Blackjack as a test bed for learning strategies in neural networks

Conference Paper · June 1998

DOI: 10.1109/IJCNN.1998.687170 · Source: IEEE Xplore

2 authors, including:

Andres Perez-Uribe
La Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud
85 PUBLICATIONS **1,263** CITATIONS

Proc. IEEE Intl. Joint Conf. on Neural Networks IJCNN'98, Anchorage, May 4-9,1998,Vol 3, pp.2022-2027.

Submitted to the IEEE International Joint Conference on Neural Networks IJCNN'98, Anchorage, May 4-9/98

Blackjack as a Test Bed for Learning Strategies in Neural Networks

Andrés Pérez-Uribe and Eduardo Sanchez
Logic Systems Laboratory, Computer Science Department
Swiss Federal Institute of Technology, CH–1015 Lausanne, Switzerland.
{Andres.Perez,Eduardo.Sanchez}@di.epfl.ch, http://lslwww.epfl.ch

Abstract— **Blackjack or twenty-one is a card game where the player attempts to beat the dealer, by obtaining a sum of card values that is equal to or less than 21 so that his total is higher than the dealer's. The probabilistic nature of the game makes it an interesting testbed problem for learning algorithms, though the problem of learning a good playing strategy is not obvious. Learning with a teacher systems are not very useful since the target outputs for a given stage of the game are not known. Instead, the learning system has to explore different actions and develop a certain strategy by selectively retaining the actions that maximize the player's performance. This paper explores the use of blackjack as a test bed for learning strategies in neural networks, and specifically with reinforcement learning techniques. Furthermore, performance comparisons with previous related approaches are also reported.**

Keywords— **Reinforcement learning, SARSA algorithm, Q-learning, Blackjack, learning strategies, artificial neural networks**

I. Introduction

Artificial neural networks are widely used for the design of adaptive, "intelligent" systems since they offer an attractive paradigm: enabling the system to "learn" to solve problems by instruction or experience. When using supervised learning algorithms, a "teacher" provides target outputs or an error vector that specifies how the learning system should modify its responses such that the error is minimized. Such learning methods have proven to be very powerful in solving certain tasks [1]. However, when a learning network is used in tasks where the environment provides a stream of input patterns, and the target outputs for each input pattern are not known, such kind of learning algorithms are not very useful [2].

Reinforcement learning methods are a different type of algorithms that enable a system to learn based on a scalar evaluation of the system performance [3]. While a supervised learning network is instructed how to change its responses in order to reduce the error, a reinforcement learning system is only evaluated by a *critic* mechanism. Applications of such learning algorithms arise in sequential-decision processes, like the development of optimal strategies for gaming and control.

Reinforcement learning has been widely used in learning games: Backgammon [4], Checkers [5], Go [6], Blackjack [7], Chess [8], Tic-tac-toe [9], etc.

In the early 1970s, Bernard Widrow [7] used the game of blackjack to make the idea of one of the first reinforcement learning algorithms called *selective bootstrap adaptation*. He demonstrated how an ADAptive LINear Element [7] *learned with a critic* to play blackjack without knowing the game and the objective of play. More recently, in [10] neural networks and Temporal Differences (TD) learning were applied to learning play games from experience.

In this paper, we report some experimental results similar to those reported in [10]. In our approach, we used the Q-learning [11] and SARSA [12] algorithms, and a slightly different state encoding. As a result, our system was able to learn strategies with a higher rate of winning games.

This paper is organized as follows: the blackjack game is described in section 2, section 3 describes the reinforcement learning basic concepts, section 4 is dedicated to presenting the SARSA learning algorithm, an on-policy reinforcement learning method, section 5 presents some experimental results and finally, section 6 presents some conclusions and future extensions of this work.

II. The game

Blackjack or twenty-one is a card game where the player attempts to beat the dealer, by obtaining a sum of card values that is equal to or less than 21 so that his total is higher than the dealer's. Each card has the same value as its index except for the ace and the picture cards. All 10's and picture cards are counted as 10. The ace can be valued as either 1 or 11, at the option of the player. At the beginning of the game, each player is dealt two cards in sequence, one face up and one face down. After looking at his first two cards, the player chooses to draw (hit) or to stop drawing cards (stand). The player may take as many hits as he wants as long as he doesn't "bust", i.e., the sum of cards in his hand does not exceed 21. Once all the players have acted on their hands, the dealer flips his or her unexposed card over and draws cards until he/she has a total of 17 or above. In our experiments (as in previous works), the game of blackjack was simplified by removing interesting aspects as "betting", and all special features as "splitting pairs", "doubling down", and "insurance", etc. The game have one dealer, and one or two players.

A. Pérez is supported by the Centre Suisse d'électronique et de Microtechnique CSEM, Neuchâtel

Proc. IEEE Intl. Joint Conf. on Neural Networks IJCNN'98, Anchorage, May 4-9,1998,Vol 3, pp.2022-2027.

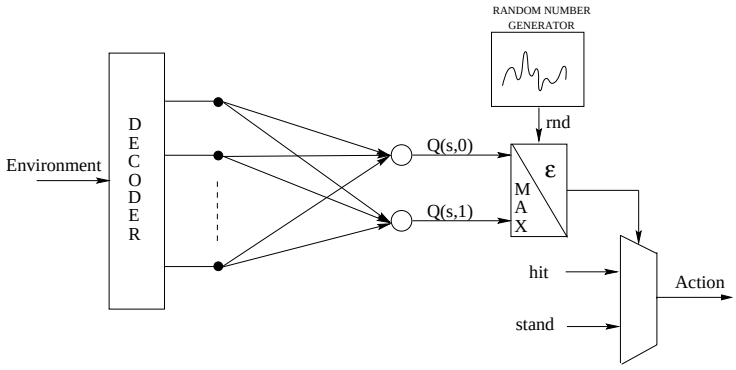

Fig. 1. Neural network architecture. The state of the environment (the deck of cards) is determined by an a-priori partition of the state space. The network learns action-values $Q(s,a)$, and selects actions following an ϵ-greedy policy.

III. Reinforcement learning

Reinforcement learning is a synonym of *learning by interaction*. During learning, the adaptive system *tries* some actions (i.e., output values) on its environment, then, it is *reinforced* by receiving a scalar evaluation (the reward) of its actions. The reinforcement learning algorithms selectively retain the outputs that maximize the received reward over time. Reinforcement learning tasks are generally treated in discrete time steps. At each time step, t, the learning system receives some representation of the environment's *state* s, it takes an action a, and one step later it receives a scalar reward r, and finds itself in a new state s'. The two basic concepts behind reinforcement learning are *trial and error search* and *delayed reward*.

One key aspect of reinforcement learning is a trade-off between *exploitation* and *exploration*. To accumulate a lot of reward, the learning system must prefer the best experienced actions, however, it has to try (to experience) new actions in order to discover better action selections for the future.

A central idea of reinforcement learning is called *temporal difference* (TD) learning. TD methods are general learning algorithms to make long-term predictions about dynamical systems. They are based on estimating *value functions*, functions of states $V(s)$ or action-states pairs $Q(s,a)$ that estimate how good is for the learning system to be in a given state or to take a certain action in a given state. Such value actions guide the action selection mechanism, the *policy* to balance exploration and exploitation, in order to maximize reward over time, and let the learning system achieve its goal.

IV. The SARSA and Q-learning algorithms

The SARSA algorithm [12] is a temporal difference (TD) method that learns action-value functions by a bootstrapping mechanism, that is, by making estimations based on previous estimations. Figure 1 presents a block diagram of the learning neural network model, and figure 2 specifies the SARSA algorithm in procedural form. At every time step, SARSA updates the estimations of the action-value functions $Q(s,a)$ using the quintuple (s,a,r,s',a'), which

1. Initialize the $Q(s,a)$ value functions arbitrarily,
2. Initialize the environment: set a state s,
3. Select an action following a certain policy (e.g. ϵ-greedy),
4. Take action a; observe reward, r, find the next state s', and select the next action a',
5. Update the estimate value $Q(s,a)$: $Q(s,a) = Q(s,a) + \alpha TD_{err}$, where $TD_{err} = r + \gamma Q(s',a') - Q(s,a)$ is the temporal difference error, α is the step size, and γ is a discount reward factor,
6. Let $s = s'$,
7. Go to step 3 until the state s is a terminal state,
8. Repeat steps 2 to 7 for a certain number of episodes.

Fig. 2. SARSA algorithm for estimating action-value functions.

gives rise to the name of the algorithm. SARSA is an on-policy version of the well known Q-learning algorithm [11], where the learned action-value function Q directly approximates the optimal action-value function, denoted by $Q^*(s,a)$.

This can be simply achieved by modifying the temporal difference error as follows :

$$TD_{err} = r + \gamma max_{a'} Q(s',a') - Q(s,a)$$

We did not find important differences between Q-learning and SARSA solutions in the blackjack task. Therefore, in the following sections we only refer to the SARSA learned strategies.

V. Experimental results

As stated above we have implemented a simplified version of the blackjack game. The deck of 52 cards is shuffled before each hand. An Ace is automatically valued as 11, unless it would cause a bust. In that case it is valued as 1. The state s of the environment corresponds to the

Q/S	4	5	6	7	8	9	10	11	
$Q(s,0)$	+0.063	+0.850	+0.860	+0.870	+0.880	+0.884	+0.899	+0.928	
$Q(s,1)$	+0.985	+0.068	+0.059	+0.068	+0.149	+0.149	+0.182	+0.317	
Q/S	12	13	14	15	16	17	18	19	20
$Q(s,0)$	+0.245	+0.078	-0.016	-0.112	-0.191	-0.208	-0.240	-0.223	-0.355
$Q(s,1)$	+1.000	+1.000	+1.000	+1.000	+1.000	+1.000	+1.000	+1.000	+1.000
Q/S	23	24	25	26	27	28	29	30	31
$Q(s,0)$	+0.831	+0.855	+0.860	+0.872	+0.0870	+0.881	+0.899	+0.901	+0.903
$Q(s,1)$	+0.059	+0.059	+0.030	+0.030	+0.077	+0.086	+0.077	+0.068	+0.077

TABLE I

LEARNED ACTION-VALUES $Q(s,a)$ AFTER 1000 RUNS OF 100 GAMES (SETUP OF SECTION B).

Strategy	**Avg(%)**	**Max(%)**	**Min(%)**
dealer's	40.7	57	25
hold	38.3	51	24
random	31.5	46	18

TABLE II

PERFORMANCE OF FIXED STRATEGIES AGAINST THE DEALER'S STRATEGY.

sum of card values in the player's hand (i.e., the score), provided that he/she does not have an ace. Therefore, $s = \{4, 5, 6, ..., 20\}$. If the player has an ace, there is a set of 9 extra states $s = \{23, 24, ..., 31\}$ determined by the sum of card values being held plus 11. Two terminal states $s = 21$ and $s = -1$ were also introduced to indicate a "perfect" score or a bust. At each stage of the game, the player chooses to hit a new card, or to stand. Such selection followed an ϵ-greedy policy, that is, the action with the highest Q value is chosen with probability $1 - \epsilon$, otherwise a random action is chosen with probability ϵ, in order to explore new actions.

A. Performance of fixed strategies

The basic rules of the blackjack are quite simple, however, it is not evident to determine an optimal playing strategy. In the first experiments (one experiment consisted of 1000 runs of 100 games) the simulated player used a set of fixed strategies. The results of such experiments (see table II) enabled us to rank the learned strategies.

Table II, presents the overall percentage of win games, the maximum, and the minimum performance during the 1000 runs, of a player using the dealer's strategy, a conservative strategy (hold) where the player always stands, and a random strategy (all of them against a dealer with a fixed strategy, as explained in the description of the game).

B. Performance of learned strategies

The learning system consisted of a lookup-table implementation of the SARSA algorithm, using the environment's state encoding previously described. In our first experiments, the reinforcement signal was defined as follows:

$$r = \begin{cases} -1 & if\ loss, \\ +1 & if\ win, \\ 0 & after\ every\ hit \end{cases}$$

Using a discount factor of $\gamma = 0.9$, a step size $\alpha = 0.01$, and an ϵ-greedy action selection, $\epsilon = 0.01$, the SARSA learning algorithm discovered the following strategy:

$$action = \begin{cases} hit & if\ (score < 11)\ or\ (an\ ACE\ is\ held) \\ stand & otherwise \end{cases}$$

Such strategy can be interpreted from the learned action-values after the 1000 runs of 100 games, given in Table I (as a matter of fact, the Q values in Table I indicate that in state 4, the learner should not hit, however, this can be explained because such state has perhaps not been visited enough times due to the low probability of receiving two consecutive two-valued cards at the beginning).

The learned strategy is very conservative, i.e., not to hit if the score is larger than 11. However, it is interesting to note how the algorithm determines such threshold without knowing the rules of the game, nor the goal of the game (just by experience and the reinforcement signal at the end of each hand). In fact, the SARSA player learns to hit only if it is sure that it will not bust; it discovered by experience the double card value of the ace (note, that states 23 to 31 correspond to a sum of the card values in hand between 2 and 10, when considering the value of an ace as 1), and the maximum card value of 10 to determine the above strategy.

Table III, presents the performance results of such algorithm with different values of ϵ: 0.1, 0.01, and a decaying exploration probability: $\epsilon(r) = 0.99^r * \epsilon(0)$, where $\epsilon(0) = 0.1$, and r is the current run ($0 < r < 1000$).

In table III, it can be seen that a high exploration probability (i.e., where $\epsilon = 0.1$) can find temporary solutions with very good results, however, this may lead to reduced overall performance, even lower than the dealer's fixed strategy. On the other hand, the overall performance of the SARSA algorithm with 0.01-greedy selection action mechanism surpasses the average rate of success of 40.6% reported in [10].

Q/S	4	5	6	7	8	9	10	11	
Q_0	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000	-0.007	-0.008	
Q_1	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000	
Q/S	12	13	14	15	16	17	18	19	20
Q_0	+0.053	+0.358	-0.376	-0.181	-0.181	-0.344	-0.388	-0.410	-0.432
Q_1	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000
Q/S	23	24	25	26	27	28	29	30	31
Q_0	+0.000	+0.000	-0.006	-0.008	-0.007	+0.000	-0.029	-0.024	+0.000
Q_1	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000

TABLE IV

Learned action-values $Q(s,a)$ after 100 runs of 100 games (setup of section C).

ϵ	**Avg(%)**	**Max(%)**	**Min(%)**
0.1	39.9	54	26
0.01	41.9	56	26
$0.1 * 0.99^r$	40.9	53	26

TABLE III

Performance of learned strategies against the dealer's strategy.

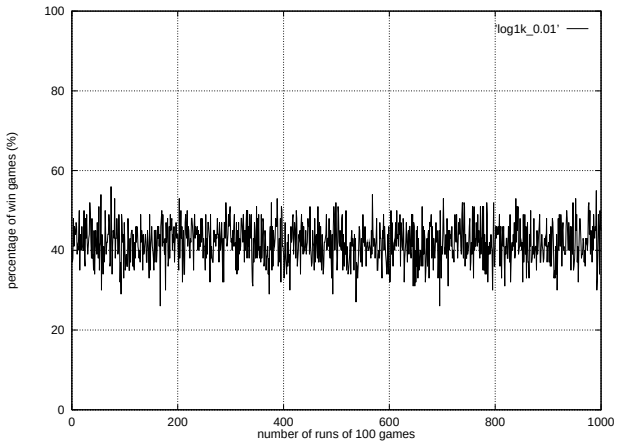

Fig. 3. Percentage of win games during 1000 runs of 100 games.

Figure 3 shows the percentage of win games during the 1000 runs of a SARSA learner with 0.01-greedy action selection mechanism.

C. Probabilistic strategies

When one considers a *nonstationary* task, reinforcement learning algorithms track the best actions as the task changes over time, instead of finding a single best action. When a task's optimal stationary policy is probabilistic, reinforcement learning algorithms find a mapping from states to discrete probability distributions over actions, which is denoted by

$$\pi : S \rightarrow PD(A)$$

where $PD(A)$ represents the set of discrete probability distributions over the set A, i.e., the actions [13].

Learning blackjack strategies need not to have a deterministic optimal policy. Instead, due to the probabilistic nature of the game it makes sense to have changing probabilistic strategies over time. The next experiments make the idea of changing strategies.

In these experiments, we used the same state encoding previously described, but this time, we used a reinforcement signal defined as follows:

$$r = \begin{cases} -1 & if\ loss \\ 0 & otherwise \end{cases}$$

Table IV presents the learned action-values with the last setup, after only 100 runs of 100 games. By interpreting such action-values, and considering that an ϵ-greedy action selection mechanism was used, we may describe the current learned strategy as follows:

if *(score> 9)or[(score> 13)and(score< 19) and(an ACE is held)] then* **stand**,
else **hit** *with 50% of probability.*

The resulting performance was surprisingly high: an average of 49.14% win games, a maximum percentage of 61% win games, and a minimum percentage of 42% win games. However, when using such strategy without further learning during 1000 runs of 100 games, the results are again poor: a mere 40% of games won in average. Nevertheless, we proceed with our experiments with continual learning, and let the system adapt to the nonstationary problem.

The learner developed changing probabilistic strategies over time. Therefore, we measured the mean of win games after 1000, 5000, 10000, and 20000 runs of 100 games to observe if the performance changed. The results are shown in table V.

D. A game with three players

In [13], a set of Markov games (i.e, a special class of games where the state signal succeeds in retaining all relevant information or summarize past sensations of the task) were proposed as a framework for multi-agent reinforce-

Number of runs	Avg(%) win games
100	49.1
1000	47.4
5000	46.8
10000	46.7
20000	46.7

TABLE V
PERFORMANCE OF LEARNED PROBABILISTIC STRATEGIES AGAINST THE DEALER'S STRATEGY.

ment learning. In such paper it was specially noted that although the simultaneously training of two adaptive agents using Q-learning was not mathematically justified, and in practice is susceptible to lock-up in local maxima, some researchers had reported amazing results with this approach [14]. Motivated by these claims, we implemented a three-player blackjack game: two SARSA-learning players and a dealer. The reinforcement signal was 0 at every hit, -1 for loss, and 1 for win; the state encoding was the same as in previous experiments, except that for the second learning player, we introduced 26 new states, in order to let the player discover different strategies when the first player had a bust or not. The state s is determined as in section B, however, if the first player had a bust, we let $s = s + 31$ (for the second player) such that states $s = \{31+4, 31+5, ...51, 31+23, 31+24, ...61\}$ correspond to that new situation.

In blackjack, players try to beat the dealer, not the other players. However, we let the second player receive a negative reinforcement if it looses against both the dealer or the first player, and observed the learned strategies.

After an experiment of 1000 runs of 100 games, the first player discovered the same conservative strategy as in section B (i.e., not to hit if the score is larger than 11). The second learning player benefited from the extra states and developed two different strategies:

if player 1 had a bust then
do not hit, if the score is larger than 11,
else **do not hit at any score**

The learned strategy is very conservative once again. If the first player stands, the second learning player learned to stand at any score, only hoping that the dealer will bust, and that his score will be larger than that of the first player.

As a matter of fact, such learned strategies were not very good, and it seems that both learning agents stuck in local maximums as stated in [13]. The first learning player (the one with 29 states, as in section B) exhibited an average of 30% of games won, the second learning player, an average of 28% win games, and the dealer, an average of 42% win games.

VI. Conclusions

This paper explored the use of blackjack as a test bed for learning strategies in neural networks. We found that the basic rules of the simplified version of blackjack were easily implemented and they were not computationaly intensive. Furthermore, the probabilistic nature of the game and the ensemble of experiments described through out the paper provided a didactical framework for learning strategies in neural networks. Finally, we reported some performance comparisons, and found out that our state encoding of the problem not only facilitated the interpretation of the learned strategies, but led us to better results.

SARSA and Q-learning algorithms are commonly implemented as look-up tables. In [15] we reported an FPGA (Field Programmable Gate Array) [16] implementation of a similar algorithm called *Adaptive Heuristic Critic* (AHC) learning, and a mechanism to automatically determine the partition of the state space. Future work would attempt to use a similar approach to automatically partition the state space of this kind of tasks. On the other hand, *evolvable hardware* techniques [17] may be used to evolve finite state machines or probabilistic state machines that implement good blackjack strategies.

References

[1] Françoise Fogelman-Soilié, "Applications of neural networks," in *Handbook of Brain Theory and Neural Networks*, Michael A. Arbib, Ed., pp. 94–98. MIT Press, 1995.

[2] V. Gullapalli, "A comparison of supervised and reinforcement learning methods on a reinforcement learning task," Computer and Information Science Department, University of Massachussetts, Amherst (unpublished paper), 1992.

[3] R. Sutton and A. Barto, *Introduction to Reinforcement Learning*, MIT Press, 1998 (to appear).

[4] Gerald Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994.

[5] A.L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal*, pp. 211–229, july 1959.

[6] Nicol N. Schraudolph, Peter Dayan, and Terrence J. Sejnowski, "Temporal difference learning of position evaluation in the game of go," in *Advances in Neural Information Processing Systems*, Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, Eds. 1994, vol. 6, pp. 817–824, Morgan Kaufmann Publishers, Inc.

[7] Bernard Widrow, Nerendra Gupta, and Sidhartha Maitra, "Punish/Reward: Learning with a Critic in Adaptive Threshold Systems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 3, no. 5, pp. 455–465, 1973.

[8] Robert A. Levinson, "A self-learning, pattern-oriented chess program," *ICCA Journal*, pp. 12(4):207–215, 1989.

[9] Justin A. Boyan, "Modular neural networks for learning context-dependent game strategies," M.S. thesis, University of Cambridge, Department of Engineering and Computer Laboatory, 1992.

[10] Daniel Kenneth Olson, "Learning to play games from experience: An application of artificial neural networks and temporal difference learning," M.S. thesis, Pacific Lutheran University, Washington, 1993.

[11] Christopher J.C.H. Watkins and Peter Dayan, "Technical note q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[12] G. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," Tech. Rep. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.

[13] Michael L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proceedings of the Eleven International Conference on Machine Learning*. 1994, pp. 157–163, Morgan Kaufmann.

Proc. IEEE Intl. Joint Conf. on Neural Networks IJCNN'98, Anchorage, May 4-9,1998,Vol 3, pp.2022-2027.

[14] Gerald Tesauro, "Practical issues in temporal difference learning," in *Advances in Neural Information Processing Systems*, John E. Moody, Steve J. Hanson, and Richard P. Lippmann, Eds. 1992, vol. 4, pp. 259–266, Morgan Kaufmann Publishers, Inc.

[15] Andres Perez and Eduardo Sanchez, "FPGA implementation of a network of neronlike adaptive elements," in *Proceedings of the International Conference on Artificial Neural Networks ICANN97*, Springer Verlag, 1997 (to appear).

[16] S. M. Trimberger, *Field-Programmable Gate Array Technology*, Kluwer Academic Publishers, Boston, 1994.

[17] Eduardo Sanchez and Marco Tomassini (Eds), *Towards Evolvable Hardware: The Evolutionary Engineering Approach*, Lecture Notes in Computer Science 1062, Springer, April 1996.