TTIC 31230, Fundamentals of Deep Learning

David McAllester, Autumn 2023

AlphaZero, MuZero and AlphaStar

The Breakthrough in Go (October 2015)

Timeline

October 2015: AlphaGo-Fan Defeats Fan Hui, European Go Champion.

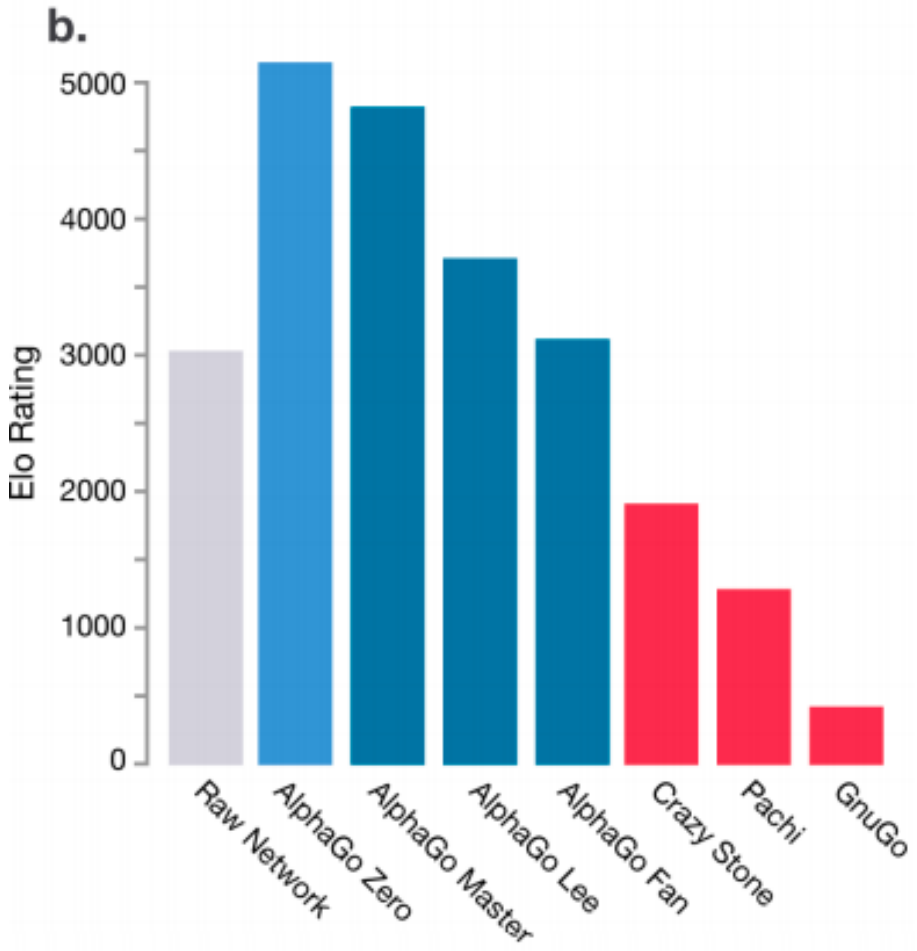
March 2016: AlphaGo-Lee Defeats Lee Sedol, world Go Champion.

April 2017: AlphaGo-Zero learns from self play only and defeats AlphaGo-Lee under match conditions 100 to 0.

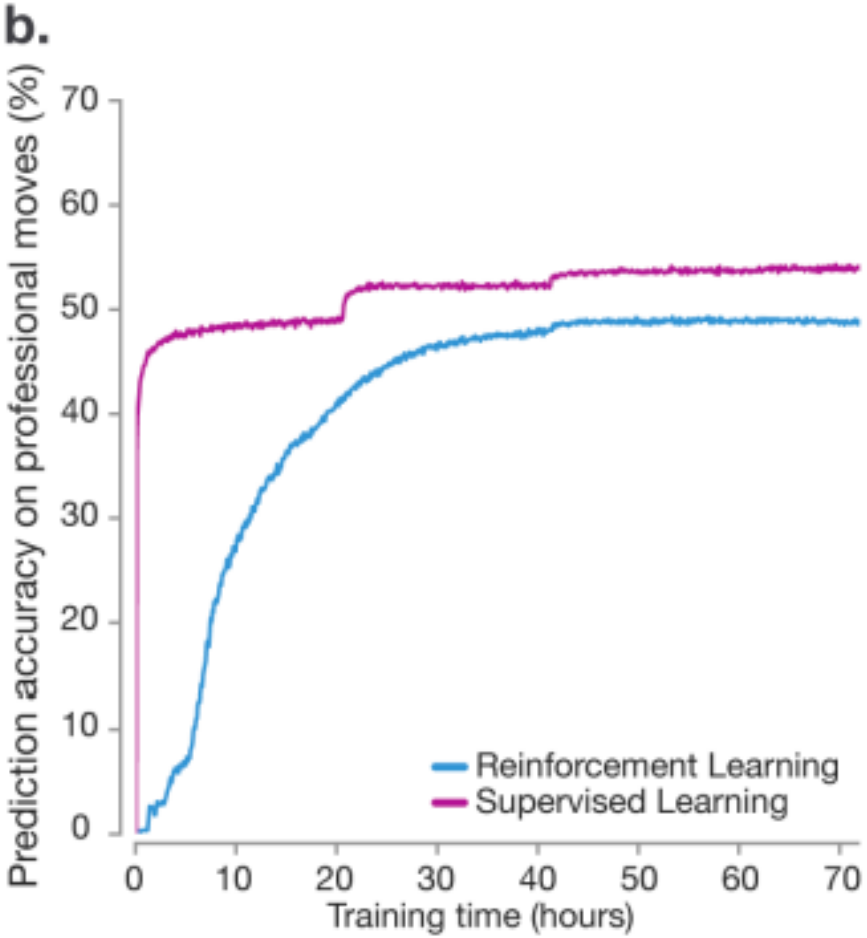
December 2017: AlphaZero Learns to play chess and shogi and defeats stockfish (already super-human) in Chess.

September 2024: The AlphaZero algorithm is used to train AlphaProof.

Go Elo Ratings (December 2017)



Learning Curve for Predicting Human Go Moves



Classical Computer Chess Algorithms

First computer chess algorithm (min-max tree search) — Claude Shannon, 1949

α-β pruning — various originators (including John McCarthy) circa 1960.

α-β pruning was the backbone of all computer chess before AlphaGo.

The branching factor in Go so large that alpha-beta tree search is infeasible.

Monte-Carlo Tree Search (MCTS)

Brugmann 1993

First major advance in computer Go.

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

Bandit problems were studied in an independent line of research.

Consider a set of choices. The standard example is a choice between different slot machines with different but unknown expected payout. The "phrase one-armed bandit" refers to a slot machine.

But another example of choices might be the moves in a game.

Bandit Problems

Consider a set of choices where 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 using a limited number of trials.

The Upper Confidence Bound (UCB) Algorithm

Lai and Robbins 1985

For each action choice (bandit) a, construct a confidence interval for its average reward based on n trials for that acrtion.

$$\mu(a) \in \hat{\mu}(a) \pm 2\sigma(a)/\sqrt{n(a)}$$

Always select

$$\underset{a}{\text{argmax}} \quad \hat{\mu}(a) + 2\sigma(a)/\sqrt{n(a)}$$

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 2009

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

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

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 Value and Policy Networks

The main innovation of AlphaGo and AlphaZero is to use deep networks for a value function and a policy function.

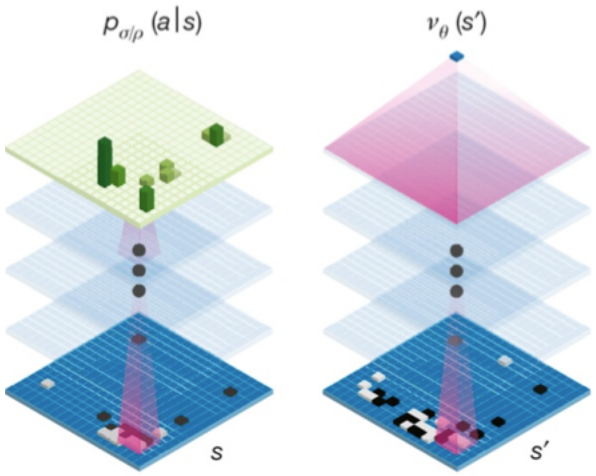
They also innovate with the structure of UCT and self-play bootstrapping.

The result is an extremely general RL algorithm.

The Value and Policy Networks

$\pi_\Phi(a|s)$ Policy network

$V_\Phi(s)$ Value network



The final AlphaZero system 40 a layer ResNet with both a policy head and a value head.

Tree Search Algorithm

Move selection involves a tree search.

Each node represents a board position s and stores the following information.

- $V_\Phi(s)$ — the value network value for the position s.
- The policy probabilities $\pi_\Phi(a|s)$ for each legal action a from that position.
- An initially empty set of children nodes.

Tree Search Algorithm

The tree is grown by a series of "simulations".

Each simulation starts at the root and recursively selects a move.

The selected move a from state s may or may not correspond to an existing child of s.

If the child exists the simulation continues down that path.

If the child does not exist a new child node is created for the selected move and the simulation terminates.

Each simulation adds one new leaf s and returns the value $V_\Phi(s)$ to all parents of s in the tree.

Tree Search Algorithm

In addition to $V_\Phi(s)$, $\pi_\Phi(a|s)$ and set of children nodes, each node s contains the following for each possible action a.

- The number $N(s,a)$ of simulations that have tried move a from s. This is initially zero.

- The average $\overline{V}(s,a)$ of $V_\Phi(s)$ plus the values returned by the the simulations that selected move a from position s. This averages $1 + N(s,a)$ numbers.

Simulations and Upper Confidence Bounds

At a node s a simulation selects the move $\text{argmax}_a \, \text{UCB}(s, a)$ where we have

$$\text{UCB}(s, a) = \overline{V}(s, a) + \lambda_u \, \pi_\Phi(a|s)/(1 + N(s, a))$$

We set λ_u be large enough that $\text{UCB}(s, a)$ will typically decrease as $N(s, a)$ increases.

Root Action Selection

When the search is completed, we must select a move from the root position to make actual progress in the game. For this we use a post-search stochastic policy

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

where β is a temperature hyperparameter.

Constructing a Replay Buffer

We run a large number of games.

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

- s is a position encountered in a game and hence 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 final outcome of the game for the player whoes move it is at position s.

The Loss Function

Training is done by SGD on the following loss function.

$$\Phi^* \;=\; \underset{\Phi}{\operatorname{argmin}}\; E_{(s,\pi,R)\sim\text{Replay},\, a\sim\pi} \begin{pmatrix} (V_\Phi(s) - R)^2 \\ -\lambda_\pi \log \pi_\Phi(a|s) \\ +\lambda_R \,||\Phi||^2 \end{pmatrix}$$

In principle this algorithm can be applied to any RL problem.

MuZero

Mastering Atari, Go, chess and shogi by planning with a learned model, Schrittweiser et al., Nature 2020.

Doing a tree search over possible actions requires knowing (or modeling) how a given action changes the state of the system.

For example, tree seach in chess requires knowing how an move changes the state.

MuZero does not assume a known state representation.

MuZero

Q-learning and advantage actor-critic do not require the ability to plan ahead (tree search).

But AlphaZero uses Monte-Carlo tree search (MCTS) to "plan" into the future.

MuZero uses the sequence of actions and observations as a representation of state.

This matches, but does not improve, playing Go and Chess.

But it improves the performance on Atari games by allowing tree search prior to action selection.

The Replay Buffer

A "state" is a sequence of observations (the game screen) and actions.

$$s = (o_1, a_1, \ldots, o_t, a_t)$$

They construct a replay buffer from rollouts using a (learned) action policy.

A replay entry e has the form

$$e = [\ s_t;\ \ a_{t+1},\ r_{t+1}, \ldots, a_{t+K}, r_{t+K},\ \ v_{t+K+1}\ ]$$

r_{t+i} is the observed reword at $t+i$ and v_{t+K+1} is the (learned) value network applied to state s_{t+K+1}

Training

The training loss function has the form

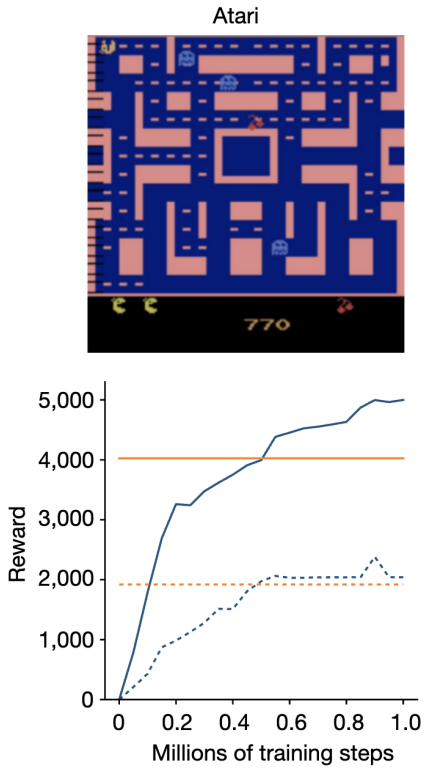
$$\Phi^* = \underset{\Phi}{\operatorname{argmin}}\, E_{s\sim\text{Rollout}}\, \mathcal{L}^\pi(s) + \mathcal{L}^V(s) + \mathcal{L}^R(s) + c||\Phi||^2$$

$\mathcal{L}^\pi$ trains the policy network to predict a_{t+1} (a cross-entropy loss).

$\mathcal{L}^V$ trains the value network to predict v_{t+K+1} (square loss).

$\mathcal{L}^R$ trains the reward network to predict each r_{t+k} (square loss).

Results



These are human normalized scores averaged over all 57 Atari games. The orange line is the previous state of the art system. Solid lines are average scores and dashed lines are median scores.

AlphaStar

Grandmaster level in StarCraft II using multi-agent reinforcement learning, Nature Oct. 2019, Vinyals et al.

StarCraft:

- Players control hundreds of units.
- Individual actions are selected from 10^{26} possibilities (an action is a kind of procedure call with arguments).
- Cyclic non-transitive strategies (rock-paper-scissors).
- Imperfect information — the state is not fully observable.

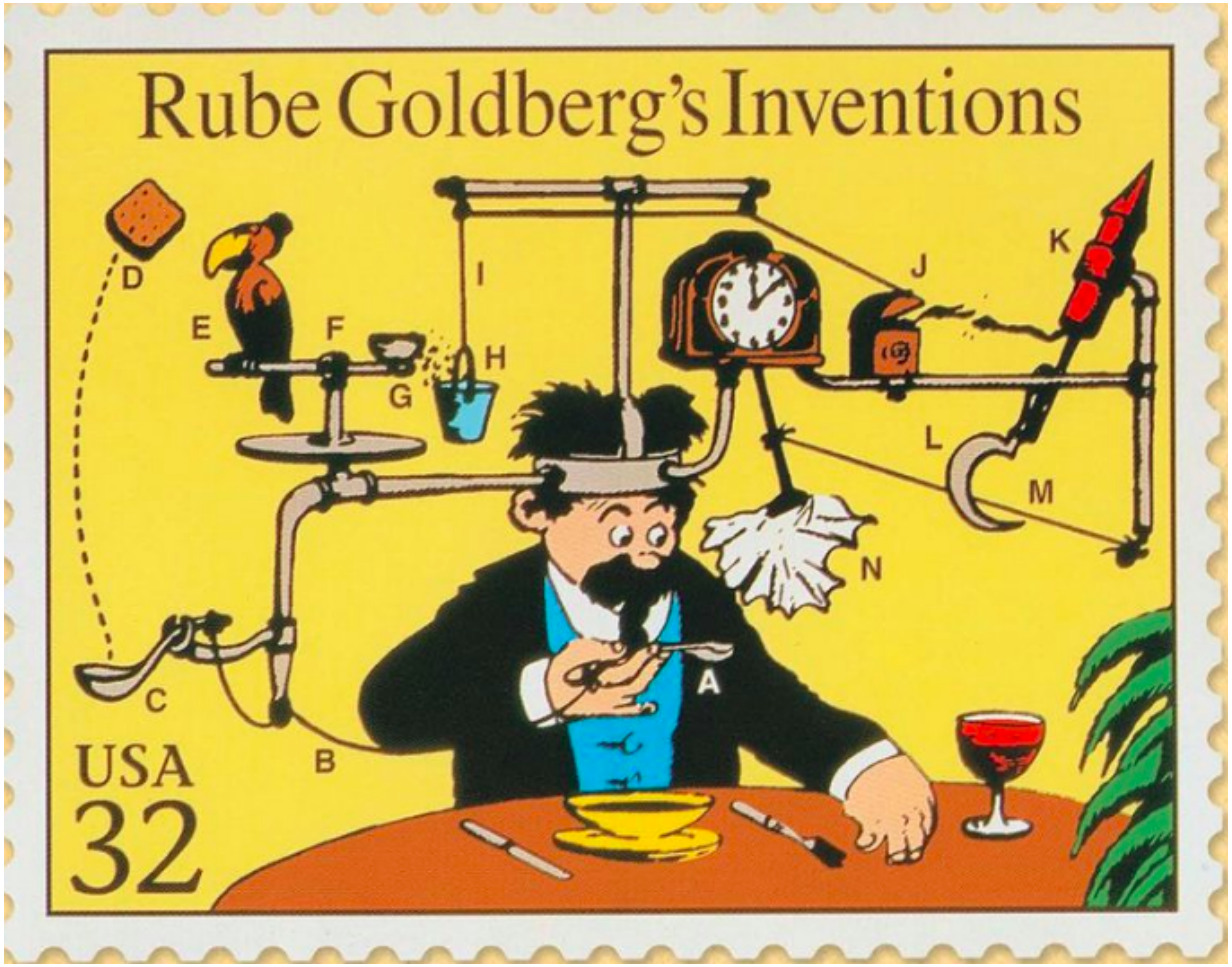
The Paper is Vague

It basically says the following ideas are used:
A policy gradient algorithm, auto-regressive policies, self-attention over the observation history,L STMs, pointer-networks, scatter connections, replay buffers, asynchronous advantage actor-critic algorithms, TD(λ) (gradients on value function Bellman error), clipped importance sampling (V-trace), a new undefined method they call UPGO that "moves policies toward trajectories with better than average reward", a value function that can see the opponents observation (training only), a "z statistic" stating a high level strategy, supervised learning from human play, a "league" of players (next slide).

The League

The league has three classes of agents: main (M), main exploiters (E), and league exploiters (L). M and L play against everybody. E plays only against M.

A Rube Goldberg Contraption?



Video

https://www.youtube.com/watch?v=UuhECwm31dM

END