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

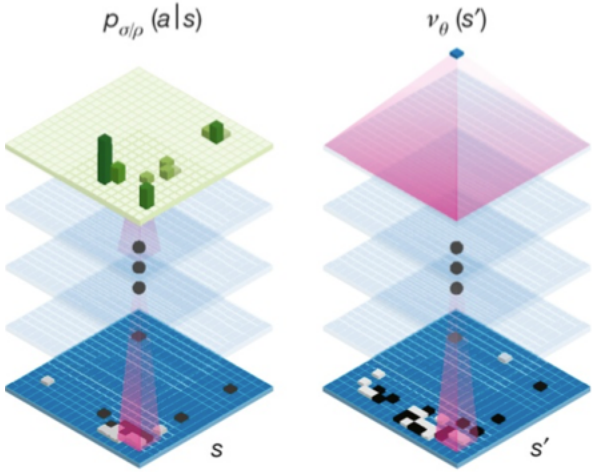
The major innovation of AlphaGo and AlphaZero is to use CNNs as evaluation functions.

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.

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.

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

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.

The AlphaZero algorithm is used to train AlphaProof (September 2024).

Training Time

Single 20 block dual-headed ResNet.

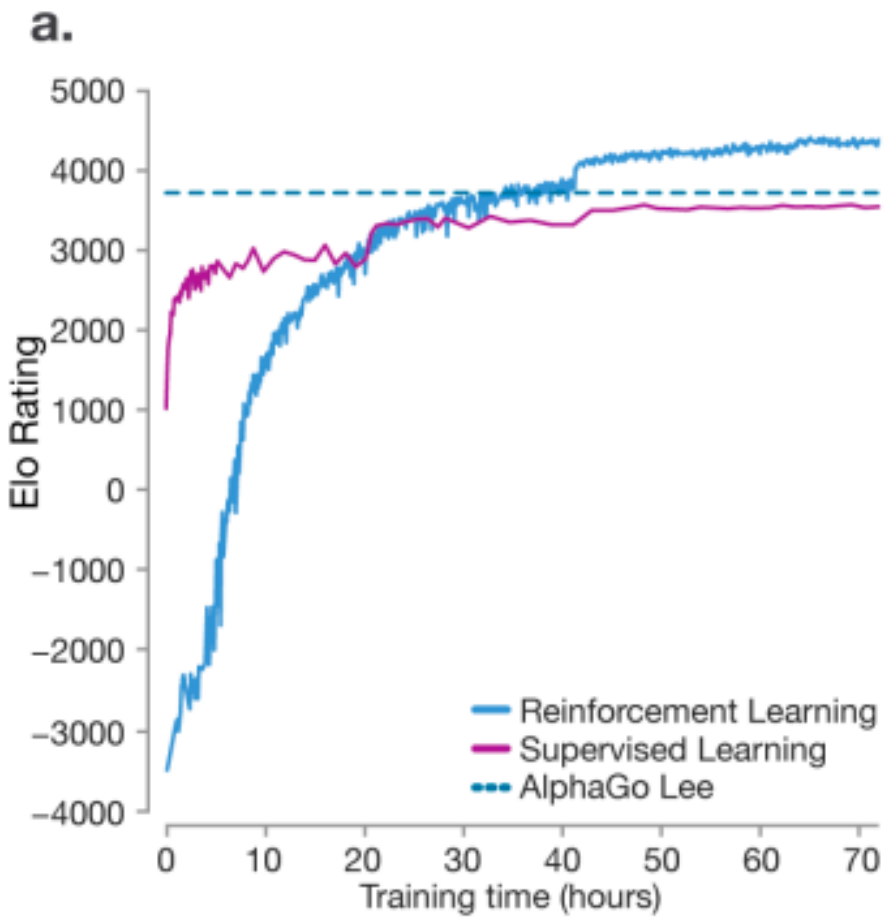
4.9 million Go games of self-play

0.4s thinking time per move

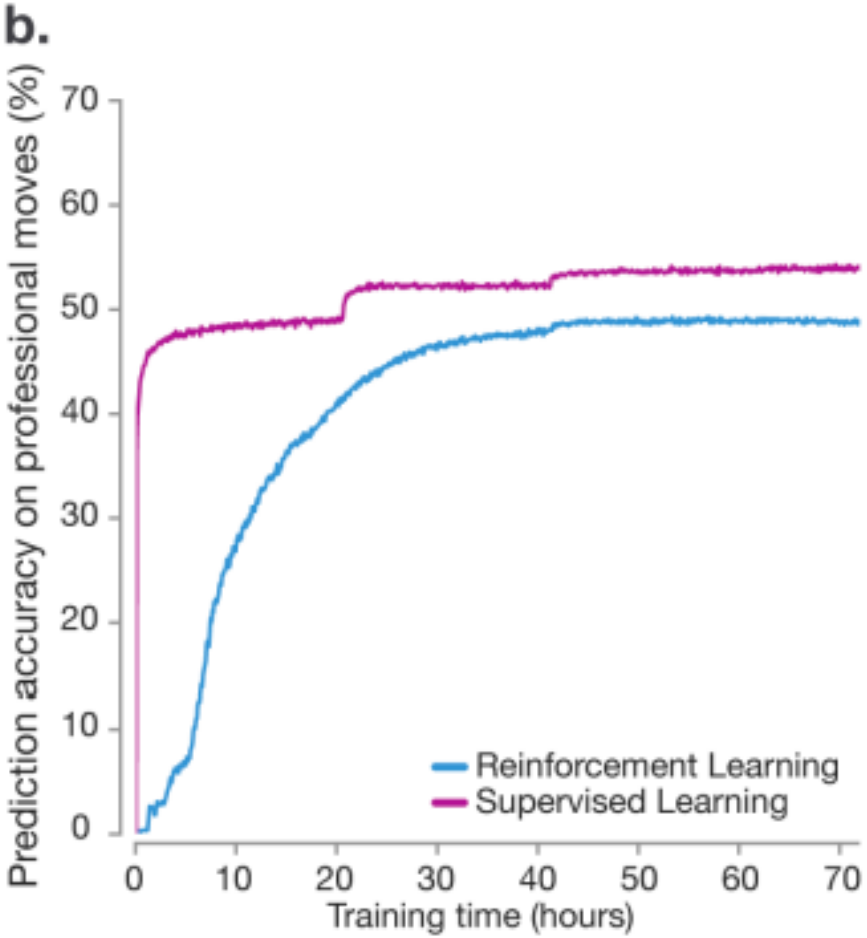
About 8 years of Go thinking time in training was completed in three days

About 1000 fold parallelism.

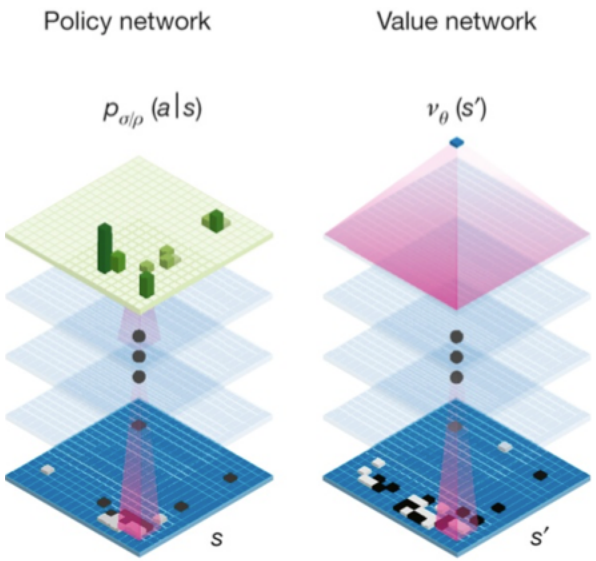
Elo Learning Curve for Go



Learning Curve for Predicting Human Go Moves

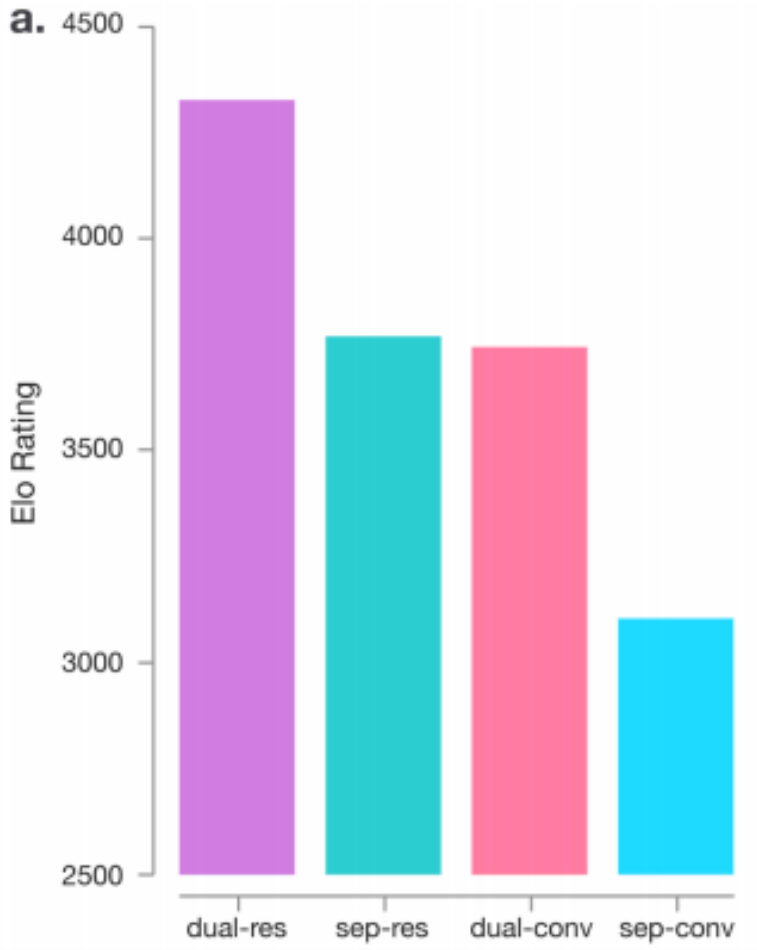


Ablation Studies



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



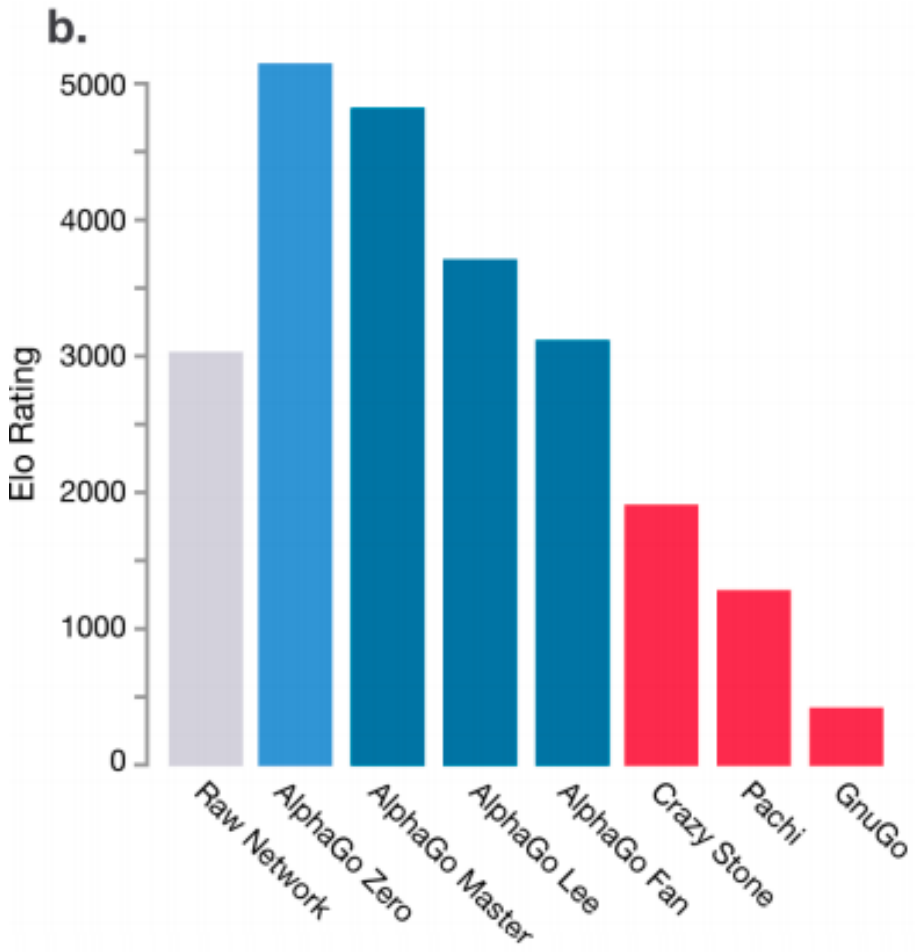
Increasing Blocks and Training

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 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

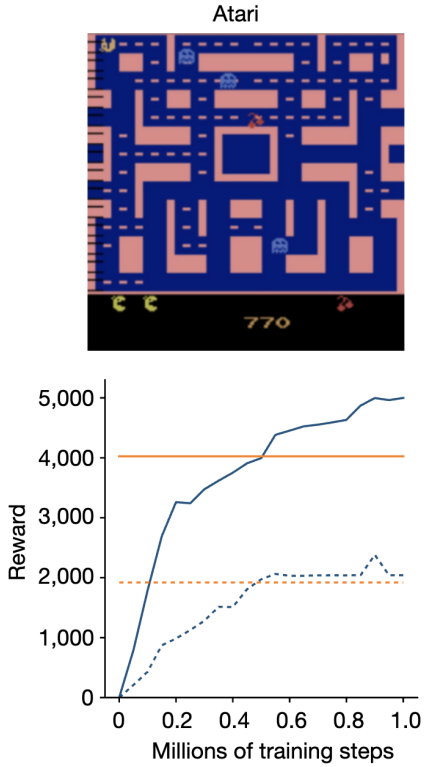
$$\Phi^* = \underset{\Phi}{\operatorname{argmin}}\, E_{s\sim\text{Rollout}}\, \mathcal{L}^\pi(s) + \mathcal{L}^V(s) + \mathcal{L}^R(s) + c||\Phi||^2$$

$\mathcal{L}^\pi$ 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.

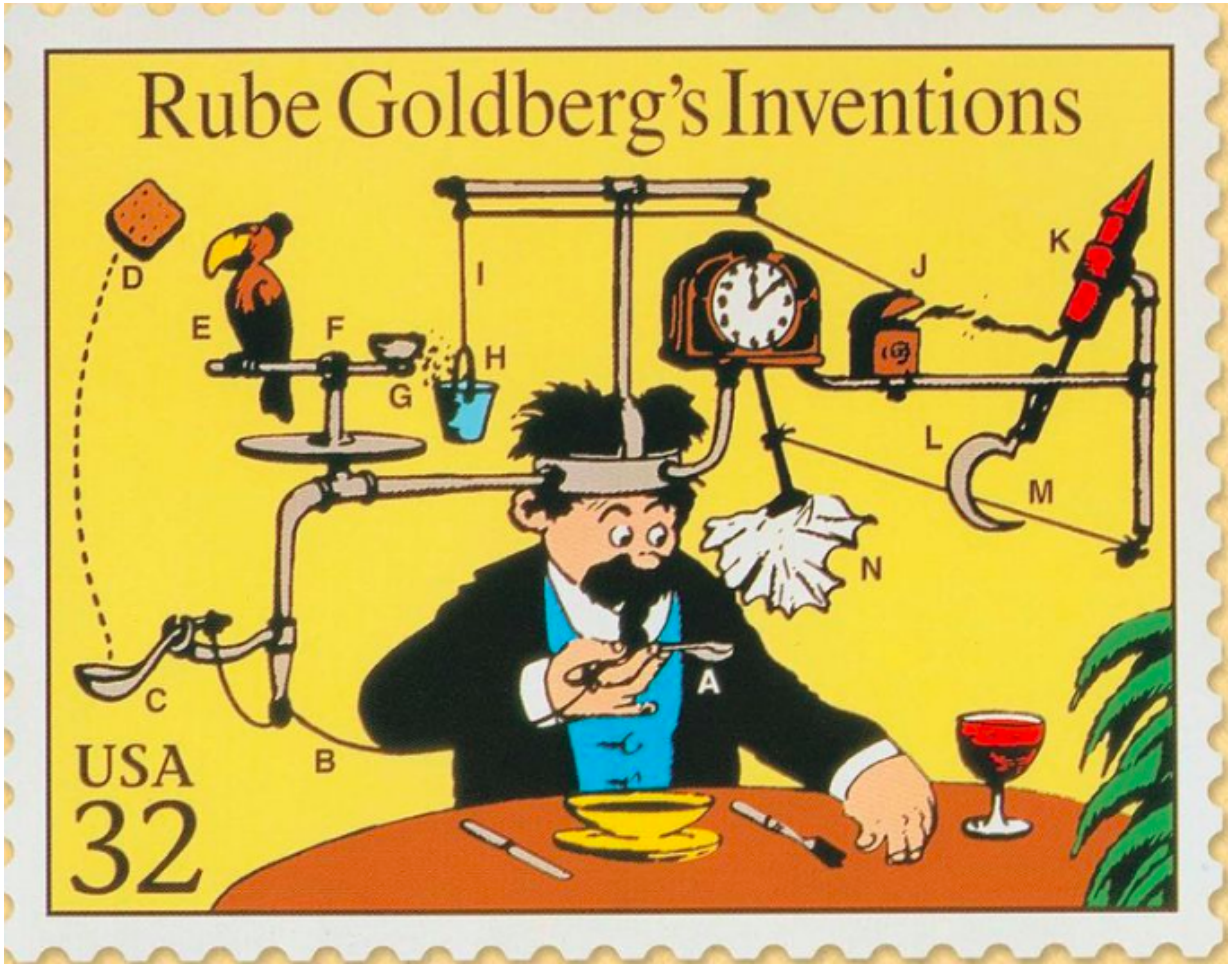
The Paper is Vague

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?



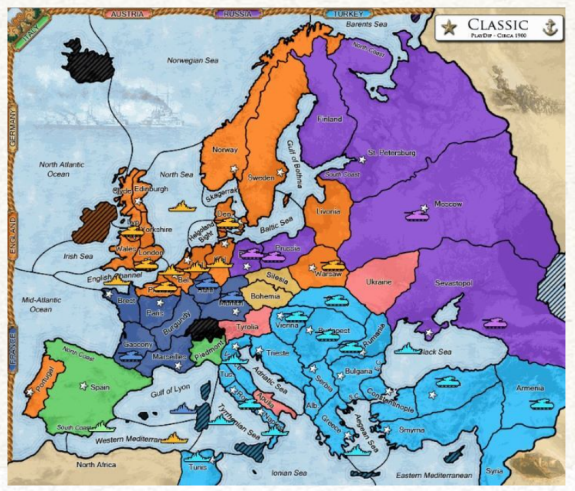
Video

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

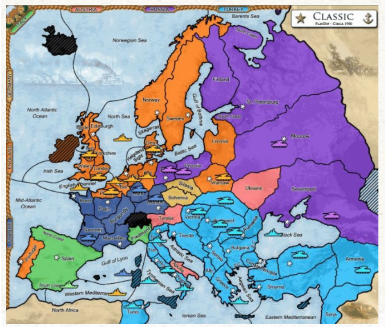
Cicero: Diplomacy

Human-level play in the game of Diplomacy by combining language models with strategic reasoning, (Meta, November 2022)

Players engages in secret negotiaions to conspire to take over the world.



Cicero: Diplomacy



The play consists of a sequence of "turns". In each turn each player converses with the others using a private messaging system. After some time the messaging ends and each player submits orders to their generals. The orders are revealed simultaneously. Promises can be broken.

Cicero: Diplomacy

Cicero is an AI agent that plays Diplomacy with humans. It is rated in the top 10% of human players.

Action Planning

We define the action of a player to be the set of orders that the player gives to their generals at the end of the turn.

Cicero plans its actions for the current turn and generates "beliefs" about the actions of the other players (mathematically equivalent to stochastic policies for the other players). Messages are generated from Cicero's planned actions and beliefs. The planned actions and beliefs can evolve during negotiation.

A Value Function

We seek a winning strategy (policy) for the agent Cicero.

A choice of actions $a_1, \ldots, a_n$ for all the players determines the next board state.

Cicero trains a value function that assigns a value $V_q(b)$ giving an estimate of the value that player q receives for board position b.

As policies are trained and estimated, the value function can be trained with standard methods — Bellman error and replay buffers.

Correlated Nash Equilibria

At the end of negotiation it is natural to search for policies, $\pi_1, \ldots, \pi_n$ conditioned on the negotiation that form a (correlated) Nash equilibrium.

This mean that each policy is a "best response" to other players policies where all policies are conditioned on the negotiation.

It is "correlated" because the actions of two players are not independent when they can negotiate an agreement.

Training Data

Cicero is trained on a dataset of 12,901,662 games from Web-Diplomacy including messages exchanged between players. (web-Diplomacy deidentified Player accounts).

An Imitation Model of Action Prediction

We want to predict actions during negotiation.

$$\text{im}^* = \underset{\text{im}}{\text{argmin}} \; E_{(x_p,a_q)\sim\text{Train}} \left[-\ln P_{\text{im}}(a_q|x_p) \right]$$

Here p and q are players. x_p is taken from a time in a negotiation and consists of all past board positions and the previous messages in this negotiation visible to player p. a_q is the action taken at the end of that round by player q.

If $q = q$ this is an imitation action policy for p.

If $q \neq p$ this is p's imitation belief about q.

piKL

Let c denote player Cicero.

Before sending each negotiation message Cicero samples 30 actions for each player q from the imitation action predictions $P(a_q|x_c)$ (including actions for Cicero).

Cicero then optimizes policies $\pi_1, \ldots, \pi_n$ according to a piKL best-response objective where π_{-p} is the collection of policies other than π_p.

$$\pi_p^* = \underset{\pi_p}{\text{argmax}}\ V(\pi_p, \pi_{-p}) - \lambda KL(\pi_p, P_{\text{im}}(a_p|x_c)$$

Imitation Message Generation

Cicero has a model $P_{\text{mess}}(y|s, r, a_s, a_r, x_s)$ where where y is the English text of the message, s is the message sender, r is the message recipient, a_s is the action taken by the message sender and a_r is the action taken by the message receiver at the end of the turn (in the future).

x_s consists of all past board positions and the previous messages visible to the sender in the current negotiation.

Imitation Message Generation

$$\text{mess}^* = \underset{\text{mess}}{\text{argmin}} \; E_{\{(x,s,r,y,a_s,a_r)\sim\text{Train}\}} \left[-\ln P_{\text{mess}}(y|s,r,a_s,a_r,x_s) \right]$$

An Imitation Agent

$$P_{\text{im}}(y|s,r,x_s) = E_{\{a_s,a_r \sim P_{\text{act}}(a_s,a_r|s,r,x_s)\}}\, P_{\text{mess}}(y|s,r,a_s,a_r,x_s)$$

In Cicero the imitation model is combined with a planning model.

Action Planning

??

The trained policies come two versions — an expensive version conditioned on the board state plus dialogue, and a cheaper version conditioned just on the board state.

Selecting the End Of Turn Action

For each turn Cicero first decides on its final action (the orders that it will give at the end of the turn) and then generates messages

The sends messages specified by intents. This is done by deciding on an "action" which specifies all of the orders that the agent will actually give.specifying all of the payers orders where x is as before (the history of board positions, and the history of messages before the message to be generated.

A message prediction model $P_{\text{mess}}(y|A, B, I, x)$ where A and B are two players, y is a message from A to B, I is a (latent) "intent" of the message, and x includes the history of board states the message and the messaged in the current turn occuring before this message.

Step 2: They get a base model — a 2.7B parameter transformer model pre-trained on text from the internet.

Step 3: The base model is fine-tuned to predict messages from the board state, the previous messages negotiation in the WebDiplomacy dataset.

⋮

Step 4: Inferring Intents

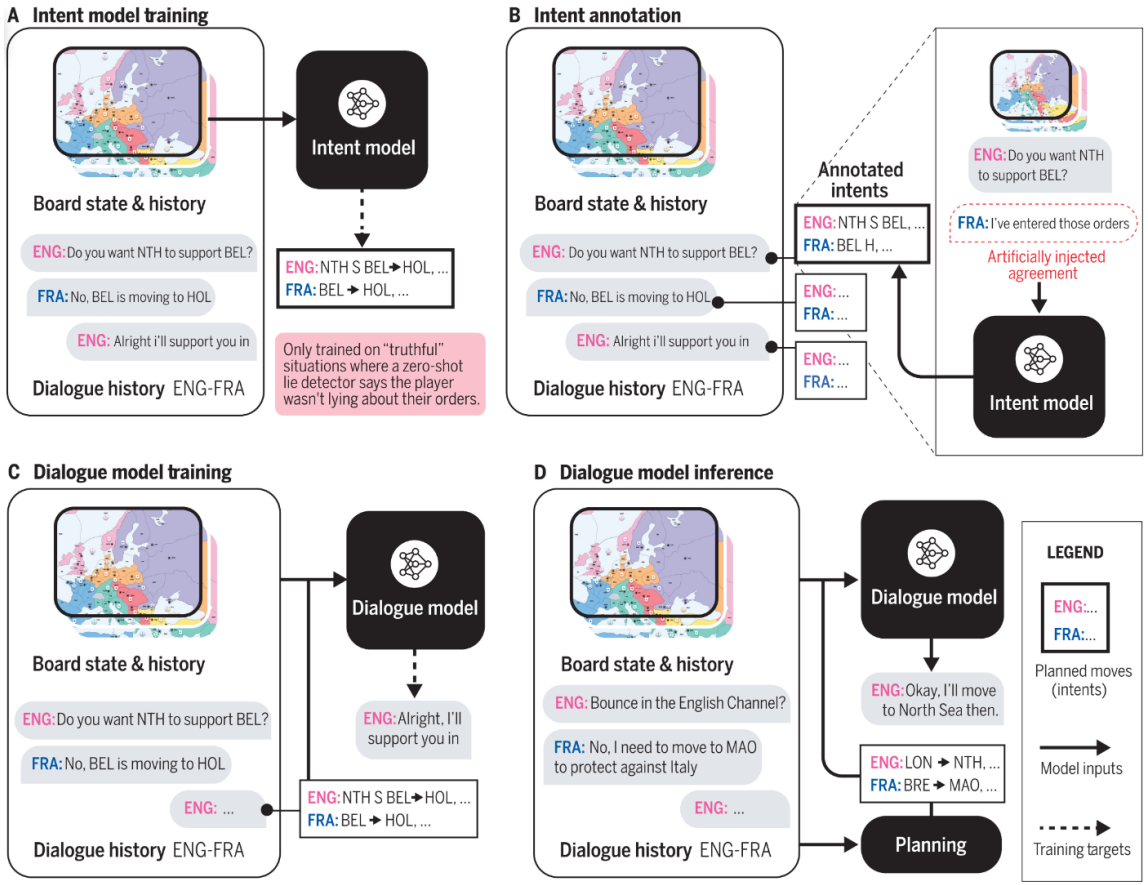
The set of legal orders for any position
A an "intent" for a message from player A to player B is a set of orders for A and a set of orders for B
These steps are explained in more detail below.

Step 4: They train a model to infer an "intent" of each message based on the orders that followed in game move.

Step 5: They train a model annotate messages with the intent of the message.

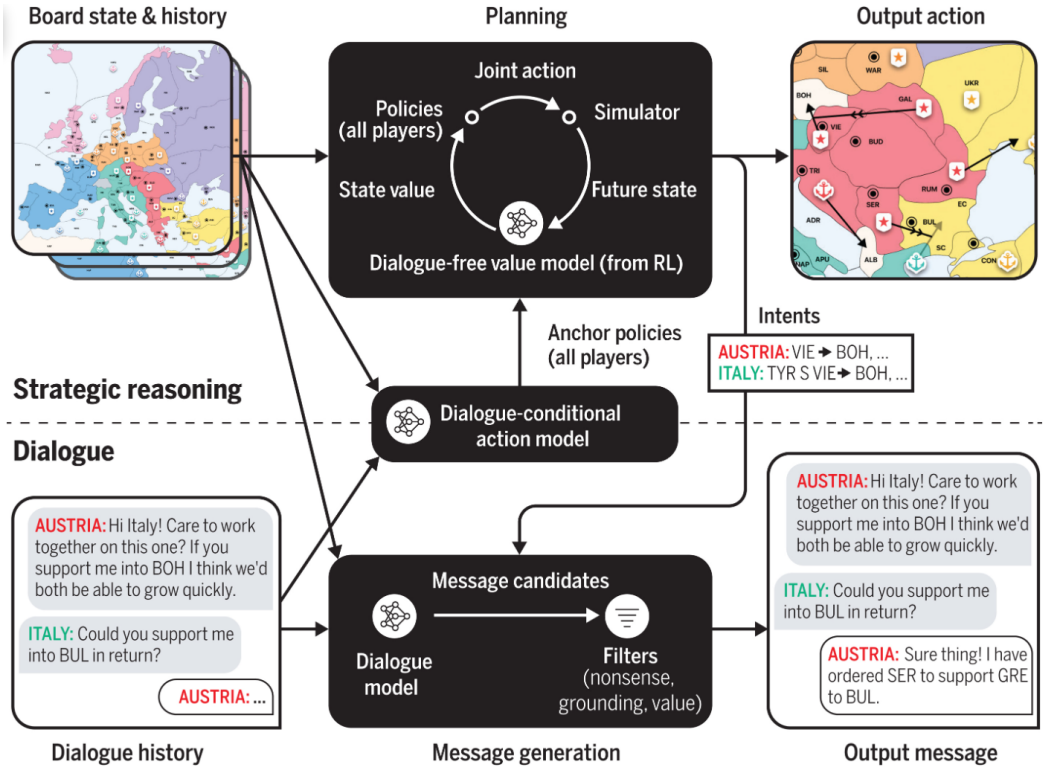
step 6:

Cicero is an Elaborate System



Cicero: Diplomacy

Cicero is an AI agent that plays Diplomacy with humans. It is rated in the top 10% of human players.



END