TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2019

AlphaZero

AlphaGo Fan (October 2015)

AlphaGo Defeats Fan Hui, European Go Champion.

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO

AlphaGo Lee (March 2016)

AlphaGo Zero vs. Alphago Lee (April 2017)

AlphaGo Lee:

- Trained on both human games and self play.
- Trained for Months.
- Run on many machines with 48 TPUs for Lee Sedol match.

AlphaGo Zero:

- Trained on self play only.
- Trained for 3 days.
- Run on one machine with 4 TPUs.
- Defeated AlphaGo Lee under match conditions 100 to 0.

AlphaZero Defeats Stockfish in Chess (December 2017)

AlphaGo Zero was a fundamental algorithmic advance for general RL.

The general RL algorithm of AlphaZero is essentially the same as that of AlphaGo Zero.

Some Background

Monte-Carlo Tree Search (MCTS)

Brugmann (1993)

To estimate the value of a position (who is ahead and by how much) run a cheap stochastic policy to generate a sequence of moves (a rollout) and see who wins.

Select the move with the best rollout value.

(One Armed) Bandit Problems
Robbins (1952)

Consider a set of choices (different slot machines).
Each choice gets a stochastic reward.

We can select a choice and get a reward as often as we like.

We would like to determine which choice is best and also to get reward as quickly as possible.

The Upper Confidence Bound (UCB) Algorithm

Lai and Robbins (1985)

For each choice (bandit) a construct a confidence interval for its average reward.

$$\mu = \hat{\mu} \pm 2\sigma/\sqrt{n}$$

$$\mu(a) \leq \hat{\mu}(a) + U(N(a))$$

Always select

$$\underset{a}{\operatorname{argmax}}\, \hat{\mu}(a) + U(N(a))$$

The Upper Confidence Tree (UCT) Algorithm
Kocsis and Szepesvari (2006), Gelly and Silver (2007)

The UCT algorithm grows a tree by running “simulations”.

Each simulation descends into the tree to a leaf node, expands that leaf, and returns a value.

In the UCT algorithm each move choice at each position is treated as a bandit problem.

We select the child (bandit) with the lowest upper bound as computed from simulations selecting that child.

Bootstrapping from Game Tree Search

Vaness, Silver, Blair and Uther, NeurIPS 2009

In bootstrapped tree search we do a tree search to compute a min-max value $V_{\rm mm}(s)$ using tree search with a static evaluator $V_\Phi(s)$. We then try to fit the static value to the min-max value.

$$\Delta\Phi = -\eta\nabla_\Phi\left(V_\Phi(s) - V_{\rm mm}(s)\right)^2$$

This is similar to minimizing a Bellman error between $V_\Phi(s)$ and a rollout estimate of the value of s but where the rollout estimate is replaced by a min-max tree search estimate.

The AlphaZero Algorithm

Silver et al. (2017)

The AlphaZero Algorithm

The AlphaZero algorithm is an RL learning method for the case where state transitions are determined by actions.

In principle, the AlphaZero algorithm could be applied to the problem of direct optimization of the BLEU score in machine translation.

Tree Search

To select an action, first construct a search tree over possible action sequences to evaluate options.

As in UCT, the tree is grown by running simulations. Each simulation descends into the tree from the root selecting an action at each state until a new leaf is created.

In an adversarial game we simulate opponent actions using the same algorithm but where rewards are reversed for opponent actions.

Simulations

Each node of the search tree is a state and each node has a set of possible actions allowed from that state.

Initially the tree is just a single starting state and all actions from that state are untried.

A simulation descends into the tree from the root selecting an action at each state.

When an untried action is selected the tree is expanded with a new leaf state and the simulation stops.

Each simulation returns the value $V_\Phi(s)$ for the new state s.

The Data Structures

Each node (state) in the search stores the following information which can be initialized by running the value and policy networks on state s.

- $V_\Phi(s)$ — the value network value for the position s.
- The policy probabilities $\pi_\Phi(s,a)$ for each legal action a.
- The number $N(s,a)$ of simulations that have tried move a from s. This is initially zero.
- For $N(s,a) > 0$, the average $\hat{\mu}(s,a)$ of the values of the simulations that have tried move a from position s.

Simulations and Upper Confidence Bounds

In descending into the tree, a simulation selects the move $\operatorname{argmax}_a U(s,a)$ where we have

$$U(s,a) = \begin{cases} \lambda_u \, \pi_\Phi(s,a) & \text{if } N(s,a) = 0 \\ \\ \hat{\mu}(s,a) + \lambda_u \, \pi_\Phi(s,a)/N(s,a) & \text{otherwise} \end{cases}$$

Upper Confidence Bounds

$$U(s,a) = \begin{cases} \lambda_u \; \pi_\Phi(s,a) & \text{if } N(s,a) = 0 \\ \\ \hat{\mu}(s,a) + \lambda_u \; \pi_\Phi(s,a)/N(s,a) & \text{otherwise} \end{cases}$$

The hyperparameter λ_u should be selected so that $U(s,a)$ will typically **decrease** as $N(s,a)$ increases.

In this regime for λ_u, we can think of $U(s,a)$ as an upper confidence bound in the UCT algorithm.

Root Action Selection

When the search is completed, we must select a move from the root position. For this we use a post-search stochastic policy

$$\pi_{s_{\mathrm{root}}}(a) \propto N(s_{\mathrm{root}}, a)^{\beta}$$

where β is temperature hyperparameter.

Constructing a Replay Buffer

We run a large number of tree-search guided episodes — episodes (games) with actions selected by tree search.

We then construct a replay buffer of triples (s, π_s, R) where

- s is a root position of a tree search.
- π_s is the distribution on a defined by $P(a) \propto N(s,a)^\beta$.
- $R \in \{-1, 1\}$ is the reward of the episode for the root player (the final outcome of the tree-search guided game).

The Loss Function

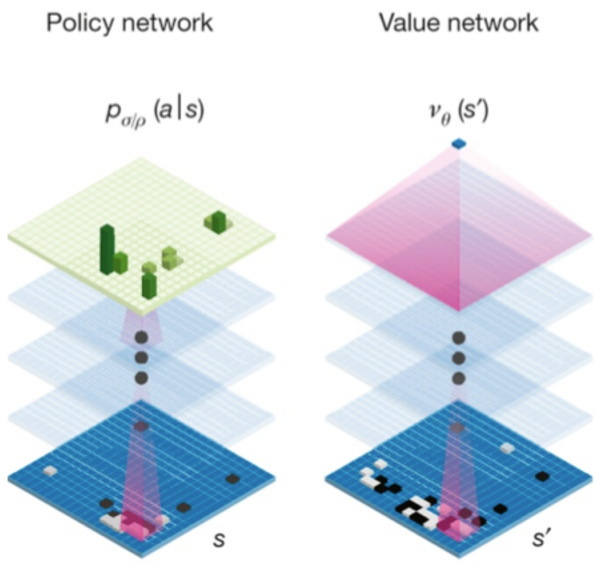
Training is done by SGD on the following loss function.

$$\Phi^* \;=\; \operatorname*{argmin}_\Phi \; E_{(s,\pi,R)\sim\mathrm{Replay},\; a\sim\pi} \left(\begin{array}{l} (V_\Phi(s) - R)^2 \\ \\ -\lambda_\pi \log \pi_\Phi(a|s) \\ \\ +\lambda_R \;||\Phi||^2 \end{array}\right)$$

The replay buffer is periodically updated with new self-play games.

Empirical Results

The Go-Chess-Shogi Networks



In AlphaZero the networks are either 20 block or 40 block ResNets and either separate networks or one dual-headed network.

Training Time

Single 20 block dual-headed ResNet on Go.

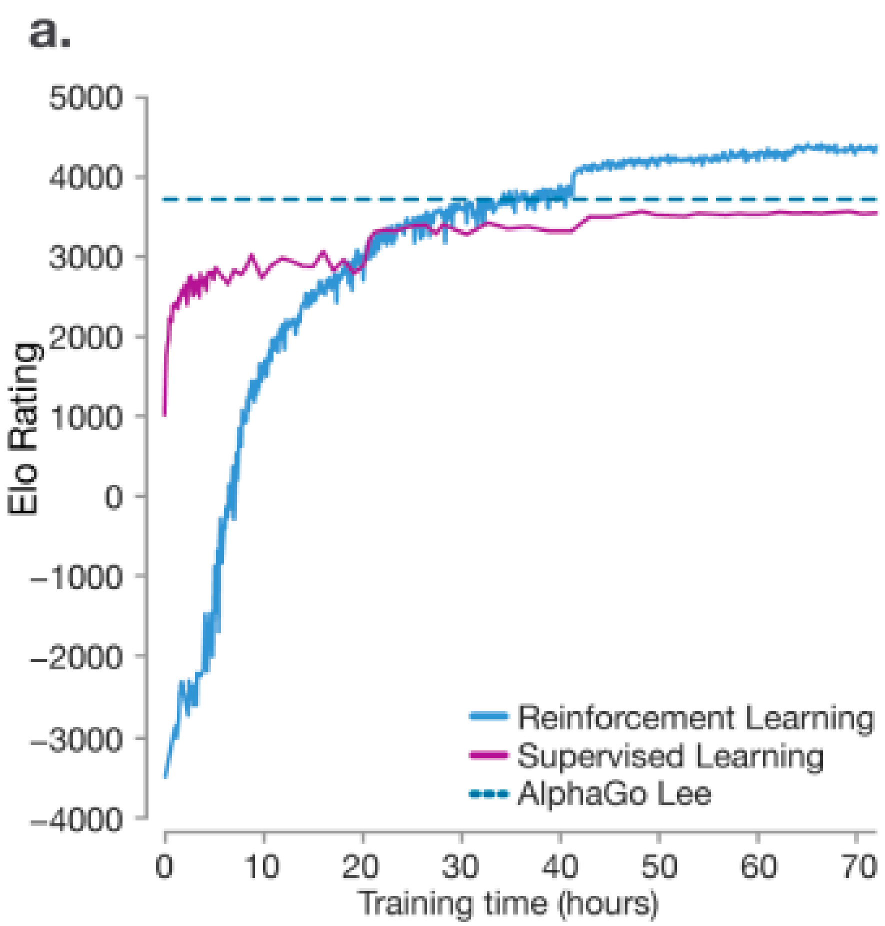
4.9 million Go games of self-play

0.4s thinking time per move

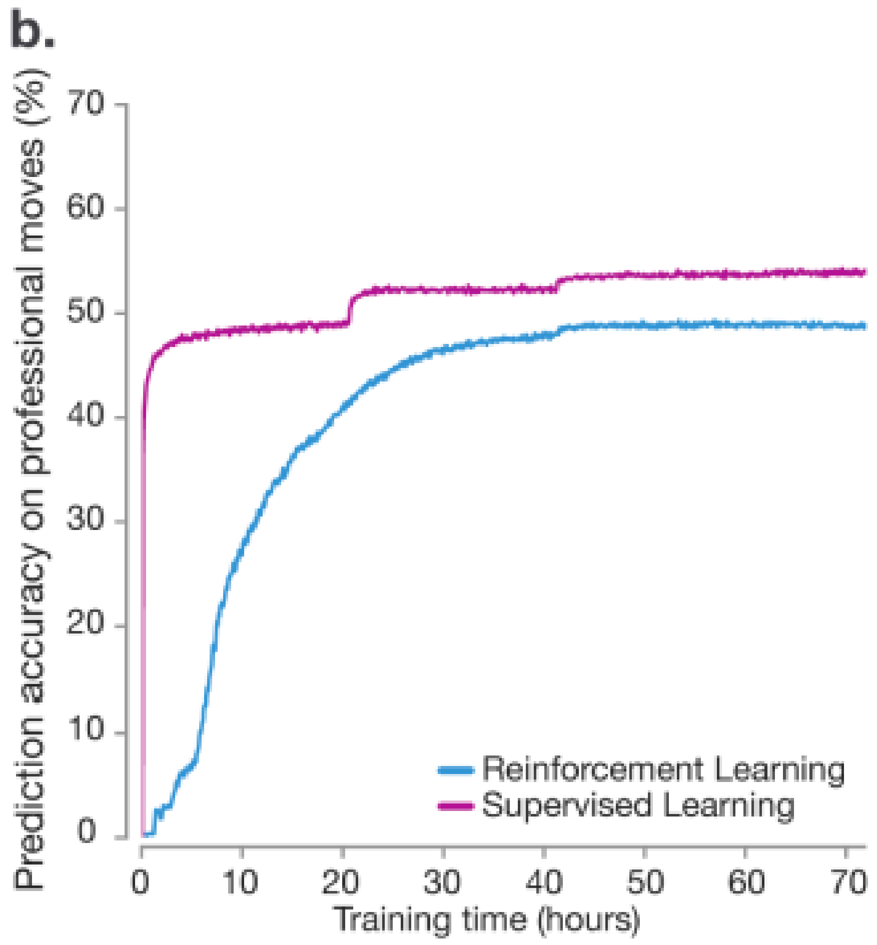
About 8 years of Go thinking time in training was completed in three days

About 1000 fold parallelism.

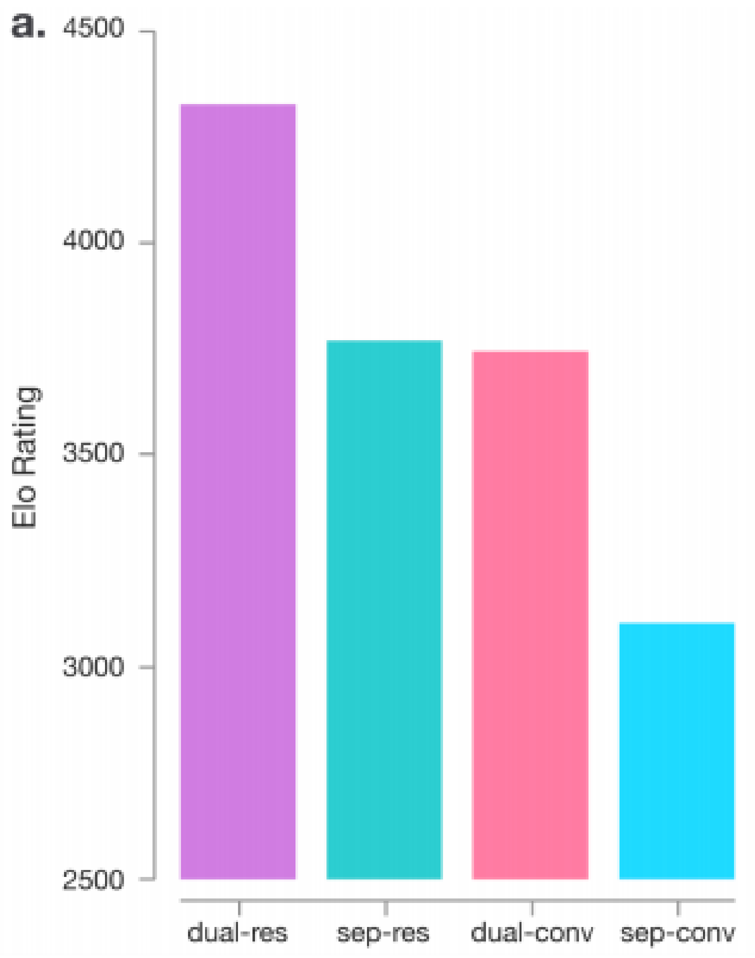
Elo Learning Curve for Go



Learning Curve for Predicting Human Go Moves



Ablation Study for Resnet and Dual-Head



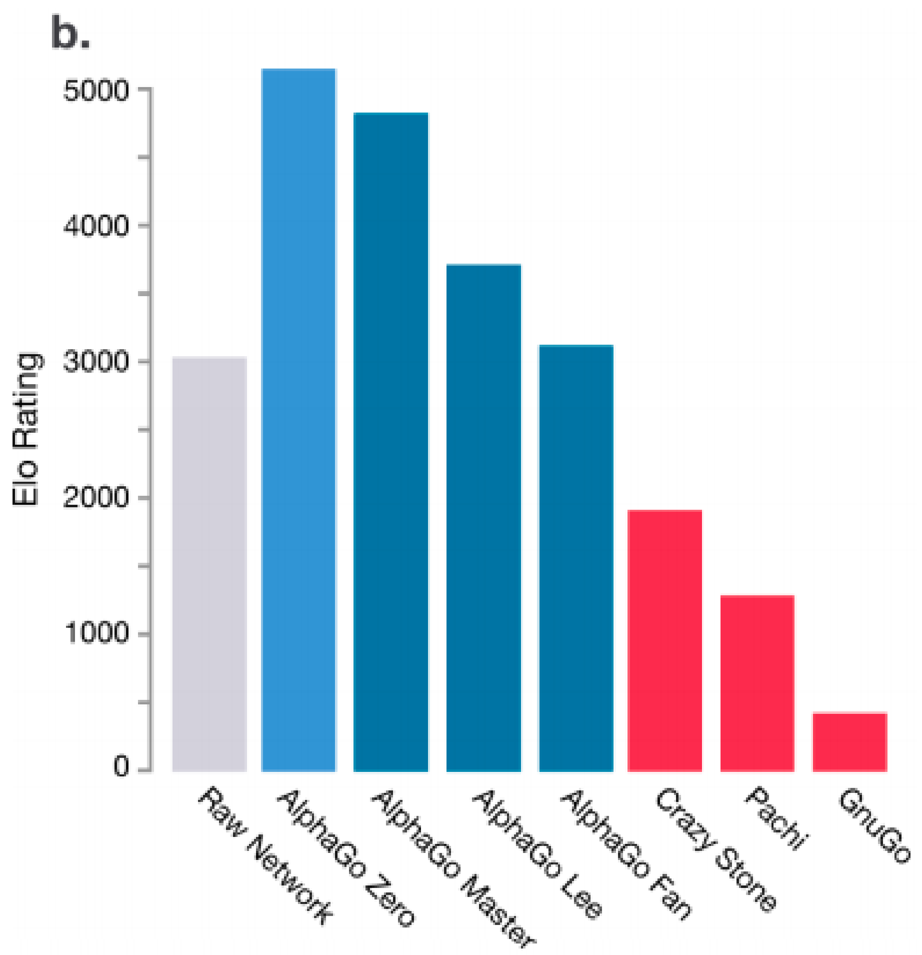
Increasing Blocks and Training

Increasing the number of Resnet blocks form 20 to 40.

Increasing the number of training days from 3 to 40.

Gives a Go Elo rating over 5000.

Final Go Elo Ratings



Is Chess a Draw?

In 2007 Jonathan Schaeffer at the University of Alberta showed that checkers is a draw.

Using alpha-beta and end-game dynamic programming, Schaeffer computed drawing strategies for each player.

This was listed by Science Magazine as one of the top 10 breakthroughs of 2007.

It is generally believed that chess is a draw. It was even conjectured that Stockfish could not be defeated ...

AlphaZero vs. Stockfish in Chess

From white Alpha won 25/50 and lost none.

From black Alpha won 3/50 and lost none.

AlphaZero evaluates 70 thousand positions per second.

Stockfish evaluates 80 million positions per second.

Analysis

Analysis

Simulations select $\mathrm{argmax}_a\ U(s,a)$.

$$U(s,a) = \begin{cases} \lambda_u\ \pi_\Phi(s,a) & \text{if } N(s,a) = 0 \\ \\ \hat{\mu}(s,a) + \lambda_u\ \pi_\Phi(s,a)/N(s,a) & \text{otherwise} \end{cases} \tag{1}$$

$$\Phi^* \ = \ \underset{\Phi}{\mathrm{argmin}}\ E_{(s,\pi,R)\sim\mathrm{Replay},\ a\sim\pi} \left(\begin{array}{l} (V_\Phi(s) - R)^2 \\ \\ -\lambda_\pi \log \pi_\Phi(a|s) \\ \\ +\lambda_R\ ||\Phi||^2 \end{array} \right) \tag{2}$$

Equation (2) establishes the meaning of $\pi_\Phi(a|s)$ as a stochastic policy.

Analysis

$$U(s,a) = \begin{cases} \lambda_u \, \pi_\Phi(s,a) & \text{if } N(s,a) = 0 \\ \\ \hat{\mu}(s,a) + \lambda_u \, \pi_\Phi(s,a)/N(s,a) & \text{otherwise} \end{cases} \tag{1}$$

But equation (1) then seems ill-typed — how can we add a reward and a probability?

Analysis

$$U(s,a) = \begin{cases} \lambda_u\ \pi_\Phi(s,a) & \text{if } N(s,a) = 0 \\ \hat{\mu}(s,a) + \lambda_u\ \pi_\Phi(s,a)/N(s,a) & \text{otherwise} \end{cases} \tag{1}$$

The types would work if we use $Q_\Phi(s,a)$ rather than $\pi_\Phi(s,a)$.

Analysis

$$\Phi^* \;=\; \underset{\Phi}{\mathrm{argmin}}\; E_{(s,\pi,R)\sim\mathrm{Replay},\; a\sim\pi} \left(\begin{array}{l} (V_\Phi(s) - R)^2 \\ \\ -\lambda_\pi \log \pi_\Phi(a|s) \\ \\ +\lambda_R \, ||\Phi||^2 \end{array}\right) \qquad (2)$$

But $\pi_\Phi(s,a)$ can model good moves accurately while modeling bad moves only well enough to determine that they are poor.

This seems difficult to duplicate with a Q function trained on some form of Bellman error such as the training of $V_\Phi(s)$.

What Happened to alpha-beta?

Grand Unification

AlphaZero unifies chess and go algorithms.

This unification of intuition (go) and calculation (chess) is surprising.

This unification grew out of go algorithms.

But are the algorithmic insights of chess algorithms really irrelevant?

Chess Background

The first min-max computer chess program was described by Claude Shannon in 1950.

Alpha-beta pruning was invented by various people independently, including John McCarthy in the late 1950s.

Alpha-beta has been the cornerstone of all chess algorithms until AlphaZero.

Alpha-Beta Pruning

```
def MaxValue(s,alpha,beta):
   value = alpha
   for s2 in s.children():
     value = max(value, MinValue(s2,value,beta))
     if value >= beta: break()
   return value

def MinValue(s,alpha,beta):
   value = beta
   for s2 in s.children():
     value = min(value, MaxValue(s2,alpha,value))
     if value <= alpha: break()
   return value
```

Strategies

An optimal alpha-beta tree is the union of a root-player strategy and an opponent strategy.

A strategy for the root player is a selection of a single action for each root-player move and a response for each possible action of the opponent.

A strategy for the opponent is a selection of a single action for each opponent move and a response for each possible action of the root player.

Proposal

Simulations should be divided into root-player strategy simulations and opponent strategy simulations.

A root-player strategy simulation is optimistic for the root player and pessimistic for the opponent.

An opponent strategy simulation is optimistic for the opponent player and pessimistic for the root-player.

Proposal

$$U(s,a) = \begin{cases} \lambda_u\, \pi_\Phi(s,a) & \text{if } N(s,a) = 0 \\ \hat{\mu}(s,a) + \lambda_u\, \pi_\Phi(s,a)/N(s,a) & \text{otherwise} \end{cases} \tag{1}$$

λ_u should be divided into λ_u^+ and λ_u^- with $\lambda_u^+ > \lambda_u^-$.

Simulations should be divided into two types — optimistic and pessimistic.

In optimistic simulations we use λ_u^+ for root-player moves and λ_u^- for opponent moves.

In pessimistic simulations we use λ_u^- for root-player moves and λ_u^+ for opponent moves.

END