https://doi.org/10.31436/iiumej.v23i1.1807

MAINTAIN AGENT CONSISTENCY IN SURAKARTA CHESS USING DUELING DEEP NETWORK WITH INCREASING BATCH

RIAN ADAM RAJAGEDE

Department of Informatics, Universitas Islam Indonesia, Sleman, Indonesia

**Corresponding author: rian.adam@uii.ac.id*

(Received: 25th January 2021; Accepted: 15th July 2021; Published on-line: 4th January 2022)

ABSTRACT: Deep reinforcement learning usage in creating intelligent agents for various tasks has shown outstanding performance, particularly the Q-Learning algorithm. Deep Q-Network (DQN) is a reinforcement learning algorithm that combines the Q-Learning algorithm and deep neural networks as an approximator function. In the single-agent environment, the DQN model successfully surpasses human ability several times over. Still, when there are other agents in the environment, DQN may experience decreased performance. This research evaluated a DQN agent to play in the two-player traditional board game of Surakarta Chess. One of the drawbacks that we found when using DQN in two-player games is its consistency. The agent will experience performance degradation when facing different opponents. This research shows Dueling Deep Q-Network usage with increasing batch size can improve the agent's performance consistency. Our agent trained against a rule-based agent that acts based on the Surakarta Chess positional properties and was then evaluated using different rule-based agents. The best agent used Dueling DQN architecture with increasing batch size that produced a 57% average win rate against ten different agents after training for a short period.

ABSTRAK: Pembelajaran Peneguhan Mendalam adalah terbaik apabila digunakan bagi mewujudkan ejen pintar dalam menyelesaikan pelbagai tugasan, terutama jika ia melibatkan algoritma Pembelajaran-Q. Algoritma Rangkaian-Q Mendalam (DQN) adalah Pembelajaran Peneguhan berasaskan gabungan algoritma Pembelajaran-Q dan rangkaian neural sebagai fungsi penghampiran. Melalui persekitaran ejen tunggal, model DQN telah beberapa kali berjaya mengatasi kemampuan manusia. Namun, ketika ejen lain berada dalam persekitaran ini, DQN mungkin kurang berjaya. Kajian ini melibatkan ejen DQN bermain papan tradisional iaitu Catur Surakarta dengan dua pemain. Salah satu kekurangan yang dijumpai adalah konsistensi. Ejen ini akan kurang bagus ketika berhadapan lawan berbeza. Kajian menunjukkan dengan penggunaan Rangkaian-Q Dwipertarungan Mendalam bersama peningkatan saiz kumpulan dapat meningkatkan konsistensi prestasi ejen. Ejen ini telah dilatih untuk melawan ejen lain berasaskan peraturan dan sifat kedudukan Catur Surakarta. Kemudian, ejen ini diuji berpandukan peraturan berbeza. Ejen terbaik adalah yang menggunakan rekaan DQN Dwipertarungan bersama peningkatan saiz kumpulan. Ianya berhasil memenangi permainan dengan purata 57% berbanding sepuluh agen lain melalui latihan jangka masa pendek.

KEYWORDS: *reinforcement learning; Dueling Deep Q-Network; increasing batch size; Surakarta chess*

1. INTRODUCTION

Deep reinforcement learning has been able to produce agents that can solve a variety of complex problems, such as simulated car driving [1], video games [2], or natural language processing [3]. A game is an environment that is often used to evaluate reinforcement learning agents' ability to interact with other agents and make decisions to achieve specific goals. Researchers can focus on algorithms with a relatively more straightforward environment before implementing them in more complex environments such as self-driving cars.

The deep learning approach successfully captures richer features and improves agent performance when combined in reinforcement learning algorithms. For example, in [2] the produced agent uses Deep Q-Learning, the combination of Convolutional Neural Network [4] and Q-Learning [5] to play and win Atari games. Another quite phenomenal one is Alpha Go [6], an agent that manages to beat the best human players in the world in a reasonably complex game, Go, using deep reinforcement learning. In various types of board games, human-level abilities have also been surpassed by intelligent agents, such as in Othello [7,8], Mancala [9], Chess [10,11], Go [6,12], or even in a game with incomplete information, such as Poker [13]. Those agents are trained using reinforcement learning, deep learning, or a combination of both. In [8], the Othello game is seen as a classification problem. The agent trained on the game database using a deep convolutional neural network to decide the best action in the given state. In contrast to [7], agents in the Othello game trained using neural fitted temporal difference learning, which is a method in reinforcement learning. In chess, the previous studies also use deep learning [11] or deep reinforcement learning [10] to produce a good performance agent.

Board games have different characteristics compared to Atari single-player video games previously researched [2]. In the majority of board games, the agent must interact with one or more players. This environmental behaviour is called a Markov Game [14]. The presence of other players in the game dramatically affects the agent in making decisions and can make it difficult for us to produce intelligent agents. In [12], agents are trained using Convolutional Neural Network and Monte Carlo Tree Search (MCTS) by self-playing in the Go game without being given explicit information by humans. Self-playing means the agent plays games against itself. The method is effective as the agent produced using this method successfully defeats humans in the Go game and the agent from the previous research [6] with a shorter training time. Another form of the agent training process that can be used is to train the agent against the rule-based player who moves based on the positional values of the board game [15] or uses a database that stores records of previous matches [8].

This research will implement and evaluate the reinforcement learning algorithm, Deep Q-Learning, in the two-player board game of Surakarta Chess. Surakarta Chess has unique game characteristics. One of them is that some positions on the board can be very profitable to defeat the opponent. From those properties, we can make rule-based agents train the reinforcement learning agent. The making of intelligent agents in Surakarta Chess has been studied previously. In those studies, agents were made based on two search methods, αβ-search [16–18] and based on MCTS [19,20]. We proposed to use Deep Q-Learning as a different approach to creating an agent in Surakarta Chess. Deep Q-Learning differs from the other methods mentioned earlier as Q-learning is not based on the search method. The use of Q-Learning on two-player board games has been studied before in several different types of games [9,15,21-24]. In [22], Deep Q-Learning still found it difficult to beat the searching method in the Hex game even though it has been trained for two weeks (about

60,000 episodes). The long training process is also shown in [24]. The model was trained to over 500,000 episodes to achieve about a 50% winning rate against heuristic agents.

This research explored Deep Q-Learning architecture variants, including Dueling Q-Network [24], and proposed increasing batch size method [25] as a regularization technique in Deep Q-Learning to speed up the training process and improve the agent performance. We trained the agent against a rule-based agent instead of a random agent, as done in [9]. In previous studies, agents were trained and evaluated against the same opponent [9,24]. In this study, to see the agent's consistency in dealing with different opponents, our agent was evaluated against different rule-based agents. Agent consistency in facing other agents in achieving a goal is crucial in various tasks, not only in games.

2. RESEARCH METHOD

The process of interaction and training in the reinforcement learning agent (RL agent) is shown in Fig.1. Agents will be trained to use Deep Q-Learning against a rule-based agent in the Surakarta Chess environment. At a timestep *t*, the RL agent will receive a board condition (a state *S*) and then perform one of the valid actions *A* based on its policy. In the next turn, the opponent will do the same thing: see the game's condition and then take action. The results of RL agent action on time steps *t* and opponent actions on time step *t*+1 make a new game state, *S'*. The agent will get two types of rewards, *R*, reward when winning a game, and reward when capturing the opponent's pieces that the system environment already defines. The RL agent learns from a collection of experiences represented by the tuple of (*S*, *A*, *R*, *S'*). All agents are implemented using PyTorch [26] and trained on Google Colab's GPU.

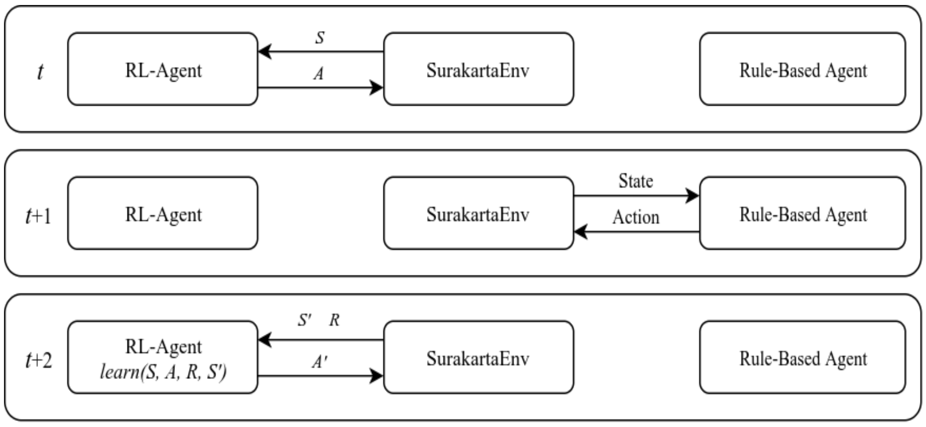

Fig. 1: Interaction between RL agent, the opponent and the environment.

2.1 Surakarta Chess

Surakarta Chess is a chess-like game from Surakarta, Indonesia, contested annually by the International Computer Game Association (ICGA). Surakarta Chess is played by two players on a 6x6 board with 8 curves and 12 pieces. The starting position of each player on the board is shown in Fig. 2 (a). In each turn, players can move their piece or capture the opponent's piece. The player can move his piece to eight adjacent positions if another piece has not occupied the destination. A player can capture the opponent's piece by moving a piece along a horizontal or vertical line then through at least one curve to reach the opponent's piece without being blocked by another piece. Illustrations of piece movement and capture are shown in Fig. 2 (b) and Fig. 2 (c).

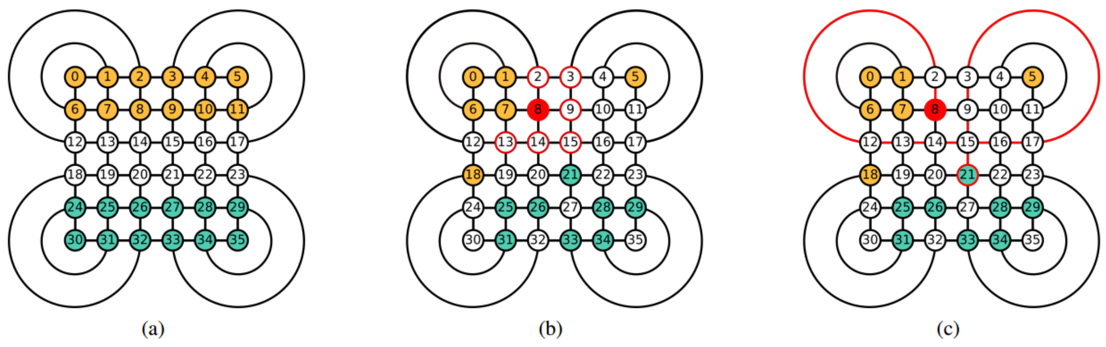

Fig. 2: (a) Surakarta Chess initial state; (b) available move for a piece in 8 shown as circles with red border; (c) capturing path from 8 to 21 shown as red path.

Because a piece must be in a particular position to capture, in previous studies, they use position values to measure how good a piece's position is [17,19]. There are three types of positions in Surakarta Chess based on their relationship to the curve. The three types of positions are shown in Fig. 3, distinguished by red, white, black, and yellow.

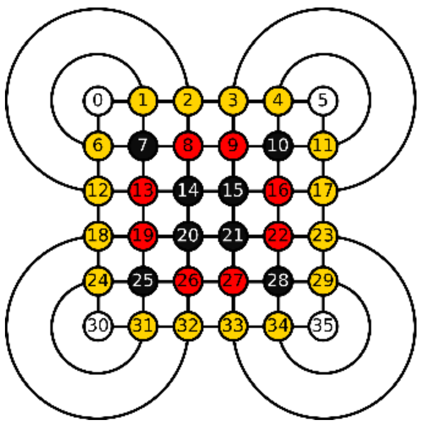

Fig. 3: Positional value in Surakarta Chess.

Surakarta Chess consists of three phases [16], namely the opening phase, the middle phase, and the final phase. In the opening phase, the player will focus on placing their pieces in an advantageous position. This is crucial because the player needs to make an unblocked path to capture the opponent's piece. The middle phase occurs after pathways are created. In this phase, players will begin to attack and defend their pieces from their opponent. The final phase is indicated by the small number of pieces on the board. In this phase, the pieces are free to move and capture the opponent's piece because the path to capture is already open.

2.2 Surakarta Chess Environment

In this study, we use *SurakartaEnv* part of Gapoera API [27], a publicly available game environment toolkit. Players can access current board information using available functions. One aspect of the information provided to players through *SurakartaEnv* is the position of the piece. *SurakartaEnv* maps the game board to a matrix with a value of -1, 0, or 1. A value of 0 is given to a position if no pieces are occupying it. A value of 1 or a value of -1 indicates the player's pieces' position depending on the player's perspective. The value 1 indicates the first player's position, while the value of -1 indicates the second player's position. Each position on the board is numbered from 0 to 35, as shown in Fig. 2. From each of these positions, 12 actions can be done, 8 of which are actions to move without capturing in 8 directions and 4 of which are actions to move across the line horizontally or vertically until reaching the curve to capture. Therefore, there are a total of 432 available actions. *SurakartaEnv* accepts an action A in the form of an integer between 0 to 431.

In addition to the position of the pieces, both players can also access other information through *SurakartaEnv*. Players can get information on which pieces that player can capture, which player pieces that the opponent can capture, and which actions are valid in each turn for both players. The rule-based agent will later use that information to make decisions in every turn. *SurakartaEnv,* by default, sets rewards to both players for each action. We use a +10 reward if the player wins the match, and -10 if the player loses, and +0.5 when the player captures the opponent's piece.

2.3 Rule-Based Agent

A rule-based agent decides its action based on game information available at the time. In this research, three rule-based agents are made that take action based on a particular priority. When an agent cannot perform any valid moves based on its priority, the agent will make a valid random move. The explanation of each priority for each agent is shown in Table 1. The red and black positions refer to Fig. 3. A safe place is a place where the player's piece is safe from opponent's attacks.

Table 1: The priority of each rule-based agent

Agent Name	**RB1**	**RB2**	**RB3**
Movement Priority	1. Capture 2. Move to black 3. Move to a safe place	1. Capture 2. Move to red 3. Move to black 4. Move to a safe place	1. Capture 2. Move to red 3. Move to a safe place

Each agent has an exploration probability parameter ε that gives agents the possibility to make a random move that is not based on their priorities. This random action is intended to create variations both in the training process and in the testing process. In this study, RL-agents will be trained against RB1 with ε = 0.1.

2.4 Reinforcement Learning Agent

Reinforcement learning agents were trained using the Deep Q-Learning method [2]. Deep Q-learning is the improvement of the Q-Learning algorithm [5] that uses a deep neural network as an approximator of the action-value function. The neural network architecture used in [2] is Convolutional Neural Network (CNN) [4]. This study will compare CNN architecture with a simple neural network without convolutional layers or commonly called multi-layer perceptron (MLP). Both architectures are shown in Fig. 4. Both architectures are simpler than those proposed in [2], given that the Surakarta Chessboard size is smaller than the Atari game image matrix. In [2], 84x84x4 inputs represent pixels on the screen from the last four frames, while in Surakarta, the input is only a 6x6 matrix. It also makes consideration for comparing MLP with CNN in Surakarta Chess.

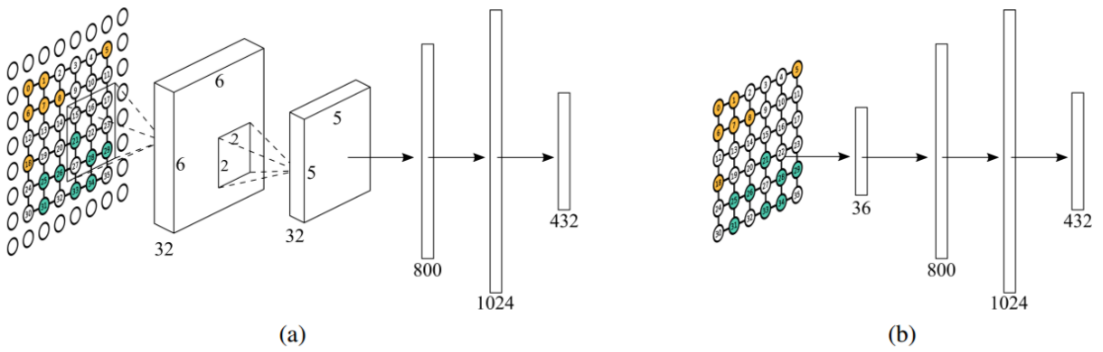

Fig. 4: (a) Q-Network Architecture with Convolutional Layer, (b) Q-Network Architecture using Multi-Layer Perceptron.

Figure 4 (a) shows the architecture consisting of one convolution layer, one pooling layer, and a fully connected layer. Inputs are padded to keep the output size the same after the convolutional process. The convolutional layer uses a 3x3 kernel size and 32 feature maps, while the pooling layer uses a 2x2 size pooling window. The neural network output is a vector with 432 elements, representing all possible actions in Surakarta Chess. In the MLP architecture, two hidden layers are used with the number of neurons similar to the neurons from the convolutional process on the CNN architecture.

We use two techniques proposed in [2], experience replay and the use of two neural networks as policy and target networks in one agent. Experience replay uses a replay memory with a capacity *C* to store the agent's experiences in the form of (*S*, *A*, *R*, *S'*). In this study, all agents are set to have the same memory replay capacity of 50,000 experiences. During the training process, the agent will take a batch of data from the replay memory as training data. We also explore the effect of increasing batch data size [25] on the model performance. Increasing batch size means that samples of experience drawn from replay memory will increase through episodes. In [25], this method can help speed up the model training process. The second technique we use is the use of policy and target networks in one agent that can increase the stability of the model. Initially, the target network is a clone of the policy network, but its weight is only updated after the policy network is updated *N* times. The algorithm used to optimize network policy is Adam Optimizer [28], with an initial learning rate of 0.0001. The weights and biases of the networks are initialized using the methods proposed in [29, 30].

We also compared another architecture called Duelling Deep Q-Network [24]. The duelling network has two separate streams. One will produce a scalar state value *V*(*s*), and the other will produce action advantages *A*(*s*, *a*). State values will provide information on whether a state is valuable or not, while action advantages provide information about the advantages of each action. The two values are then combined to get *Q*(*s*, *a*) using Eq. (1). In previous research [24], a duelling network was proposed using CNN. This research will also compare the model performance when using MLP on the duelling network. The Duelling Network architecture is shown in Fig. 5.

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|A|} \sum_{a'} A(s,a') \right) \quad (1)$$

During the training process, both agents that use DQN and Duelling Q-Network will use ε-greedy as a policy to decide the action in a state. The value of ε is the agent's probability of taking random actions to explore states. In this study, the value of ε initially used was 13%. This value will be reduced after some episodes, so the agent acts based more on the state action value. Each agent was trained for 1250 episodes. In episodes 75, 150, and 250, the value of ε will be reduced by 3.7%. As a result, after 250 episodes, the value of ε is only 1.9%.

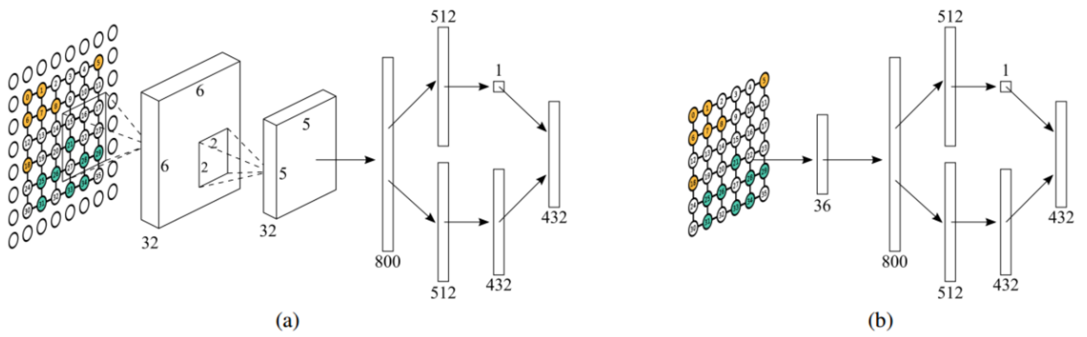

Fig. 5: (a) Dueling Q-Network Architecture with Convolutional Layer, (b) MLP on Dueling Q-Network Architecture.

3. RESULTS AND DISCUSSION

3.1 Model Performance

RL agents will be trained and their parameters tuned against RB1 and their performance evaluated against other agents. In the first experiment, we compared the effect of increasing batch size [25]. This experiment used DQN with a convolutional network as a model architecture. Three agents were compared. The first was an agent without increasing batch size, the second was an agent with a less frequently increasing batch size, and the third was a model with a more frequent increasing batch size. The result of the agents against RB1 is shown in Table 2.

Table 2: Performance comparison of increasing batch size usage

	Increase frequency	**Average score from last episodes**			
		Last 50	**Last 100**	**Last 150**	**Last 200**
Model 1	-	5.80	5.39	5.39	5.60
Model 2	250	5.40	4.82	5.53	5.96
Model 3	150	**6.80**	**6.45**	**6.59**	**6.38**

The agent's score is the number of opponent pieces captured by the agent until the game is finished. Model 1 did not use increasing batch size, while models 2 and 3 used it. Model 2 used a less frequently increasing batch size that increases the batch size 50% from the current size every 250 episodes, while Model 3 used a more frequently increasing batch size that increased the batch size 50% every 150 episodes. This process was conducted until it reached 1000 episodes. All models had an initial batch size of 48. The graph of increasing batch size is shown in Fig. 6. Table 2 shows that increasing the batch size during training can help the model improve its performance.

Another parameter that affects the agent's performance is the value of *N*, which is the number of updates made on the policy network before updating the target network. We tested three *N* values: 10, 210, and 700. The *N* values used were much smaller than those used in [2] because we used fewer iterations in the training phase. In a single game, we assumed the agent had 70 turns so that when $N = 210$, the target network was updated around every three games. The score for each model against RB1 for each episode appears in Fig. 7. It showed that if we used a too small or too large *N*, the agent performance became unstable and tended to decrease over episodes. Whereas when $N = 210$, the score obtained by the model was ascending with relatively more stability. The three agents shown in Fig. 7 used DQN with a convolutional layer and increasing batch size.

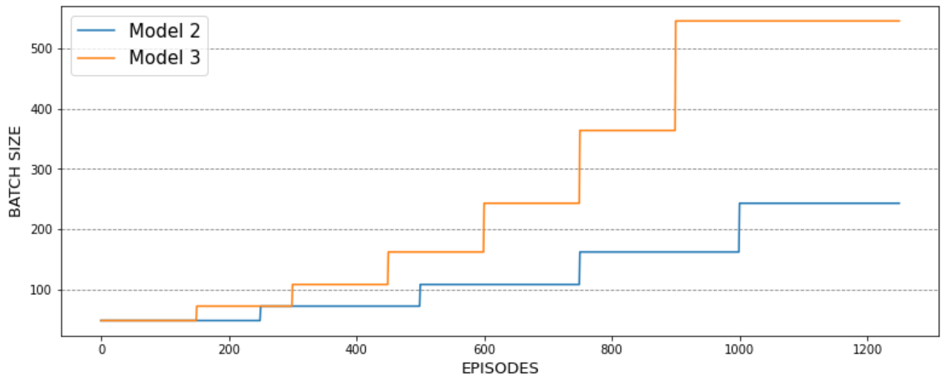

Fig. 6: Batch size throughout episodes.

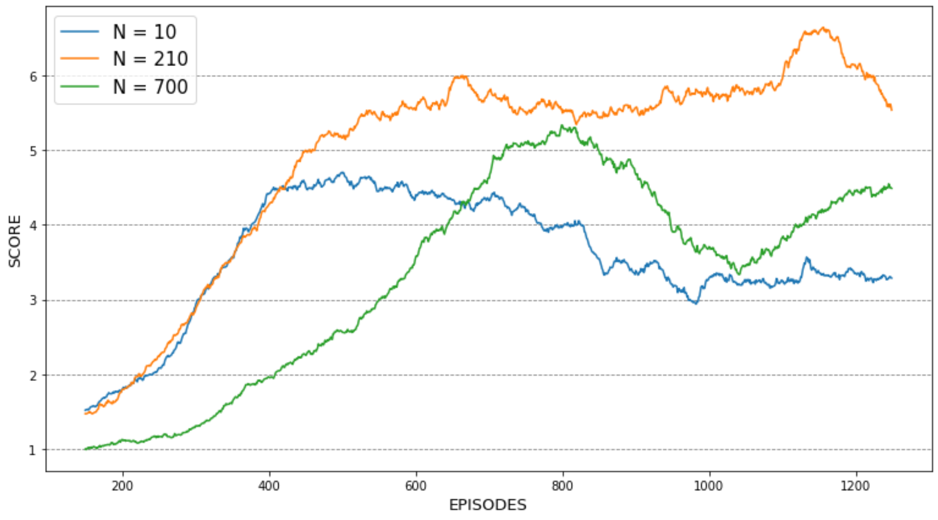

Fig. 7: Performance comparison of N, using moving average scores for every 150 episodes.

This research also compared four agents with different neural network architectures. Table 3 shows that the use of the duelling network significantly increased the agent's performance. The scores shown are averages of the last few scores. For example, in the "last 50" column, the value in that column is the average of the RL agent scores for the last 50 episodes, from episodes 1201 to 1250.

The use of a duelling network with a convolutional layer outperformed other agents. Even the duelling network without a convolutional layer still outperformed the regular DQN with a convolutional layer. Scores of each model for every episode are shown in Fig. 8.

Table 3: Performance comparison of network architecture

Model type	Average score from last episodes			
	Last 50	**Last 100**	**Last 150**	**Last 200**
DQN-CNN	6.80	6.45	6.59	6.38
DQN-MLP	5.78	5.55	5.49	5.49
Dueling-CNN	**9.36**	**8.69**	**8.25**	**8.21**
Dueling-MLP	8.08	7.45	**8.25**	6.63

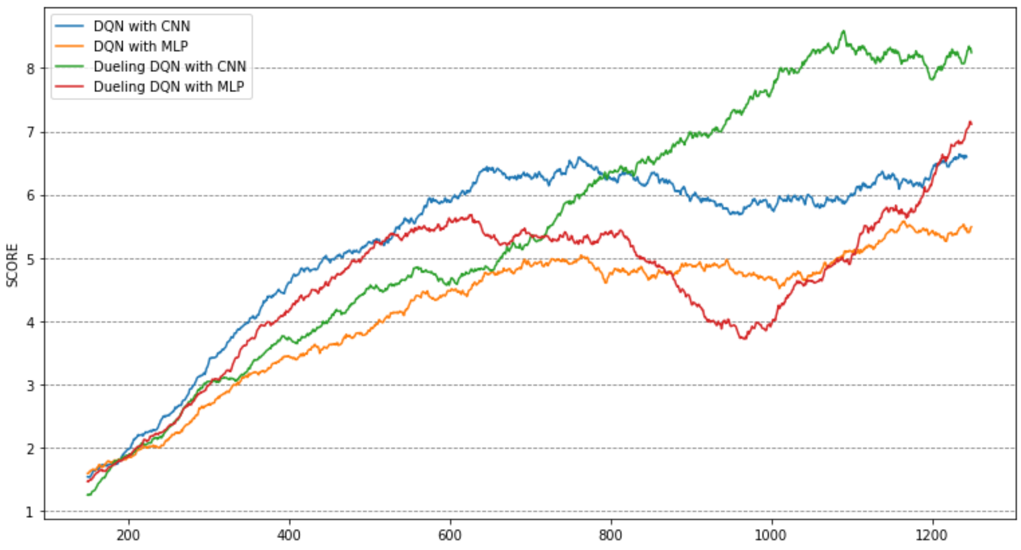

Fig. 8: Performance comparison of Q-Learning models, using moving average scores for every 150 episodes.

3.2 Score Against Other Agents

After completing the training process, we evaluated four RL agents with different network architectures using the parameter settings mentioned earlier. These agents competed against five different rule-based agents and one random agent. The random agent was an agent that always took a valid random movement in each turn. Each RL agent played 30 games for each rule-based agent. The RL agent played as the first player who makes the first move. In Table 4, it shows the number of RL agent wins in 30 games against other agents. We evaluated five agents:

- two agents without duelling networks (DQN-CNN and DQN-MLP)
- two agents with duelling networks (Duelling-CNN and Duelling-MLP)
- an agent for Duelling-CNN that was trained without an increasing batch

The last agent we mentioned was used to compare the effect of an increasing batch on agent performance.

From the table, we can see that all five agents can defeat the random agent (RA) in all games for a 100% win rate. It also shows that agents trained using the duelling network had the most wins against nine other agents. When an opponent had a greater ε value, meaning that the opponent had a greater chance of moving randomly, agents were expected to exploit this vulnerability to win more. However, as shown in the table, only agents who used the duelling network tended to be able to use it as an opportunity to win the match, whereas, in the DQN-MLP or DQN-CNN model, an increase in the value of ε did not benefit the model significantly. The agent that did not use increasing batches had poor performance because of difficulties in the training process, as shown in Table 4.

Duelling network agents can also adapt to agents that have different rules, RB2 and RB3. Both rule-based agents had a slight difference in priority for determining their action compared to RB1. This adaptation did not occur in other models that experienced a high number of losses against rule-based agents aside from the training process. The average win rate is calculated by averaging the win rate of an RL agent against the other agents.

Table 4: Performance comparison of models against other rule-based agents. The shaded column is the agent that is countered during the training process

Model type	Number of RL agent wins										Avg. Win. Rate (%)
	RA	RB1			RB2			RB3			
		ε=0.1	ε=0.2	ε=0.3	ε=0.1	ε=0.2	ε=0.3	ε=0.1	ε=0.2	ε=0.3	
DQN-CNN	30	2	7	5	1	3	2	3	2	1	18.6
DQN-MLP	30	7	4	0	0	0	1	0	0	0	13.9
Dueling-CNN	**30**	**13**	**16**	**23**	**12**	**12**	**19**	**9**	**18**	**19**	**57.0**
Dueling-MLP	30	11	13	13	5	10	11	4	7	13	39.0
Dueling-CNN (no inc. batch)	30	0	1	1	0	4	1	0	0	0	12.3

3.3 Model Consistency

To see an agent's consistency in facing other agents, we used the Coefficient of Variations (CV), as shown in Eq. (2), where σ is the standard deviation, and μ is the mean of the data. The smaller the CV value, the more consistent the model. We calculated the CV between the three rule-based agents by separating them based on their epsilon value. The results are shown in Table 5.

$$CV = \frac{\sigma}{\mu} \times 100 \tag{2}$$

For example, in Table 5, Duelling-CNN had a CV of 14.99 in column with ε = 0.1. This was calculated from {13, 12, 9}, the number of Duelling-CNN agent wins against RB1 ε = 0.1, RB2 ε = 0.1, and RB3 ε = 0.1, respectively. From Table 5, we can see that the Duelling-CNN model had a relatively small CV value compared to other agents. Except when the value ε = 0.3, the Duelling-MLP agent had the smallest CV value. The CV value of agent Duelling-CNN without increasing batch in column ε=0.1 could not be calculated because the agent had no victory.

Table 5: Coefficient of variations between rule-based agents

Model type	**Coefficient of variation**		
	ε=0.1	**ε=0.2**	**ε=0.3**
DQN-CNN	40.82	54.00	63.73
DQN-MLP	141.42	141.42	141.42
Duelling-CNN	**14.99**	**16.26**	9.27
Duelling-MLP	46.36	24.49	**7.64**
Duelling-CNN w/o increasing batch	-	101.98	70.71

4. CONCLUSION

In this research, a deep reinforcement learning approach was implemented to create agents in Surakarta Chess. We explored several Deep Q-Learning architectures to find the best agent. The results show that the use of duelling networks and increasing batch size can improve agent performance and consistency in the Surakarta Chess game. In a short period, agents with duelling network architecture and increasing batch size mechanisms succeeded in obtaining a high average win rate, 57%, against ten different agents. We also found that the best model had better consistency even though it played against agents with different strategies. In the future, duelling networks and increasing batch size can be evaluated in more complex environments.

REFERENCES

[1] Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D Continuous control with deep reinforcement learning. In 4th International Conference on Learning Representations: 2-4 May 2016; San Juan, Puerto Rico. Retrieved from http://arxiv.org/abs/1509.02971.

[2] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D. (2015) Human-level control through deep reinforcement learning. Nature, 518 (7540): 529–533. doi: 10.1038/nature14236.

[3] Yogatama D, Blunsom P, Dyer C, Grefenstette E, Ling W Learning to compose words into sentences with reinforcement learning. In 5th International Conference on Learning Representations: 24-26 April 2017; Toulon, France. Retrieved from https://openreview.net/forum?id=Skvgqgqxe.

[4] LeCun Y, Bottou L, Bengio Y, Haffner P. (1998) Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86 (11): 2278–2324.

[5] Watkins CJ, Dayan P. (1992) Q-learning. Machine learning, 8 (3–4): 279–292.

[6] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D. (2016) Mastering the game of Go with deep neural networks and tree search. Nature, 529 (7587): 484–489. doi: 10.1038/nature16961.

[7] van den Dries S, Wiering MA. (2012) Neural-fitted TD-leaf learning for playing othello with structured neural networks. IEEE Transactions on Neural Networks and Learning Systems, 23 (11): 1701–1713. doi: 10.1109/TNNLS.2012.2210559.

[8] Liskowski P, Jaskowski W, Krawiec K. (2018) Learning to play othello with deep neural networks. IEEE Transactions on Games, 10 (4): 354–364. doi: 10.1109/TG.2018.2799997.

[9] Kasim MF. (2016) Playing the game of congklak with reinforcement learning. In Proceedings of 8th International Conference on Information Technology and Electrical Engineering (ICITEE). IEEE. pp 1–5. doi: 10.1109/ICITEED.2016.7863309.

[10] Lai M. (2015) Giraffe: Using deep reinforcement learning to play chess. Master's thesis, Imperial College London, Retrieved from https://arxiv.org/pdf/1509.01549.pdf.

[11] David OE, Netanyahu NS, Wolf L. (2016) DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess. In Villa AEP, Masulli P, Pons Rivero AJ eds Artificial Neural Networks and Machine Learning – ICANN 2016. Cham, Springer International Publishing. pp 88–96. doi: 10.1007/978-3-319-44781-0_11.Retrieved from http://link.springer.com/10.1007/978-3-319-44781-0_11.

[12] Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y, Lillicrap T, Hui F, Sifre L, van den Driessche G, Graepel T, Hassabis D. (2017) Mastering the game of Go without human knowledge. Nature, 550 (7676): 354–359. doi: 10.1038/nature24270.

[13] Moravčík M, Schmid M, Burch N, Lisỳ V, Morrill D, Bard N, Davis T, Waugh K, Johanson M, Bowling M. (2017) DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. Science, 356 (6337): 508. doi: 10.1126/science.aam6960.

[14] Littman ML. (1994) Markov games as a framework for multi-agent reinforcement learning. In Machine learning proceedings 1994. Elsevier. pp 157–163.Retrieved from https://doi.org/10.1016/b978-1-55860-335-6.50027-1.

[15] Somasundaram TS, Panneerselvam K, Bhuthapuri T, Mahadevan H, Jose A. (2018) Double Q–learning Agent for Othello Board Game. In Proceedings of 10th International Conference on Advanced Computing (ICoAC). IEEE. pp 216–223. doi: 10.1109/ICoAC44903.2018.8939117.

[16] QIU H, WANG Y, GAO F, Quan QIU. (2013) Research on search engine techniques of Surakarta game. The Journal of China Universities of Posts and Telecommunications, 20 117–120. doi: 10.1016/S1005-8885(13)60235-6.

[17] Liu C, Wang J, Zhang Y. (2014) Search strategy research and analysis for surakarta chess game. In Proceedings of The 26th Chinese Control and Decision Conference (2014 CCDC). IEEE. pp 3362–3366.

[18] Winands MH. (2015) The Surakarta bot revealed. In Computer Games. Springer. pp 71–82.

[19] Zuo G, Wu C. (2016) A heuristic Monte Carlo tree search method for surakarta chess. In Proceedings of Chinese Control and Decision Conference (CCDC). IEEE. pp 5515–5518. doi: 10.1109/CCDC.2016.7531982.

[20] Li S, Qi Y, Bo J, Fu Y. (2019) Design and Implementation of Surakarta Game System Based on Reinforcement Learning. In Proceedings of Chinese Control and Decision Conference (CCDC). IEEE. pp 6326–6329. doi: 10.1109/CCDC.2019.8832340.

[21] Draskovic D, Brzakovic M, Nikolic B. (2019) A comparison of machine leaming methods using a two player board game. In Proceedings of IEEE EUROCON 2019-18th International Conference on Smart Technologies. IEEE. pp 1–5. doi: 10.1109/EUROCON.2019.8861927.

[22] Young K, Vasan G, Hayward R. (2016) Neurohex: A deep q-learning hex agent. In Computer Games. Springer. pp 3–18.Retrieved from https://doi.org/10.1007/978-3-319-57969-6_1.

[23] Arvidsson O, Wallgren L. (2010) Q-Learning for a Simple Board Game. PhD Thesis, Bachelor's Thesis, School of Computer Science and Engineering, KTH Royal Institute of Technology, Retrieved from http://www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/rapport/arvidsson_oskar_OCH_wallgren_linus_K10047.pdf.

[24] Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M, Freitas N. (2016) Dueling network architectures for deep reinforcement learning. In Proceedings of International conference on machine learning. PMLR. pp 1995–2003.Retrieved from http://proceedings.mlr.press/v48/wangf16.html.

[25] Smith SL, Kindermans P-J, Ying C, Le QV. (2018) Don't Decay the Learning Rate, Increase the Batch Size. In 6th International Conference on Learning Representations: 30 April - 3 May 2018; Vancouver, BC, Canada. Retrieved from https://openreview.net/forum?id=B1Yy1BxCZ.

[26] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S. (2019) PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach H, Larochelle H, Beygelzimer A, Alché-Buc F d\textquotesingle, Fox E, Garnett R eds Proceedings of Advances in Neural Information Processing Systems 32. Curran Associates, Inc. pp 8026–8037.Retrieved from http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[27] Rajagede RA, Mahardhika GP. (2021) Gapoera: Application Programming Interface for AI Environment of Indonesian Board Game. In arXiv. Retrieved from https://arxiv.org/abs/2110.11924.

[28] Kingma DP. (2015) Adam: A Method for Stochastic Optimization. In 3rd International Conference on Learning Representations; San Diego, CA, USA. Retrieved from http://arxiv.org/abs/1412.6980.

[29] Glorot X, Bengio Y. (2010) Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the 13th international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings. pp 249–256.

[30] He K, Zhang X, Ren S, Sun J. (2015) Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision. pp 1026–1034.