A multi-size convolution neural network for RTS games winner prediction

Jie Huang [a] and Weilong Yang

College of System Engineering, National University of Defense Technology, Changsha, China

Abstract. Researches of AI planning in Real-Time Strategy (RTS) games have been widely applied to human behavior modeling and combat simulation. Winner prediction is an important research area for AI planning, which ensures the decision accuracy. In this paper, we introduce an effective architecture -- multi-size convolution neural network (MSCNN)-- into winner prediction. It can capture more feature for game states, because of the various sizes of filters in MSCNN. Experiments show that the modified evaluating algorithm can effectively improve the accuracy of winner prediction for RTS games.

1 Introduction

Real-Time Strategy (RTS) games are popular real-time combat simulation games in which players instruct units to gather resources, build structures, destroy opponent's buildings to win the game. As typical agent-based game, RTS games pose a huge challenge for AI researchers due to the large state space, limited decision time and dynamic adversarial environment involved. AI planning becomes an important research area for real-time adversarial planning and non-determination decision [1].

Winner prediction is an important part of AI planning. As basic algorithm for player's planning and decision, winner prediction is similar to state evaluation. Current prediction function use evaluating algorithms by taking related factors into consideration to obtain a state score. This method is applicable to simple and fixed game scene. However, in RTS games, the evaluating factors are constantly changing, for which fixed evaluating algorithm cannot accurately describe the game state at different game phases. For example, the current evaluating function does not consider the spatial relationship between units, resulting in that units at different positions still are calculated by carried resources and hit-point value.

This paper focuses on the evaluating process in RTS games and constructs MSCNN based on game state. MSCNN can improve the utilization of computing resources inside the network, because of the various sizes of filters which can work in parallel. As a result, it achieves state-of-the-art performance on experiments.

The structure of this paper is shown as follows: Section II discusses related work. Section III shows the MSCNN architecture and the feature for game states. Section IV details our experiments and discussion. Section V concludes this work.

2. Related work

2.1 Winner prediction in RTS games

RTS games are regarded as a simplification of combat simulation and could therefore serve as a test bed for investigating activities such as real-time adversarial planning and decision making under uncertainty [2]. Research has been conducted to modify game tree search and Monte Carlo algorithm to RTS games planning [3][4][5]. Among these planning algorithms, evaluating algorithm is needed to calculate the current and play-out game state. By evaluating the play-out game state, we predict the final winner.

Alexander [6] first described evaluating process in RTS game state, proposing an evaluating function by calculating the hit-point and attack ability of units. LTD (Life-Time Damage) is a modified evaluating algorithm, which based on the lifetime damage each unit can inflict. And by weaken the contribution of hit-point, LTD2 algorithm is proposed [7]. Tung [8] combined the LTD method and the unit portfolio evaluation to evaluate game state, but did not consider the relationship between two algorithms.

Taking nonterminal position into consideration, Stanescu [9] modified the state evaluating algorithm by taking other properties into account, such as resources, visibility, security, and goals. By putting forward the concept of visibility and detection, a formula is given to calculate the efficiency of detection. Graham Erickson [10] proposed a global evaluating function model for predicting which side would win after a period of game play-out. Taking military features, economic and player skill into consideration, the logistic regression is used to learn the model weights. Alberto [11] used a contrastive

[a] Corresponding author: huangjie_gh@126.com

evaluating algorithm to obtain the score by calculating the difference between player and its opponent. Bakkes [12] generated evaluating function by using a central data store of samples of gameplay experiences.

Besides traditional evaluating function method with expert knowledge, learning method is also used to evaluate game state. Li [13] used a GA-based reinforcement learning method named ELM to calculate the optimal unit combination strategy, and building strategy. Marius [14] proposed the use of neural network method for evaluation, the spatial relationships between units were also considered in the category. Learning method can achieve good performance but need pre-learning and large dataset, and if the game parameters changes, it will be useless unless the dataset for changed parameters is provided.

2.2 Convolution neural network

Convolutional neural networks (CNNs) have been widely used in machine learning tasks, often demonstrating superior performance compared to traditional methods. Many of those applications focus on classification related tasks, e.g. on image recognition [15], on speech [16][17][18] and on machine translation [19][20].

CNNs are different from fully connect networks with three important ideas to improve its performance: sparse interaction, parameter sharing and equivariant representation. Sparse interaction contrasts with traditional fully connect neural networks where next layer neurons connect all the ones in above layer, and as a result every output neuron is interactive with every input neuron. In CNN, there is a filter which is usually much smaller than the input size to scan all the input so that the output is only influenced by a narrow window of the input in the scope of the filters each time. Parameter sharing refers to the filter parameters similar to the weight matrix elements of fully connect neural networks and are reused in the convolution operations, while the ones in fully connect neural network only used once to calculate the output. Equivariant representation refers to the idea of k-MaxPooling, which is added at the end of CNN to make sure that it can gets the best features.

By now, deep learning models were becoming extremely useful in categorizing the content of images and video frames. Most skeptics had accepted that deep learning was here to stay. Given the usefulness of these techniques, internet giants like Google were very interested in efficient and large deployments of architectures on their server farms. In February 2015 Batch-normalized Inception was introduced as Inception V2. In December 2015 they released a new version of the Inception modules and the corresponding architecture, which better explains the original GoogLeNet architecture, giving a lot more detail on the design choices. Then Christian and team proposed a new version of Inception. The Inception module after the stem is rather similar to Inception V3. Next, Xception improves on the inception module and architecture with a simple and more elegant architecture that is as effective as ResNet and Inception V4. We can see that crafting neural network architectures is of paramount importance for the progress of the deep learning field [21].

3 Multi-size convolution neural network for game state

3.1 Data

As Stanescu's experiment setting [14], our dataset is obtained base on the free-software μRTS (https://githubs.com/ santiontanon/ microrts), which has been used by several researchers to validate new algorithms for RTS games. Figure 1 shows a screenshot of a μRTS game, in which two players compete to destroy opponent's units.

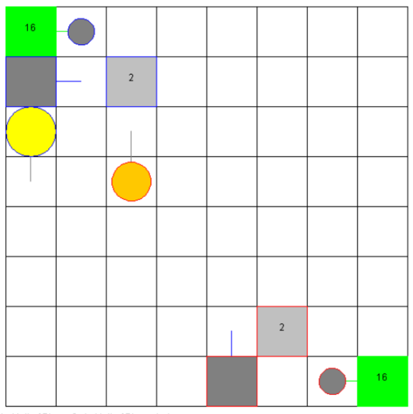

Figure 1: A screenshot of the μRTS game environment. The two players are distinguished according to the blue and red outline colors. The green squares are resources. The white squares are bases. The gray circles are workers, the yellow circles are heavy attackers, and the orange circles are light attackers.

We conduct a round-robin tournament between 9 different μRTS bots. Each algorithm plays 50 games against all other algorithms in maps 8 × 8 tiles. Each tournament consists of (9×8)/2 = 36 matches. In each tournament, 24 different initial starting conditions are used. All scenarios start with one base and one worker for each player, but with different, symmetric, initial positions. The time limits are as followed: maximums of 100ms, 200ms, 100 playouts and 200 playouts per search episode. In total 36×24×4 = 3456 different games were played. Among them, the games with result of are discarded. Because we have significantly less data for the learning algorithm, we chose to sample 3 random positions from each game. As a result, for game i we add {(s_{i1}, w_i), (s_{i2}, w_i), (s_{i3}, w_i)} to the dataset, and slightly over 10000 positions are generated. The dataset was split into a test set (2000 positions) and a training set (the remaining 8000 positions).

3.2 Feature

Each position s is pre-processed into a set of 8×8 feature planes. These features correspond to the raw board represen-tation and contain information about each tile of the μRTS type, current health points, game frames until actions are completed and resources. All integers, such as unit health points, are split into K different 8×8 planes of binary values using the one-hot encoding. The full set of feature planes is listed in Table 1.

Table 1. Input feature planes for the neural network

Feature	# of planes	Description
Unit type	6	Base, Barracks, worker, light, ranged, heavy
Unit health	2	1 or 2
Unit owner	2	1 or 2
Resources	7	1,2,3,4,5,6-9 or >=10

3.3 Network and training details

In the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14), GoogLeNet [22] achieves the new state-of-the-art performance for classification and detection. It is made up of several inception modules (IM) which staked upon each other. For IM, convolutional filters of varied size are employed to capture different abstraction features. IM is composed of four groups of convolution operations with filter sizes 1,3 and 5. Feature maps are produced by concatenating all convolutions outputs to form the input of the next stage. Since directly applying more convolutional filters and concatenating them are computationally expensive, 1*1 convolutional filters at the bottom of the module is used, which reduces the input dimensionality and hence decreases the computation cost dramatically. It also captures the correlated features in the same region, and image patterns are responded by 3*3 and 5*5 convolutions at larger scales. Because of the carefully crafted designing, this architecture increases the depth and width of the network while keeping the computational budget constant. In [22], the authors suggest that current incarnations of the inception module are restricted to filter sizes 1*1, 3*3 and 5*5 and it is based more on convenience rather than necessity.

Inspired by above IM, we propose our model-- multi-size convolution neural network -- to adapt to the winner prediction task (Fig. 2 and Fig. 3). The input to the neural network is an 8*8*4 image stack consisting of 4 feature planes. Four groups of convolutional layers with different sizes of filters are employed at the same time. Every convolutional layer pads the input with zeros to keep the size of outputs unchanged. 4 feature planes then are convolved with 64/32/1 respectively filters(filter) of size 1*1 with stride 1. They also are convolved with 64/96/96/32/1(48/64/32/1) filters(filter) of size 3*3(5*5). Both are followed by leaky rectified linear units (LReLUs). Meanwhile, last group includes a filter of size 1*1 with an average pooling layer before convolution, again followed by an LReLU. All outputs are concatenated to obtain an 8*8*4 image. Then follow two fully connected linear layers, with 256 and 64 LReLUs, respectively. The output layer is a fully connected layer with two units, and a softmax function is applied to obtain the winning probabilities for player 0 and player 1.

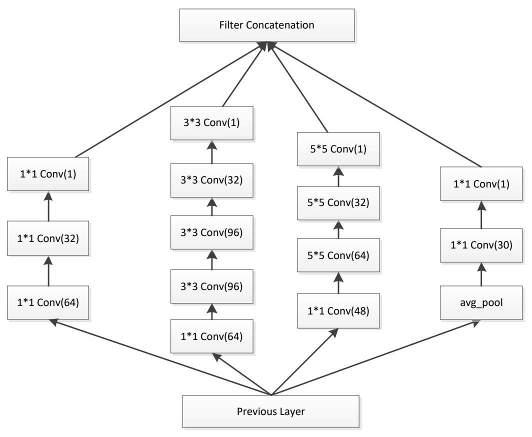

Figure 2. multi-size convolution neural network.

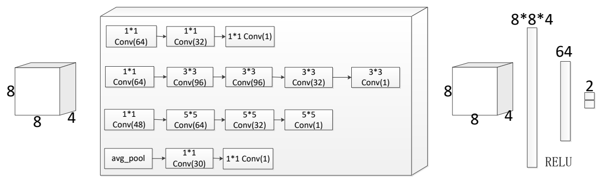

Figure 3. multi-size convolution neural network for RTS games winner prediction.

4 Experiments and results

In order to verify the proposed algorithm, an empirical study based on μRTS game is carried out, and the performance of DHEN is compared to the performances of other state-of-the-art evaluating algorithms developed for RTS games.

4.1 Experiment environment and setting

Setup. In our experiments, MSCNN is trained and tested using the machine learning library PyTorch. During training, the learning rate is 0.01. We employ Adagrad as optimizer with weight decay (0.0001). All convolutions have batch normalization (0.001).

Baseline. We compare with CNN in [14] and LSTM.

(1) CNN: An architecture in [14] employs two convolutional layers with 64 and respectively 32 filters of size 3*3.

(2) LSTM: An architecture combines two layers LSTM and a classifier layer.

4.2 Experimental results and analysis

Table 2 give the results of different systems for this datasets, respectively. Here we are going to study the effectiveness of MSCNN by the experimental results. Table 2 shows that MSCNN outperforms LSTM and CNN. This phenomenon can be explained by the unique architecture of MSCNN. Usually, the most straightforward way of improving the performance of deep neural networks is by increasing their size, including the depth and width. However, bigger size means a larger number of parameters and increased network size. A larger number of parameters make the enlarged network more prone to overfitting, and bigger network size increases use of computational resources. A fundamental way of solving both of these issues would be to the use of small filter sizes, pooling layers, and same-padded convolution (the same width and height, each layer padded with zeros following convolution). However, MSCNN perfectly fulfills our requirements for an efficient neural network by utilizing various sizes of convolution filters, which can work in parallel.

Table 2. Accuracy of different systems for winner prediction.

Idx	System	Accuracy
1	CNN	0.6394
2	LSTM	0.6352
3	MSCNN	0.6652

Figure 4 shows the convergence of the loss function during training. The red curve represents the loss value for each step, and the blue curve represents the loss fit curve obtained using locally weighted regression. It can be seen from the figure that after 600 steps, the loss is basically maintained at a lower value, and the training of the neural network also reaches a convergence state thereafter.

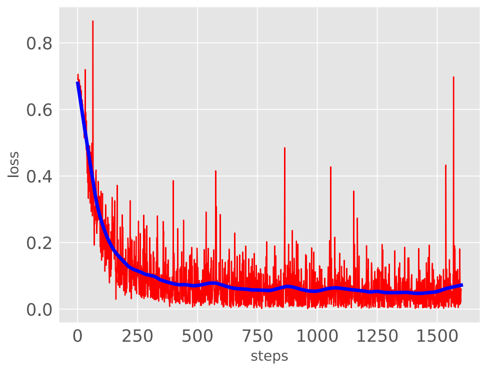

Figure 4. the loss value for each step and the loss fit curve.

5 Conclusions

In summary, we have performed both experimental and theoretical study of evaluating RTS game state. A multi-size convolution neural network is proposed to handle the evaluation problem accompany with the planning process. The experimental results in µRTS game successfully verify the algorithm's validation.

Next we will use the generative adversarial network (GAN) to test the prediction problem in multiple states. GAN was proposed by Ian Goodfellow in 2014, which greatly advanced the development process of unsupervised learning. GAN generates fake data by generative model, and judge authenticity by adversarial model. It is hoped that the data generated by the generative model can be the same as the real data.

Acknowledgment

The work described in this paper is sponsored by the National Natural Science Foundation of China under Grant No. 61473300.

References

1. M. Buro, *Call for AI research in RTS games*, In Proceedings of the AAAI-04 Workshop on Challenges in Game AI 2004, pp. 139--142, 2004.
2. G. Erickson and M. Buro, *Global state evaluation in StarCraft*, in Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, pp. 112-118, 2014.
3. S. Ontanon, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, *A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft*, Computational Intelligence & Ai in Games, **vol. 5,** pp. 293-311, 2013.
4. M. Chung, M. Buro, and J. Schaeffer, *Monte Carlo Planning in RTS Games*, in IEEE Symposium on Computational Intelligence and Games, pp. 117--124, 2005.
5. R. K. Balla and A. Fern, *UCT for tactical assault planning in real-time strategy games*, IJCAI 2009, Proceedings of the International Joint Conference on Artificial Intelligence, Pasadena, California, Usa, pp. 40-45, July, 2010.
6. D. Churchill, A. Saffidine, and M. Buro, *Fast Heuristic Search for RTS Game Combat Scenarios*, 2012.
7. A. Kovarsky and M. Buro, *Heuristic Search Applied to Abstract Combat Games*, Lecture Notes in Computer Science, **vol. 3501,** pp. 66-78, 2005.
8. M. Buro, *Adversarial hierarchical-task network planning for complex real-time games*, in International Conference on Artificial Intelligence, pp. 1652-1658, 2015.
9. D. N. Tung, Q. N. Kien, and T. Ruck, Heur*istic Search Exploiting Non-additive and Unit Properties for RTS-game Unit Micromanagement (Preprint)*, Journal of Information Processing, **vol. 23,** pp. 2-8, 2015.
10. M. Stanescu, S. P. Hernandez, G. Erickson, R. Greiner, and M. Buro, *Predicting army combat outcomes in StarCraft*, in AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, pp. 86-92, 2013.
11. A. Uriarte and S. Ontañón, *Game-tree search over high-level game states in RTS games*, in AIIDE, pp. 73-79, 2014.
12. S. Bakkes, P. Spronck, and J. V. D. Herik, *Phase-dependent evaluation in RTS games*, 2013.
13. Y. J. Li, P. H. F. Ng, and S. C. K. Shiu, *A fast evaluation method for RTS game strategy using fuzzy extreme learning machine*, Natural Computing, **vol. 15,** pp. 1-13, 2015.
14. M. Stanescu, N. A. Barriga, A. Hess, and M. Buro, *Evaluating real-time strategy game states using convolutional neural networks*, in Computational Intelligence and Games, 2017.

15. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, **vol. 86,** pp. 2278-2324, 1998.
16. H. Schwenk, Continuous space language models, *Computer Speech & Language*, **vol. 21,** pp. 492-518, 2007.
17. G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. R. Mohamed, N. Jaitly, et al., *Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups*, IEEE Signal Processing Magazine, **vol. 29,** pp. 82-97, 2012.
18. A. Graves, A. R. Mohamed, and G. Hinton, *Speech recognition with deep recurrent neural networks*, **vol. 38,** pp. 6645-6649, 2013.
19. J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, and J. Makhoul, *Fast and Robust Neural Network Joint Models for Statistical Machine Translation, in Meeting of the Association for Computational Linguistics*, pp. 1370-1380, 2014.
20. M. Sundermeyer, T. Alkhouli, J. Wuebker, and H. Ney, *Translation Modeling with Bidirectional Recurrent Neural Networks*, in Conference on Empirical Methods in Natural Language Processing, 2014.
21. https://www.topbots.com/a-brief-history-of-neural-network-architectures/
22. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, et al., *Going deeper with convolutions*, pp. 1-9, 2014.