Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Supplementary materials for

Xiali LI, Yanyin ZHANG, Licheng WU, Yandong CHEN, Junzhi YU, 2024. TibetanGoTinyNet: a lightweight U-Net style network for zero learning of Tibetan Go. *Front Inform Technol Electron Eng*, 25(7):924-937.
https://doi.org/10.1631/FITEE.2300493

1 Introduction

1.1 Policy learning is similar to image segmentation task

Both image segmentation and strategy learning produce a tensor of size $\{H, W, C\}$. In the image segmentation task, H and W are the height and width of the image, respectively, and C is the number of channels. Each pixel in the image corresponds to a vector, with the vector's length being equal to the number of segmentation categories (denoted by C). The pixel is assigned to the category with the highest value in the vector. In the Tibetan Go board policy learning, H and W are the height and width of the Tibetan Go board respectively, and the moves of the pieces on the Tibetan Go board are naturally mapped on each coordinate point. The action with the largest value is most likely to be executed. Therefore, the idea of image segmentation can be applied to the Tibetan Go board zero learning task. So we choose the well-established U-Net (Ronneberger et al., 2015) in the field of image segmentation as the foundation to study a lightweight network suitable for zero learning of Tibetan Go under limited computing resources.

1.2 Tibetan Go

Tibetan Go is the tournament game in national Tibetan chess competitions and grading tournaments in China. However, compared with Go and Chess, research on Tibetan Go is still in its infancy, the degree of digitization is very low, and the datasets required for deep reinforcement learning training and deep learning models with high performance are lacking.

1.3 The general game playing system

In the field of computer games, researchers use diverse and unstructured programming languages, techniques, and architectures, making it difficult to compare and communicate with each other. The general game playing (GGP) system serves to research artificial intelligence (AI) in games. The Ludii generalized game gystem (Piette et al., 2020) is even more general, scalable, understandable, and efficient. Programming Tibetan Go using the Ludii general-purpose game system language (Soemers et al., 2022) simplifies and symbolizes the complex logical elements of Tibetan Go. This approach can be extended to the programming of many other games, thus expanding the field of experimental research on zero learning in computer games, especially Tibetan Go board games.

2 AlphaGo family and its improvements

AlphaGo Zero effectively trains a Go AI from scratch using only the rules of the game because of the successful combination between Monte Carlo tree search (MCTS) and deep neural network.

ELF OpenGo (Tian et al., 2019) is an open-source implementation of AlphaGo Zero for the game of Go. KataGo (Wu, 2019) is an open-source implementation of AlphaGo Zero that facilitates learning in many ways. Polygames (Cazenave et al., 2020) employs the fully convolutional network (FCN) to maintain the

location information and obtain optimal performance in many forms of board games. A simple version of U-Net was provided by Polygames. It uses additive operations instead of concat operations when blending different levels of feature maps. TibtanGoTinyNet proposed in this work was inspired by the self-play and deep reinforcement learning without expert knowledge, and improvements to the policy–value network of the game were made using lightweight network technologies.

3 Preliminary

3.1 Ludii

Ludii is a general game system designed to play, evaluate, and design a wide spectrum of games, including board games, card games, dice games, and mathematical games.

Games are characterized as structured collections of ludemes (units of game-related information). This enables the complete range of traditional strategy games from around the world to be modeled in a single playable database for the first time. Most board games, if written in the Ludii platform language and implemented through the Ludii interface by Polygames deep reinforcement learning environment, allow the model to act on a wide range of board games (Soemers et al., 2021).

3.2 Difference in rules between Go and Tibetan Go

Capturing two or three cannot recapture one. When one player captures a piece, the other cannot immediately place a piece inside the opponent's eye (nakade). False Eye is live piece. There are also special rules different from Go with regard to counting the territory at the end of the game. If a player makes some alive groups and forms the shape of one of the eight Tibetan Auspicious Symbols (the Eight Auspicious Symbols are specific shapes known as Survana Matsya, Dhvaja, Chattra, Shankha, Padma, Kalasha, Shrivatsa, and Chakra), an additional eight points will be added to the player's occupying points. If a player occupies all four corners, the player will gain an additional 20 occupying points. If a player occupies not only all four corners but also Tengen, 25 more pieces will be added to the surrounding points. If a player has more than one alive group than their opponent, the difference between the alive groups will be subtracted from the total number of occupying points.

3.3 The 9×9 Tibetan Go can embody the characteristics of the 17×17 Tibetan Go

Many concepts such as "qi," "forbidden points," "ko," "eyes," and "life-and-death" are equally important in both versions. The difference is that the 17×17 board requires more attention to the overall situation, while the 9×9 board, as a simplified version, will develop faster in the central part of the board. As there is a situation in which the first player has an advantage in Tibetan Go, we use a method of increasing the score of the second player to balance the winning rates of both black and white players.

The space complexity of the original 17×17 Tibetan Go is approximately 7.73×10^{173}, while the space complexity of the simplified 9×9 Tibetan Go is 4.43×10^{38}, greatly reducing the complexity. For studying lightweight self-play reinforcement learning models of Tibetan Go under conditions of limited computing power and small game data, the simplified 9×9 Tibetan Go is more suitable. To balance the winning rates of both black and white players in Tibetan Go, we adopt the "handicap stones" rule from Go games and give one extra stone to black when calculating the final score, which has achieved the effect of balancing the winning rates of both black and white players in experiments.

4 Network structure

Fig. S1 illustrates the relationship between the modules in TibetanGoTinyNet and shows the components of the network more visually.

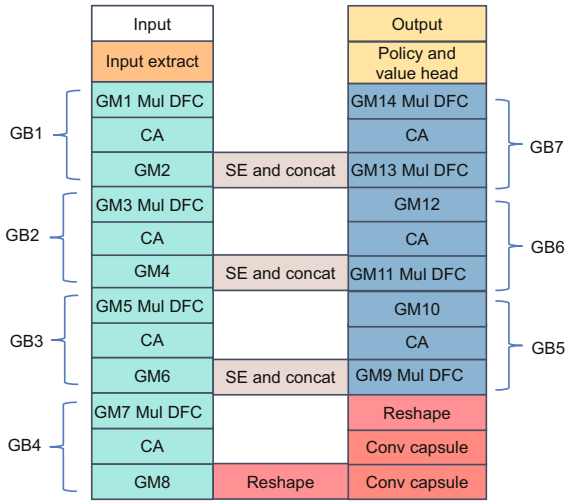

Fig. S1 Hierarchical module of the proposed TibetanGoTinyNet (GM: ghost module; CA: coordinate attention; GB: GhostV2 bottleneck)

4.1 Capsule network

To overcome the shortcomings of traditional convolution in spatial information availability by utilizing the characteristics of capsule networks, and to better extract features while controlling network parameters and maintaining the lightweight characteristics of the network, we only use the capsule network layer in the extraction of the highest level features. The number and length of capsule categories in the capsule encoder path of the network are set to (8, 16) and (3, 24). The number of capsule varieties in the last capsule layer is equal to the number of categories in the output of the policy head. In our model, the shape of the feature map entered into the last capsule layer is $H \times W \times C \times A$, where C is the number of capsule varieties and A is the dimension of each capsule.

4.2 The channel attentions mechanism

In the Tibetan Go zero learning task, the representation of states is complex and requires the use of dozens of channels, but not every channel is of equal importance. Unlike images which are RGB three channels, board games encode more channels for the situation, for example, 36 in this experiment. Therefore it is particularly important to select the more valuable channels to give high weight to the board game. So to extract more important features in the training and give more weight to the important channels, we use the channel attentions mechanism SE in skip connection to increase the weight of some of the important channels. This is different from many other U-Net style networks which neglect to enhance the network from the skip connection.

4.3 Dual-output network

Fig. S2 shows the beginning and end modules of the network. The dual outputs are value and policy, as illustrated in Fig. S2b and Fig. S2c, respectively. The policy head's output is designed to make the number of channels adapt to the number of operations on the game board. The feature map goes through a conv2D layer and then is flattened. Finally, the output tensor is sorted by the softmax function. The input tensor of the value head first is flattened and then goes through a fully connected layer. Finally, the tensor is output after the operation of the activation function tanh.

Different from the mostly used output design in general U-Net style networks, our proposed network has dual outputs. The dural outputs are policy and value. The policy head's output is designed to make the number of channels adapt to the number of operations on the board. The feature map goes through a conv2D layer and then is flattened. Finally, the output tensor is sorted by the softmax function. The input

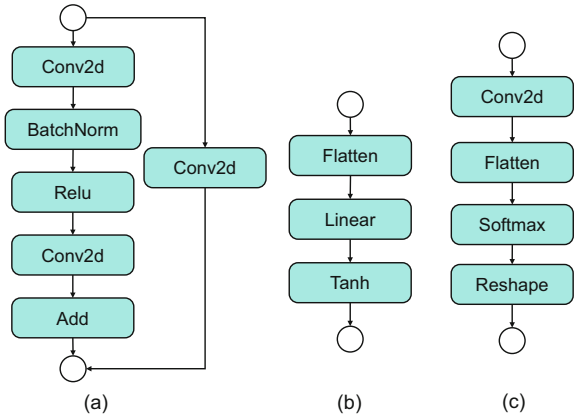

Fig. S2 The beginning and end modules of the network: (a) input extract block; (b) value head; (c) policy head

tensor of the value head first is flattened and then goes through a fully connected layer. Finally, the tensor is output after the operation of the activation function tanh.

5 Experiments and discussion

5.1 Baseline models

In this work, we compare TibetanGoTinyNet with U-Net, Res-UNet, Res-UNet Attention, Ghost-UNet, and Ghost Capsule-UNet.

1. U-Net: U-Net is an encoder–decoder structure, simple but effective, originally designed to solve image segmentation problems. The lightweight and simple structure of U-Net performs well in small-scale tasks.

2. Res-UNet: Combining residual blocks with U-Net, the traditional convolution is replaced with residual blocks as the backbone of U-Net. This method can combine the long-term skip connection of U-Net with the residual block's short-term skip connection.

3. Res-UNet Attention: U-Net is improved by adding AG (attention gate) at the skip connection. The traditional convolution is replaced with residual blocks as the backbone of U-Net.

4. Ghost-UNet: Ghost blocks are combined with U-Net and the traditional convolution is replaced with ghost blocks as the backbone of U-Net.

5. Ghost Capsule-UNet: U-Net is improved by adding capsule net at the bottleneck and GhostNetV2 is combined with U-Net to enhance the backbone of U-Net.

5.2 Performance of TibetanGoTinyNet model with different hyperparameters

To study the impact of learning rates and rollouts on the model, we conducted experiments with different hyperparameters. First, we trained Res-UNet, Res-UNet Attention, Ghost-Unet, and Ghost Capsule-UNet models under five different learning rates, namely, 0.1, 0.01, 0.005, 0.001, and 0.0001, with rollout count set to 200. The models were trained for 150 epochs. Under the same learning rate and rollout settings, the trained TibetanGoTinyNet model was played against other models in 100 matches. The results of the matches are shown in Table S1. From Table S1, it can be observed that except for the learning rate of 0.1, TibetanGoTinyNet achieved relatively high winning rates against other models under the remaining four learning rates. Specifically, the winning rates were higher at learning rates of 0.001 and 0.005, while winning rates were lower at learning rates of 0.01 and 0.0001.

To explore the optimal range of learning rates, we trained the TibetanGoTinyNet model with different learning rates: 0.1, 0.01, 0.005, 0.001, and 0.0001. We then played the TibetanGoTinyNet model trained with a learning rate of 0.001 against the models trained with other learning rates. The results are shown

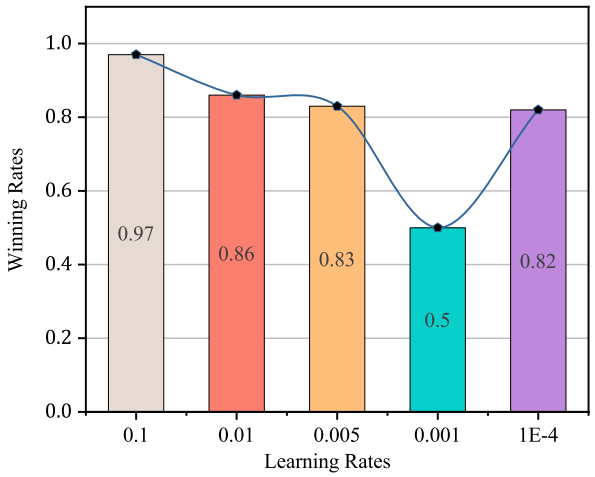

Fig. S3 TibetanGoTinyNet trained at a learning rate of 0.001 compared to the model trained at other learning rates

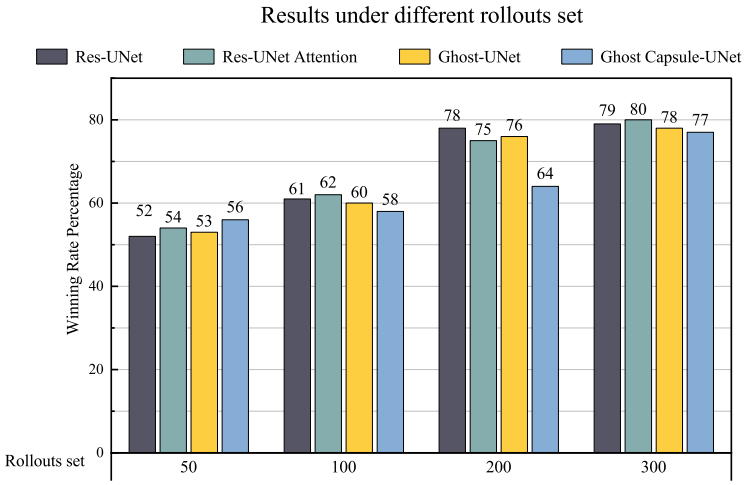

Fig. S4 The results of TibetanGoTinyNet against other models under several rollout set training conditions

Table S1 Winning rates of TibetanGoTinyNet against other models under different learning rate conditions

Model	Params	Learning Rates				
		0.1	0.01	0.005	0.001	0.0001
Res-UNet	1.226M	49%	58%	62%	78%	55%
Res-UNet Attention	1.242M	51%	64%	66%	75%	59%
Ghost-UNet	0.672M	48%	60%	61%	76%	59%
Ghost Capsule-UNet	0.765M	52%	56%	58%	64%	57%

in Fig. S3. Fig. S3 indicates that the model trained with a learning rate of 0.001 outperformed the models trained with other learning rates. When the learning rate was set to 0.1, the model did not converge due to a large learning rate. Learning rates of 0.01 and 0.005, although relatively high, were not optimal. On the other hand, a learning rate of 0.0001 resulted in low efficiency due to slow learning speed.

We also investigated the influence of rollouts on the experimental results. We selected rollout values of 50, 100, 200, and 300 to train TibetanGoTinyNet, Res-UNet, Res-UNet Attention, Ghost-UNet, and Ghost Capsule-UNet models, with each model trained for 150 epochs. The TibetanGoTinyNet model trained with different rollout values was played against other models trained with the same number of epochs. The results are shown in Fig. S4. From the figure, it can be observed that regardless of the rollout values, the winning rate of the TibetanGoTinyNet model was above 50%, and it increased as the rollout values increased. This is because the neural network guides the Monte Carlo tree search (MCTS) for decision-making. With larger

rollout values, the neural network's guidance on the MCTS becomes more effective, and with sufficient computational resources, the TibetanGoTinyNet model can better demonstrate superior performance.

References

Cazenave T, Chen YC, Chen GW, et al., (2020). Polygames: improved zero learning. *ICGA J*, 42(4):244–256.

Piette É, Soemers D, Stephenson M, et al., (2020). Ludii—the ludemic general game system. http://arxiv.org/abs/1905.05013v3

Ronneberger O, Fischer P, Brox T, (2015). U-Net: convolutional networks for biomedical image segmentation. 18$^{\text{th}}$ Int Conf on Medical Image Computing and Computer-Assisted Intervention, p.234–241.

Soemers DJNJ, Mella V, Browne C, et al., (2021). Deep learning for general game playing with Ludii and polygames. *ICGA J*, 43(3):146–161.

Soemers DJNJ, Piette É, Stephenson M, et al., (2022). The Ludii game description language is universal. https://arxiv.org/abs/2205.00451

Tian Y, Ma J, Gong Q, et al., (2019). ELF OpenGo: an analysis and open reimplementation of AlphaZero. Int Conf on Machine Learning, p.6244-6253.

Wu DJ, (2019). Accelerating self-play learning in go. https://arxiv.org/pdf/1902.10565