ALPHAGO ZERO: A NOTE WITH PSEUDO CODE

Yuxi Li (yuxili@gmail.com)

1 INTRODUCTION

AlphaGo (Silver et al., 2016) set a landmark in AI by defeating a top human Go player with 18 international titles in March 2016. AlphaGo Zero (Silver et al., 2017) further improved previous versions by learning a superhuman computer Go program without human knowledge.

AlphaGo (Silver et al., 2016; 2017) is based on deep learning, reinforcement learning (RL), and Monte Carlo tree search (MCTS). This wave of deep reinforcement learning was ignited by deep Q-network (Mnih et al., 2015); see Li (2017) for a comprehensive overview.[1]

See Sutton and Barto (2017) for a detailed and intuitive description of AlphaGo (Silver et al., 2016); See Deepmind's official blogs of AlphaGo (Silver et al., 2016) and AlphaGo Zero (Silver et al., 2017) respectively at https://deepmind.com/research/alphago/, and at https://deepmind.com/blog/alphago-zero-learning-scratch/. See Littman (2015) for a survey of RL.

2 PSEUDO CODE

We present a brief, conceptual pseudo code in Algorithm 1 for training in AlphaGo Zero, conducive for easier understanding. Refer to the original paper (Silver et al., 2017) for details.

AlphaGo Zero can be understood as an approximation policy iteration, incorporating MCTS inside the training loop to perform both policy improvement and policy evaluation. MCTS may be regarded as a policy improvement operator. It outputs move probabilities stronger than raw probabilities of the neural network. Self-play with search may be regarded as a policy evaluation operator. It uses MCTS to select moves, and game winners as samples of value function. Then the policy iteration procedure updates the neural network's weights to match the move probabilities and value more closely with the improved search probabilities and self-play winner, and conduct self-play with updated neural network weights in the next iteration to make the search stronger.

The features of AlphaGo Zero (Silver et al., 2017), comparing with AlphaGo (Silver et al., 2016), are: 1) it learns from random play, with self-play reinforcement learning, without human data or supervision; 2) it uses black and white stones from the board as input, without any manual feature engineering; 3) it uses a single neural network to represent both policy and value, rather than separate policy network and value network; and 4) it utilizes the neural network for position evaluation and move sampling for MCTS, and it does not perform Monte Carlo rollouts. AlphaGo Zero deploys several recent achievements in neural networks: residual convolutional neural networks (ResNets), batch normalization, and rectifier nonlinearities.

AlphaGo Zero has three main components in its self-play training pipeline executed in parallel asynchronously: 1) optimize neural network weights from recent self-play data continually; 2) evaluate players continually; 3) use the strongest player to generate new self-play data.

When AlphaGo Zero playing a game against an opponent, MCTS searches from the current state, with the trained neural network weights, to generate move probabilities, and then selects a move.

3 DISCUSSIONS

AlphaGo Zero is a reinforcement learning algorithm. It is neither supervised learning nor unsupervised learning. The game score is a reward signal, not a supervision label. Optimizing the loss

[1] Our goal to put this note with pseudo code on arXiv is to facilitate easier understanding of AlphaGo Zero training algorithm and to make several clarifications. We plan to incorporate this into our next update of Li (2017) by the end of 2017. We appreciate feedbacks received so far, and welcome comments and criticisms.

function l is supervised learning. However, it performs policy evaluation and policy improvement, as one iteration in policy iteration.

AlphaGo Zero is not only a heuristic search algorithm. AlphaGo Zero is a policy iteration procedure, in which, heuristic search, in particular, MCTS, plays a critical role, but within the scheme of reinforcement learning policy iteration, as illustrated in the pseudo code in Algorithm 1. MCTS can be viewed as a policy improvement operator.

AlphaGo attains a superhuman level. It may confirm that professionals have developed effective strategies. However, it does not need to mimic professional plays. Thus it does not need to predict their moves correctly.

The inputs to AlphaGo Zero include the raw board representation of the position, its history, and the colour to play as 19×19 images; game rules; a game scoring function; invariance of game rules under rotation and reflection, and invariance to colour transposition except for komi. An additional and critical input is solid research and development experiences.

AlphaGo Zero utilized 64 GPU workers (each maybe with multiple GPUs) and 19 CPU parameter servers (each with multiple CPUs) for training, around 2000 TPUs for data generation, and 4 TPUs for execution.

AlphaGo requires huge amount of data for training, so it is still a big data issue. However, the data can be generated by self play, with a perfect model or precise game rules.

Due to the perfect model or precise game rules for computer Go, AlphaGo algorithms have their limitations. For example, in healthcare, robotics and self driving problems, it is usually hard to collect a large amount of data, and it is hard or impossible to have a close enough or even perfect model. As such, it is nontrivial to directly apply AlphaGo algorithms to such applications.

On the other hand, AlphaGo algorithms, especially, the underlying techniques, namely, deep learning, reinforcement learning, and Monte Carlo tree search, have many applications. Silver et al. (2016) and Silver et al. (2017) recommended the following applications: general game-playing (in particular, video games), classical planning, partially observed planning, scheduling, constraint satisfaction, robotics, industrial control, and online recommendation systems. AlphaGo Zero blog mentioned the following structured problems: protein folding, reducing energy consumption, and searching for revolutionary new materials. See Li (2017) for more applications of AlphaGo algorithms, and the underlying techniques, in particular, deep reinforcement learning.

AlphaGo has made tremendous progress, and set a landmark in AI. However, we are still far away from attaining artificial general intelligence (AGI).

REFERENCES

Li, Y. (2017). Deep Reinforcement Learning: An Overview. *ArXiv e-prints*.

Littman, M. L. (2015). Reinforcement learning improves behaviour from evaluative feedback. *Nature*, 521:445–451.

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

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

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., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, 550:354–359.

Sutton, R. S. and Barto, A. G. (2017). *Reinforcement Learning: An Introduction (2nd Edition, in preparation)*. MIT Press.

Input: the raw board representation of the position, its history, and the colour to play as 19×19 images; game rules; a game scoring function; invariance of game rules under rotation and reflection, and invariance to colour transposition except for komi
Output: policy (move probabilities) p, value v

initialize neural network weights θ_0 randomly
//AlphaGo Zero follows a policy iteration procedure
for *each iteration* i **do**

// termination conditions:
// 1. both players pass
// 2. the search value drops below a resignation threshold
// 3. the game exceeds a maximum length

initialize s_0
for *each step* t*, until termination at step* T **do**

// MCTS can be viewed as a policy improvement operator
// search algorithm: asynchronous policy and value MCTS algorithm (APV-MCTS)
// execute an MCTS search $\pi_t = \alpha_{\theta_{i-1}}(s_t)$ with previous neural network $f_{\theta_{i-1}}$
// each edge (s, a) in the search tree stores a prior probability $P(s, a)$, a visit count $N(s, a)$, and an action value $Q(s, a)$
while *computational resource remains* **do**
select: each simulation traverses the tree by selecting the edge with maximum upper confidence bound $Q(s, a) + U(s, a)$, where $U(s, a) \propto P(s, a)/(1 + N(s, a))$
expand and evaluate: the leaf node is expanded and the associated position s is evaluated by the neural network, $(P(s, \cdot), V(s)) = f_{\theta_i}(s)$; the vector of P values are stored in the outgoing edges from s
backup: each edge (s, a) traversed in the simulation is updated to increment its visit count $N(s, a)$, and to update its action value to the mean evaluation over these simulations, $Q(s, a) = 1/N(s, a) \sum_{s'|s,a \to s'} V(s')$, where $s'|s, a \to s'$ indicates that a simulation eventually reached s' after taking move a from position s
end
// self-play with search can be viewed as a policy evaluation operator: select each move with the improved MCTS-based policy, uses the game winner as a sample of the value
play: once the search is complete, search probabilities $\pi \propto N^{1/\tau}$ are returned, where N is the visit count of each move from root and τ is a parameter controlling temperature; play a move by sampling the search probabilities π_t, transition to next state s_{t+1}
end

score the game to give a final reward $r_T \in \{-1, +1\}$
for *each step* t *in the last game* **do**
$z_t \leftarrow \pm r_T$, the game winner from the perspective of the current player
store data as (s_t, π_t, z_t)
end
sample data (s, π, z) uniformly among all time-steps of the last iteration(s) of self-play

//train neural network weights θ_i
//optimizing loss function l performs both policy evaluation, via $(z - v)^2$, and policy improvement, via $-\pi^T \log p$, in a single step
adjust the neural network $(p, v) = f_{\theta_i}(s)$:
to minimize the error between the predicted value v and the self-play winner z, and
to maximize similarity of neural network move probabilities p to search probabilities π
specifically, adjust the parameters θ by gradient descent on a loss function
$(p, v) = f_{\theta_i}(s)$ and $l = (z - v)^2 - \pi^T \log p + c\|\theta_i\|^2$
l sums over the mean-squared error and cross-entropy losses, respectively
c is a parameter controlling the level of $L2$ weight regularization to prevent overfitting

evaluate the checkpoint every 1000 training steps to decide if replacing the current best player (neural network weights) for generating next batch of self-play games
end

Algorithm 1: AlphaGo Zero training pseudo code, based on Silver et al. (2017)