The end is near

- **This week**
  - Finish RL
- **Next week**
  - Random other material
  - Project presentations in pods
- **Dec 12**
  - Review
  - Projects due
- **Dec 20: final**

Deep Q-Learning
and RL game play

Lyle Ungar

with a couple slides from Eric Eaton

DQL/DQN
AlphaGo
AlphaZero, MuZero
Cicero

Remember Q-Learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R + \gamma Q\left(s', \mu(s')\right) - Q(s,a)\right)$$

Converges when this is zero

where

$$Q\left(s', \mu(s')\right) = \max_{a'} Q(s', a')$$

Aside: Reminder of off vs. on policy

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R + \gamma Q\left(s', \mu(s')\right) - Q(s,a)\right)$$

Converges when this is zero

SARSA - *on policy* (ε–greedy for a and future evaluation)

$Q\left(s', \mu(s')\right) = Q\left(s', \pi(s')\right) = max_{a'} Q(s', a')$ with prob *1-*ε

$random_{a'} Q(s', a')$ with prob ε

Q-learning - *off policy* (ε–greedy for a, greedy for future evaluation)

Greedy: $Q\left(s', \mu(s')\right) = max_{a'} Q(s', a')$

Deep Q-Learning (DQN)

Inspired by

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

Represent $Q(s,a)$ ***by a neural net***

Estimate using gradient descent with loss function:

$$\left(R + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)^2$$

The policy, $\pi(a)$**, is then given by maximizing the predicted Q-value**

Separate Q- and Target Networks

Issue: Instability (e.g., rapid changes) in the Q-function can cause it to diverge

Idea: use two networks to provide stability ("self-play")

- The Q-network is updated regularly
- The target network is an older version of the Q-network, updated occasionally

$$\left(\left(R(s,a,s') + \gamma \max_{a'} Q(s',a')\right) - Q(s,a)\right)^2$$

computed via target network

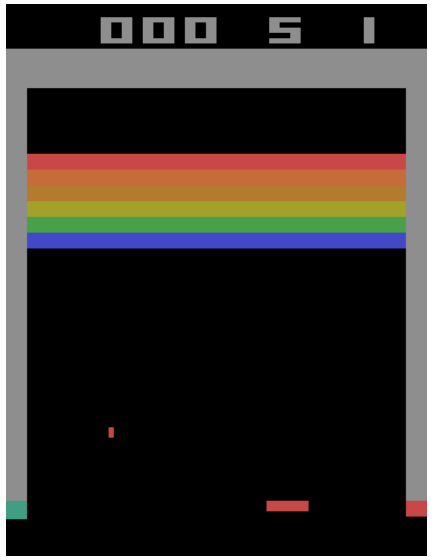
computed via Q-network

Deep Q-Learning (DQN) Algorithm

Initialize replay memory $\mathcal{D}$
Initialize Q-function weights θ
for episode $= 1 \ldots M$, do
  Initialize state s_t
  for $t = 1 \ldots T$, do
    $a_t \leftarrow \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \max_a Q^*(s_t, a; \theta) & \text{with probability } 1 - \epsilon \end{cases}$ ε-greedy
    Execute action a_t, yielding reward r_t and state s_{t+1}
    Store $\langle s_t, a_t, r_t, s_{t+1} \rangle$ in $\mathcal{D}$
    $s_t \leftarrow s_{t+1}$
    Sample random minibatch of transitions $\{\langle s_j, a_j, r_j, s_{j+1} \rangle\}_{j=1}^{N}$ from $\mathcal{D}$
    $y_j \leftarrow \begin{cases} r_j & \text{for terminal state } s_{j+1} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) & \text{for non-terminal state } s_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
  end for
end for

Based on https://arxiv.org/pdf/1312.5602v1.pdf

DQN on Atari Games



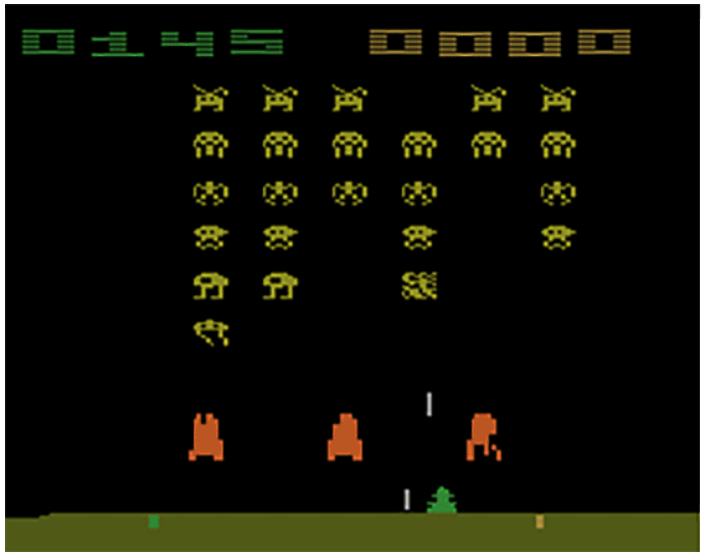
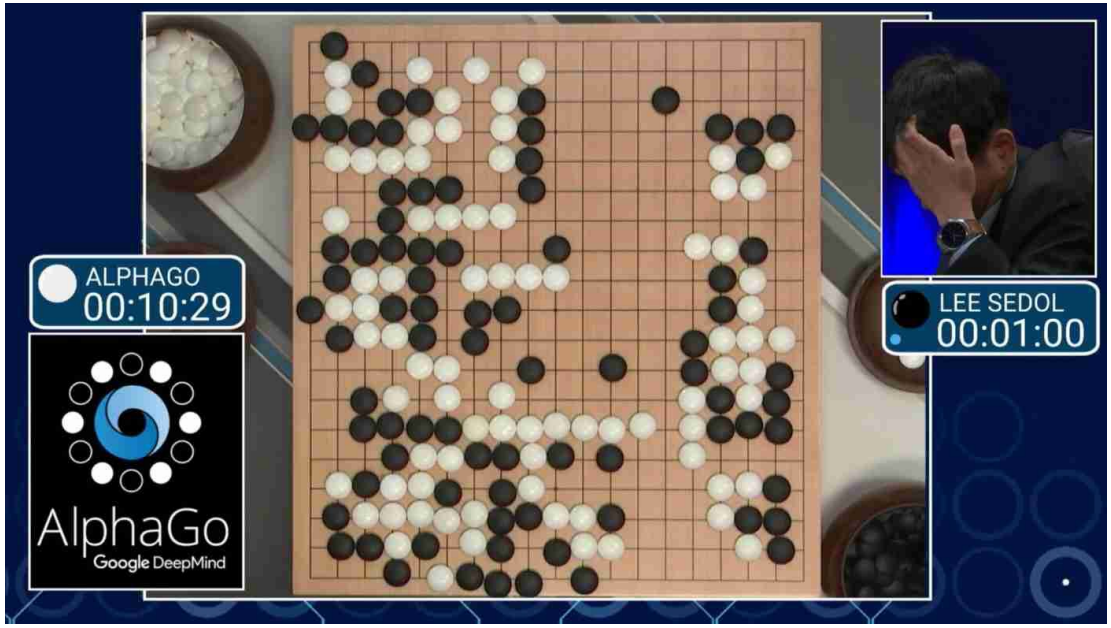


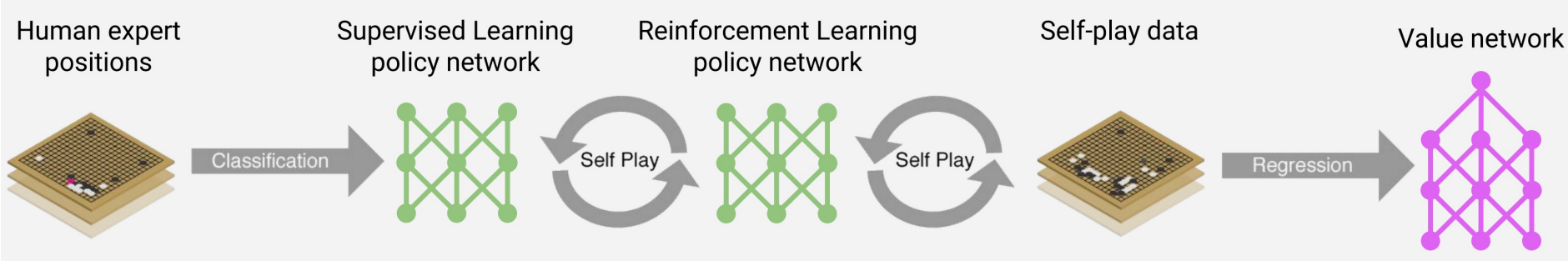
Image Sources:
https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-architectures-4c1b3fb7f756
https://deepmind.com/blog/going-beyond-average-reinforcement-learning/
https://jaromiru.com/2016/11/07/lets-make-a-dqn-double-learning-and-prioritized-experience-replay/

AlphaGo



https://medium.com/@jonathan_hui/alphago-how-it-works-technically-26ddcc085319 **2016**

AlphaGo - Complicated



1. Train a CNN to predict (supervised learning) moves of human experts

2. Use as starting point for policy gradient (self-play against older self)

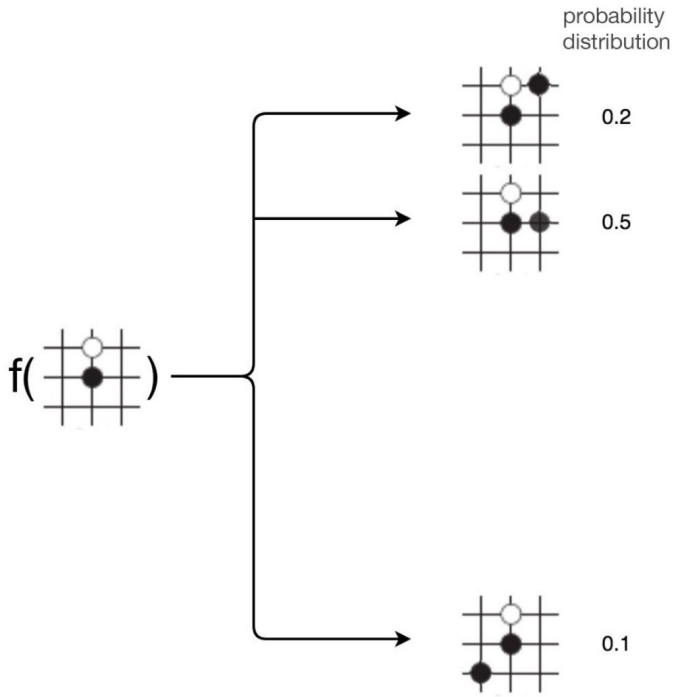
3. Train value network with examples from policy network self-play

4. Use Monte Carlo tree search to explore possible games

Image from DeepMind's ICML 2016 tutorial on AlphaGo: https://icml.cc/2016/tutorials/AlphaGo-tutorial-slides.pdf

Learn policy

- ***a*** = *f(**s**)*
- ***a*** = where to play (19*19)
- ***s*** = description of board



supervised = “behavioral cloning”

State ~ 19*19*48

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

AlphaGo – lots of hacks

- **Bootstrap**
  - Initialize with policy learned from human play
- **Self-play**
- **Speed matters**
  - Rollout network (fast, less accurate game play)
  - Monte Carlo search
- **It still needed fast computers**
  - > 100 GPU weeks

AlphaZero

- **Self-play with a single, continually updated neural net**
  - No annotated features - just the raw board position
- **Uses Monte Carlo Tree Search**
  - Using *Q(s,a)*
- **Does policy iteration**
  - Learns *V(s)* and $\pi(s)$
- **Beat AlphaGo (100-0) after just 72 hours of training**
  - On 5,000 TPUs

https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go **2017**

Monte Carlo Tree Search (MCTS)

- **In each state s_{root}, select a move, $a_t \sim \pi_t$ either proportionally** (exploration) **or greedily** (exploitation)
- **Pick a move a_t with**
  - low visit count (not previously frequently explored)
  - high move probability (under the policy
  - high value (averaged over the leaf states of MC plays that selected *a* from *s*) according to the current neural net
- **The MCTS returns an estimate *z* of $v(s_{root})$ and a probability distribution over moves, $\pi = p(a|s_{root})$**

AlphaZero loss function

NNet: $(\mathbf{p}, v) = f_\theta(s)$

- Minimizes the error between the value function *v(s)* and the actual game outcome *z*
- Maximizes the similarity of the policy vector ***p****(s)* to the MCTS probabilities $\pi(s)$.
- L_2 regularize the weights θ

$$l = (z - v)^2 - \pi^{\mathrm{T}} \log \mathbf{p} + c\|\theta\|^2.$$

AlphaZero

Believe it or not, we now have all elements required to train our unsupervised game playing agent! Learning through self-play is essentially a policy iteration algorithm- we play games and compute Q-values using our current policy (the neural network in this case), and then update our policy using the computed statistics.

Here is the complete training algorithm. We initialise our neural network with random weights, thus starting with a random policy and value network. In each iteration of our algorithm, we play a number of games of self-play. In each turn of a game, we perform a fixed number of MCTS simulations starting from the current state s_t. We pick a move by sampling from the improved policy $\vec{\pi}_t$. This gives us a training example $(s_t, \vec{\pi}_t, \_)$. The reward _ is filled in at the end of the game: +1 if the current player eventually wins the game, else -1. The search tree is preserved during a game.

At the end of the iteration, the neural network is trained with the obtained training examples. The old and the new networks are pit against each other. If the new network wins more than a set threshold fraction of games (55% in the DeepMind paper), the network is updated to the new network. Otherwise, we conduct another iteration to augment the training examples.

And that's it! Somewhat magically, the network improves almost every iteration and learns to play the game better. The high-level code for the complete training algorithm is provided below.

https://web.stanford.edu/~surag/posts/alphazero.html

MuZero

- **Slight generalization of AlphaZero**
  - Also uses Monte Carlo Tree Search (MCTC), self-play
- **Learns Go, Chess, Atari …**
- **Learns 3 CNNs**
  - **Representation model**: observation history $\longrightarrow$ $state_t$
  - **Dynamics model:** $state_t$ * $action_t$ $\longrightarrow$ $state_{t+1}$
  - **Policy**: $state_t$ $\longrightarrow$ $action_t$

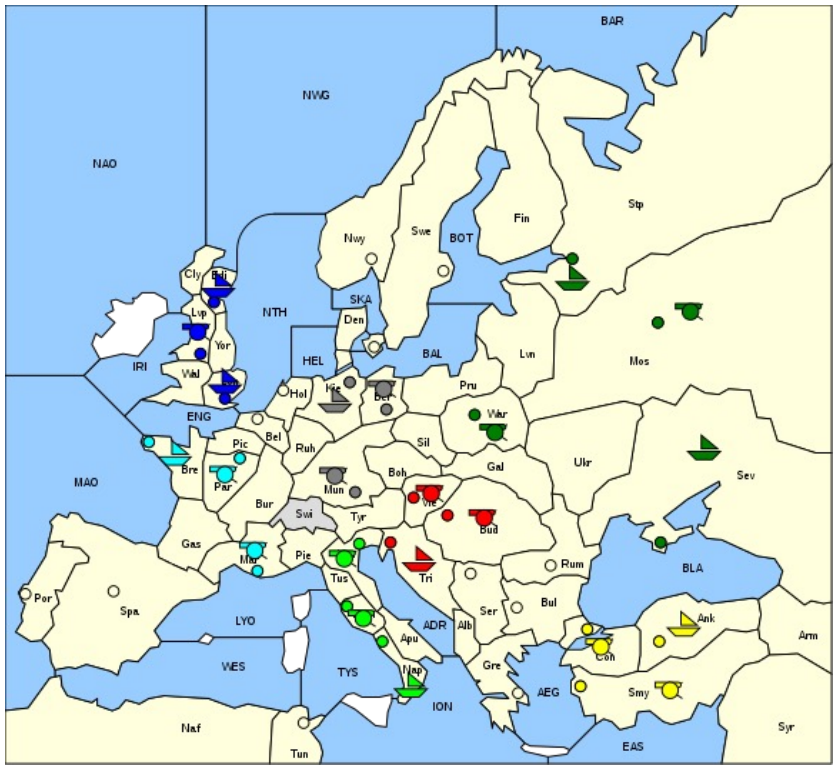
StarCraft

- StarCraft-playing AI model consists of 18 agents, each trained with 16 Google v3 TPUs for 14 days.
- Thus, at current prices ($8.00 / TPU hour), the company spent $774,000 on this model

Diplomacy

- Players negotiate
- Then all players secretly write down their moves
- Then all moves are revealed and put into effect simultaneously.

Social interaction and interpersonal skills make up an essential part of the play.



https://en.wikipedia.org/wiki/Diplomacy_%28game%29

Cicero: Human-level play in the game of *Diplomacy* by combining language models with strategic reasoning

◆ Dialog

- pre-trained language model, fine tuned on dialogue data from human games of *Diplomacy*.
- augmented with inferred intents

◆ Strategy

- Planning relies on a value and policy function trained via self-play RL which penalized the agent for deviating too far from human behavior (in order to maintain a human-compatible policy).

https://www.science.org/doi/10.1126/science.ade9097

Dialog in Cicero

We took R2C2 as our base model – a 2.7B parameter Transformer-based encoder-decoder model pre-trained on text from the Internet using a BART de-noising objective. The base pre-trained model was then further trained on WebDiplomacy via standard Maximum Likelihood Estimation. Specifically, with a dataset $\mathcal{D}$={[$\mathbf{x}(i)$,$\mathbf{y}(i)$]} the model was trained to predict a dialogue message $\mathbf{y}^{(i)}$ from player $\mathcal{A}$ to player $\mathcal{B}$ at time t, given all of the following represented as text $\mathbf{x}^{(i)}$:

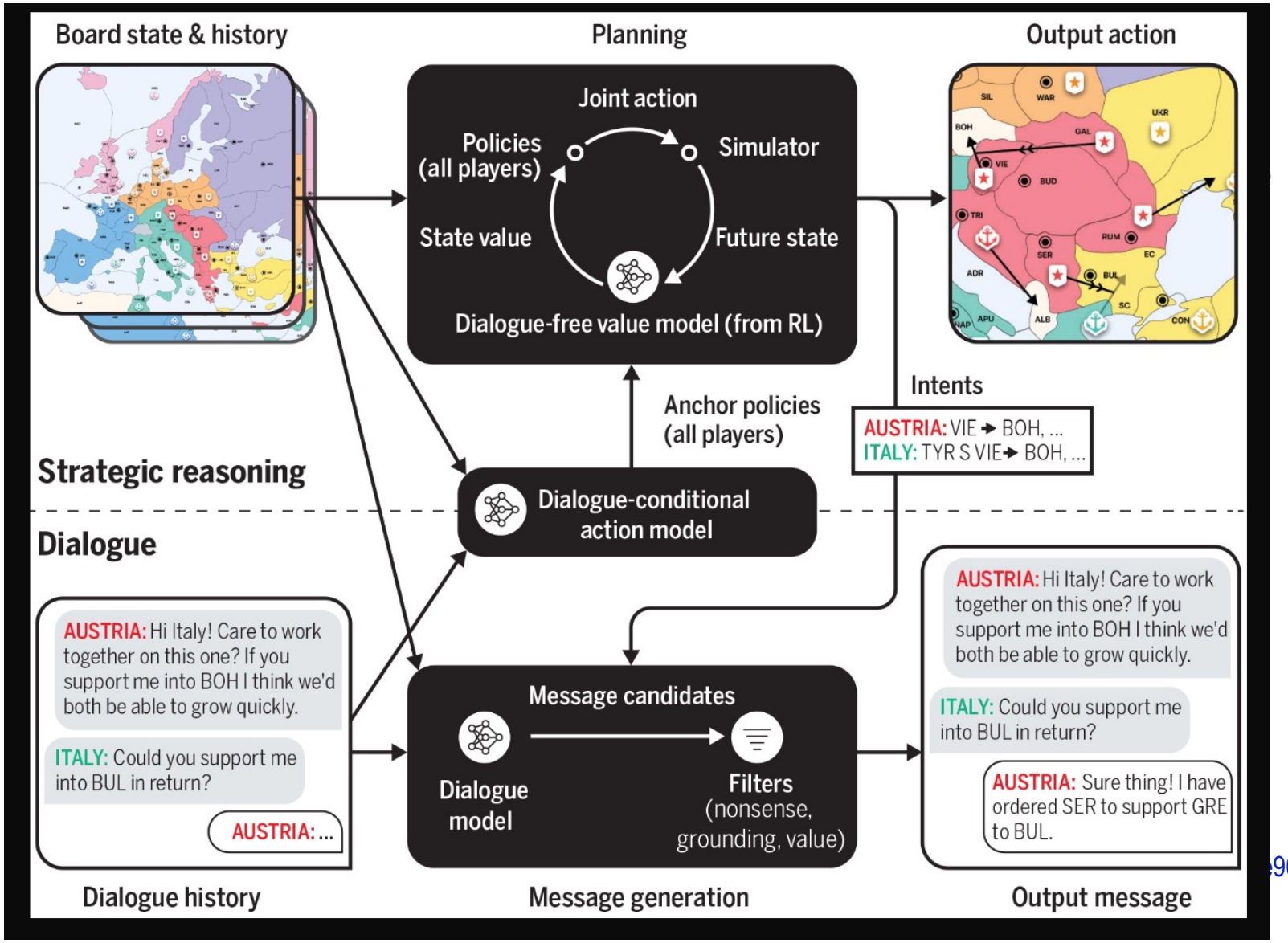
- dialogue history
- game state and action history
- player rating
- game and message metadata
- intents

https://www.science.org/doi/10.1126/science.ade9097

Strategy in Cicero

- [Strategy] requires predicting how humans will play. A popular approach in cooperative games is to model the other players' policies via supervised learning on human data, which is commonly referred to as behavioral cloning (BC). However, pure BC is brittle, especially since a supervised model may learn spurious correlations between dialogue and actions
- To address this problem, Cicero used variants of *piKL* to model the policies of players. *piKL* is an iterative algorithm that predicts policies by assuming each player i seeks to both maximize the expected value of their policy π_i and minimize the KL divergence between π_i and the BC policy.
- we used piKL during self-play to keep the policies human-compatible.

https://www.science.org/doi/10.1126/science.ade9097


Board state & history
Planning
Output action
Joint action
Policies (all players)
Simulator
State value
Future state
Dialogue-free value model (from RL)
Anchor policies (all players)
Intents
AUSTRIA: VIE → BOH, ...
ITALY: TYR S VIE → BOH, ...
Strategic reasoning
Dialogue-conditional action model
Dialogue
AUSTRIA: Hi Italy! Care to work together on this one? If you support me into BOH I think we'd both be able to grow quickly.
ITALY: Could you support me into BUL in return?
AUSTRIA: ...
Message candidates
Dialogue model
Filters (nonsense, grounding, value)
AUSTRIA: Hi Italy! Care to work together on this one? If you support me into BOH I think we'd both be able to grow quickly.
ITALY: Could you support me into BUL in return?
AUSTRIA: Sure thing! I have ordered SER to support GRE to BUL.
Dialogue history
Message generation
Output message

e9097

Cicero conversation

Applied RL Summary

- **Superhuman game playing using DQL**
  - But it does not generalize well to even slightly different games

- **Why is DeepMind losing $500 million/year?**