Solving the Double Dummy Bridge Problem with Shallow Autoencoders

Jacek Mańdziuk(✉) and Jakub Suchan

Faculty of Mathematics and Information Science, Warsaw University of Technology, Koszykowa 75, 00-662 Warsaw, Poland
j.mandziuk@mini.pw.edu.pl, kubasuchan@hotmail.com

Abstract. This paper presents a new approach to solving the Double Dummy Bridge Problem (DDBP). The DDBP is a hard classification task utilized by bridge playing programs which rely on Monte Carlo simulations. The proposed method employs shallow autoencoders (AEs) during an unsupervised pretraining phase and Multilayer Perceptron networks (MLPs) with three hidden layers, built on top of these trained AEs, in the final fine-tuning training. The results are compared with our previous study in which MLPs with similar architectures, but with no use of AEs and pretraining, were employed to solve this task. Several conclusions concerning efficient weight topologies and fine-tuning schemes of the proposed model, as well as interesting weight patterns discovered in the trained networks are presented and explained.

Keywords: Autoencoder · Double Dummy Bridge Problem
Classification

1 Introduction

Game AI is a popular and fast growing field of Artificial Intelligence (AI) research which concerns various aspects of machine game playing. While historically this research domain was mainly focused on perfect-information games such as chess or checkers, in recent years games in which only partial game-related information is available to each player (i.e. certain significant game aspects are concealed from them) gained momentum. Imperfect-information games include, besides others, most of card games - with the game of bridge being one of the most popular examples.

Rules of Bridge. Bridge is a popular trick-taking card game played by four contestants (referred to as *North*, *East*, *South* and *West* or N, E, S, W, for short) in teams of two (NS vs EW), using a standard 52 card deck. The teams serve as adversaries during the whole game and firstly participate in a *bidding phase*.

Afterwards the team which has proposed a higher contract is bound to attain their prior declaration. Their competitors are expected to spare no efforts in

L. Cheng et al. (Eds.): ICONIP 2018, LNCS 11304, pp. 268–280, 2018.
https://doi.org/10.1007/978-3-030-04212-7_23

order to hinder their success. This constitutes a *play phase*. In this phase, the teammate of the player who won the bidding puts their cards face-up down on the table and ceases henceforth to actively participate in the game. The player left to the bidding winner starts the play phase (makes the initial lead). The participants try to match the suit of the card that was played in the current trick. After everyone has done so the player with the highest-ranked card - including trumps - claims ownership of the trick, furthermore he becomes the one to play first in the next trick.

The goal of the game is to take as many tricks as possible, and certainly the highest number of tricks x that can be collected by a pair equals 13, in which case the opponent pair scores $13 - x$ tricks. Please consult [8] for a detailed explanation of the game rules.

Double Dummy Bridge Problem. Various attempts were made in AI literature to mimic the strategy used by humans during the bidding phase, yet typically a computer program remains inferior to its human counterpart [2,19] in that matter. In Machine Learning (ML) framework, the problem of bidding the appropriate contract can be transformed into a specific learning problem. Please observe that, if, disregarding the rules of Bridge, we assume that all information is available to the players (i.e. the game is a perfect-information one), the problem of bidding the contract becomes deterministic under the assumption that all players follow the optimal strategy in the play phase. In other way, assuming that all information is available to the players the number of tricks to be taken by each of the two playing pairs can be unambiguously assessed for a specified game (with the above-mentioned perfect rationality of the players). This situation is an illustration of the Double Dummy Bridge Problem (DDBP). More precisely, the DDBP consists in answering the question about the number of tricks that will be taken in a given game by the pair NS. The DDBP has been deemed a real challenge by many AI researches, not only because of its complexity, but also due to its high sensitivity to even minute changes in the distribution of cards among the players. For instance, exchanging positions of just two cards in a deal may significantly affect the expected results (i.e. the DDBP solution).

The practical value of fast DDBP solvers comes into play in simulation-based bridge programs in which the outcome of the game is assessed based on massive simulations of possible game scenarios, each of which requires solving a certain DDBP instance related to assumed distribution of cards. Such an approach has been utilized in *partition search* algorithm [3,9] or *cost-sensitive classifiers* and *upper-confidence-bound* algorithms [11].

Related Work. In the literature, there have been several attempts to solve the DDBP using example based learning. In particular, previous approach consisted in applying various Multilayer Perceptron (MLP) architectures with several coding schemes in the input layer and a Resilient Backpropagation (RProp) learning algorithm [14–16]. The best one among the tested MLP architectures $(208 - 52 - 13 - 1)$ accomplished an accuracy of 53.11% for the so-called *suit contracts* and 37.80% for the *no trump contracts*. In the case of suit contracts

these results appeared to be superior to those accomplished by the human bridge grandmasters solving exactly the same sets of DDBP instances [13].

Our approach was researched further by others [5], this time with the focus on optimization of the training method - confirming the superiority of the RProp algorithm. Another related work [6] compared the impact of various activation functions on the output error. Yet other neural network approaches to solve the DDBP were proposed in [7] and [17], where respectively the Elman network and the Cascade Correlation network are employed.

Motivation and Research Goals. Encouraged by the promising results, in particular for suit contracts, in this paper we revisit the DDBP, but this time with shallow autoencoders (AEs) as the neural network architecture. In order to make the comparison fair the same best-performing input coding from our previous experiment is used in the current approach. Also the architecture is similar, i.e. shallow AEs with one hidden (intermediate) layer are utilized. This way, we attempt to make a direct comparison of the efficacy of AE training with an unsupervised pretraining phase and the MLP architecture trained in a supervised manner.

In summary, the main contribution of this work is threefold:

- verification of the suitability of the AE-based approach to a very sensitive classification problem, such as the DDBP;
- comparison of the AE's efficacy in solving the DDBP with the classification outcomes obtained previously using the MLP [16];
- shedding light on the intrinsic differences between supervised MLP training and AE training in the considered classification task.

The remainder of the paper is structured as follows. The next section introduces the proposed AE-based approach with a detailed presentation of the coding scheme and several variants of AE network architectures tested in the experimental evaluation of the method. Section 3 presents the experimental setup and analysis of results. Conclusions and directions for further research are summarized in the last section.

2 Autoencoder-Based Architectures

This section describes the AE architectures used in our attempts to solve the DDBP. On a general note, training of an AE consists of two phases. In the first one, the goal is to reconstruct the given input, i.e. to learn the identity function on the training set of examples. Yet, due to the hidden layers being less abundant in neurons this process leads to feature extraction. With every layer being smaller the feature extraction becomes gradually more extensive. In the process of extraction the data is compressed with every consecutive layer, however at a certain depth the data compression may become increasingly lossy and hamper the overall model performance in the recognition phase. *This was actually the case in our experiments, as our initial approaches involving deep*

autoencoders resulted in highly increasing training error in subsequent AE layers. For this reason we decided to restrict the AE architecture to one hidden layer, which appeared to be most effective in the preliminary experiments.

After the above-described pretraining phase (consisting in unsupervised feature extraction) the *encoding AE layers* form the final classification network with an additional output layer and are trained in a supervised manner in the same way as MLP networks. The *decoding AE layers* used for input reconstruction have no application for solving this problem and are therefore discarded in the final supervised training phase [18].

2.1 A Deal Representation in the Input Layer

In the input AE layer the deal representation (cards distribution) is fed to the network in the form of 208 binary inputs. This input was divided into 4 sections, each representing the cards of one player, provided in a fixed order, i.e. first W (52 inputs) then N (52 inputs), followed by E and S. Each such section corresponds to the whole deck of cards aligned in the following order: $A\spadesuit, \ldots, 2\spadesuit$, $A\heartsuit, \ldots, 2\heartsuit$, $A\diamondsuit, \ldots, 2\diamondsuit$, $A\clubsuit, \ldots, 2\clubsuit$. Consequently, each card in a deal is represented by 4 inputs (one per player): the input representing the player who actually possesses the card is equal to 1 and the other 3 input neurons associated with that card, representing the three remaining players, are assigned input values equal to 0. This way the player who possesses a given card in a deal is pointed out. In other words, for each player there are exactly 13 inputs equal to 1 - on positions corresponding to the cards he/she possesses, and 39 inputs equal to 0 - on the remaining positions. The above coding proved to be highly efficient in our previous experiments [16] and in subsequent works by other authors [6]. Please consult [16] for a comprehensive discussion on the advantages of this deal coding over the alternative, more concise representations.

In order to encode the remaining deal-related information, i.e. the leading hand (the player who makes the opening lead) and the trump suit we initially extended the input representation by 9 neurons, first 5 of which indicated the game type (No trump, Spades, Hearts, Diamonds, Clubs) and the remaining 4 denoted the player making the opening lead. However, in a set of initial experiments this method proved inefficient as the network could not fully conceive the significance of these dedicated inputs. For this reason, in the final experimental setup we followed the idea presented in [12,16] in which by default the fourth player (S) makes the lead. If the play should start with a lead from N the cards of N and S players are swapped as well as the cards of W and E. As a consequence, the leading player (the fourth one in the input representation) becomes N. As for the representation of the game type, separate networks are trained for trump and no trump (NT) games. In the latter case it is always assumed that the trump color is the first one in each player hand's representation, i.e. Spades. In order to make the another one (say Diamonds) the trump suit, the cards representing Diamonds and Spades are swapped in each hand (please notice that names of suits are just labels and can be exchanged with no consequences). So effectively, the first suit remains the trump one.

2.2 Hidden (Feature) Layers

Hidden layers of the AE network serve as feature extractors that focus on the most relevant aspects of the processed input data and pass it forward for further processing and compression. The size of each subsequent hidden layer is reduced compared to the preceding one, which leads to building gradually more general, high-level feature-based representations in subsequent layers. At the same time, this compression process is prone to certain degree of information loss which, to a large extent, depends on the *compression rate* (CR) between the subsequent layers.

Compression Rate. In order to find efficient AE architectures a bunch of preliminary experiments were conducted, aimed at finding the most suitable CR between layers. First of all, it turned out that for the DDBP **the information loss is high already when the second hidden layer is added**. For this reason we decided to use shallow AE architectures with one hidden layer. Furthermore, **in order to make a direct comparison** with our previous MLP-based approach [16] two more "standard" hidden layers of sizes 52 and 13, respectively were added in the final architectures, which were not trained in AE manner.

In order to find the efficient CR value we applied a grid search procedure within the interval $CR \in [1.1, 3.0]$. As could be expected, the most efficient CRs were found in the middle area of the tested ranges. Higher values of CR resulted in too strong compression and information loss, while low CR values did not offer a relevant advantage compared to the baseline input representation. Based on the outcomes we decided to use two CRs in the final experiments: $CR = 4/3$ and $CR = 2$, which led to 156 and 104 neurons in the first hidden layer (AE compression layer), respectively.

2.3 Output Layer

The above described AE architecture, i.e. $208 - 156/104 - 208$ was used in the pretraining phase in order to build a meaningful feature-based representation in the AE compression layer. Once this unsupervised training process was completed the decoding layer was discarded and replaced by two standard hidden layers (with 52 and 13 units, resp.) and an output layer leading to the **final architecture used for the classification task**: $208 - 156/104 - 52 - 13 - 14$

Initially, two cases were considered for the output layer. Firstly, a classifying *softmax* layer composed of 14 output neurons - one per class (possible DDBP outcomes are integers from $[0, 13]$). Secondly, a layer consisting of one *sigmoid* neuron whose output corresponds to the number of tricks scaled into the range $[0, 1]$. In the latter case the $[0, 1]$ range was divided into 14 segments of pairwise equal lengths and the training (goal) signal for each class was set in the middle of the respective subinterval. In the preliminary tests both approaches yielded comparable results, so we arbitrarily decided to stick to the first option, i.e. one output neuron per class.

2.4 Topology of Connections

One of the main research goals of this paper is to experimentally compare the MLP-based DDBP solution presented in our earlier works and an application of AE architectures solving the same task. In order to make this comparison straightforward we used the same input and output representations and similar network architecture in terms of the number of layers as well as their sizes. In our previous experiments [16] the topologies of the most effective architectures were the ones where the connections between the input and the first hidden layer were restricted to individual hand (player's cards) representations. More precisely each 52-neuron representation of a given hand in the input layer was fully connected with 1/4 of the 1hl neurons (26 or 39 depending on the particular setup), without any connections to other 1hl units. This way the 1hl served as a compression layer where the initial 52-neuron hand representation was transformed to a 26 or 39 unit one, respectively.

For each of the fully connected AE architecture selected for the final experiments a corresponding model with the above described connection topology.

2.5 Network Architectures

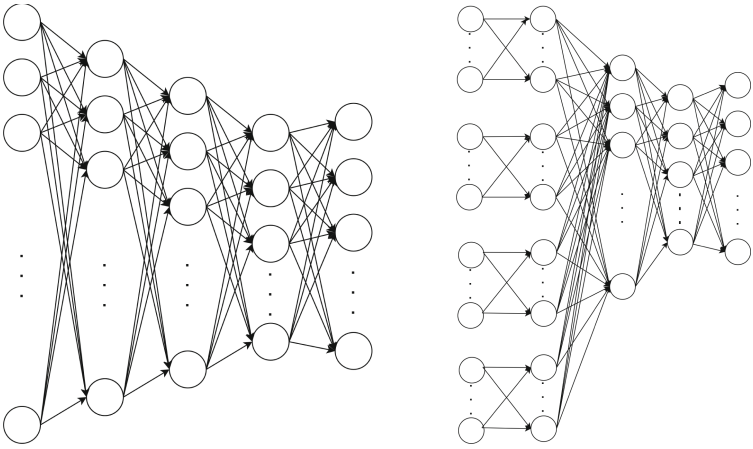
In summary of the above description the following network structures were devised as the base models for experimental evaluation:

- Input layer composed of 208 units coding a deal as described in Sect. 2.1. Output layer composed of 14 sigmoid neuron denoting the number of tricks to be taken by the NS pair (cf. Sect. 2.3).
- First hidden layer composed of either 104 or 156 units.
- Topology between input and 1hl being either full connection (F) or dedicated connection (D) pattern (see Sect. 2.4).
- 2hl and 3hl composed of 52 and 13 neurons, respectively.

In all tested models sigmoid neurons are utilized. All together 4 AE models were tested in experiments varying by size of the 1hl (2 options) and topology of connections (2 options). Fully connected AE models are presented in Fig. 1a and those with dedicated connections between input and 1hl in Fig. 1b.

3 Experimental Setup and Results

One of the main observations reported in our previous work was the very distinct quality of the results for suit (trump) contracts and NT ones. More interestingly this observation is true not only with respect to our tests with the MLPs [16], but also among professional human players solving DDBP instances [13]. This property, whose nature is still to be fully discovered, has also been confirmed in our preliminary tests. This is why separate AE networks (of the same architecture) were used for training and testing on suit and NT contracts, respectively. Hence, the number of experimental setups discussed in the previous section was



(a) AE network of type F (fully connected): 208-156/104-52-13-14.

(b) AE network of type D (with dedicated connections): $52x4$-39/$26x4$-52-13-14.

Fig. 1. Two types of considered network architectures. In each case, the size of the 1hl is equal to 104 or 156, depending on the experimental setup.

doubled to 8 different scenarios. The source code and the experimental results presented in this paper are available in our project's Github repository [1].

Training and Testing Data. The data is taken from the Ginsberg's GIB Library [10] in the form of a text file. Each sample represents one deal, i.e. cards assigned to each of the 4 hands/players. Each such deal is accompanied by 20 integers (from the set $\{0, 1, \ldots, 13\}$ denoting the number of tricks to be taken by the NS pair in 20 possible game configurations: 4 trump suits or NT game as well as 4 possible leading (opening) hands. This way, for a given card distribution among the players, any possible game contract can be considered.

The GIB Library consists of over 700 000 deals represented in the above-described way. Following our previous experiments with MLPs, for suit/trump games we randomly selected 100 000 deals for training and another 100 000 for testing. These 100 000 training deals were used in 8 game scenarios related to the choice of a trump suit and leading hand (N or S). In effect, the training process in suit AE models was performed on a set of 800 000 deals. In the case of NT models, 400 000 deals was randomly selected (and multiplied by two possible leading hands) for training and another 100 000 for testing.

Training phase 1 - Training of an Autoencoder. Training is performed in two phases. In the first one a shallow AE with one compression layer (of size 104 or 156 depending on the model) is trained using the data described above. Once the training is completed the hidden layer provides a compressed representation of the 208-dimensional input data.

Since the input and output have the same cardinality and the input vectors are binary and sparse, *crossentropy* is used as an error function:

$$H(x,z) = -\sum_{k=1}^{d}[x_k \log z_k + (1-x_k)\log(1-z_k)]. \quad (1)$$

where x_k denotes a desired output, z_k is the actual output and d is the size of a training sample. The Root Mean Square Propagation (RMSProp) algorithm with a learning rate $\eta_1 = 0.001$ is used in this phase. The value of η_1 was selected based on a set of preliminary experiments. Training is terminated when the change in the error function values falls below 0.01.

Training phase 2 - Final Training of a Classifier. In the second training phase, two new hidden layers and an output layer are added to the encoding part of the shallow AE described in the above point comprising the following architecture: $208 - 104/156 - 52 - 13 - 14$, which is trained again based on the whole set of examples in order to serve as a classifier for unknown DDBP deals.

This training phase is performed using the RMSProp algorithm, according to one of the two possible scenarios. Either the whole network is trained or the weights developed in the pretraining phase are frozen and only the remaining part of the network ($104/156 - 52 - 13 - 14$) undergoes weight modification. While the former is the standard approach, the motivation behind the latter is the following: since the AE pretraining error was close to zero the trained compression layer of the AE can actually serve as an alternative input representation and therefore the weights incoming to this layer need not be modified at this stage.

In either case training is performed with the learning rate $\eta_2 = 0.004$ for the trump deals and $\eta_2 = 0.002$ for the no trump ones. These learning rates were set based on preliminary experiments performed with various values of $\eta_2 \in [0.001, 0.1]$. Training is considered completed when the error function on the validation set increases 25 times. The state of the network corresponding to the overall highest performance, i.e. the lowest error value is considered as the training outcome.

3.1 Results

The results are presented for 16 experiment setups and divided into two tables. Table 1 provides the outcomes for suit contracts and Table 2 for no trump ones. In each case 8 network configurations which correspond to the above-described design decisions are tested.

Conclusions. Several interesting observations can be made based on presented results. **(1)**: they confirm a crucial role of the selected compression rate on the error value in the pretraining phase. On the other hand, quite surprisingly, this initial pretraining error does not have much influence on the final system error, after the fine-tuning phase. **(2)**: in suit contracts the results are close to those

Table 1. Best results for **trump (suit) deals** in 8 possible system configurations. The results in columns *F* and *D* denote the *full* and *dedicated* connection topology between the input and 1hl, resp. In either case there are two possible sizes of the 1hl (104 and 156 units). In each of them the weights between the input and 1hl may or may not be frozen during the fine tuning phase. *Compression error* and *fine tuning error* report the *cross-entropy* errors after completion of the training procedure in *phase 1* and *phase 2*, resp. The best reference results for the MLP presented in [16] are equal to 53.11%, 96.48%, 99.88% for the exact, one trick off and two tricks off, resp.

Model topology	*F*				*D*			
1hl size	104		156		104		156	
Weights partly frozen	No	Yes	No	Yes	No	Yes	No	Yes
Test set								
Suit (♠) - exact	43.66%	39.75%	47.93%	29.06%	51.28%	23.93%	48.97%	32.19%
Suit (♠) - off by 1	90.46%	87.33%	93.83%	72.75%	95.33%	63.50%	94.37%	77.76%
Suit (♠) - off by 2	99.07%	98.45%	99.63%	92.41%	99.72%	86.40%	99.69%	95.07%
Compression error	0.400	0.24	9.560	9.599	46.98	45,90	14.23	13.21
Fine tuning error	0.048	0.050	0.043	0.056	0.043	0.059	0.044	0.054

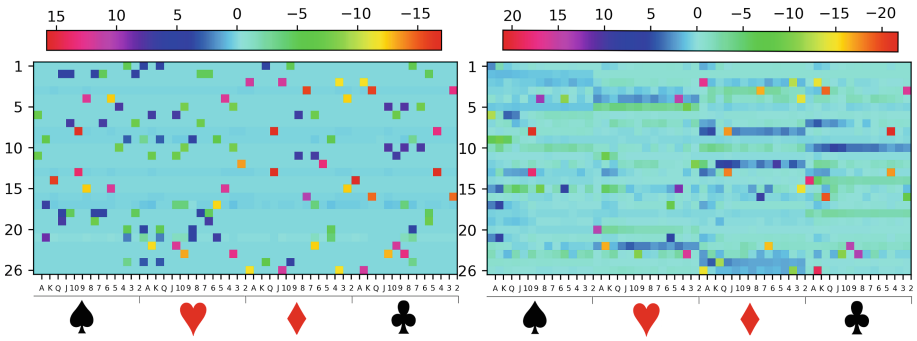
Table 2. Best results for **no trump deals** in 8 possible system configurations. See the caption of Table 1 for a detailed specification of the table's layout. The best reference results for the MLP presented in [16] are equal to 37.80%, 84.31%, 97.34% for the exact, one trick off and two tricks off, resp.

Model topology	*F*				*D*			
1hl size	104		156		104		156	
Weights partly frozen	No	Yes	No	Yes	No	Yes	No	Yes
Test set								
NT - exact	36.6%	35.13%	37.30%	28,40%	41.73%	22.46%	39.04%	29.48%
NT - off by 1	82.15%	80.12%	82.04%	70.47%	86.18%	59.73%	84.45%	72.51%
NT - off by 2	95.96%	95.29%	95.61%	90.17%	96.63%	82.90%	96.43%	91.66%
Compression error	0.293	0.325	9.580	9.597	45.13	44.56	13.99	14.49
Fine tuning error	0.052	0.053	0.051	0.057	0.049	0.060	0.051	0.056

obtained in our previous study [16] in the case of perfect accuracy (i.e. 51.28% vs 53.11%), and are on par in the two remaining cases (one/two tricks off). At the same time we managed to slightly surpass the aforementioned results in the case of NT contracts raising its value from 37.80% to 41.73% for perfect classification case and from 84.31% to 86.07% with a one-trick error margin. In both experiments the same deal representation and the same numbers of deals were used for training and testing. Also the resulting architectures were very close with the main difference being a two phase training procedure including an AE

pretraining phase in the current studies. **(3)**: our initial plans to employ deep autoencoders turned out to be ineffective as any attempt of further compression, extending beyond one hidden layer, resulted in significant raise of the cross-entropy error function. We believe that these three above observations prove that the DDBP is a hard classification task for both MLPs and AEs. **(4)**: we observed that networks with dedicated sets of weights between the input and the first hidden layer are more effective than their counterparts with traditional, fully connected topology of weights. **(5)**: despite close-to-zero loss value in the pretraining phase of 104 models the idea of freezing the weights between the input and 1hl in the second (fine-tuning) phase was not successful. This came to us as a bit of surprise since the 104 representation (due to the meaningless loss values) seemed to be a good candidate for an alternative deal representation. The reason of failure is subject of further investigation since at the moment we do not possess a convincing explanation. **(6)**: similarly to the results of our previous study [16], NT contracts appeared to be much more demanding than suit ones. The reason for that is attributed to different nature of both types of deals: in trump ones the advantage stemming from possession of trump cards can be easier translated to the number of taken tricks, while playing NT contracts is generally more demanding and requires subtle maneuvers - more frequent use of *a finesse* is a typical example.

Weight Patterns. Following our previous study with pure MLP networks (without pretrained autoencoders) [16] we explored the weight spaces of the trained networks in the quest for meaningful patterns, explainable by human bridge players. Figure 2 visualizes the weights of a randomly selected player in one of the randomly selected experiments ($D/104/trump$). Due to space limits we are not able to delve more deeply into this topic, but two general conclusions may be easily drawn. Firstly, after the pretraining phase there are some number of strong (positive or negative) weights, most of which *are not altered* in subsequent training (c.f. left and right subfigures). Secondly, after the final training one can easily spot quite many neurons in the 1hl which are *dedicated to particular suits*, i.e. are "focused" on 13 consecutive inputs representing a given suit. This specific attention is visible in the form of "stripes" in the right subfigure covering the weights from A to 2 of a given suit. Both of the above observations are in line with the MLP-related study [16]. Furthermore, the pertinence of suit lengths (which is the crux of the second observation) is critical in human assessment of hand strength in *trump* deals.



(a) Weights after pretraining phase **(b)** Weights after fine-tuning phase

Fig. 2. Visualization of weights representing one player's cards for D/104 network and trump suits data. Each point refers to the respective weight between one of the 52 inputs assigned to that player's cards and one of the dedicated 26 neurons in the 1hl. For instance the weight between $Q\spadesuit$ and 14th 1hl neuron is strongly positive (red color) while the one between $Q\clubsuit$ and the 4th 1 h neuron is strongly negative (orange color). (Color figure online)

4 Summary and Future Work

The main goal of this paper is verification of suitability of AEs for solving the DDBP - a hard classification task with sensitive input-output relation. To this end, several configurations of different network architectures and detailed differences in the training algorithm are proposed and experimentally evaluated. The detailed conclusions are presented and discussed in the previous section.

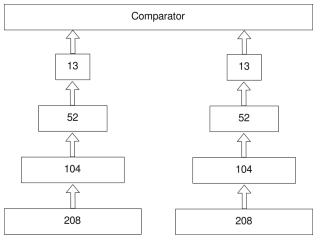


Fig. 3. Schematic presentation of a comparator-based approach to solving the DDBP planned as our next research step.

On a general note, the results of applying shallow AEs in the unsupervised pretraining phase and MLPs in the final fine-tuning phase turned out to be comparable to those of training MLPs directly in a supervised manner. Specific weight patterns observed in weight spaces of the trained MLPs are also visible in the current setup of the system, albeit their detection is not as evident.

Currently we plan to adopt an approach utilized in the DeepChess project [4]: instead of directly answering the question about the number of NS tricks, the system will be trained to compare two deals and answer an easier, qualitative question: *which of the two deals promises higher number of tricks for* NS*?* (see Fig. 3 for a possible implementation). With the ability of efficient qualitative prediction one could compare the unknown deal with a few reference deals with known numbers of NS tricks to answer the initial, quantitative question.

Acknowledgments. This work was supported by the Polish National Science Centre grant 2017/25/B/ST6/02061.

References

1. DDBP Github repository. https://github.com/holgus103/DDBP/
2. Amit, A., Markovitch, S.: Learning to bid in bridge. Mach. Learn. **63**(3), 287–327 (2006)
3. Beling, P.: Partition search revisited. IEEE Trans. Comput. Intell. AI Games **9**(1), 76–87 (2017)
4. David, O.E., Netanyahu, N.S., Wolf, L.: DeepChess: end-to-end deep neural network for automatic learning in chess. In: Villa, A.E.P., Masulli, P., Pons Rivero, A.J. (eds.) ICANN 2016. LNCS, vol. 9887, pp. 88–96. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44781-0_11
5. Dharmalingam, M., Amalraj, R.: Articifial neural network architecture for solving the double dummy bridge problem in contract bridge. Int. J. Adv. Res. Comput. Commun. Eng. **2**(12), 4683–4691 (2013)
6. Dharmalingam, M., Amalraj, R.: A solution to the double dummy contract bridge problem influenced by supervised learning module adapted by artificial neural network. ICTACT J. Soft Comput. **5**, 836–843 (2014)
7. Dharmalingam, M., Amalraj, R.: Supervised Elman neural network architecture for solving double dummy bridge problem in contract bridge. Int. J. Sci. Res. (IJSR) **3**(6), 2745–2750 (2014)
8. Francis, H., Truscott, A., Francis, D. (eds.): The Official Encyclopedia of Bridge, 5th edn. American Contract Bridge League Inc., Memphis (1994)
9. Ginsberg, M.L.: http://www.gibware.com
10. Ginsberg, M.L.: Library of double-dummy results. http://www.cirl.uoregon.edu/ginsberg/gibresearch.html
11. Ho, C.Y., Lin, H.T.: Contract bridge bidding by learning. In: AAAI Workshop: Computer Poker and Imperfect Information (2015)
12. Mańdziuk, J., Mossakowski, K.: Example-based estimation of hand's strength in the game of bridge with or without using explicit human knowledge. In: IEEE Symposium on Computational Intelligence in Data Mining, Honolulu, Hawaii, USA, pp. 413–420 (2007)
13. Mańdziuk, J., Mossakowski, K.: Neural networks compete with expert human players in solving the double dummy bridge problem. In: 2009 IEEE Symposium on Computational Intelligence and Games, pp. 117–124, September 2009
14. Mossakowski, K., Mańdziuk, J.: Artificial neural networks for solving double dummy bridge problems. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 915–921. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24844-6_142

15. Mossakowski, K., Mańdziuk, J.: Neural networks and the estimation of hands' strength in contract bridge. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Żurada, J.M. (eds.) ICAISC 2006. LNCS (LNAI), vol. 4029, pp. 1189–1198. Springer, Heidelberg (2006). https://doi.org/10.1007/11785231_124
16. Mossakowski, K., Mańdziuk, J.: Learning without human expertise: a case study of the double dummy bridge problem. IEEE Trans. Neural Netw. **20**(2), 278–299 (2009)
17. Muthusamy, D.: Double dummy bridge problem in contract bridge: an overview. Artif. Intell. Syst. Mach. Learn. **10**(1), 1–7 (2018)
18. Ng, A., Ngiam, J., Foo, C.Y., Mai, Y., Suen, C.: UFLDL tutorial. http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial
19. Yegnanarayana, B., Khemani, D., Sarkar, M.: Neural networks for contract bridge bidding. Sadhana **21**(3), 395–413 (1996)