Transferable Reinforcement Learning for Board Games

Natthaphon Hongcharoen

Thesis presented for Faculty of Engineering and Technology

Panyapiwat Insitute of Management

In study of bachelor of science, Computer Engineering

Study year 2018

Abstract

The modern Reinforcement Learning has become widely interested recently, after the AlphaGo[22] became the first program to defeat a world champion in the game of Go. And by 2017, the AlphaGo Zero[24] and AlphaZero[23] programs achieved superhuman performance in the game of Go and Chess, by solely trained from games of self-play which require no human knowledge. But in contrast, both AlphaGo Zero and AlphaZero required extremely powerful processor as they need to random move in the early state of training which cost more expend. While in Computer Vision domain, we often use pretrained model from large dataset such as Imagenet[8] and retrain it for desired task which cost less time and achieved more accuracy than train it from scratch. In this thesis, we experiment a method to reuse the trained model of a game such as Othello, Connect4 or Gomoku and re-training with one different game by hoping it to be faster.

1 Introduction

1.1 Background

The modern Reinforcement Learning such as AlphaGo has showed outstanding performance by won against world champion in game of Go a decade earlier than predicted[15][21]. And the AlphaGo Zero has showed that it possible to train Reinforcement Learning without any human knowledge while still achieved good performance[24][23]. In contrast of good result, for AlphaGo, by using ‘supervised learning from human expert games’[22] means it's require the expert games that might be difficult to obtained or inthe worst case, it might be impossible to get. And in case of AlphaGo Zero, according to original paper[24], the model need to be trained with 29 million games of self-play. Parameters were updated from 3.1 million mini-batches of 2,048 positions each. In Gian-Carlo Pascutto's experiments, every 2000 self-play moves generated by GTX-1080ti GPU would take 93 seconds, which means it need 1700 years to play all 29 million games[17]. It practically impossible to train AlphaGo Zero in personal level.

Meanwhile in Computer Vision domain, transferring weights which pretrained in large datasets and then re-train the model in preferred dataset is a widely used method as there are ImageNet[8] pretrained weights in many popular libraries[1][9][6]. The reason is it often improve regularizing and better performance and need less sample to train[27]. In the same way, Transfer Learning should be able to used in Deep Reinforcement Learning algorithm which use Deep Convolutional Neural Networks and help improve training speed and model's performance.

1.2 Objective

1. Create Reinforcement Learning models for board games such as Othello, Connect4 or Gomoku with AlphaZero algorithm.
2. Create Transferred Reinforcement Learning models from trained games.
3. Evaluate Transferred models performance against normal models.

1.3 Scope

1. Create at least 2 AlphaZero models and at least 2 Transferred models.
2. Evaluate performance of each algorithm in same iteration and same version.

2 Related Works

2.1 AlphaGo and modern Reinforcement Learning

By March 2016 DeepMind AlphaGo has become the first computer program to won against the world champion in game of Go by defeated Lee Sedol, the winner of 18 world titles, which many years earlier than predicted[15][21]. The AlphaGo introduced a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state- of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. DeepMind also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away[22].

All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s, under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game's breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess $(b \approx 35,\ d \approx 80)$[2] and especially Go $(b \approx 250,\ d \approx 150)$[2], exhaustive search is infeasible[26][18], but the effective search space can be reduced by two general principles. First, the depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s. This approach has led to superhuman performance in chess[5], checkers[19] and othello[4], but it was believed to be intractable in Go due to the complexity of the game[16]. Second, the breadth of the search may be reduced by sampling actions from a policy $p(a|s)$ that is a probability distribution over possible moves a in position s. For example, Monte Carlo rollouts[25] search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p. Averaging over such rollouts can provide an effective position evaluation, achieving superhuman performance in backgammon[25] and Scrabble[20], and weak amateur level play in Go[3].

Monte Carlo tree search (MCTS)[7][13] uses Monte Carlo rollouts to estimate the value of each state in a search tree. As more simulations are executed, the search tree grows larger and the relevant values become more accurate. The policy used to select actions during search is also improved over time, by selecting children with higher values. Asymptotically, this policy converges to optimal play, and the evaluations converge to the optimal value function[13].

AlphaGo pass in the board position as a 19 × 19 image and use convolutional layers to construct a representation of the position. By sing these neural networks to reduce the effective depth and breadth of the search tree: evaluating positions using a value network, and sampling actions using a policy network. The neural networks trained using a pipeline consisting of several stages of machine learning. Begin by training a supervised learning (SL) policy network p_σ directly from expert human moves. This provides fast, efficient learning updates with immediate feedback and high-quality gradients. And also train a fast policy p_π that can rapidly sample actions during rollouts. Next, train a reinforcement learning (RL) policy network p_ρ that improves the SL policy network by optimizing the final outcome of games of self-play. This adjusts the policy towards the correct goal of winning games, rather than maximizing predictive accuracy. Finally, train a value network v_θ that predicts the winner of games played by the RL policy network against itself.

2.2 AlphaGo Zero, Mastering the Game of Go without Human Knowledge

AlphaGo Zero differs from first version AlphaGo in several important aspects. First and foremost, it is trained solely by self-play reinforcement learning, starting from random play, without any supervision or use of human data. Second, it only uses the black and white stones from the board as input features. Third, it uses a single neural network, rather than separate policy and value networks. Finally, it uses a simpler tree search that relies upon this single neural network to evaluate positions and sample moves, without performing any Monte-Carlo rollouts. To achieve these results, we introduce a new reinforcement learning algorithm that incorporates lookahead search inside the training loop, resulting in rapid improvement and precise and stable learning[24].

AlphaGo Zero uses a deep neural network f_θ with parameters θ. This neural network takes as an input the raw board representation s of the position and its history, and outputs both move probabilities and a value, $(p, v) = f_\theta(s)$. The vector of move probabilities p represents the probability of selecting each move (including pass), $p_a = P_r(a|s)$. The value v is a scalar evaluation, estimating the probability of the current player winning from position s. This neural network combines the roles of both policy network and value network[22] into a single architecture. The neural network consists of many residual blocks[11] of convolutional layers[14] with batch normalisation[12] and rectifier non-linearities[10]

หนังสืออ้างอิง

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*, CoRR, abs/1603.04467 (2016).

[2] L. V. Allis, *Searching for solutions in games and artificial intelligence*, 1994.

[3] B. Bouzy and B. Helmstetter, *Monte-carlo go developments*, in Advances in Computer Games, Springer US, 2004, pp. 159--174.

[4] M. Buro, *From simple features to sophisticated evaluation functions*, in Proceedings of the First International Conference on Computers and Games, CG '98, London, UK, UK, 1999, Springer-Verlag, pp. 126--145.

[5] M. Campbell, A. J. Hoane, Jr., and F.-h. Hsu, *Deep blue*, Artif. Intell., 134 (2002), pp. 57--83.

[6] F. Chollet, *Keras deep learning for humans*. http://keras.io/.

[7] R. Coulom, *Efficient selectivity and backup operators in monte-carlo tree search*, in Proceedings of the 5th International Conference on Computers and Games, CG'06, Berlin, Heidelberg, 2007, Springer-Verlag, pp. 72--83.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *Imagenet: A large-scale hierarchical image database*, 2009 IEEE Conference on Computer Vision and Pattern Recognition, (2009).

[9] Facebook, *Tensors and dynamic neural networks in python with strong gpu acceleration*. http://pytorch.org/.

[10] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, *Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit*, Nature, 405 (2000), pp. 947--951.

[11] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, pp. 770--778.

[12] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, in Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, JMLR.org, 2015, pp. 448--456.

[13] L. Kocsis and C. Szepesvari, *Bandit based monte-carlo planning*, in Proceedings of the 17th European Conference on Machine Learning, ECML'06, Berlin, Heidelberg, 2006, Springer-Verlag, pp. 282--293.

[14] Y. LeCun, *Gradient-based learning applied to document recognition*, 1986.

[15] A. Levinovitz, *The mystery of go, the ancient game that computers still can't win*. https://www.wired.com/2014/05/the-world-of-computer-go/, 2014.

[16] M. Muller, *Computer go*, Artif. Intell., 134 (2002), pp. 145--179.

[17] G.-C. Pascutto, *[computer-go] zero performance*. http://computer-go.org/pipermail/computer-go/2017-October/010307.html, 2017.

[18] J. Schaeffer, *The games computers (and people) play*, Advances in Computers, 52 (2000), pp. 189--266.

[19] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron, *A world championship caliber checkers program*, Artificial Intelligence, 53 (1992), pp. 273 -- 289.

[20] B. Sheppard, *World-championship-caliber scrabble*, Artif. Intell., 134 (2002), pp. 241--275.

[21] D. Silver and D. Hassabis, *Alphago: Mastering the ancient game of go with machine learning*. https://ai.googleblog.com/2016/01/alphago-mastering-ancient-game-of-go.html, 2016.

[22] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, *Mastering the game of go with deep neural networks and tree search*, Nature, 529 (2016), pp. 484--489.

[23] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, *Mastering chess and shogi by self-play with a general reinforcement learning algorithm*, CoRR, abs/1712.01815 (2017).

[24] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. H. A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, *Mastering the game of go without human knowledge*, Nature, 550 (2017), pp. 354--359.

[25] G. Tesauro and G. R. Galperin, *On-line policy improvement using monte-carlo search*, in Advances in Neural Information Processing Systems 9, M. C. Mozer, M. I. Jordan, and T. Petsche, eds., MIT Press, 1997, pp. 1068--1074.

[26] H. van den Herik, J. W.H.M.Uiterwijk, and J. van Rijswijck, *Games solved: now and in the future*, Aritificial Intellegence, 134 (2002), pp. 277--311.

[27] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, *How transferable are features in deep neural networks?*, CoRR, abs/1411.1792 (2014).