# Artificial Neural Networks for Solving Double Dummy Bridge Problems

Krzysztof Mossakowski and Jacek Mańdziuk

Faculty of Mathematics and Information Science, Warsaw University of Technology,
Plac Politechniki 1, 00-661 Warsaw, POLAND
{mossakow, mandziuk}@mini.pw.edu.pl

**Abstract.** This paper describes the results of applying artificial neural networks to the double dummy bridge problem. Several feedforward neural networks were trained using resilient backpropagation algorithm to estimate the number of tricks to take by players $NS$ in fully revealed contract bridge deals. Training deals were the only data presented to the networks. The best networks were able to perfectly point the number of tricks in more than one third of deals and gained about 80% accuracy when one trick error was permitted. Only in less than 5% of deals the error exceeded 2 tricks.

## 1 Introduction

This paper presents the first step on the way to construct a computer program playing the game of contract bridge, using artificial neural networks. The first step relies on a verification of neural networks' ability to estimate the number of tricks that can be taken by one pair of players in a deal, in assumption of optimal play of all players. The problem is simplified by revealing all hands - the so-called *double dummy problem*.

The next step in a construction of a contract bridge program, will be an estimation of a number of tricks during and after bidding. In this step only partial information about other players' hands will be available.

The most important, and also the most interesting issue in contract bridge, is a play. A game strategy, good enough to win, or to have the biggest chance to win, regardless of opponents play, should be developed. The play would be the third part of a construction of a computer program playing contract bridge.

There are many computer programs playing the game of a contract bridge, unfortunately most of them are much worse than human players. To the best of our knowledge there are only two of them that can win against strong amateur-level players: Bridge Baron [1] and GIB [2]. GIB, Ginsberg's Intelligent Bridge-player, also succeeded in playing against professionals. We are not aware of any program developed based on neural networks playing at the "decent" level.

## 2 The Data

The data used in solving double dummy bridge problems was taken from GIB Library [3], which includes 717,102 deals with revealed all hands. Additionally the library provides a number of tricks taken by the pair $NS$ for each deal under the assumption of a perfect play of both parties. In all experiments the attention was fixed on a number of tricks taken by a pair $NS$ with $W$ player defender's lead for notrump play.

The set of deals was divided into three groups. The first $500,000$ deals were assigned to training. Deals numbered from $500,001$ to $600,000$ were assigned to validation, and the rest of deals to testing. Usually the training set contained $10,000$ deals, however for some number of networks $100,000$ deals were used.

## 3 Neural Networks

In all experiments feed-forward networks created, trained and tested using JNNS (Java Neural Network Simulator) [4] were used. In most cases logistic (unipolar sigmoid) activation function was used for all neurons except for the case of representation of data using negative numbers, where the hyperbolic tangent (bipolar sigmoid) activation function was applied.

The number of input neurons was specified by the chosen method of deal's representation. The numbers of hidden layers and neurons varied. In most of the experiments the output layer was composed of a single neuron representing the estimated number of tricks taken by a pair $NS$. All networks were trained using Rprop algorithm[5], with the following choice of method's parameters: initial and maximum values of an update-value factor were equal to 0.1 and 50.0, resp., and weight decay parameter was equal to $1E-4$.

## 4 Experiment Description and Results

In this section results of several experiments with various ways of coding a deal (hands' cards) and the ways of coding an estimated number of tricks to be taken are described. All results are presented in the form: $[(A \mid B \mid C), (D \mid E \mid F)]$. The first three numbers represent resp. the fractions *in percent* of training deals for which the network was mistaken by no more than 2 tricks ($A\%$), no more than 1 trick ($B\%$) and was perfectly right ($C\%$). The other three numbers indicate analogous results for testing data.

### 4.1 Coding a Deal

Two approaches to coding a deal as a set of real numbers suitable for neural network input representation were applied.

In the first approach each card of each hand was represented by two real numbers: the value (2, 3,. . ., *King*, *Ace*) and the suit (*Spades*, *Hearts*, *Diamonds*, *Clubs*). Both real numbers were calculated using a uniform linear transformation to the range $[0.1, 0.9]$ (ranges $[0, 1]$ and $[0.2, 0.8]$ were also tested, but no significant difference was noticed). Representation of each hand required 26 values, so the total number of input values was 104 (26x4). The simplest neural network,

**Table 1.** Results obtained for various coding schemes and network architectures. Column denoted by $D$ represents the number of training deals.

| Network type | Results (in %) | $D$ |
|---|---|---|
| (26x4)-(13x4)-1 | [(94.77 \| 77.45 \| 31.91), (94.77 \| 77.50 \| 32.05)] | 100,000 |
| (26x4)-(13x4)-(7x4)-13-1 | [(96.02 \| 80.14 \| 33.57), (93.87 \| 75.70 \| 31.04)] | 10,000 |
| 52-1 | [(94.17 \| 76.22 \| 31.06), (94.15 \| 76.15 \| 31.29)] | 100,000 |
| 52-25-1 | [(96.27 \| 81.02 \| 34.60), (95.81 \| 79.95 \| 34.02)] | 100,000 |
| 104-1 | [(94.81 \| 77.62 \| 32.19), (94.76 \| 77.52 \| 32.19)] | 100,000 |
| 104-30-4-1 | [(97.04 \| 82.84 \| 36.25), (95.64 \| 79.63 \| 33.74)] | 100,000 |
| 104-14 | [(84.76 \| 63.75 \| 26.72), (84.72 \| 63.28 \| 24.79)] | 100,000 |
| 104-33-14 | [(93.59 \| 76.96 \| 35.61), (93.58 \| 76.23 \| 31.82)] | 100,000 |

without hidden neurons - (26x4)-1, accomplished result [(75.86 | 51.63 | 18.37), (75.90 | 51.78 | 18.56)].

More complex network - (26x4)-(13x4)-(7x4)-13-1, presented in Fig. 1(a), yielded much better results: [(96.02 | 80.14 | 33.57), (93.87 | 75.70 | 31.04)]. Comparably good results were also accomplished by simpler network without the two last hidden layers, i.e. (26x4)-(13x4)-1: [(94.77 | 77.45 | 31.91), (94.77 | 77.50 | 32.05)].

In the other way of coding a deal each card had its own input neuron assigned, which input value denoted the hand containing this card. The hands were coded as follows: $N : 1.0$, $S : 0.8$, $W : -1.0$, $E : -0.8$. The following order of cards was used: $2S$ (two *Spades*), $3S$, $4S$, ..., $KS$ (king of *Spades*), $AS$ and so on for *Hearts*, *Diamonds* and *Clubs* (other orders were also tested, but no significant difference in results was observed). The simplest neural network without hidden neurons, i.e. 52-1 accomplished the result [(94.17 | 76.22 | 31.06), (94.15 | 76.15 | 31.29)]. Adding one hidden layer: 52-25-1 improved the result to [(96.27 | 81.02 | 34.60), (95.81 | 79.95 | 34.02)].

A slight modification of the above way of coding a deal was done by extending the input representation to 104 neurons. The first 52 input neurons represented assignment to a pair (value 1.0 for $NS$ pair and $-1.0$ for $WE$), and the other 52 ones represented a hand in a pair (1.0 for $N$ or $W$ and $-1.0$ for $S$ or $E$). The simplest network: 104-1 accomplished the result [(94.81 | 77.62 | 32.19), (94.76 | 77.52 | 32.19)], and two-hidden layer network $104 - 30 - 4 - 1$, presented in Fig. 1(b), yielded the result [(97.04 | 82.84 | 36.25), (95.64 | 79.63 | 33.74)].

### 4.2 The Way of Coding the Number of Tricks

The first approach to coding the number of tricks was the use of a linear transformation from integer values: $0, \ldots, 13$ to the range of real values $[0.1, 0.9]$.

Alternative way of coding consisted in providing one output neuron per each possible number of tricks, i.e. a total of 14 output neurons. Here the idea was to treat the prediction problem as the type of pattern classification/clustering problem with only one output being active. During training value 1.0 was used to indicate the correct output class (number of tricks), and 0.0 (or $-1.0$) to

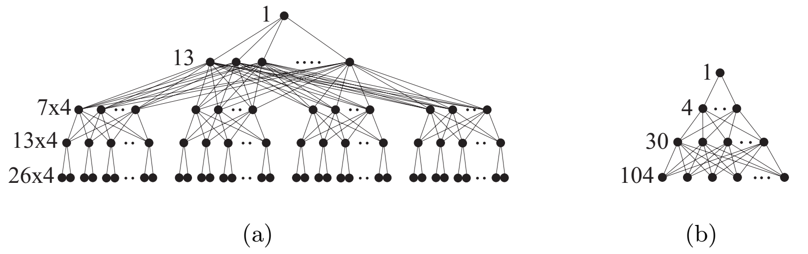

**Fig. 1.** Sample neural network architectures.

represent the other (incorrect) output classes. The simplest network: 104-14 accomplished the result [(84.76 | 63.75 | 26.72), (84.72 | 63.28 | 24.79)]. Adding one hidden layer: 104-33-14 improved the results significantly to [(92.70 | 74.21 | 34.85), (92.54 | 73.74 | 29.78)] and [(93.59 | 76.96 | 35.61), (93.58 | 76.23 | 31.82)], resp. for the case of using −1.0 and 0.0 to represent incorrect outputs in the training phase.

## 5 Analysis of Results

The number of iterations required to learn the training set without overfitting depended mostly on the way of coding a deal. The networks with coding (26x4) needed a few tens of thousands iterations, and networks with coding by cards assignment (52 or 104) only several hundred ones.

Results obtained for various input codings and neural architectures described in section 4 are summarized in Table 1. The best results were accomplished by networks with coding a deal by card assignment, but results achieved for the other coding are only slightly worse. The conclusion drawn from the table could be, that achieving the results at the level of (96 | 80 | 34) on the testing set seems to be a difficult task. At first glance, the result of 34% of the faultless prediction may seem discouraging. However it must be emphasized, that neural networks were trained using information about deals only. Actually, no information about the rules of the play was explicitly provided. In particular, no information about some nuances of the play, e.g. finesses or squeezes was coded in the input. Only cards in hands and numbers of tricks to be taken were presented in the training data.

### 5.1 Simple Estimator of the Number of Tricks

To have a point of reference very simple, naive estimator of the number of tricks was proposed. It was based on Work point count for the pair $NS$ (ace - 4 points, king - 3, queen - 2, jack - 1; total number of points in a deal - 40). The number of tricks to take by $NS$ was estimated by $(13/40) * points\_of\_NS$. This estimator for the testing set of 100,000 deals achieved the result of (86.19 | 61.32 | 22.52).

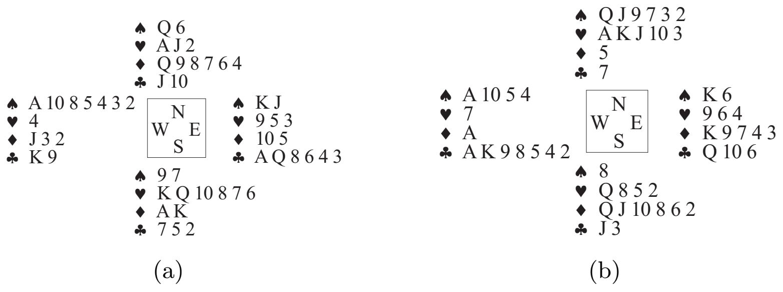

**Fig. 2.** Sample deals.

## 5.2 Hand Patterns

Work point count is widely used to estimate the power of a hand, but one of its most important drawbacks is omitting hand patterns. Short and long suits are very important during the play. Table 2 presents medium values of standard deviation values for hand suits' lengths. A general conclusion, which can be drawn from this statistics is that true values of very short and very long suits are quite difficult to be estimated by neural networks used in the experiments.

On the other hand, comparing the results of naive estimator (see Section 5.1) with the ones presented in Table 1 indicates that the networks definitely go beyond the naive Work point - based estimation of the hand's strength.

## 5.3 Sample Deals

In this section two examples of sample deals from [3] are presented - see Fig. 2.

The first deal (Fig. 2(a)) is the deal for which the number of tricks taken by $NS$ in notrump play can be either 12 (when $N$ or $S$ makes defender's lead) or 0 (when the opponent player makes the defender's lead). The network's output was 6. Since in all data presented in the experiments $W$ always made a defender's lead the expected result was 0. Therefore the network highly overestimated hands $NS$. Certainly the naive estimator also made a big mistake, it claimed 7 tricks for $NS$. This example shows that in solving even a double dummy problem there may be several nuances that should be carefully taken into account, e.g. defining the hand that makes defender's lead.

The second sample deal (Fig. 2(b)) shows that long suits not always caused problems for trained networks. The network predicted 2 tricks for $NS$ when $W$ makes a defender's lead, which is a correct value. Naive estimator claimed 6 tricks for $NS$.

**Table 2.** Mean values of standard deviations for hand suits' lengths (columns $M$) in the training set for (26x4)-(13x4)-(7x4)-13-1 network composed of 10,000 deals. Columns $D$ represent the difference between network's output and the target number of tricks.

| $D$ | Number of deals | $M$ | $D$ | Number of deals | $M$ |
|---|---|---|---|---|---|
| $-4$ | 23 | 1,803186 | 1 | 2303 | 1,379000 |
| $-3$ | 150 | 1,577670 | 2 | 754 | 1,467983 |
| $-2$ | 834 | 1,424109 | 3 | 178 | 1,611750 |
| $-1$ | 2354 | 1,353548 | 4 | 40 | 1,724390 |
| 0 | 3357 | 1,318973 | 5 | 6 | 1,607476 |
| | | | 6 | 1 | 1,879716 |

## 6 Conclusions

In this piece of research the underlying assumption is to reduce human's influence on neural networks to the minimum. This is the reason of avoiding special preprocessing of deals. It seems to be reasonable to present to networks information about short suits (e.g. humans add 2 points for a void and 1 point for a singleton) but authors decided to let the networks discover this information by themselves if it is really important. The difference between results of neural networks and naive estimation of hands' strengths shows, that neural networks were able to notice hand patterns as important information. However statistics presented in Table 2 show that there is still some room for improvement in this area.

Based on the assumption that only raw data is presented to the network the result of (95.81 | 79.95 | 34.02) on the test set achieved in the experiments looks promising. Only in less than 5% of deals the network was wrong by more than 2 tricks.

Several other experiments with double dummy problem are planned. First of all some other neural network architectures would be tested, e.g. Self-Organizing Maps. There is also a need to test plays other than notrump.

In the next step of research only one hand will be fully presented along with some incomplete information about other hands. This part of research will simulate bidding. The last stage will be the play phase.

## References

1. Smith, S., Nau, D., Throop, T.: Computer bridge: A big win for ai planning. AI Magazine **19** (1998) 93–105
2. Ginsberg, M.: Gib: Imperfect information in a computationally challenging game. Journal of Artificial Intelligence Research **14** (2001) 303–358
3. Ginsberg, M. (http://www.cirl.uoregon.edu/ginsberg/gibresearch.html)
4. (http://www-ra.informatik.uni-tuebingen.de/software/JavaNNS/welcome_e.html)
5. Riedmiller, M., Braun, H.: A fast adaptive learning algorithm. Technical report, University Karslruhe, Germany (1992)