ISSN 1750-9823 (print)
International Journal of Sports Science and Engineering
Vol. 1 (2007) No. 2, pp. 87-96

Use of Artificial Neural Networks for Predicting the Outcome of Cricket Tournaments

D. Roy Choudhury [1], Preeti Bhargava [2 +], Reena [2], Samta Kain [1]

[1] Department of Computer Engineering, Delhi College of Engineering, New Delhi, India

[2] Department of Information Technology, Delhi College of Engineering, New Delhi, India

(*Received May 14 2007, accepted June 16 2007*)

Abstract. The present study aims to train artificial neural networks that assist in predicting the outcome of multi (mainly three) team tournaments. To train the neural networks we use match results for various matches played by the teams in the past 10 years. This is done keeping in mind that the squads or teams haven't changed much over the past 10 years. Once trained, the current tournament's match information will be run through the neural networks. Possible input variables include, for each team, number of matches played, number of matches won, number of matches lost, recent standings of teams, conditions of locations of matches, number of times a team reached the quarterfinal stages of a tournament, number of times a team reached the semifinal stages of a tournament, number of tournaments a team has won. The domains used for training and testing include overall performance in the tournament and in the final match of the tournament. When it's time to predict a tournament's outcome, we run the data through all the networks and add up the score for each team. The team with the highest score is the winner.

Keywords: Artificial neural networks, cricket, sports.

1. Introduction

Artificial Neural Networks (ANNs) have had widespread use in sports applications namely predicting the order of finish in horse racing [1], selecting winning dogs [1] and modeling swimming performance [2]. Where cricket is concerned, work has been done on predicting the match outcome in one day international cricket matches using the Duckworth – Lewis method, while the game is in progress [3]. The use of ANNs in sports has been a topic of intense study with a number of authors highlighting the importance of pattern recognition using ANNs ([4], [5]). In our study we use them for predicting the outcome of international cricket triangular tournaments.

2. Artificial neural networks

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

Neural network simulations appear to be a recent development. However, this field was established before the advent of computers, and has survived at least one major setback and several eras.

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions. Other advantages include:

- Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial

[+] Corresponding author. *Tel.*: 91-09891052439, *Email:* preetibhargava@hotmail.com experience.

- Self- Organization: An ANN can create its own organization or representation of the information it receives during learning time.
- Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
- Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

2.1. A framework for distributed representation of a neural network

An artificial network consists of a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections (see Figure 1).A set of major aspects of a parallel distributed model can be distinguished :

- a set of processing units ('neurons,' 'cells');
- a state of activation y_k for every unit, which equivalent to the output of the unit;
- connections between the units. Generally each connection is defined by a weight w_{jk} which determines the effect which the signal of unit j has on unit k;
- a propagation rule, which determines the effective input s_k of a unit from its external inputs;
- an activation function F_k, which determines the new level of activation based on the effective input $s_k(t)$ and the current activation $y_k(t)$ (i.e., the update);
- an external input (aka bias, offset)
- a method for information gathering (the learning rule);
- an environment within which the system must operate, providing input signals and if necessary error signals.

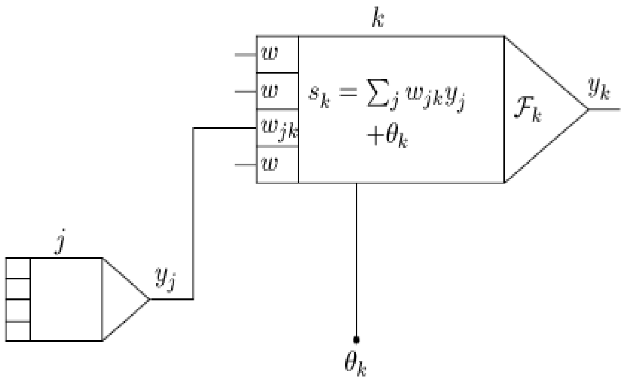


Fig. 1. The basic components of an artificial neural network.
The propagation rule used here is the `standard' weighted summation.

2.1.1 Processing units

Each unit performs a relatively simple job: receive input from neighbors or external sources and use this to compute an output signal which is propagated to other units. Apart from this processing, a second task is the adjustment of the weights. The system is inherently parallel in the sense that many units can carry out their computations at the same time. Within neural systems it is useful to distinguish three types of units: input units (indicated by an index i) which receive data from outside the neural network, output units (indicated by an index o) which send data out of the neural network, and hidden units (indicated by an index h) whose input and output signals remain within the neural network. During operation, units can be updated either synchronously or asynchronously. With synchronous updating, all units update their activation simultaneously; with asynchronous updating, each unit has a (usually fixed) probability of updating its activation at a time t, and usually only one unit will be able to do this at a time.

2.1.2 Connections between units

In most cases we assume that each unit provides an additive contribution to the input of the unit with which it is connected. The total input to unit k is simply the weighted sum of the separate outputs from each of the connected units plus a bias or offset term θ.

The contribution for positive w_{jk} is considered as an excitation and for negative w_{jk} as inhibition. In some cases more complex rules for combining inputs are used, in which a distinction is made between excitatory and inhibitory inputs. We call units with a propagation rule sigma units.

2.1.3 Activation and output rules

We also need a rule which gives the effect of the total input on the activation of the unit. We need a function F_k which takes the total input $s_k(t)$ and the current activation $y_k(t)$ and produces a new value of the activation of the unit k:

$$y_k (t + 1) = F_k(y_k(t); s_k(t))$$

2.1.4 Network topologies

This section focuses on the pattern of connections between the units and the propagation of data. As for this pattern of connections, the main distinction we can make is between:

- Feed-forward networks, where the data flow from input to output units is strictly feedforward. The data processing can extend over multiple (layers of) units, but no feedback connections are present, that is, connections extending from outputs of units to inputs of units in the same layer or previous layers.
- Recurrent networks that do contain feedback connections. Contrary to feed-forward networks, the dynamical properties of the network are important. In some cases, the activation values of the units undergo a relaxation process such that the network will evolve to a stable state in which these activations do not change anymore. In other applications, the changes of the activation values of the output neurons are significant, such that the dynamical behavior constitutes the output of the network.

2.1.5 Training of artificial neural networks

A neural network has to be configured such that the application of a set of inputs produces (either 'direct' or via a relaxation process) the desired set of outputs. Various methods to set the strengths of the connections exist. One way is to set the weights explicitly, using a priori knowledge. Another way is to 'train' the neural network by feeding it teaching patterns and letting it change its weights according to some learning rule.

2.2. Perceptron

Networks with threshold activation functions - A single layer feed-forward network (see Figure 2) consists of one or more output neurons o, each of which is connected with a weighting factor w_{io} to all of the inputs i. In the simplest case the network has only two inputs and a single output, of the network is formed by the activation of the output neuron, which is some function of the input.

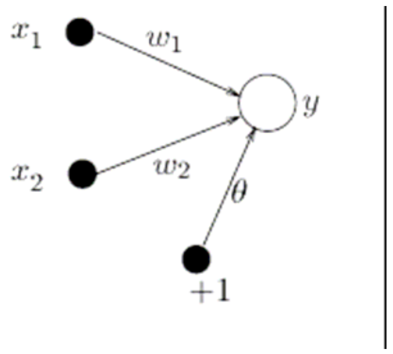


Fig. 2: Single layer network with one output and two inputs.

2.2.1 Perceptron learning rule and convergence theorem

Suppose we have a set of learning samples consisting of an input vector x and a desired output d(x). For a classification task the d(x) is usually +1 or -1. The perceptron learning rule is very simple and can be stated as follows:

1. Start with random weights for the connections;
2. Select an input vector x from the set of training samples;
3. If y = d(x) (the perceptron gives an incorrect response), modify all connections w_i according to: w_i =

$d(x)x_i$;

4. Go back to 2.

Besides modifying the weights, we must also modify the threshold θ. This θ is considered as a connection w_θ between the output neuron and a 'dummy' predicate unit which is always on: $x_\theta = 1$. Given the perceptron learning rule as stated above, this threshold is modified according to:

θ = {0 if the perceptron responds correctly;

{d(x) otherwise.

2.2.2 Multilayer Perceptrons

In this architecture, the units each perform a biased weighted sum of their inputs and pass this activation level through a transfer function to produce their output, and the units are arranged in a layered feedforward topology. The network thus has a simple interpretation as a form of input-output model, with the weights and thresholds (biases) the free parameters of the model. Such networks can model functions of almost arbitrary complexity, with the number of layers, and the number of units in each layer, determining the function complexity. Important issues in Multilayer Perceptrons (MLP) design include specification of the number of hidden layers and the number of units in these layers.

2.2.3 Training Multilayer Perceptrons

Once the number of layers, and number of units in each layer, has been selected, the network's weights and thresholds must be set so as to minimize the prediction error made by the network. This is the role of the *training algorithms*. This automatically adjusts the weights and thresholds in order to minimize this error. This process is equivalent to fitting the model represented by the network to the training data available. The error of a particular configuration of the network can be determined by running all the training cases through the network, comparing the actual output generated with the desired or target outputs. The differences are combined together by an *error function* to give the network error. The most common error functions are the *sum squared error* (used for regression problems), where the individual errors of output units on each case are squared and summed together, and the cross entropy functions (used for maximum likelihood classification).

In traditional modeling approaches (e.g., linear modeling) it is possible to algorithmically determine the model configuration that absolutely minimizes this error. The price paid for the greater (non-linear) modeling power of neural networks is that although we can adjust a network to lower its error, we can never be sure that the error could not be lower still.

A helpful concept here is the error surface. Each of the *N* weights and thresholds of the network (i.e., the free parameters of the model) is taken to be a dimension in space. The *N+1*th dimension is the network error. For any possible configuration of weights the error can be plotted in the *N+1*th dimension, forming an *error surface*. The objective of network training is to find the lowest point in this many-dimensional surface.

In a linear model with *sum squared error* function, this error surface is a parabola (a quadratic), which means that it is a smooth bowl-shape with a single minimum. It is therefore "easy" to locate the minimum.

Neural network error surfaces are much more complex, and are characterized by a number of unhelpful features, such as local minima (which are lower than the surrounding terrain, but above the global minimum), flat-spots and plateaus, saddle-points, and long narrow ravines.

It is not possible to analytically determine where the global minimum of the error surface is, and so neural network training is essentially an exploration of the error surface. From an initially random configuration of weights and thresholds (i.e., a random point on the error surface), the training algorithms incrementally seek for the global minimum. Typically, the gradient (slope) of the error surface is calculated at the current point, and used to make a downhill move. Eventually, the algorithm stops in a low point, which may be a local minimum (but hopefully is the global minimum).

2.3. The Learning Process

We can categorize the learning situations in two distinct sorts. These are:

2.3.1 Supervised learning or Associative learning

In this form of learning, the network is trained by providing it with input and matching output patterns. These input-output pairs can be provided by an external teacher, or by the system which contains the network (self-supervised).

2.3.2 Unsupervised learning or Self- organization

In this form of learning, an output unit is trained to respond to clusters of pattern within the input. In this paradigm the system is supposed to discover statistically salient features of the input population. Unlike the supervised learning paradigm, there is no a priori set of categories into which the patterns are to be classified; rather the system must develop its own representation of the input stimuli.

3. Architecture for training of an artificial neural network

The best-known example of a neural network training algorithm is back propagation. Modern second-order algorithms such as conjugate gradient descent and Levenberg-Marquardt are substantially faster (e.g., an order of magnitude faster) for many problems, but back propagation still has advantages in some circumstances, and is the easiest algorithm to understand.

3.1. The Back Propagation Algorithm

In back propagation, the gradient vector of the error surface is calculated. This vector points along the line of steepest descent from the current point, so we know that if we move along it a "short" distance, we will decrease the error. A sequence of such moves (slowing as we near the bottom) will eventually find a minimum of some sort. The difficult part is to decide how large the steps should be.

Large steps may converge more quickly, but may also overstep the solution or (if the error surface is very eccentric) go off in the wrong direction. In contrast, very small steps may go in the correct direction, but they also require a large number of iterations. In practice, the step size is proportional to the slope (so that the algorithm settles down in a minimum) and to a special constant: the learning rate. The correct setting for the learning rate is application-dependent, and is typically chosen by experiment; it may also be time-varying, getting smaller as the algorithm progresses.

The algorithm is also usually modified by inclusion of a momentum term: this encourages movement in a fixed direction, so that if several steps are taken in the same direction, the algorithm "picks up speed", which gives it the ability to (sometimes) escape local minimum, and also to move rapidly over flat spots and plateaus.

The algorithm therefore progresses iteratively, through a number of epochs. On each epoch, the training cases are each submitted in turn to the network, and target and actual outputs compared and the error calculated. This error, together with the error surface gradient, is used to adjust the weights, and then the process repeats. The initial network configuration is random, and training stops when a given number of epochs elapses, or when the error reaches an acceptable level, or when the error stops improving.

3.2. Levenberg-Marquardt Algorithm

The Levenberg-Marquardt algorithm was designed to approach second-order training speed without having to compute the Hessian matrix. When the performance function has the form of a sum of squares (as is typical in training feedforward networks), then the Hessian matrix can be approximated as

$$\mathbf{H} = \mathbf{J}^T\mathbf{J}$$

and the gradient can be computed as

$$\mathbf{g} = \mathbf{J}^T\mathbf{e}$$

where **J** is the Jacobian matrix that contains first derivatives of the network errors with respect to the weights and biases, and **e** is a vector of network errors. The Levenberg-Marquardt algorithm uses this approximation to the Hessian matrix in the following Newton-like update:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}]^{-1}\mathbf{J}^T\mathbf{e}$$

When the scalar μ is zero, this is just Newton's method, using the approximate Hessian matrix. When μ is large, this becomes gradient descent with a small step size. Newton's method is faster and more accurate near an error minimum, so the aim is to shift toward Newton's method as quickly as possible. Thus, μ is decreased after each successful step (reduction in performance function) and is increased only when a tentative step would increase the performance function. In this way, the performance function is always reduced at each iteration of the algorithm.

4. Over-learning and Generalization

One major problem with the approach outlined above is that it doesn't actually minimize the error which is the expected error the network will make when new cases are submitted to it. In other words, the most desirable property of a network is its ability to generalize to new cases. In reality, the network is trained to minimize the error on the training set, and short of having a perfect and infinitely large training set, this is not the same thing as minimizing the error on the real error surface - the error surface of the underlying and unknown model.

The most important manifestation of this distinction is the problem of over-learning, or over-fitting. It is easiest to demonstrate this concept using polynomial curve fitting rather than neural networks, but the concept is precisely the same.

A polynomial is an equation with terms containing only constants and powers of the variables. For example:

$$y=2x+3$$
$$y=3x^2+4x+1$$

Different polynomials have different shapes, with larger powers (and therefore larger numbers of terms) having steadily more eccentric shapes. Given a set of data, we may want to fit a polynomial curve (i.e., a model) to explain the data. The data is probably noisy, so we don't necessarily expect the best model to pass exactly through all the points. A low-order polynomial may not be sufficiently flexible to fit close to the points, whereas a high-order polynomial is actually too flexible, fitting the data exactly by adopting a highly eccentric shape that is actually unrelated to the underlying function.

Neural networks have precisely the same problem. A network with more weights models a more complex function, and is therefore prone to over-fitting. A network with fewer weights may not be sufficiently powerful to model the underlying function. For example, a network with no hidden layers actually models a simple linear function.

How is the right complexity of network determined? A larger network will almost invariably achieve a lower error eventually, but this may indicate over-fitting rather than good modeling. The answer is to check progress against an independent data set, the selection set. Some of the cases are reserved, and not actually used for training in the back propagation algorithm. Instead, they are used to keep an independent check on the progress of the algorithm. It is invariably the case that the initial performance of the network on training and selection sets is the same (if it is not at least approximately the same, the division of cases between the two sets is probably biased). As training progresses, the training error naturally drops, and providing training is minimizing the true error function, the selection error drops too. However, if the selection error stops dropping, or indeed starts to rise, this indicates that the network is starting to overfit the data, and training should cease. When over-fitting occurs during the training process like this, it is called over-learning. In this case, it is usually advisable to decrease the number of hidden units and/or hidden layers, as the network is over-powerful for the problem at hand. In contrast, if the network is not sufficiently powerful to model the underlying function, over-learning is not likely to occur, and neither training nor selection errors will drop to a satisfactory level.

The problems associated with local minima, and decisions over the size of network to use, imply that using a neural network typically involves experimenting with a large number of different networks, probably training each one a number of times (to avoid being fooled by local minima), and observing individual performances. The key guide to performance here is the selection error. However, following the standard scientific precept that, all else being equal, a simple model is always preferable to a complex model.

A problem with this approach of repeated experimentation is that the selection set plays a key role in selecting the model, which means that it is actually part of the training process. Its reliability as an independent guide to performance of the model is therefore compromised - with sufficient experiments, you may just hit upon a lucky network that happens to perform well on the selection set. To add confidence in the performance of the final model, it is therefore normal practice (at least where the volume of training data allows it) to reserve a third set of cases - the test set. The final model is tested with the test set data, to ensure that the results on the selection and training set are real, and not artifacts of the training process. Of course, to fulfill this role properly the test set should be used only once - if it is in turn used to adjust and reiterate the training process, it effectively becomes selection data!

This division into multiple subsets is very unfortunate, given that we usually have less data than we would ideally desire even for a single subset. We can get around this problem by resampling. Experiments can be conducted using different divisions of the available data into training, selection, and test sets. There are a number of approaches to this subset, including random (monte-carlo) resampling, cross-validation, and bootstrap. If we make design decisions, such as the best configuration of neural network to use, based upon a number of experiments with different subset examples, the results will be much more reliable. We can then either use those experiments solely to guide the decision as to which network types to use, and train such networks from scratch with new samples (this removes any sampling bias); or, we can retain the best networks found during the sampling process, but average their results in an ensemble, which at least mitigates the sampling bias.

5. Data Selection

All the stages data - training, verification and test data must be representative of the underlying model (and, further, the three sets must be independently representative). The old computer science adage "garbage in, garbage out" could not apply more strongly than in neural modeling. If training data is not representative, then the model's worth is at best compromised. At worst, it may be useless. It is worth spelling out the kind of problems which can corrupt a training set:

- **The future is not the past.** Training data is typically historical. If circumstances have changed, relationships which held in the past may no longer hold.
- **All eventualities must be covered.** A neural network can only learn from cases that are present. If people with incomes over \$100,000 per year are a bad credit risk, and your training data includes nobody over \$40,000 per year, you cannot expect it to make a correct decision when it encounters one of the previously-unseen cases. Extrapolation is dangerous with any model, but some types of neural network may make particularly poor predictions in such circumstances.
- **A network learns the easiest features it can.** A classic illustration of this is a vision project designed to automatically recognize tanks. A network is trained on a hundred pictures including tanks, and a hundred not. It achieves a perfect 100% score. When tested on new data, it proves hopeless. The reason? The pictures of tanks are taken on dark, rainy days; the pictures without on sunny days. The network learns to distinguish the (trivial matter of) differences in overall light intensity. To work, the network would need training cases including all weather and lighting conditions under which it is expected to operate - not to mention all types of terrain, angles of shot, distances.
- **Unbalanced data sets.** Since a network minimizes an overall error, the proportion of types of data in the set is critical. A network trained on a data set with 900 good cases and 100 bad will bias its decision towards good cases, as this allows the algorithm to lower the overall error (which is much more heavily influenced by the good cases). If the representation of good and bad cases is different in the real population, the network's decisions may be wrong. A good example would be disease diagnosis. Perhaps 90% of patients routinely tested are clear of a disease. A network is trained on an available data set with a 90/10 split. It is then used in diagnosis on patients complaining of specific problems, where the likelihood of disease is 50/50. The network will react over-cautiously and fail to recognize disease in some unhealthy patients. In contrast, if trained on the "complainants" data, and then tested on "routine" data, the network may raise a high number of false positives. In such circumstances, the data set may need to be crafted to take account of the distribution of data or the network's decisions modified by the inclusion of a *loss matrix*. Often, the best approach is to ensure even representation of different cases, then to interpret the network's decisions accordingly.

6. One day international cricket

One-day cricket is a version of the sport of cricket that is completed in one day, as distinct from Test cricket and first-class cricket which can take up to five days to complete.

6.1. One Day International Cricket Rules

In a one-day cricket match, each team bats only once, and each innings is limited to a set number of overs, usually fifty in a One-day International and between forty and sixty in a List A domestic one-day match. Other changes to the game include additional restrictions on where fielders may be placed (preventing teams from placing every fielder on the edge of the field to prevent boundaries), a restriction on the number of overs that may be bowled by any one bowler and stricter rules on wide balls and short deliveries (to prevent teams from restricting scoring by bowling deliveries that batsmen have no chance to score from). In many games a white ball is used rather than the traditional red; the need to paint rather than stain the white ball gives it subtly different characteristics in flight as it wears.

6.2. Bowling restrictions

As mentioned above, in almost all competitive one-day games, a restriction is placed on the number of overs that may be bowled by any one bowler. This is to prevent a side playing two bowlers with extremely good stamina who can then bowl the entirety of their side's overs, thus skewing the composition of a side. The classical composition of a cricket team is five specialist batsmen, five specialist bowlers and a wicket-keeper: in order to maintain this, the usual limitation is set so that a side must include at least five bowlers. For example, the usual limit for twenty-over cricket is four overs per bowler, for forty-over cricket eight per bowler and for fifty-over cricket ten per bowler. There is at least one notable exception to this convention. Pro Cricket in the United States restricts bowlers to five overs each, thus leaving a side requiring only four bowlers.

6.3. One Day International Cricket Tournaments

One-day International matches are usually played in brightly coloured clothing (leading some to give it the unflattering nickname pyjama cricket), and often in a "day-night" format where the first innings of the day occurs in the afternoon and the second occurs under Stadium lights. One-day international tournaments occur in various forms:

6.3.1 The World Cup

Involves all Test nations and qualifying associate nations

Consists of a round-robin group stage, a Super Six stage, semifinals, and finals.

Held once in four years

International Cricket Council determines venue

6.3.2 International Cricket Council Champions Trophy

- Involves all Test nations and qualifying associate nations
- Consists of knockout games (if a team loses a single game, it is "knocked out" of the tournament)
- Held once in four years between World Cups
- International Cricket Council determines venue

6.3.3 One-day International Series

- Involves two nations
- Consists of three to seven games, all matches are played even if series result is determined
- Played when one nation "tours" another
- Usually played in one of the two participating nations

6.3.4 Triangular Tournament

- Involves three nations
- Consists of a round-robin group stage, each team playing the other two or three times, and a final
- Played in one of the three participating nations or in neutral venue.

7. Training and testing of neural networks for cricket tournament forecasting

Our approach is to train the networks with live data ([8], [9]) of one day triangular cricket tournaments of the past 10 years. This has been done because the team composition or the squads have been consistent over the past 10 years. The simulation has been done in MATLAB [10].

7.1. Input variables used

- After careful analysis, the following input variables were identified:
- Number of matches played
- Number of matches won
- Number of matches lost
- Number of times finals reached in a tournament
- Number of times semifinals reached in a tournament
- Number of times quarters reached in a tournament

- Recent ranking
- Number of tournaments won
- Number of tournaments played
- Number of tournaments lost
- Number of times home ground advantage

For training, the teams were considered in combinations of three.

7.2. Architecture of ANN

The training was accomplished by 4 feed forward neural networks each with three neurons in a single hidden layer (see Figure 3) as this is known to be a universal approach element [11]

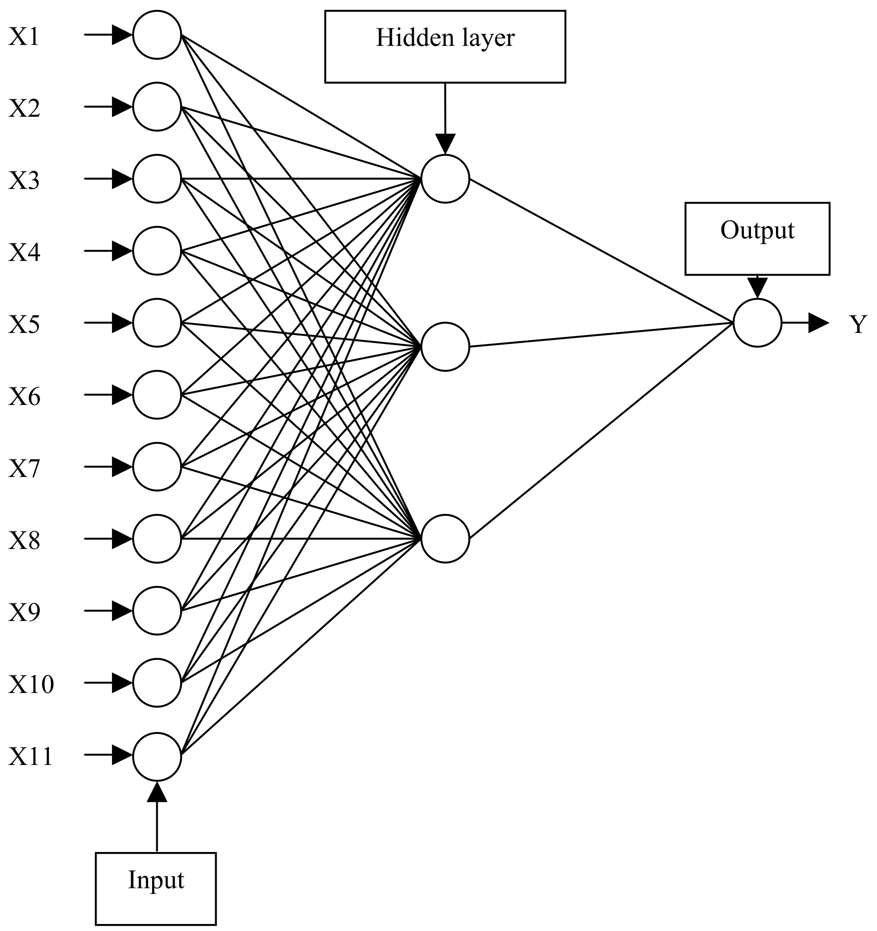


Fig 3: One of the neural networks used

The four networks used are:

- Best Team Network
- Worst Team Network
- Best Net Run Rate* Network
- Worst Net Run Rate Network

These parameters are used because in a tournament the performance of a team can be judged on the basis of its standing in the tournament - this represents its performance in the final match of the tournament which decides the winner of the tournament. The NRR maintained in the tournament represents the overall performance of the team in the tournament. Individual scores in matches can determine the outcome of that match alone but not the whole tournament.

*The net run rate in a single game is the run rate per over that a team scores, minus the run rate per over that is scored against them. Hence for a single match the winning team will have a positive NRR and the losing team will have a negative NRR. However, net run rate is only useful when aggregated over two or more matches and in this scenario it can be thought of as a measure of the aggregate or average of a team's performances over several matches.

7.3. Testing Results

We analyzed 60 major tournaments played by the various teams for the past 10 years. Records of 40 of the tournaments have been used for training the neural network and the remaining for testing. For testing any combination, the input data was simulated with each network and the resulting scores of all the four networks were added. A GUI accomplishes this work. The scores lie in the range 0 to 5. The team with the highest score will have rank 1 in the tournament, the team with the next best score will have rank 2 and the team with the lowest score will have rank 3 (See Figures 4 and 5)

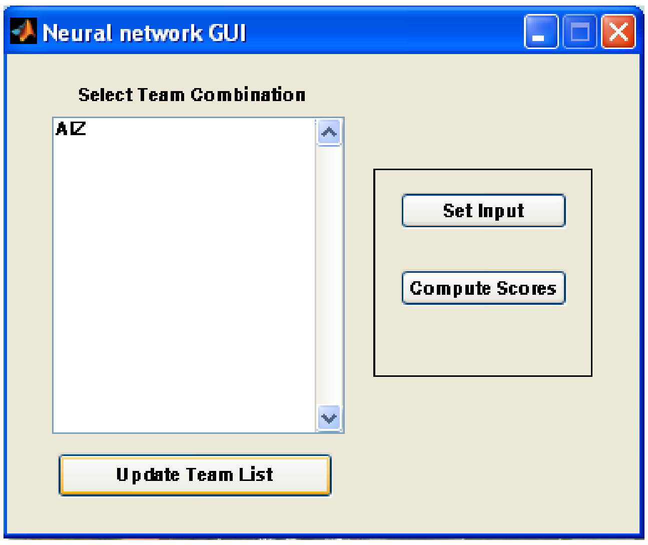
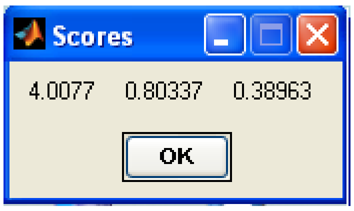


Fig. 4: GUI for using ANNs to predict the outcome of cricket tournaments



4.0077 0.80337 0.38963

Fig. 5: Output scores for the tournament

Out of the 20 tournaments which we used for testing, the overall efficiency of prediction comes to about 84.6%. The efficiency is likely to go up if only the tournaments for the past 5 years are used for analysis. However, due to lack of sufficient training as few tournaments are played every year, the data for the past 10 years has been taken.

8. References:

[1] www.calsci.com/Applications

[2] António José Silva, Aldo Manuel Costa, Paulo Moura Oliveira, Victor Machado Reis, José Saavedra, Jurgen Perl, Abel Rouboa and Daniel Almeida Marinho, "*The use of neural network technology to model swimming performance*", Journal of Sports Science and Medicine, March 2007 - Volume 6, Issue 1

[3] Michael Bailey, Stephen R. Clarke, *"Predicting the match outcome in one day international cricket matches, while the game is in progress"*, Journal of Sports Science and Medicine, July 2006- Volume 5, Issue 4

[4] Perl J., "*Artificial Neural Networks in Sports: New Concepts and Approaches*", International Journal of Performance Analysis in Sport, Volume 1, Number 1

[5] Perl J., Weber K., *"A Neural Network approach to pattern learning in sport"*, International Journal of Computer Science in Sport, Volume 3, Edition 1

[6] Artificial Neural Networks by B. Yegnanarayana, Prentice Hall of India

[7] An Introduction to Artificial Neural Networks, The University of Amsterdam Press

[8] www.icc-cricket.com

[9] www.cricinfo.com

[10] www.mathworks.com

[11] K. Hornik, M. Stinchcombe, H. White, *"Multilayer feedforward networks are universal approximators"*, ACM Neural Networks 1989 - Volume 2 Issue 5