - function). In this case, the output is in the range (0,1), and the input is sensitive in a range not much larger than (-1,+1). The function is also smooth and easily differentiable, facts that are critical in allowing the network training algorithms to operate (this is the reason why the step function is not used in practice).



The limited numeric response range, together with the fact that information has to be in numeric form, implies that neural solutions require preprocessing and post-processing stages to be used in real applications. Two issues need to be addressed:

Scaling. Numeric values have to be scaled into a range which is appropriate for the network. Typically, raw variable values are scaled linearly. In some circumstances, nonlinear scaling may be appropriate (for example, if you know that a variable is exponentially distributed, you might take the logarithn.

In some circumstances, non-linear scaling may be appropriate (for example, if you know that a variable is exponentially distributed, you might take the logarithm). Non-linear scaling is not supported in *ST Neural Networks*. Instead, you should scale the variable using *STATISTICA*'s data transformation facilities before transferring the data to *ST Neural Networks*.

Nominal variables. Nominal variables may be two-state (e.g., *Gender*={*Male*,*Female*}) or many-state (i.e., more than two states). A two-state nominal variable is easily represented by transformation into a numeric value (e.g., *Male*=*0*, *Female*=*1*). Many-state nominal variables are more difficult to handle. They can be represented using an ordinal encoding (e.g., *Dog*=*0*,*Budgie*=*1*,*Cat*=*2*) but this implies a (probably) false ordering on the nominal values - in this case, that *Budgies* are in some sense midway between *Dogs* and *Cats*. A better approach, known as *one-of-N* encoding, is to use a number of numeric variables to represent the single nominal variable. The number of numeric variables equals the number of possible values; one of the *N* variables is set, and the others cleared (e.g., *Dog*={*1,0,0*}, *Budgie*={*0,1,0*}, *Cat*={*0,0,1*}). *ST Neural Networks* has facilities to convert both two-state and many-state nominal variables for use in the neural network. Unfortunately, a nominal variable with a large number of states would require a prohibitive number of numeric variables for *one-of-N* encoding, driving up the network size and making training difficult. In such a case it is possible (although unsatisfactory) to model the nominal variable using a single numeric ordinal; a better approach is to look for a different way to represent the information.

Prediction problems may be divided into two main categories:

Classification. In classification, the objective is to determine to which of a number of discrete classes a given input case belongs. Examples include credit assignment (is this person a good or bad credit risk), cancer detection (tumor, clear), signature recognition (forgery, true). In all these cases, the output required is clearly a single nominal variable. The most common classification tasks are (as above) two-state, although many-state tasks are also not unknown.

Regression. In regression, the objective is to predict the value of a (usually) continuous variable: tomorrow's stock price, the fuel consumption of a car, next year's profits. In this case, the output required is a single numeric variable.

6. Conclusion

Neural networks can actually perform a number of regression and/or classification tasks at once, although commonly each network performs only one. In the vast majority of cases, therefore, the network will have a single output variable, although in the case of many-state classification problems, this may correspond to a number of output units (the post-processing stage takes care of the mapping from output units to output variables). If you do define a single network with multiple output variables, it may suffer from cross-talk (the hidden neurons experience difficulty learning, as they are attempting to model at least two functions at once). The best solution is usually to train separate networks for each output, then to combine them into an ensemble so that they can be run as a unit.

References

1. **Kay(1994a)** Information Theoretic Neural Networks for Contextually Guided Unsupervised Learning: Mathematical and Statistical Considerations. Research report.
2. Neural Networks : Simon Haykin, a comprehensive foundation
3. P.D. WasserMan, Neural Computing: Theory and Practice , Van Nortrand Reinhold
4. Bose, Neural Network fundamentals with graphs, algorithms and applications
5. R.H.Neilson, Neurocomputing, Addison Wesley
6. J. Anderson et al., Neurocomputing Vol. 1 & Vol 2, MIT Press, 1986 & 1988