TTIC 31230, Fundamentals of Deep Learning

David McAllester, Autumn 2023

AlphaZero, MuZero and AlphaStar

AlphaGo Fan (October 2015)

AlphaGo Defeats Fan Hui, European Go Champion.

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

Early Computer Chess

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 2015.

The game of Go is clearly not approachable by these methods.

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, NeurIPS 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

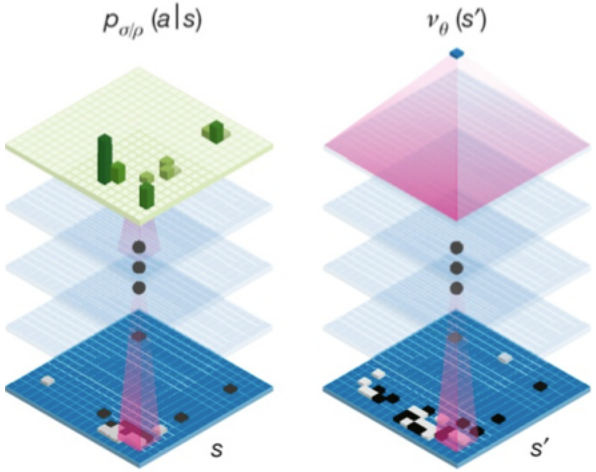
The major innovation of AlphaGo and AlphaZero is to use CNNs as evaluation functions.

We have a policy network computing $\pi_\Phi(a|s)$ and a value network computing $V_\Phi(s)$.

The Value and Policy Networks

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

$V_\Phi(s)$ Value network



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

Top Level Algorithm

To play a game (against yourself) select a move at each position.

Each move selection involves growing a tree of possible future positions.

The tree is "sparse" in the sense that the children of a node are a (possibly empty) subset of the possible next positions.

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

Each simulation starts at the root and recursively selects a move at each node until the selected move adds a new node to the tree.

Simulations

During a simulation moves are selected by maximizing an upper value as in the UCT algorithm.

Each simulation adds one new node to the tree.

Each simulation returns the value $V_\Phi(s)$ (from the value network) for the newly generated node.

The Data Structures

Each node in the tree data structrure stores the following information which is 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 $\pi_\Phi(a|s)$ for each legal action a.
- The number $N(s,a)$ of simulations that have tried move a from s. This is initially zero.
- The average $\hat{\mu}(s,a)$ of $V_\Phi(s)$ and 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 $\text{argmax}_a U(s,a)$ where we have

$$U(s,a) = \hat{\mu}(s,a) + \lambda_u \, \pi_\Phi(a|s)/(1+N(s,a))$$

We have that $U(s,a)$ will typically decrease as $N(s,a)$ increases.

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 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.

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

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

The AlphaZero Algorithm

The AlphaZero algorithm is a general RL learning method. It can be applied to any problem of sequential decision making.

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

Training Time

Single 20 block dual-headed ResNet.

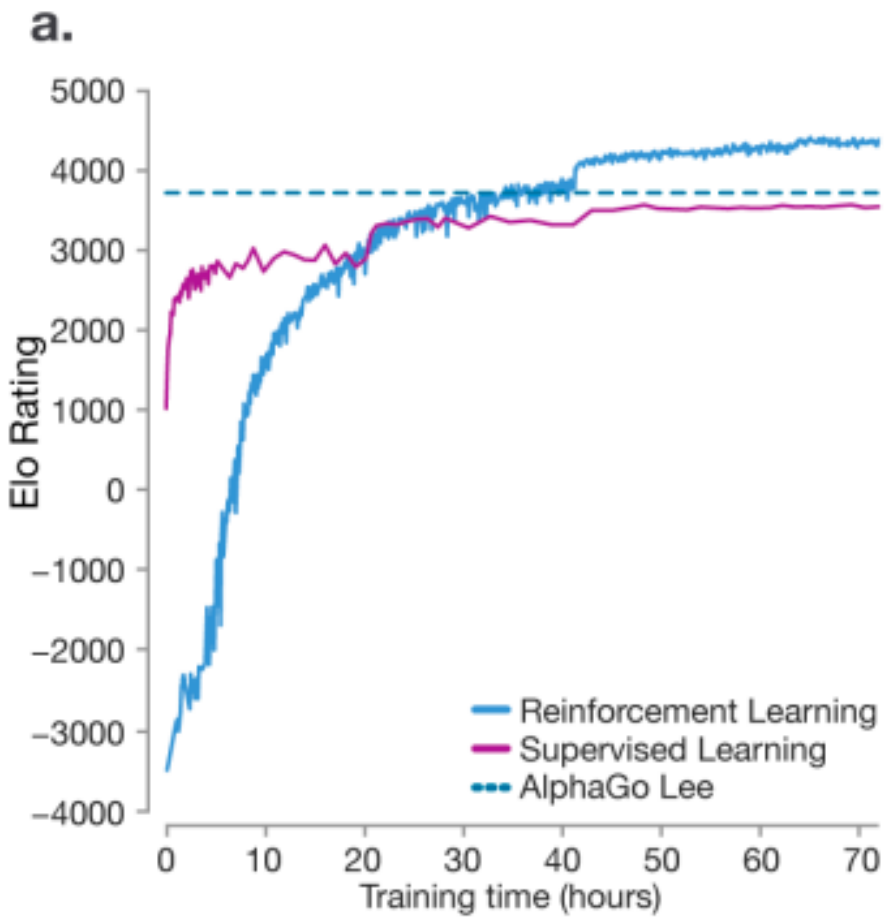
4.9 million Go games of self-play

0.4s thinking time per move

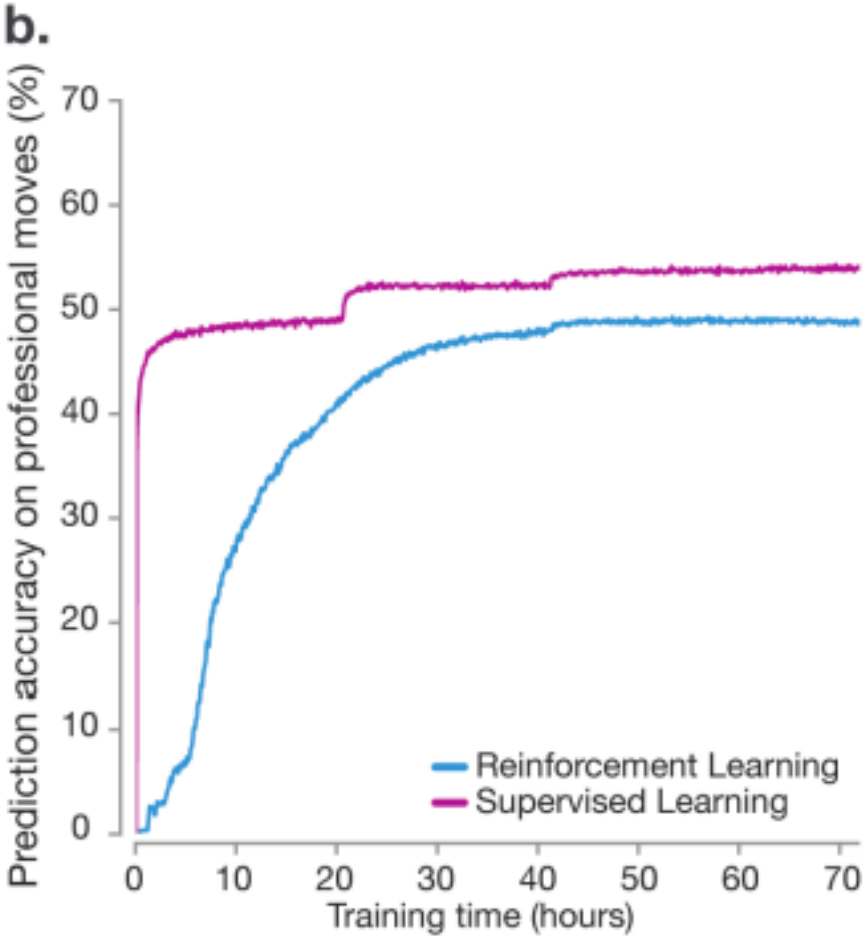
About 8 years of Go thinking time in training was completed in three days

About 1000 fold parallelism.

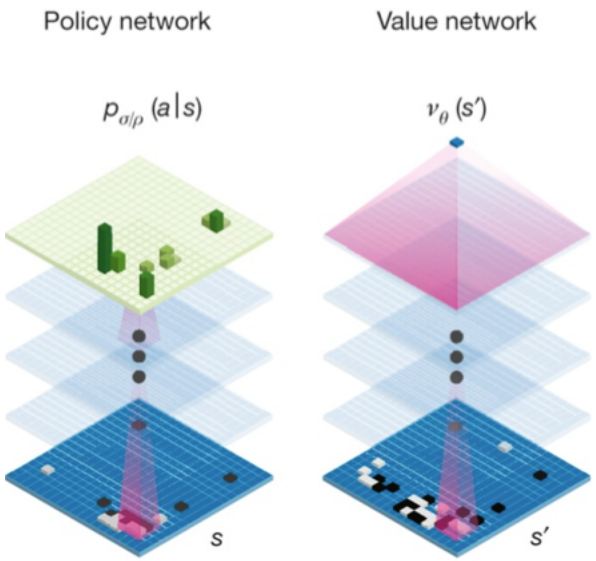
Elo Learning Curve for Go



Learning Curve for Predicting Human Go Moves

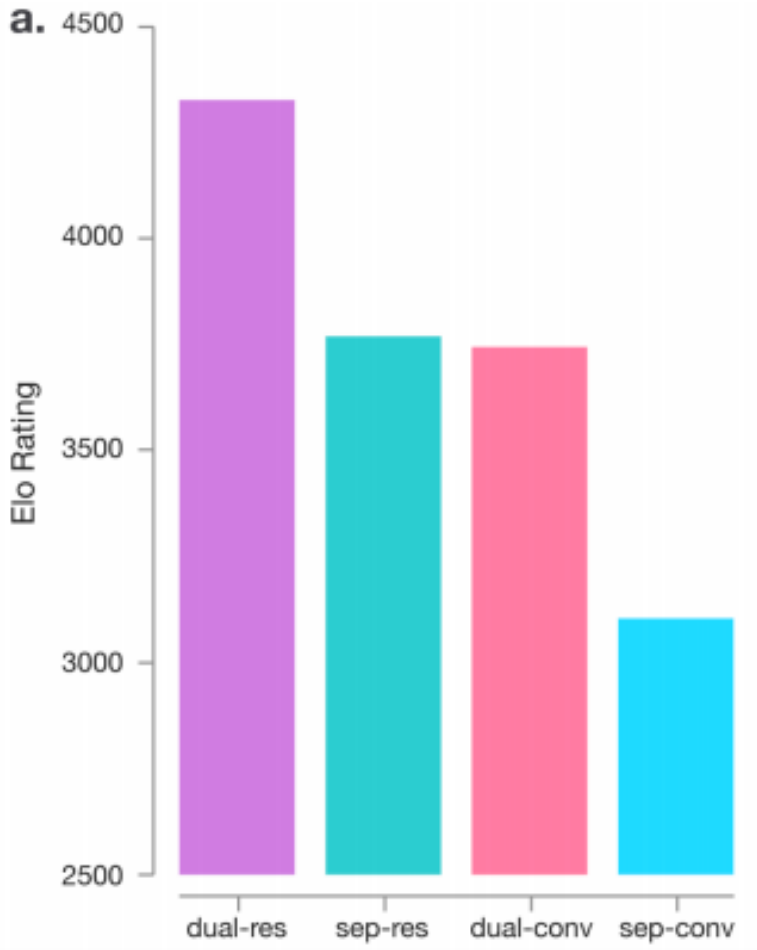


Ablation Studies



We evaluate 20 layer networks which are either traditional CNNs or resnet and are either two separate networks or one network with dual heads.

Ablation Studies



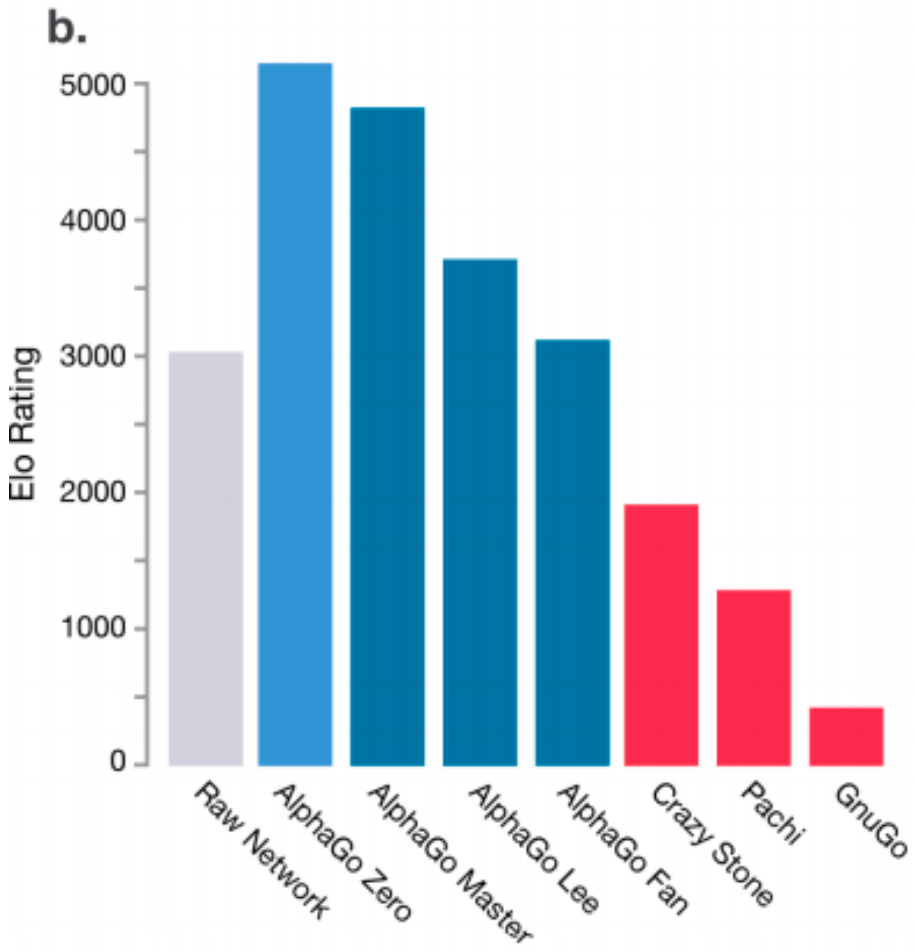
Increasing Blocks and Training

Increasing the number of Resnet blocks from 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.

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 ti $t+i$ 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

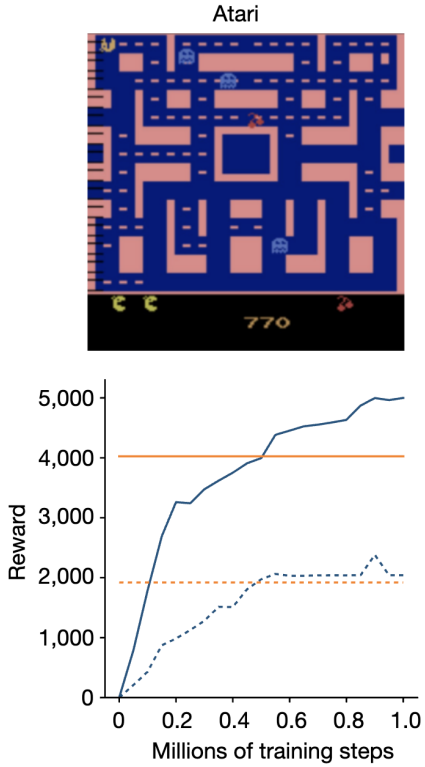
$$\Phi^* = \underset{\Phi}{\operatorname{argmin}}\, E_{s\sim\text{Rollout}}\, \mathcal{L}^\pi(s) + \mathcal{L}^V(s) + \mathcal{L}^R(s) + c||\Phi||^2$$

$\mathcal{L}^\pi$ 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.

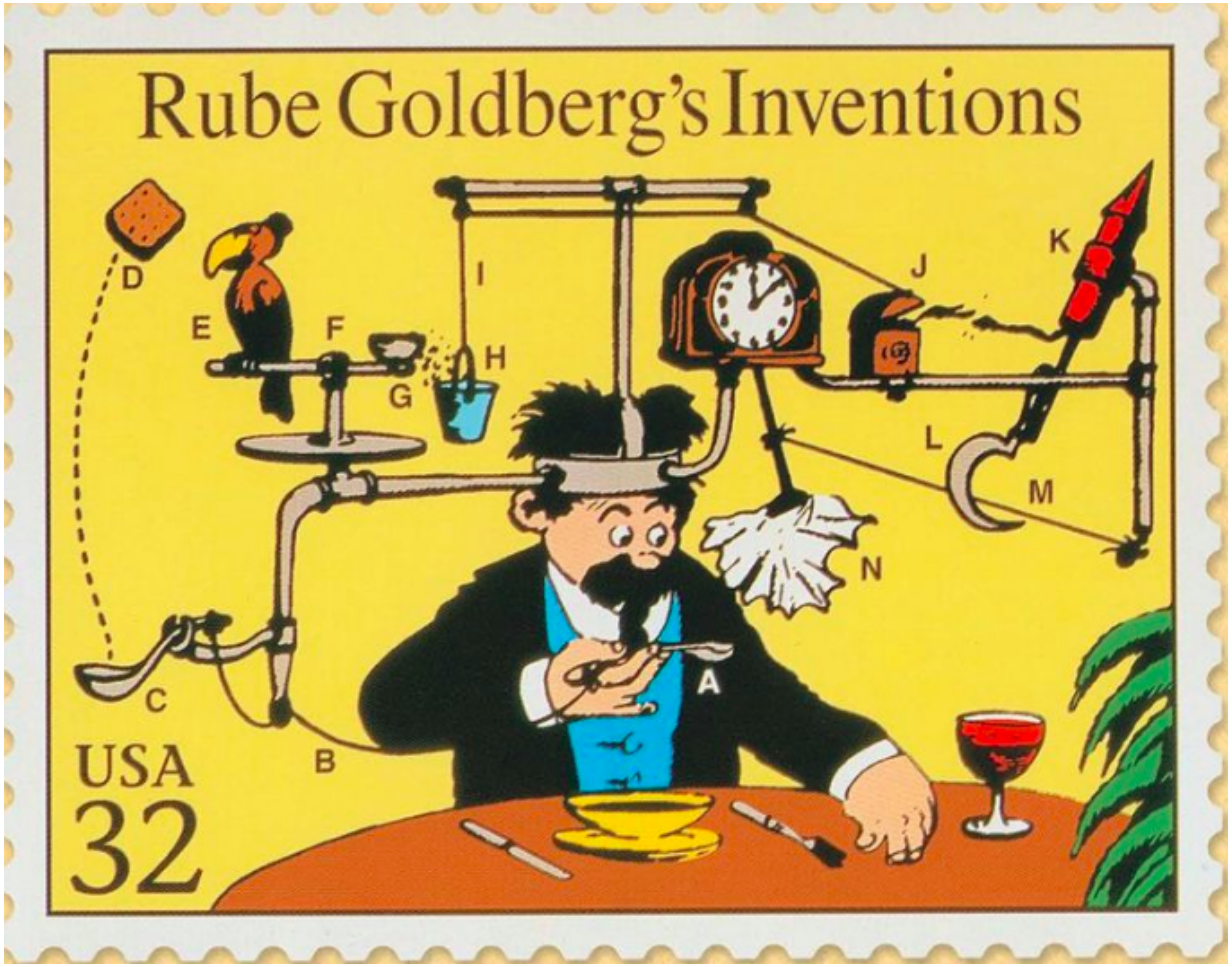
The Paper is Vague

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