Enhancing MCTS with Convolutional Autoencoder and Linear Approximator in XCOM-Inspired Environments

Yiwei Zhang [1,2] and Richard Zhao [2]

[1] *Beijing Normal University, Beijing, China.*
[2] *University of Calgary, Calgary, Alberta, Canada.*

Abstract

Monte Carlo Tree Search is a widely acknowledged algorithm with video games being a common application. While MCTS has seen various enhancements, its integration with temporal difference learning (TD learning) in complex search environments still has interesting research potential. This paper introduces a novel method, Conv-AE+FA, merging convolutional autoencoders with a linear function approximator, to optimize game states. Applied to a game setting inspired by the commercial game XCOM, our study not only delves into the use of TD learning for guiding MCTS but also underscores the improved guidance in the MCTS search achieved by the use of convolutional autoencoders with linear approximation.

Keywords

MCTS, TD Learning, Convolutional Autoencoder, Linear Function Approximation

1. Introduction

Monte Carlo Tree Search (MCTS) has gained extensive recognition in the realm of artificial intelligence (AI) for its proficiency as an efficient, any-time search algorithm. Its applications stretch across numerous sectors, with video games standing out as one of the most prominent areas of usage. Over the years, the academic and scientific community has proposed a plethora of advancements to enhance the efficacy of MCTS, including the proposition to combine the benefits of MCTS with reinforcement learning, paving the way for a more robust, adaptable, and intelligent search mechanism.

Continuing the pursuit of refining MCTS, various improvements have been proposed [1]. Temporal difference (TD) learning has emerged as a noteworthy strategy to guide the search process. However, as the game environment becomes progressively more complicated, the utility of TD learning begins to wane. This paper ventures into an innovative approach, deploying convolutional autoencoders in conjunction with a linear function approximator to generalize game states in TD learning, thus potentially overcoming its limitations in complex settings. We apply this novel approach to a game environment inspired by the commercial turn-based strategy game XCOM. XCOM provides an easy-to-define and versatile environment for comparing AI techniques, and improvements on its AI can be transferred to other domains of a similar nature.

This paper presents the following contributions:

1. An exploration of TD learning as a partial guidance to MCTS, in an environment inspired by a commercial game.
2. A demonstration of the effectiveness of combining a convolutional autoencoder and a linear function approximator in enhancing the guidance in the MCTS search process.

AIIDE Workshop on Experimental Artificial Intelligence in Games (EXAG), October 08, 2023, University of Utah, Salt Lake City, Utah, USA.

EMAIL: yiweizh@mail.bnu.edu.cn (Yiwei Zhang); richard.zhao1@ucalgary.ca (Richard Zhao).
ORCID: 0000-0001-8257-4291 (Richard Zhao).

Copyright © 2023 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

2. Related Works

The application of AI techniques to gaming can be seen through many different methods, including search, planning, and learning strategies. One such method, MCTS, has been regularly investigated within the realm of AI game playing. Coulom [3] pioneered the integration of Monte Carlo evaluation with tree search, culminating in the development of the MCTS algorithm. The Upper Confidence Bounds Applied to Trees (UCT) algorithm was introduced by Kocsis and Szepesvári [6], serving as a variant of the MCTS algorithm. This variant employs a selection policy rooted in the Upper Confidence Bound (UCB) formula to strike an optimal balance between exploration and exploitation within the search tree. The UCT algorithm has demonstrated significant success across diverse domains, inclusive of game playing, planning, and optimization problems [14]. In the context of narrow tactical lines in sudden-death games such as Lines of Action (LOA), an MCTS offshoot, MCTS Solver, has been developed [17]. Research has also been conducted into the effective application of MCTS in classic and modern board games as well as video games [2].

In reinforcement learning (RL), TD learning represents a frequently employed method for modifying the value function estimation associated with a state or action, predicated on the perceived reward and the projected value of the succeeding state or action. TD-Gammon famously applied TD learning to the game of Backgammon [15]. The integration of TD learning with function approximation techniques is a common practice in estimating value functions for expansive state or action spaces [14].

Combining MCTS and RL can produce powerful algorithms. AlphaGo inspired many approaches in this area [13]. One strategy involves employing RL to acquire a heuristic evaluation function that directs MCTS toward the more fruitful sections of the game tree. Alternatively, RL can be used to ascertain the value of nodes within the tree, a technique that facilitates the pruning of the tree and focuses the search on more advantageous branches [16].These methodologies have found successful application in a number of games [3].

More recently, Saadat and Zhao [10] presented the online MCST-TD algorithm, combining TD learning with MCTS by using TD learning to learn values of states online, and concurrently, using current estimated state values to provide partial guidance to MCTS to enable MCTS to adapt to specific opponent strategies. While the authors showed that this adaptation can happen quickly on a small test environment of a 6 by 6 game board, its effectives was unclear on larger environments.

Autoencoders are no strangers to game AI research. Jain et al. [5] used autoencoders in game content generation, recognition and repair. Sarkar et al. [11] used variational autoencoders to generate levels in the style of existing games and blending levels across different games. Mak et al. [8] used autoencoders in aiding game design by generating game maps and avatars. Seth et al. [12] deployed adversarial autoencoders to identify players with irresponsible behaviors. Our work aims to explore the use of autoencoders, specifically convolutional autoencoders, in providing a better state representation for TD learning.

3. Descriptions of XCOM and XCOM-Inspired Environments

XCOM is a series of renowned turn-based tactical decision-making video games. In 1994, MicroProse released the first version under the initial title "UFO: Enemy Unknown." Over the years, many sequels have been released, including "XCOM: Enemy Unknown" in 2012, "XCOM 2" in 2016, and "XCOM: Chimera Squad" in 2020. Players assume the role of a commander of an international organization, defending against alien invasions and battling alien troops. This commercially successful game series has garnered high acclaim due to its interactive narratives, impeccable combat scenarios, and emergent strategic gameplay. Beyond entertainment, XCOM also serves as a context for academic research. Given its emphasis on positional play strategy and tactics, XCOM is an excellent platform to study how adaptive AI decision-making systems evaluate the game state and select actions in complex and uncertain conditions.

Past researchers [10] tested their work on miniXCOM, a simplified environment inspired by XCOM. In this work, we recreated the miniXCOM environment (Figure 1), but also a much larger environment that is roughly equivalent in complexity to an actual map in an XCOM game (Figure 2). We called these XCOM-Inspired Environments. The large board has a dimension of 18 by 18, with 4 squad members on each side. The layout of this board is inspired by the actual first level of XCOM2, where the playable area is roughly 20 by 18. We modified the area to be symmetrical on both sides to provide a fair testing environment.

Each map is represented by a grid and has walls that can be viewed as barriers or covers. We use a two-dimensional array to represent the board. Each grid block is assigned a specific value according to its state. For an empty block, it is 0. A wall block is -2. The block value is 1 when it is occupied by a human squad member. Conversely, the block is -1 when it has an alien squad member. With these four different values, the 2D array is able to represent all possible states on the game board. Moreover, there are two AI agents: the human squad and the alien squad. Each squad has several members. They have three different types of actions, which are controlled by various AI systems. Squad members are able to *move* on the board, *shoot* the uncovered enemy and execute *move and shoot* in a sequence if they can find an uncovered enemy after choosing a move destination. In each turn, agent can only issue one command to one member in its squad.

- Move action: Every squad member has four directions to choose: Up, Down, Left, and Right. Moving one block in a certain direction costs one step. A parameter called *Max-Move* stipulates the maximum moving steps in each turn. Humans and aliens cannot move onto the wall. For the mini board, Max-Move is 4. For the large board, Max-Move is 6.
- Shoot action: In each turn, every squad member can shoot at most one opponent and kill it immediately as long as there are no walls between them. From a mathematical perspective, a line is drawn between the squad member and target opponent. If this line does not intersect any walls, the enemy is uncovered and the squad member can shoot it successfully.

These two agents continually make decisions and execute actions until all opposing squad members are eliminated, enabling them to win the game. The simulated XCOM game provides an ideal scenario for conducting our experiments and comparing various decision-making algorithms.

4. Methodology

This section explains the different techniques involved in the current research, and how they each contribute to the combined method of *Conv-AE+FA*, the proposed novel method at playing the XCOM-Inspired Environments.

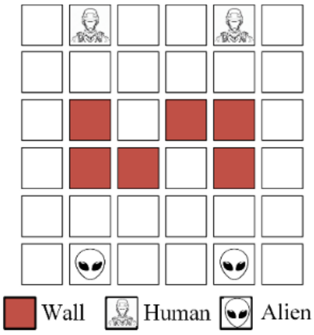

Figure 1: Mini game board setup [10].

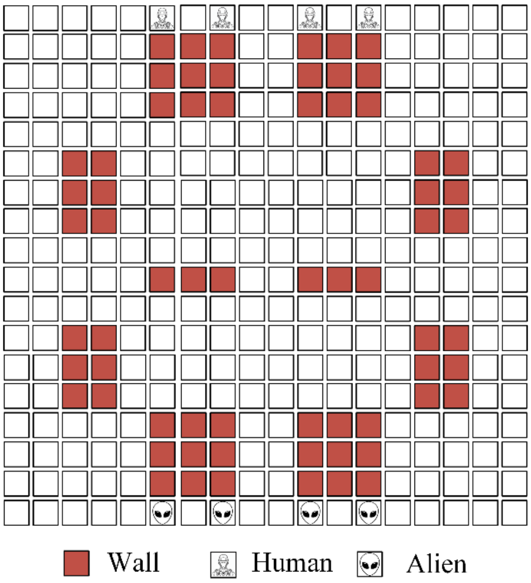

Figure 2: Large game board setup we used in our experiments, adapted from the first map of XCOM2.

4.1. MCTS and UCT

MCTS is a heuristic tree search algorithm designed to identify the optimal move by iteratively constructing a game tree, utilizing rollouts evaluation and selective search. This method involves simulating numerous game terminations from a designated state, recording the outcomes, and subsequently exploring the game tree to enhance the decision-making mechanism. This process encompasses four iterative phases: Selection, Expansion, Simulation, and Backpropagation.

1. Selection: According to a specific selection rule, the algorithm chooses the best child as expansion node until the child node is unexplored or it is a terminal state.

2. Expansion: the chosen node is added in the game tree if it has some unexplored actions.
3. Simulation: This step starts from the expanded node. For each rollout, it repeats to play a move according to a fixed rollout policy until terminal state and records the result.
4. Backpropagation: Back up process changes the attributes of the selected simulation node based on the result, such as increasing the number of visits and updating the node value estimate.

UCT is a type of selection rule in MCTS. It helps MCTS manage the trade-off between the exploration of new nodes and exploitation of known paths. Formula (1) shows how to calculate the UCT value.

$$UCT = \frac{Q(v')}{N(v')} + C\sqrt{\frac{2\,ln\,N(v)}{N(v')}} \qquad (1)$$

v' is one of the children of the node v. $Q(v)$ is the total reward of node v. $N(v)$ is the number of being visited of node v. C is a constant number to control the rate of exploration. The node with largest UCT value will be selected to expand in the next step.

In general, with the UCT selection policy, MCTS can concentrate on promising parts of the game tree and has an overview of search horizon compared to some other search methods based on alpha-beta search.

4.2. Convolutional Autoencoder

Convolutional neural network (CNN) [7] is a deep, feed-forward artificial neural network effective for processing grid-like data, such as pattern recognition and feature learning for high-dimension images. There are three major layers in a classical CNN architecture, which imitates the structure of human brains to extract spatial features from input data: convolutional layer, pooling layer, and fully connected layer.

Convolutional layers encompass numerous small learnable filters, interchangeably referred to as kernels. The filters traverse the image in alignment with a predefined stride and a two-dimensional feature map emerges, encapsulating abstracted features derived from the primary input data. The purpose of the pooling layer is to reduce spatial dimensions, thereby mitigating the risk of overfitting while reinforcing the extracted features. The fully connected layer yields a linear output, transforming the multidimensional features derived from preceding layers into an n-by-1 vector. In our work, we use this output as a representation of our game states.

An autoencoder [4] is an unsupervised learning algorithm designed for many tasks such as feature extraction and data generation. This algorithm compresses input data and reconstructs them. An autoencoder approach fundamentally consists of two primary components, an encoder and a decoder. The encoder and the decoder are learnable artificial neural networks. Figure 3 shows an autoencoder.

Encoder $f_\theta : \mathbb{R}^m \rightarrow \mathbb{R}^k$ maps input X into compressed representation Z.

Decoder $g_\phi : \mathbb{R}^k \rightarrow \mathbb{R}^m$ maps representation Z into reconstruction $\hat{X}$. (Usually, $k \ll m$.)

We utilize certain objective functions, like mean square loss, to update the encoder and decoder. Formula (2) implies that the ideal encoder and decoder should be the pair that reduces the discrepancy between the input data and the reconstructed data to the minimum.

The optimization formula:

$$\theta_{enc}, \phi_{dec} = \underset{\Theta,\Phi}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} \left\| x_i - g_\phi\big(f_\theta(x)\big)_i \right\|_2^2 \qquad (2)$$

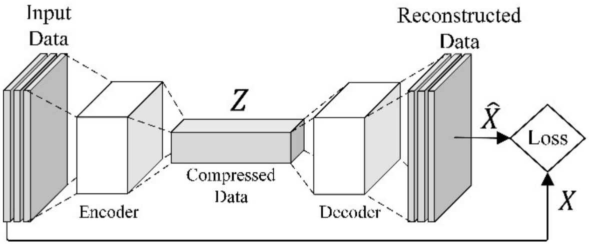

Figure 3: Structure of an autoencoder.

Upon completion of training, when feeding a real game state x into the encoder, it yields a compressed data output known as the feature vector, which is beneficial for our downstream tasks.

A convolutional autoencoder [9] combines the strengths of CNNs and autoencoders to learn hierarchical representations of input data. It leverages the power of extracting features from images and the capability to train an image feature extractor without labels. In our XCOM-Inspired Environments, the game board can be viewed as an image composed of four different pixel types so it is a suitable environment to utilize a CNN for analyzing the game board image. The logic of a convolutional autoencoder mirrors that of the standard autoencoder: making the reconstruction data $\hat{X}$ closely resemble to original input X after passing through entire autoencoder architecture.

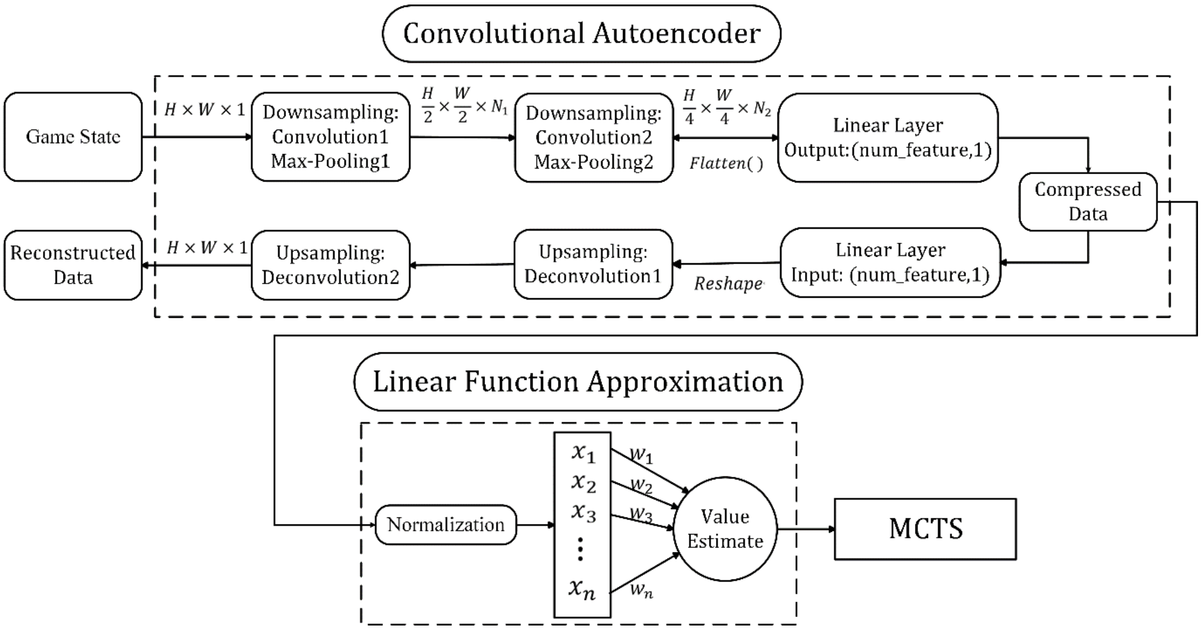
Convolutional Autoencoder
Game State
$H \times W \times 1$
Downsampling: Convolution1 Max-Pooling1
$\frac{H}{2} \times \frac{W}{2} \times N_1$
Downsampling: Convolution2 Max-Pooling2
$\frac{H}{4} \times \frac{W}{4} \times N_2$
Flatten( )
Linear Layer Output:(num_feature,1)
Compressed Data
Linear Layer Input: (num_feature,1)
Reshape
Upsampling: Deconvolution1
Upsampling: Deconvolution2
$H \times W \times 1$
Reconstructed Data
Linear Function Approximation
Normalization
x_1
x_2
x_3
x_n
w_1
w_2
w_3
w_n
Value Estimate
MCTS

Figure 4: Structure of Conv-AE+FA: Convolutional Autoencoder and Linear Function Approximator

The major distinction from a standard autoencoder lies on the application of CNNs as the mapping function f_θ for the encoder and transposed CNNs as the function g_ϕ for the decoder. We show the details of the encoder and decoder in Tables 1 and 2.

Table 1
Details of Convolutional Autoencoder-Mini. Other parameters: $N_1 = 16$, $N_2 = 8$, num_feature $= 6$.

Layer	Type	Kernel	Stride	Padding	Output-Padding
1	Conv1	3	1	1	NA
2	Max-Pool1	2	2	0	NA
3	Conv2	3	1	1	NA
4	Max-Pool2	2	1	0	NA
5	TransConv1	3	2	1	0
6	TransConv2	3	2	1	1

Table 2
Details of Convolutional Autoencoder-Large. Other parameters: $N_1 = 8$, $N_2 = 16$, num_feature $= 10$.

Layer	Type	Kernel	Stride	Padding
1	Conv1	3	1	1
2	Max-Pool1	2	2	0
3	Conv2	3	1	1
4	Max-Pool2	2	2	0
5	TransConv1	2	2	0
6	TransConv2	2	2	0
7	TransConv3	3	1	0

4.3. Linear Function Approximation

TD learning aims to get an optimal policy and guide the AI agent by learning the value function. When using function approximation for the value function, the approximator provide a value estimate for each state. We use the state values determined by the function approximator to assist in selecting the best node in the MCTS algorithm. Function approximation is often used to handle reinforcement learning problems with large or continuous states, as tabular methods are impractical to assess the value of each state. We used a linear function approximator so that TD learning can still retain the property of online real-time learning.

The diagram showing the entire process used by Conv-AE+FA is illustrated in Figure 4. We re-scale the features extracted by autoencoder through Z-score normalization. Linear function approximation represents the value function with a linear combination of weights and transformed features. Instead of updating a separate value estimate for each state as in tabular TD learning, the agent learns a weight vector. Formula (3) shows that when we feed the game state matrix into the encoder of the trained convolutional autoencoder, the compressed data is viewed as the feature vector representing the state.

$$\begin{aligned} f_{encoder}(State) &= \boldsymbol{x}(S) \\ &= [x_1(S) \quad x_2(S) \quad \cdots \quad x_n(S)] \end{aligned} \qquad (3)$$

The estimated value of a state is given by the dot product of weights and the feature vector.

The TD learning method updates the parameter vector based on the observed TD error, which shows in the Formula (4).

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\big(R + \gamma\, \hat{v}(S', \boldsymbol{w}) - \hat{v}(S, \boldsymbol{w})\big)\boldsymbol{x}(S) \quad (4)$$

S is current state and S' is the next state. R is the reward for state. α is the learning rate and γ is the discount factor. The weight vector is updated to minimize this TD error, effectively learning a value function that can generalize from seen states to unseen states based on their features.

After getting the estimate of the game states, we add this value into the UCT formula used by MCTS as in Formula (5).

$$\frac{Q(v')}{N(v')} + C\sqrt{\frac{2\ln N(v)}{N(v')}} + \hat{v}(S, \boldsymbol{w}) \quad (5)$$

The selection of nodes in MCTS is partially guided by the added term of $\hat{v}(S, \boldsymbol{w})$, which represents state information.

5. Experiments

We conducted experiments on both the 6×6 mini board and our larger 18×18 board in the XCOM-Inspired Environments. The 6×6 mini board has two squad members on each side where as the 18×18 board has four squad members on each side, resulting in a much larger state space.

5.1. Dataset and Training

To train the convolutional autoencoder, a large dataset is required. All training samples should simulate real game states so that the autoencoder can learn useful features. We created two datasets for the two board sizes. The mini-dataset has 10,000 samples on the 6×6 mini board and the large-dataset has 60,000 samples on the 18×18 large board. Considering that actual game states include varying numbers of humans and aliens, we generate all possible scenarios. Humans and aliens are all randomly generated on the game board while ensuring they cannot attack each other. Tables 3 and 4 illustrate the distribution of volumes for different combinations of humans and aliens.

We used the Adam optimizer with learning rate 0.001. After 10 epochs, the loss function converged and we saved the parameters of the encoder for use in the MCTS process.

Table 3
A total of 10,000 mini boards are generated randomly with the following distribution of numbers of humans and aliens.

	Alien: 1	2
Human: 1	1500	2000
2	2000	4500

Table 4
A total of 60,000 large boards are generated randomly with the following distribution of numbers of humans and aliens.

	Alien: 1	2	3	4
Human: 1	3500	2500	2000	1500
2	5500	5500	3500	3500
3	3500	3500	4500	4500
4	3500	3500	3500	6000

5.2. Experiment Details

We conducted six experiments, three on the mini board and three on the large board. We compared our proposed approach, MCTS with Conv-AE and linear function approximation (Conv-AE+FA) with other methods. The opponents are, respectively: MCTS (baseline), MCTS-TD, and MCTS with manually extracted features and linear function approximation (MF+FA). We set a fixed limit on the number of iterations the algorithm can perform and stop the search once this limit is reached. The limit is 100 in our experiments.

For the approach with manually extracted features, we would like to measure the effectiveness of feature extraction from Conv-AE against features chosen manually based on human knowledge. We hypothesize that there may be many drawbacks with manually choosing features. Despite human knowledge, it is difficult to determine whether the chosen features are sufficient for the problem. Based on the expertise of a player of the XCOM games, we created five features: the number of the humans and aliens, the number of possible actions and the number of rows occupied by humans and aliens. These features together present the distribution of humans and aliens.

In each experiment, a single run comprises a total of 40 game rounds. We grouped the results after every 10 rounds, partitioning each run into four segments. For the mini board, we conducted 20 runs of 40 rounds and averaged the results. For the large board, we conducted 10 runs. In the interest of fairness, humans and aliens alternate in initiating the first action. During each run, humans take the first move in 20 rounds, while in the remaining rounds, the aliens act first. A game round concludes in a draw if neither squad achieves victory following 25 moves. For function approximation, the weight vectors are randomly initialized between 0 and 1 at the beginning of each run. The details of hyperparameters are in Table 5. Shooting an enemy produces a reward of 10 to the agent and a reward of -10 to the opponent. The reward for the action to move is 0.

Table 5

Parameter values used in the experiments.

Parameters	Mini board	Large board
Learning rate	0.1	0.1
Exploration factor	$1/\sqrt{2}$	$1/\sqrt{2}$
Number of features (generated by Conv-AE)	6	10
Number of features (designed manually)	5	5

5.3. Mini Board Results

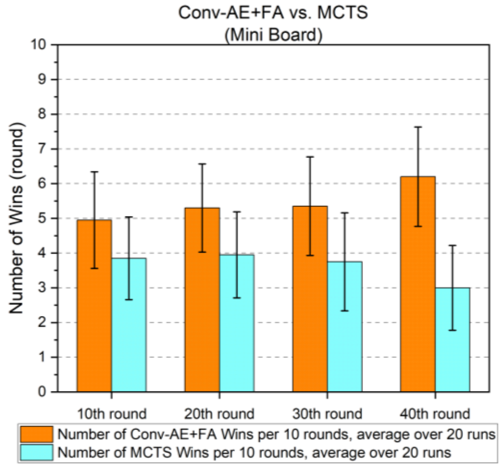

Figure 5: Conv-AE+FA compared to MCTS on mini board.

Figure 5 shows the results of Conv-AE+FA vs. MCTS on the mini board. The error bars represent 1 standard deviation. Conv-AE+FA held an advantage over MCTS in every 10 rounds (statistically significant at 95% confidence at 20 rounds and beyond, using paired two-tailed t-tests). As the weight vector updates, Conv-AE+FA adapts to the opponent and becomes stronger.

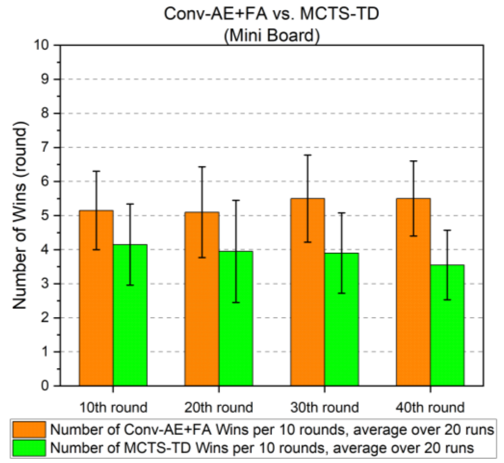

Figure 6: Conv-AE+FA compared to MCTS-TD on mini board.

Figure 6 shows the results of Conv-AE+FA vs. MCTS-TD. Since MCTS-TD is also an adaptive algorithm, it performed better than MCTS against Conv-AE+FA. Conv-AE+FA provides state information of the entire game board while in past work [10], MCTS-TD only represented the state by a 3 by 3 grid centered around the current location of an agent (to reduce state space). Conv-AE+FA outperformed MCTS-TD and the results are statistically significant at 95% confidence at 30 rounds and beyond.

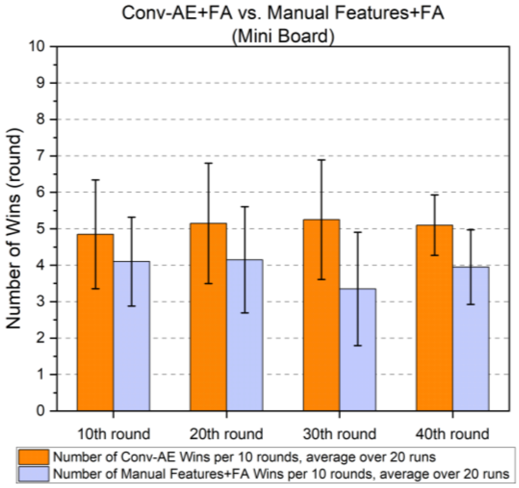

Figure 7: Conv-AE+FA compared to MF+FA on mini board.

Figure 7 shows the results of Conv-AE+FA vs. MF+FA (Manual Features and Function Approximation). Although manual features also contain state information of the entire game board, it remains uncertain whether the information is diverse and beneficial for our decision-making process. Consequently, Conv-AE+FA demonstrated superior performance over MF+FA and the results are statistically significant at 95% confidence at 30 rounds and beyond.

5.4. Large Board Results

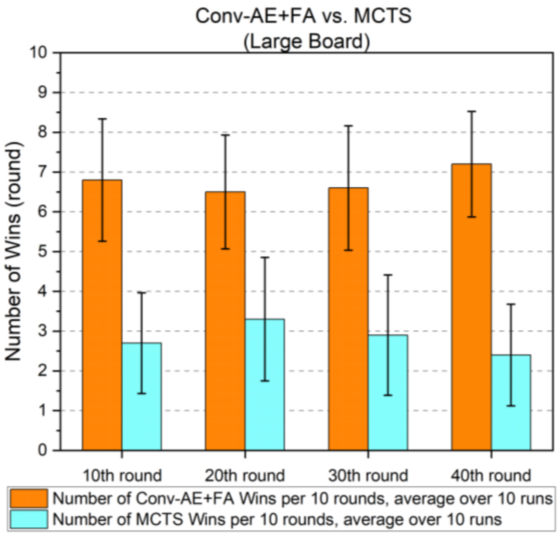

Figure 8: Conv-AE+FA compared to MCTS on the large board.

Figure 8 shows the results of Conv-AE+FA vs. MCTS on the large board. Conv-AE+FA holds a larger advantage over MCTS in every 10 rounds. The winning rate is higher compared to the mini board. This show that in a more complex environment, CNNs improve the ability to analyze the game state, extract hierarchical features and provide enhanced guidance to MCTS.

Figure 9 shows the results of Conv-AE+FA vs. MCTS-TD on the large board. Compared to the experiments on the mini board, our approach has an obvious advantage over MCTS-TD because linear function approximation is more scalable to high-dimensional state spaces. It generalizes across states based on their features, which can be a more efficient representation. MCTS-TD with a 3-by-3 local grid representation cannot obtain important information of current state beyond its immediate neighbors.

Figure 10 shows the results of Conv-AE+FA vs. MF+FA (Manual Features and Function Approximation) on the large board. Compared to the experiments on mini board, our approach has an obvious advantage. Convolutional autoencoders are capable of learning a hierarchy of features due to their deep, layered structure. Conv-AE automatically learns to extract features that are useful for reconstruction, which can also be useful for other tasks. Table 6 shows the combined results for a clearer comparison. All results shown in Table 6 are statistically significant using paired two-tailed t-tests at 95% confidence level.

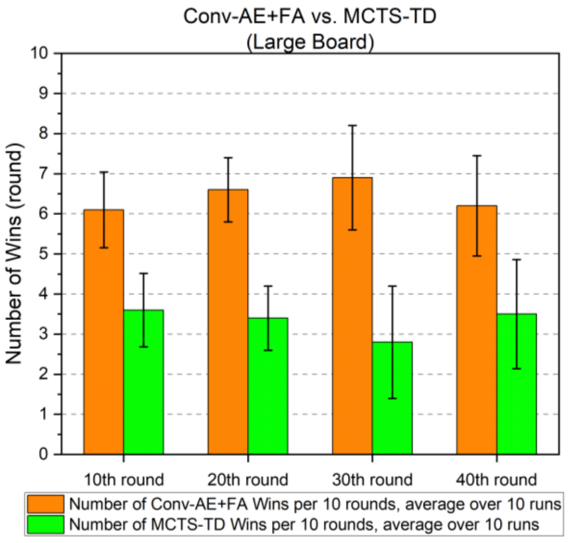

Figure 9: Conv-AE+FA compared to MCTS-TD on the large board.

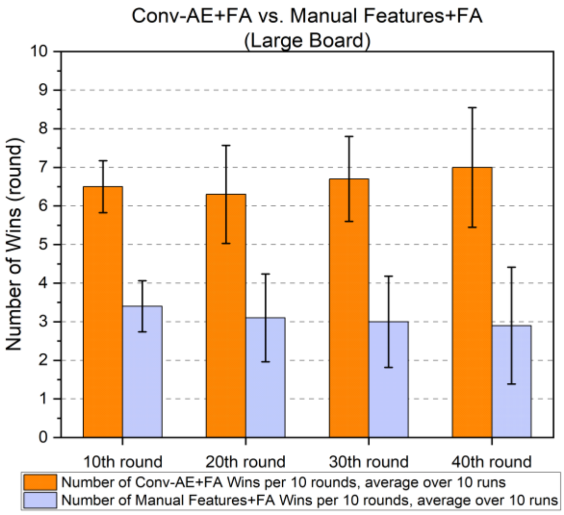

Figure 10: Conv-AE+FA compared to MF+FA on the large board.

5.5. Discussions

According to the description of the experiments, the human squad is always the proposed algorithm while alien squad has three different techniques. To provide contrast for the effects of these three techniques, we include a ratio comparison graph. As shown in Formula (6), the ratio for each technique is determined by dividing the winning number of the alien squad, indicative of the efficacy of the compared technique, by the winning number of the human squad representing the performance of our proposed Conv-AE+FA technique.

$$ratio = \frac{The\ number\ of\ Alien\ Wins}{The\ number\ of\ Human\ Wins} = \frac{The\ effect\ of\ previous\ technique}{The\ effect\ of\ proposed\ technique} \quad (6)$$

When the ratio is less than 1, the human squad wins more than the alien squad. It signifies that Conv-AE+FA is stronger than the previous technique. In contrast, a ratio exceeding 1 suggests that our algorithm is underperforming. When comparing various techniques, a higher ratio indicates that particular technique performs better when confronted by Conv-AE+FA.

According to Figure 11, it is not surprised that Conv-AE provided a crucial role in extracting useful features to be used by TD learning, which then provided guidance to MCTS in its search. In fact, the advantages of Conv-AE+FA becomes more apparent on the large board, a more complex environment where strategies become more important.

Table 6

Combined results of large board for all rounds (standard deviations in brackets). Some rounds ended in a draw due to neither side winning at time out.

Winning rate	MCTS	MCTS-TD	MF+FA	Conv-AE+FA
Conv-AE+FA vs. MCTS	28.25% (6.99%)			67.75% (8.40%)
Conv-AE+FA vs. MCTS-TD		33.25% (6.62%)		64.50% (6.40%)
Conv-AE+FA vs. MF+FA			31.00% (5.72%)	66.25% (5.15%)

In each board, MCTS-TD and MF+FA both performed better than MCTS. In mini board, MF+FA was better than MCTS-TD. However, a notable observation is that on the large board, MCTS-TD performed well compared to MF+FA, even though MCTS-TD relied solely on local state information, whereas MF+FA used manual features derived from the entire state space. This suggests that human expert input does not always enhance the learning process and can, at times, be detrimental.

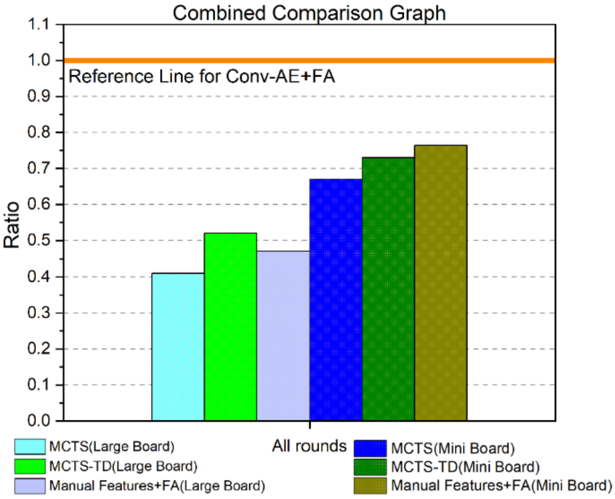

Figure 11: Combined comparison of the different techniques, using Conv-AE+FA as the benchmark value of 1.

6. Conclusions

In this research, we examine the effectiveness of two augmentations to MCTS with TD learning: a convolutional autoencoder to extract features of the game board, and linear function approximation to represent the game state in reinforcement learning. While MCTS partially guided by TD learning allows the algorithm to adapt to an opponent while the game is being played, we show that in a larger state space, a convolutional autoencoder is effective at extract features of the state compared to manually created features, and that combined with linear function approximation, this can bring statistically significant improvements in the results of MCTS.

This work is not without limitations. The experiments were conducted on specific maps. While we believe that the results are generalizable to different layouts of maps, this should be analyzed in further work. Furthermore, explainable AI is an important goal of AI research. Future research should examine in details on the features produced by the convolutional autoencoder to provide explanations on their effectiveness.

7. References

[1] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games 4.1 (2012): 1-43.

[2] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, Monte Carlo Tree Search: A New Framework for Game AI. Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 4.1 (2008): 216-217.

[3] R. Coulom, Efficient selectivity and backup operators in Monte-Carlo tree search. Proceedings of the 5th international conference on Computers and games (2006): 72-83.

[4] G. E. Hinton and R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks. Science 313.5786 (2006): 504–507.

[5] R. Jain, A. Isaksen, C. Holmgård, and J. Togelius, Autoencoders for level generation, repair, and recognition. Proceedings of the ICCC workshop on computational creativity and games (Vol. 9), 2016.

[6] L. Kocsis and C. Szepesvári, Bandit Based Monte-Carlo Planning. Machine Learning: ECML 2006, volume 4212 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2006.

[7] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, Backpropagation applied to handwritten zip code recognition. Neural computation, 1.4 (1989): 541-551.

[8] H. W. L. Mak, R. Han, and H. H. Yin, Application of variational autoencoder (VAE) model and image processing approaches in game design. Sensors 23.7 (2023): 3457.

[9] X. Mao, C. Shen, and Y. B. Yang, Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. Advances in neural information processing systems 29, 2016.

[10] K. Saadat and R. Zhao, Exploring Adaptive MCTS with TD Learning in miniXCOM. AIIDE Workshop on Experimental AI in Games (EXAG), 2022.

[11] A. Sarkar and S. Cooper, Generating and blending game levels via quality-diversity in the latent space of a variational autoencoder. Proceedings of the 16th International Conference on the Foundations of Digital Games (2021): 1-11.

[12] D. Seth, S. Eswaran, T. Mukherjee, and M. Sachdeva, A Deep Learning Framework for Ensuring Responsible Play in Skill-based Cash Gaming. 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA) (2020): 454-459.

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and S. Dieleman, Mastering the game of Go with deep neural networks and tree search. nature 529.7587 (2016): 484-489.

[14] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, Monte Carlo tree search: A review of recent modifications and applications. Artificial Intelligence Review 56.3 (2023): 2497-2562.

[15] G. Tesauro, Temporal difference learning and TD-Gammon. Communications of the ACM 38.3 (1995): 58-68.

[16] T. Vodopivec and B. Šter, Enhancing upper confidence bounds for trees with temporal difference values. 2014 IEEE conference on computational intelligence and games (2014): 1–8.

[17] M. H. M. Winands, Y. Björnsson, and J. T. Saito, Monte-Carlo Tree Search Solver. Computers and Games, volume 5131 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2008.