TTIC 31230, Fundamentals of Deep Learning

David McAllester, April 2017

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 Algorithmic Concepts

Rollout position evaluation (Bruegmann, 1993)

Monte Carlo Tree Search (MCTS) (Bruegmann, 1993)

Upper Confidence Bound (UCB) Bandit Algorithm (Lai and Robbins 1985)

Upper Confidence Tree Search (UCT) (Kocsis and Szepesvari, 2006)

Rollouts and MCTS (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.

Take an average value of many rollouts.

Do a selective tree search using rollout averages for position evaluation.

(One Armed) Bandit Problems

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 UCB algorithm (1995 Version)

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 UCT algorithm (2006)

Build a search tree by running “simulations”.

Each simulation uses the UCB rule to select a child of each node until a leaf is reached.

The leaf is then expanded and a value is computed for the leaf.

This value is “backed up” through the tree adding a value and increment the count of each ancestor node.

AlphaGo

AlphaGo trained:

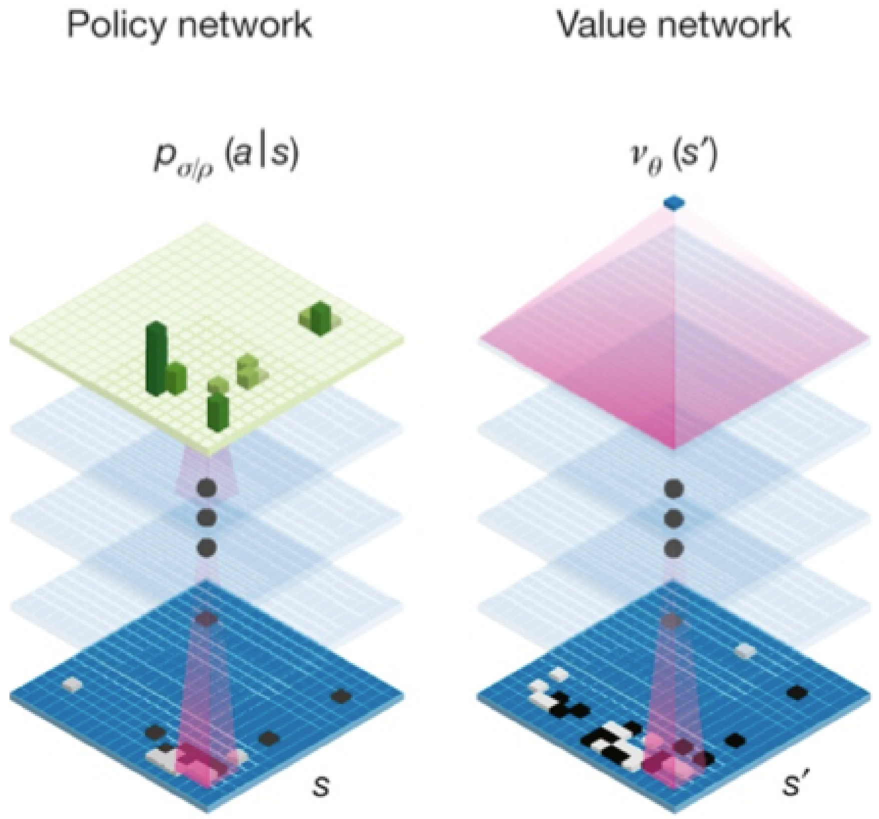
- a fast rollout policy.
- an imitation policy network.
- a self-play policy network.
- a value network trained to predict self-play rollout values.

AlphaGo

Competition play is done using UTC search using the four components just mentioned.

No tree search is used in training.

AlphaGo Policy and Value Networks



[Silver et al.]

The layers use 5×5 filters with Relu on 256 channels

Fast Rollout Policy

Softmax of linear combination of (hand designed) pattern features.

An accuracy of 24.2%, using just 2μs to select an action, rather than 3ms for the policy network.

Imitation Policy Learning

A 13-layer policy network trained from from 30 million positions from the KGS Go Server.

Self-Play Policy

Run the policy network against version of itself to get an (expensive) rollout $a_1, b_1, a_2, b_2, \ldots, a_N, b_N$ with value z.

No tree search is used here.

$$\Theta_\pi \mathrel{+}= z \, \nabla_{\Theta_\pi} \ln \pi(a_t | s_t; \Theta_\pi)$$

This is just REINFORCE.

Regression Training of Value Function

Using self-play of the final RL policy we generate a database of 30 million pairs (s, z) where s is a board position and $z \in \{-1, 1\}$ is an outcome and each pair is from a different game.

We then train a value network by regression.

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}}\, E_{(s,z)}\,(V(s,\Theta) - z)^2$$

Monte Carlo Tree Search (MCTS)

Competition play is then done with UCT search using the four predictors described above.

A simulation descends the tree using

$$\underset{a}{\operatorname{argmax}}\ Q(s,a) + cP(s,a)\frac{\sqrt{N(s)}}{1+N(a)}$$

where $P(s,a)$ is the **imitation learned** action probability.

Monte Carlo Tree Search (MCTS)

When a leaf is expanded it is assigned value

$$(1-\lambda)V(s)+\lambda z$$

where $V(s)$ is from the the self-play learned value network and z is value of a rollout from s using the fast rollout policy.

Once the search is deemed complete, the most traversed edge from the root is selected as the move.

AlphaGo Zero

- The self-play training is based on UCT tree search rather than rollouts.
- No rollouts are ever used — just UCT trees under the learned policy and value networks.
- No database of human games is ever used, just self-play.
- The networks are replaced with Resnet.
- A single dual-head network is used for both policy and value.

Training Time

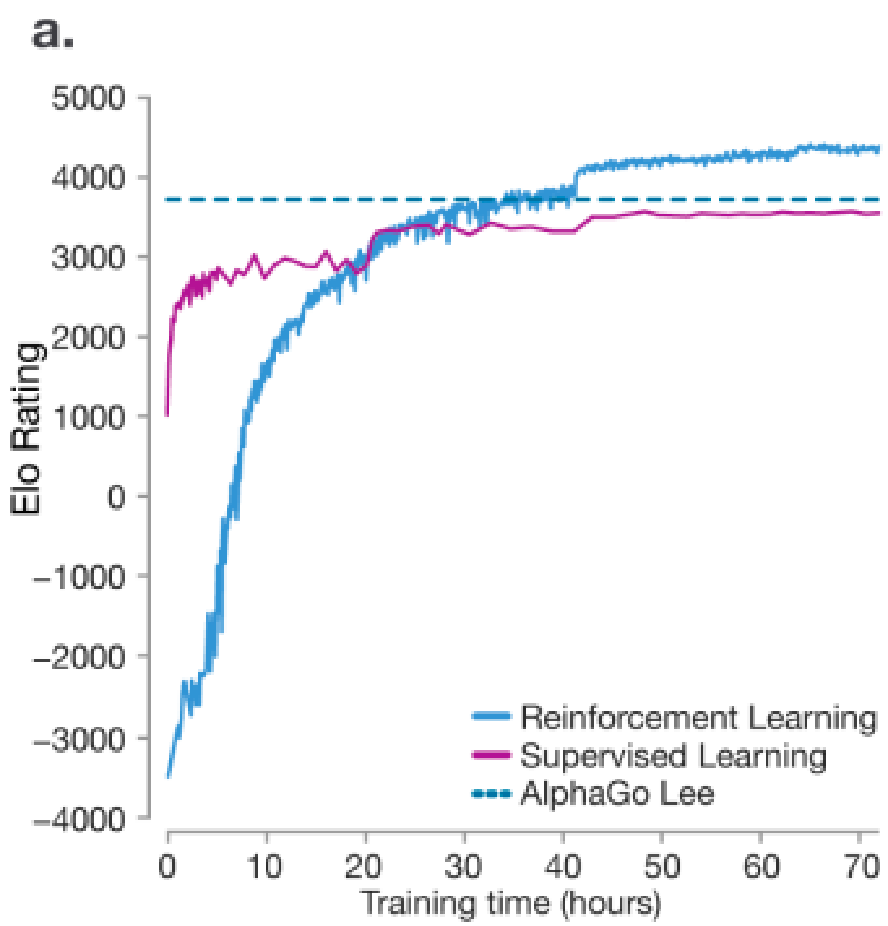
4.9 million games of self-play

0.4s thinking time per move

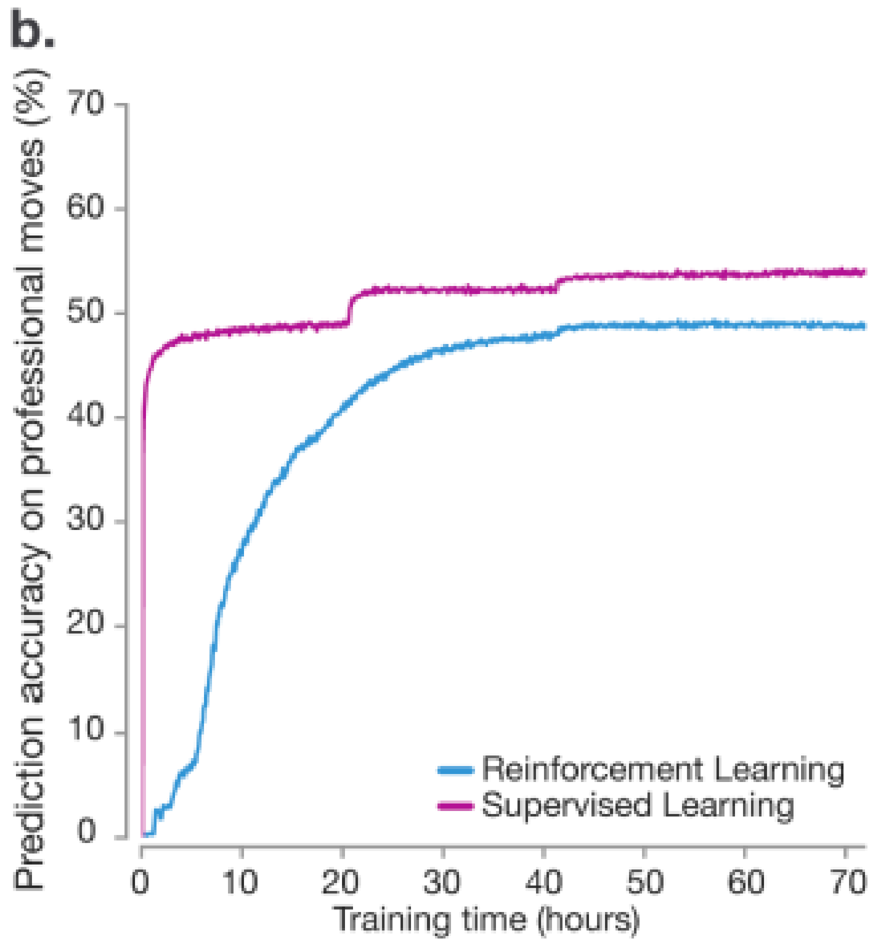
About 8 years of thinking time in training.

Training took just under 3 days — about 1000 fold parallelism.

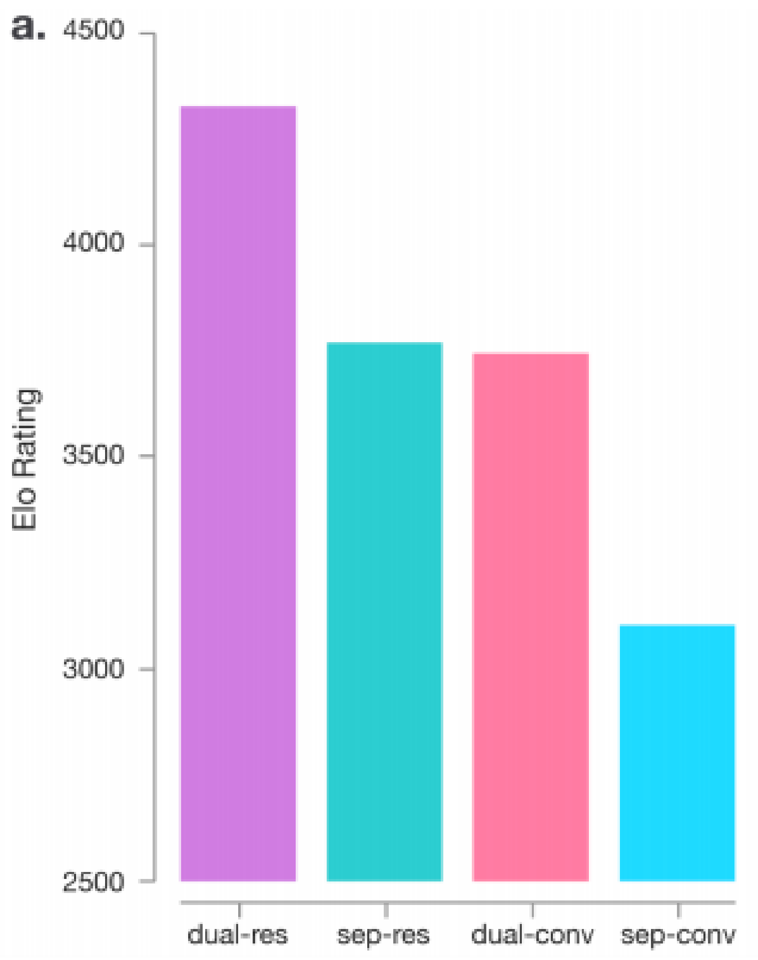
Elo Learning Curve



Learning Curve for Predicting Human Moves



Ablation Study for Resnet and Dual-Head



Learning from Tree Search

UTC tree search is used to generate a complete self-play game.

Each self-play game has a final outcome z and generates data (s, π, z) for each position s in the game where π is the final move probability of that position and z is the final value of the game.

This data is collected in a replay buffer.

Learning from Tree Search

Learning is done from this replay buffer using the following objective on a single dual-head network.

$$\Phi^* \;=\; \operatorname*{argmin}_{\Phi}\; E_{(s,\pi,z)\sim \text{Replay},\; a\sim\pi} \left(\begin{array}{l} (v_\Phi(s) - z)^2 \\ \\ -\lambda_1 \log Q_\Phi(a|s) \\ \\ +\lambda_2 ||\Phi||^2 \end{array} \right)$$

Exploration

Exploration is maintained by selecting moves in proportion to visit count for the first 30 moves rather than the maximum visit count.

After 30 moves the max count is selected.

Throughout the game noise is injected into the root move probabilities for each move selection.

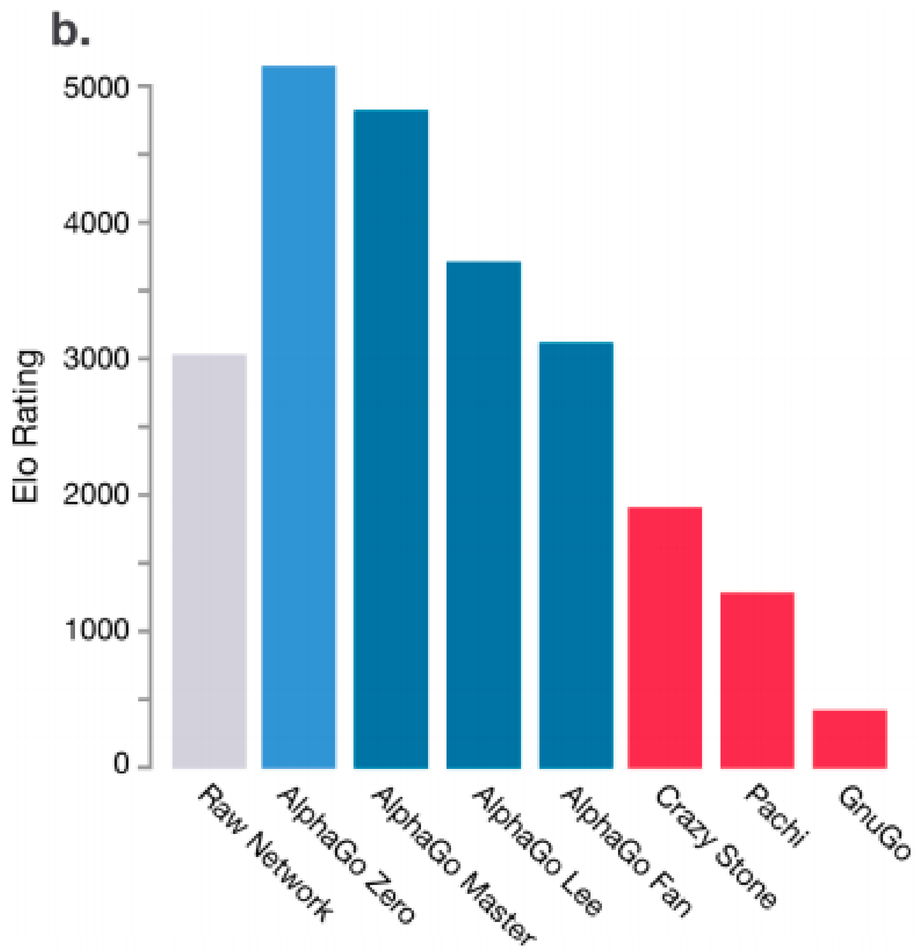
Increasing Blocks and Training

Increasing the number of Resnet blocks form 20 to 40.

Increasing the number of training days from 3 to 40.

Gives an Elo rating over 5000.

Final Elo Ratings



AlphaZero — Chess and Shogi

Essentially the same algorithm with the input image and output images modified to represent to game position and move options respective.

Minimal representations are used — no hand coded features.

Three days of training.

Tournaments played on a single machine with 4 TPUs.

Alpha vs. Stockfish

From white Alpha won 25/50 and lost none.

From black Alpha won 3/50 and lost none.

Alpha evaluates 70 thousand positions per second.

Stockfish evaluates 80 million positions per second.

Checkers is 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.

Is chess also a draw?

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, about 1956-1960.

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
```

Conspiracy Numbers

Conspiracy Numbers for Min-Max search, McAllester, 1988

Consider a partially expanded game tree where each leaf is labeled with a static value.

Each node s has a min-max value $V(s)$ determined by the leaf values.

For any N define an upper confidence $U(s,N)$ to be the greatest value that can be achieved for s by changing N leaf nodes.

We define $N(s,U)$ to be the least N such that $U(s,N) \geq U$.

Conspiracy Algorithm

Define an upper-confidence leaf for s and U any leaf that occurs in a set of $N(s,U)$ leaves that can change $V(s)$ to U.

Algorithm:

Fix a hyper-parameter N.

Repeatedly expand an upper-confidence leaf for the root S and value $U(s,N)$ and a lower-confidence leaf for s value $L(s,N)$.

Simulation

To find an upper-confidence leaf for the root and value U:

At a max node pick the child minimizing $N(s, U)$.

At a min node select any child s with $V(s) < U$.

Refinement

Let the static evaluator associate leaf nodes with values $U(s, N)$ and $L(s, N)$

END