Self Training

September 10, 2017

Research Review

Ngoc Minh VU

Version 0.1

– Draft –

Abstract

In this review we go through two fascinating papers in AI field: Deep Blue and AlphaGo.

Contents

1 Deep Blue II

This paper describe the first chess machine that defeated then-reigning World Chess Champion Garry Kasparov in a six-game match in 1997

1.1 Brief summary of chess machines

The following table summary some important aspect of some notable chess machines

Name	Year	Aspects
ChipTest	1980	Created at CMU lab, it has • single-chip chess • **500k position per second** (pps)
Deep Thought	1988	Created at CMU lab, it has • single-chip chess **700k pps** • first chess machine to beat a Grandmaster
Deep Thought 2	1991-1995	Created at IBM Watson lab, it has four improvements: • multiprocessing: 24 chess engines • enhanced evaluation hardware (but still too simple) • improved search software • extended book: make reasonable opening moves
Deep Blue I	1995-1996	Created at IBM Watson lab, it has • single-chip chess search engine • 36 nodes IBM RS/6000 with 216 chess chips • **50-100 millions pps** It lost Kasparov by a fairly decissive 4-2 score.
Deep Blue II	1997	Created at IBM Watson lab combining with knownledge from Grandmaster consultant. It has • better single-chip chess search engine with more features than Deep Blue I's chip • double number of chess chips: 480 chess chips • **100-200 millions pps** • a set of tools to aid in debugging, evaluation tuning and visualization tools Deep Blue II defeated Kasparov by a score of 3.5-2.5.

Next we look at some notable aspect of Deep Blue II.

1.2 Deep Blue II

Deep Blue II relies on many ideas in ealier chess program such as quiescence search, iterative deepening, transposition tables and NegaScout. However it has the following main improvement

Name	Improvement
Chess chip	It has three parts: the move generator, the evaluation function and the search control with following properties • generated moves are reasonably ordered • evaluation is composed of “fast” and “slow” method • it implements null-window alpha-beta search The chess chip allows **hardware search** which is very simple and fast.
Software Search	Based on the experiences with previous chess machines, it was designed based on the following principles • Extend forcing/forced pairs (ffp) of moves • Forced moves are expectation dependent • Fractional extensions: depends on the forcing’s level an ffp might not get a full 2-ply extension but rather a smaller 1.75 ply • Delayed extensions: when there is a series of ffps, extensions are delayed until multiple ffps occur in a given path • Dual credit: each side has its own credit • Preserve the search envelope: avoid an oscillating search The **software search** allows more complex search which is combining with **hardware search** (fast) that allows us to do **4-5 ply searches plus quiescence** in middlegame positions and somewhat deeper searches in endgames
Parallel search	The system has 30 nodes with 480 chips, in order to fully use its power, it designs a parallel search algorithm as follows • three layers static processor tree: one master node controlling the other 29 nodes, which in turn control 16 chess chip each. • centralized control is managed on SP nodes • defining cases to be searched parallel • synchronization
Other	The system also improved the following • Better evaluation function based on previous match • Using opening book to get openning moves and tested with Deep Blue to chose ones working well. • Using extended book which allows Deep Blue II to learn from 700000 game database • Using endgame databases in both hardware (chess chip) and software • Using normal time control i.e when time-out occurs the current best move is played. It also define “panic time” which is used when certain condition meets.

1.3 Conclusion

The success of Deep Blue II was based on experienced of previous chess machine combining with knownledge of chess Grandmaster and high capacity of the system.

2 AlphaGo

The paper on AlphaGo describes a break through in AI field where for 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.

The paper introduces a new approach that uses **'value networks'** to evaluate board positions and **'policy networks'** to select moves, these networks are deep neural networks that are trained by a **novel combination** of **supervised learning** from human expert games, and **reinforcement learning** from games of self-play.

Also, with above trained networks, the paper introduces a new search algorithm that combines **Monte Carlo simulation** with value and policy network.

The neural networks are trained using the following pipeline

- Using expert human moves database, we train a supervised learning (SL) policy network p_σ and p_π to predict next move (which is a classification problem). The p_π using less layer than p_σ so it can run very fast: it selects an action using just $2\mu s$ rather than $3ms$ for p_σ.
- Using trained p_σ as initialization, we train a reinforcement learning (RL) policy network p_ρ by policy gradient learning to maximize the outcome against previous version of the policy network. During this process, a new data set is generated by playing games of self-play.
- Finally, a value network v_θ is trained by regression to predict the expected outcome (that is, whether the current player wins) in the positions from the self-play data set.

Let's examine these novel techniques in more detail.

2.1 Supervised learning of policy networks

The SL policy network $p_\sigma(a|s)$ alternates between **convolutional layers** with weight σ and **rectifier nonlinearities**. We trained a **13-layer** policy network using stochastic gradient ascent on randomly sampled state-action pairs (s, a) to maximize the likelihood of the human move a selected in the state s

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma}$$

The network predicted expert moves on a held out test set with an accuracy of 57.0% using all input features and 55.7% using only raw board position and move history as inputs (which **beats the state-of-the-art** from other research groups of 44.4%).

2.2 Reinforcement learning of policy networks

The second state of the training pipeline aims at improving the policy network by policy gradient reinforement learning (RL). The RL policy network p_ρ is identical in structure to the SL policy network, and its weights ρ are initialized to the same value $\rho = \sigma$.

We then play games between the current policy network p_ρ and a **randomly selected previous iteration** of the policy network. This randomization helps preventing over-fitting to the current policy.

We use a reward function $r(s)$ is zero for all non-terminal time steps $t < T$. The outcome $z_t = \pm r(s_T)$ is the terminal reward at the end of the game from the perspective of the current player at time step t: +1 for winning and -1 for losing.

Weights are then updated in the direction that maximizes expected outcome

$$\Delta\rho \propto \frac{\partial \log p_\rho(a|s)}{\partial\rho} z_t$$

We test the RL policy against other programs

RL policy v.s others

Opponent	Win Rate
SL policy	80%
Pachi (strongest open-source Go)	85%

2.3 Reinforcement learning of value networks

The final stage of the training pipeline focuses on **position evaluation**, estimating a value function $v^p(s)$ that predicts the outcome from position s of games played by using policy p for both players

$$v^p(s) = \mathbb{E}\left[z_t | s_t = s, a_{t\ldots T} \sim p\right]$$

We approximate the value function using a value network $v_\theta(s)$ with weights θ. This neural networks has a similar architecture to the policy network, but outputs **a single prediction** (instead of a probability distribution). We train the weights of the value network by regression on state-outcome pairs (s, z), using SGD to minimize the mean squared error (MSE) between the predicted value $v_\theta(s)$ and the corresponding outcome z

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$

Note that this approach when trained on KGS data leads to **over-fitting** where MSE of **0.37 on the test set v.s 0.19 on the training set**. To overcome this problem, the self-play data set is also used which leads to MSE of 0.226 and 0.234 on training and test set respectively, indicating minimal over-fitting.

2.4 Searching with policy and value networks

AlphaGo combines the policy and value networks in an MCTS algorithm with the following principle

- Each edge (s, a) of a state-action pair, we store an action value $Q(s, a)$, a visit count $N(s, a)$ and prior probability $P(s, a)$. The tree is straversed by simulation (that is, descending the tree in complete games without backup), starting form the root state.
- At each time step t an action a_t is selected from state s_t as

  $$a_t = \arg\max_a (Q(s_t, a) + u(s_t, a))$$

  so as to maximize **action value** plus a bonus

  $$u(s, a) \propto \frac{P(s, a)}{1 + N(s, a)}$$

that is proportional to the prior probability but **decays with repeated visits to encourage exploration**.

- At a leaf node s_L at step L the value of the leaf node is evaluated in two very different ways: first by the value network $v_\theta(s_L)$ and second by the outcome z_L of a random rollout played out until terminal step T using the fast rollout policy p_π, these evaluation is combined with a mixing parameter λ
- At the end of simulation, we update $N(s,a)$ and $Q(s,a)$

$$N(s,a) = \sum_{i=1}^{n} \mathbb{1}(s,a,i)$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} \mathbb{1}(s,a,i) V(s_L^i)$$

To efficiently combine MCTS with deep neural networks, AlphaGo use an asynchronous **40 multi-threaded search** that executes simulations on **48 CPUs** and computes policy and value networks in parallel on **8 GPUs**. There is a distributed version of AlphaGo with **40 search threads**, **1202 CPUs** and **176 GPUs**.

2.5 Evaluating AlphaGo

The single-machine AlphaGo is **stronger than any previous Go program**, winning 494 out of 495 games (99.8% win-rate). To provide a grater challenge to AlphaGo, we also played games with four handicap stones; it won 77%, 86% and 99% against Crazy Stone, Zen and Pachi, respectively.

The distributed version of AlphaGo was significantly stronger, winning 77% of games against single-machine AlphaGo and 100% of its games against other programs.

We tried variants of AlphaGo with $\lambda = 0,\ 0.5$ and 1 and $\lambda = 0.5$ performed best.

2.6 Conclusion

The AlphaGo have introduced a new search algorithm that successfully combines neural network evaluations with Monte Carlo rollouts. This leads to the first Go program that has reached a professional level in Go.

During the match against Fan Hui, AlphaGo **evaluated thousands of times fewer positions** than Deep Blue did in its chess match against Kasparov; however it won the match by

- **selecting those positions more intelligently**, using the policy network
- **evaluating them more precisely**, using the value network

Furthermore, while Deep Blue relied on a hand-crafted evaluation function, the neural networks of AlphaGo are trained directly from game play purely through general-purpose supervised and reinforcement learning methods.

3 Summary

So we went through two AI breakthrough one in chess and on in Go. The two is summarized in the following table

	Deep Blue	AlphaGo
Evaluation	Hand-crafted	Self-trained with NNs
Search	traditional minimax/alphabeta	MCTS with deep neural network
Hardware	Multi-processing with CPUs and chess chip	Distributed with lots of CPUs and GPUs
V.s human	3.5-2.5	5-0

Looking at above result, we can foresee a future where AI can do much better than human which hopefully will help us to solve more currently un-solvable problem.