Mastering the Game of Stratego with Model-Free Multiagent Reinforcement Learning (DeepNash)

백승언

28 Nov, 2022

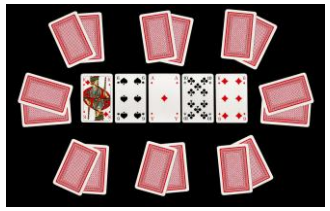
[1]: https://arxiv.org/pdf/2206.15378.pdf

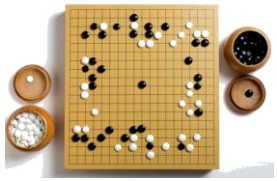
Contents

Backgrounds

Previous RL milestones in games

- **Milestones**
  - Backgammon (1992) → TD(λ)
  - Chess (1997) → TD(λ)
  - Atari game (2013) → DQN
  - Go (2016) → MC-Tree search
  - Star II (2019) → IMPALA, UPGO
  - Poker (2020) → CFR





RL in two-player zero-sum imperfect information games

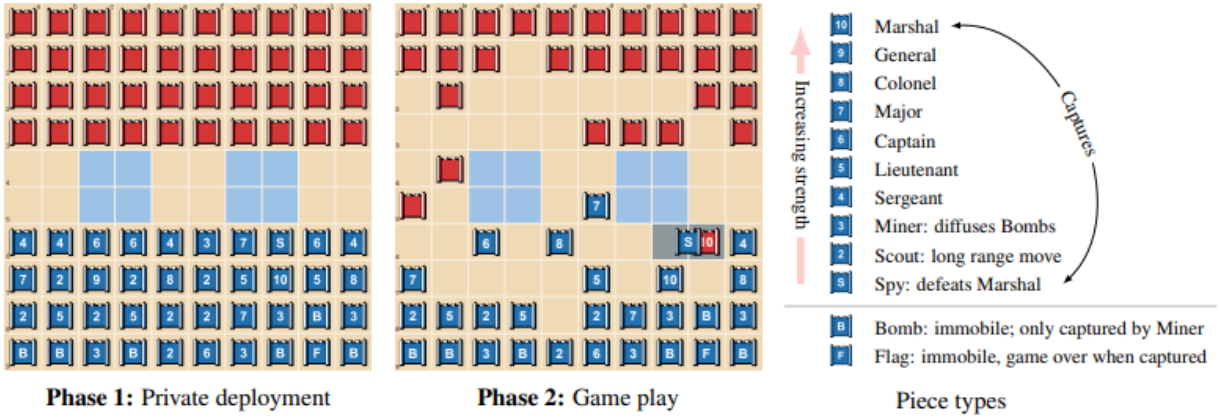
- **RL methods have been applied two-player zero-sum perfect information games**
  - TD(λ) in backgammon, chess
  - Search based methods in GO, Shogi

- **Recently, some RL approaches as follow have shown success in imperfect information games**
  - **Regret minimization based approaches**
    - Minimizing the regret if the difference between the average value of the sequence of actions it generates, and that the best alternate in hindsight, approaches zero over time.
      - Meta's ReBeL
  - **Best-response based approaches**
    - The best response is computed by training against a meta-distribution over the current population policies
      - DeepMind's AlphaStar, OpenAI's Five
  - **Policy gradient based approaches**
    - Approximates counterfactual regret minimization or Nash equilibrium via a policy gradient
      - **DeepMind's DeepNash**

About the game of Stratego

- **Two-player board game where each player aims to capture the opponent's flag**
  - Description of basic rule
    - There are a total of 12 types of pieces and each player start the Stratego with 40 pieces
    - Each pieces could move in 3x3 square without diagonal direction
      - Exceptionally, scout could long range move, flag and bomb could not move
    - If certain piece move to area where other piece lay in, battle would occur and weak piece would be removed
    - Player wins when the opponent's flag is captured or when there are no opponent's pieces to move

  - Unlike conventional board games like Shogi and Chess, Stratego has various tactical/strategic point
    - In deployment phase, both players **secretly place** their pieces on the board
    - The type of pieces are **concealed** and revealed when attacking and/or attacked
    - Some types of pieces **could not moved**: bomb, flag
    - Pieces could not move to **inhibited area**: lake

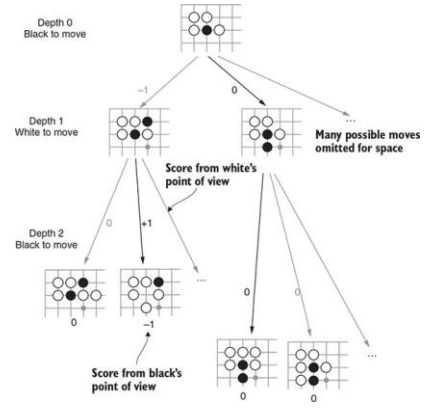


About Stratego

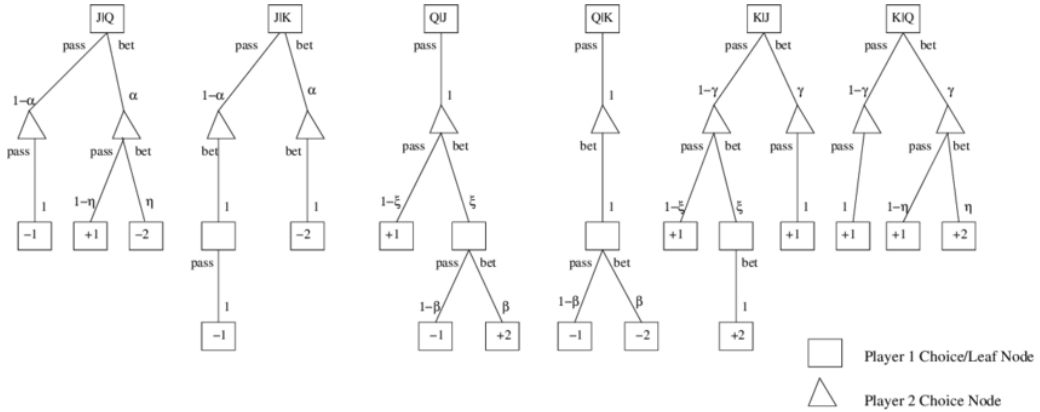
DeepNash

Introduction to DeepNash (I)

- **Stratego is one of the few iconic board games that Artificial Intelligence(AI) has not yet mastered**
  - This popular game has an enormous game tree on the order of 10^{535} nodes, and has the additional complexity of requiring decision-making under imperfect information (Go: 10^{360}, Poker: 10^{164})
  - Currently, it is not possible to use SOTA model-based perfect information planning technique, nor SOTA imperfect information search technique that break down the game into subgame in Stratego
  - Developing intelligent agents that learns end-to-end to make optimal decision under imperfect information in Stratego, from scratch, without human demonstration data, remained one of the grand challenges of AI research



Game tree of Go

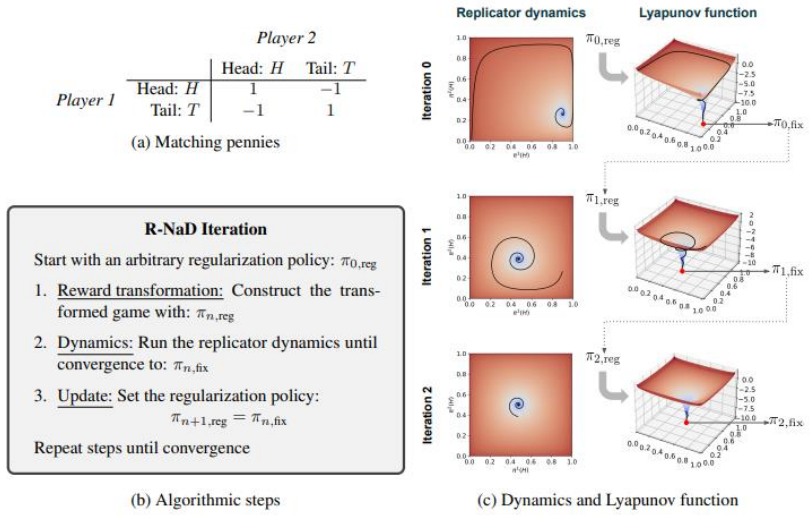


Game tree of Poker

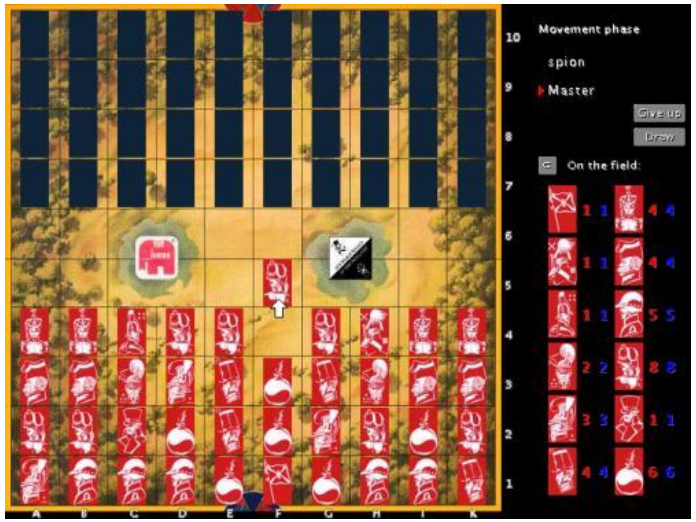
Introduction to DeepNash (II)

- **The authors proposed the DeepNash, which beats the human experts and traditional AI bots**
  - DeepNash combined existing R-NaD algorithm with a deep neural network architecture for converging policies to an **ϵ-Nash equilibrium**, which means it learns to play at a highly competitive level

  - DeepNash are systematically evaluated its performance against various SOTA Stratego bots and humen expert players on the Gravon games platform(online platform)
    - All current SOTA botas have been defeated by DeepNash (win rate of over **97%**)
    - DeepNash ranks among the **top 3 players**, both on the annual (2022) and all-times leader boards



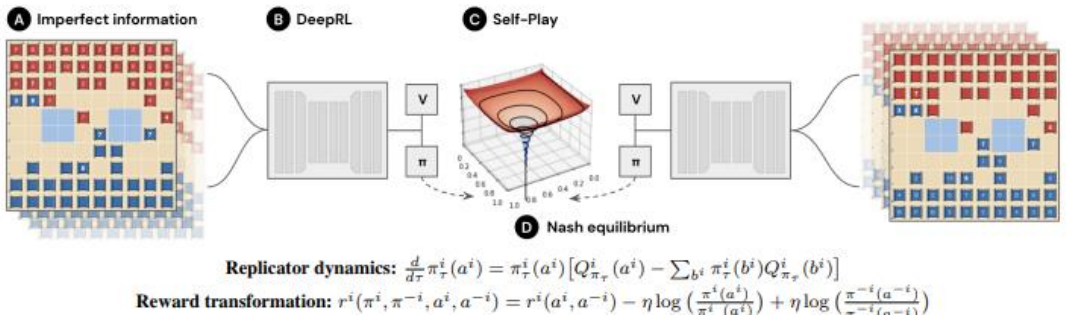
The R-NaD algorithm with the matching pennies game

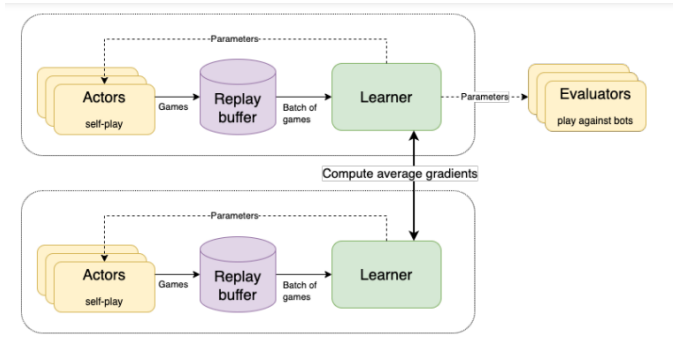


Stratego in Gravon platform

Methodology (I) – Overview

- **DeepNash takes an end-to-end learning approach to solve Stratgeo, by incorporating the learning of the deployment phase and game play phase**
  - 1. Putting the pieces tactically on the board at the start of a game
  - 2. In the learning of the game-play phase, using an integrated model-free RL and game-theoretic approach
    - Aim of the agent is to learn an approximate Nash equilibrium through self-play without search(model-free)
    - Scaling up the reinforcement learning approach with R-NaD to Deep Reinforcement Learning
  - 3. Whole training process are performed with IMPALA(with V-trace) for efficiency
  - 4. Post processing of policy using some human/domain knowledge at test time





Overview of the DeepNash approach

Training architecture

Human knowledge !

Methodology (II) – Regularized Nash Dynamics(R-NaD) algorithm

- **Problem formulation**
  - In a two-player zero-sum normal form games, two player(player $i = 1$ or $i = 2$) simultaneously play actions $a^i \in A\ \left(a = (a^1, a^2) = \left(a^i, a^{-i}\right)\right)$ with policies $\pi^i \in \Delta A\ (\pi = (\pi^1, \pi^2) = \left(\pi^i, \pi^{-i}\right)$, where $-i$ encodes the opponent of player i)

  - At turn t the player receive a reward signal $(r_t^1(a^1, a^2),\ r_t^2(a^1, a^2) = -r_t^1(a^1, a^2))$ and the current player $i = \psi_t$ observes the game state through an observation o_t and selects an action a_t based on policy $\pi(.|o_t)$

  - If policy π is played we define the Q-function to be the expected reward for player i for action a_i and the value function to be the expected reward
    - $Q_\pi^i(a^i) = E_{a^{-i}\sim\pi^{-i}}\left[r_\pi^i(a^i, a^{-i})\right],\ \ V_\pi^i = E_{a\sim\pi}\left[r_\pi^i(a)\right] = E_{a^i\sim\pi^i}\left[Q_\pi^i(a^i)\right]$

  - By definition, a policy π^* is a Nash equilibrium if for all π and for all i we have $V^i_{\pi^i,\pi^{*-i}} - V^i_{\pi^*} \leq 0$
    - In other words, a Nash equilibrium is a joint policy such that no player has an incentive to change its policy if all the other players stick to their policy

  - In model-free RL, the trajectories $T = [(o_t, a_t, (r_t^1, r_t^2), \pi(.|o_t)), \psi_t)]_{0\leq t<t_{max}}$ are the only data the agent will leverage to learn the parameterized policy

Methodology (III) – Regularized Nash Dynamics(R-NaD) algorithm

- **R-NaD relies on three key steps: reward transformation, dynamics, update**
  - **Reward transformation step**
    - Modifying the rewards as follows based on a regularization policy π_{reg}
      - $r^i(\pi^i, \pi^{-i}, a^i, a^{-i}) = r^i(a^i, a^{-i}) - \eta \log\left(\frac{\pi^i(a^i)}{\pi^i_{reg}(a^i)}\right) + \eta \log\left(\frac{\pi^{-i}(a^{-i})}{\pi^{-i}_{reg}(a^{-i})}\right), \quad with\ \eta > 0, \quad i \in [1, 2]$
      - η: regularization parameter, i: the player index
  - **Dynamics step**
    - Letting the system evolve according to the replicator dynamics system on this modified game
    - Replicator dynamics are a descriptive learning process, that are known as Follow the Regularized Leader and defined as follows:
      - $\frac{d}{d\tau}\pi^i_\tau(a^i) = \pi^i_\tau(a^i)\left[Q^i_{\pi_\tau}(a^i) - \Sigma_{b^i}\pi^i_\tau(b^i)Q^i_{\pi_\tau}(b^i)\right]$
      - $Q^i_{\pi_\tau}(a^i)$ is the quality or fitness of an action
      - $Q^i_{\pi_\tau}(a^i) = E_{a^{-i}\sim\pi^{-i}_\tau}\left[r^i(\pi^i_\tau, \pi^{-i}_\tau, a^i, a^{-i})\right]$

Player 2

Player 1	Head: H	Tail: T
Head: H	1	-1
Tail: T	-1	1

(a) Matching pennies

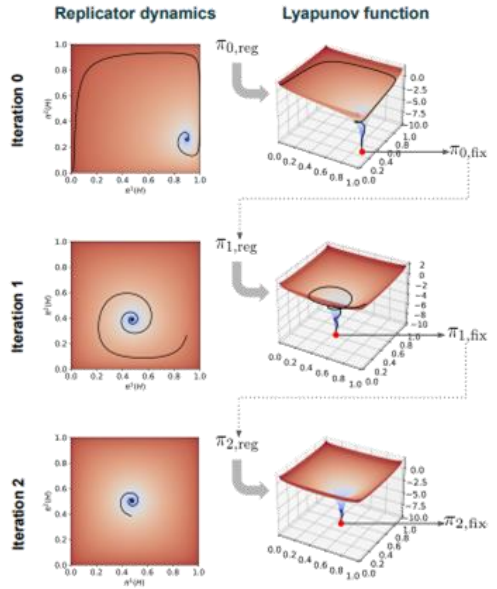
R-NaD Iteration

Start with an arbitrary regularization policy: $\pi_{0,\text{reg}}$

1. Reward transformation: Construct the transformed game with: $\pi_{n,\text{reg}}$
2. Dynamics: Run the replicator dynamics until convergence to: $\pi_{n,\text{fix}}$
3. Update: Set the regularization policy: $\pi_{n+1,\text{reg}} = \pi_{n,\text{fix}}$

Repeat steps until convergence

(b) Algorithmic steps



(c) Dynamics and Lyapunov function

The R-NaD algorithm with the matching pennies game

Methodology (IV) – Regularized Nash Dynamics(R-NaD) algorithm

- **R-NaD relies on three key steps: reward transformation, dynamics, update**
  - **Dynamics step**
    - Thanks to the reward transformation, this system has a unique fixed point π_{fix} and convergence to it is guaranteed, which can be proven by Lyapunov function as follows
      - $H_{\pi_{fix}}(\pi) = \Sigma_{i=1}^{2}\Sigma_{a^i \in A^i}\pi_{fix}^{i}(a^i)\log\left(\frac{\pi_{fix}^{i}(a^i)}{\pi^i(a^i)}\right)$, $\frac{d}{d\tau}H_{\pi_{fix}}(\pi_\tau) \leq -\eta H_{\pi_{fix}}(\pi_\tau)$
  - **Update step**
    - Using the fixed policy as a regularization policy for the next iteration
      - $\pi_{n+1,reg} = \pi_{n,fix}$

- **R-NaD repeat these three steps for generating a sequence of fixed point(policy)**
  - Obtained fixed policy can be proven to converge to a **Nash equilibrium** of the original (modified) game.
  - In R-NaD paper, $n \to \pi_{n,fix}$, $n \to \Sigma_{i=1}^{2}\mathrm{KL}(\pi_{nash}^{i}, \pi_{fix}^{i})$ was proven to converge to 0

		Player 2	
		Head: H	Tail: T
Player 1	Head: H	1	-1
	Tail: T	-1	1

(a) Matching pennies

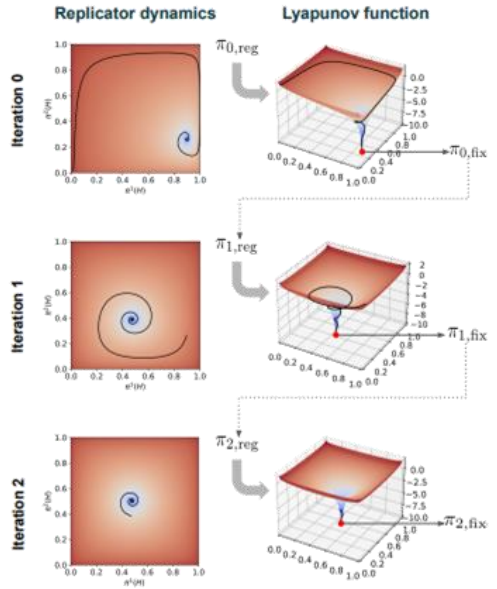
R-NaD Iteration

Start with an arbitrary regularization policy: $\pi_{0,\text{reg}}$

1. Reward transformation: Construct the transformed game with: $\pi_{n,\text{reg}}$
2. Dynamics: Run the replicator dynamics until convergence to: $\pi_{n,\text{fix}}$
3. Update: Set the regularization policy: $\pi_{n+1,\text{reg}} = \pi_{n,\text{fix}}$

Repeat steps until convergence

(b) Algorithmic steps



(c) Dynamics and Lyapunov function

The R-NaD algorithm with the matching pennies game

Methodology (V) – Observation representation

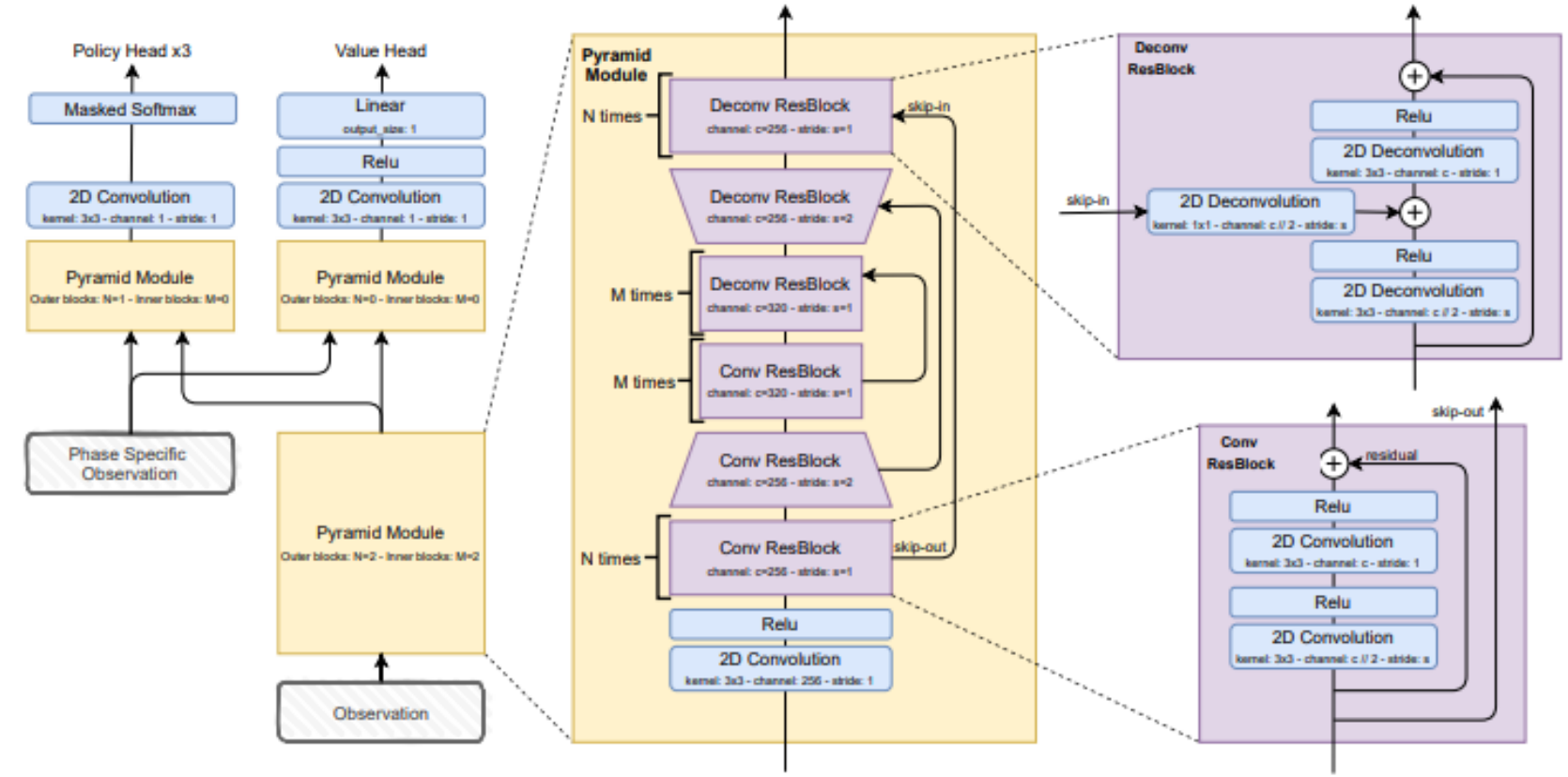
- **Observation of DeepNash includes private information, public information, move history, and so forth**
  - Players observe the game as a sequence of board positions where they see both the location and type of their own pieces, but only the location of the opponent's pieces(strategic of tactical information)
    - Private information
      - Piece type assignment [one hot encoding of 1~12]
    - Public information
      - Probability of piece type
      - $$Pub_i[r,c,t] = \begin{cases} 0 & \text{if there is no piece belonging to player } i \text{ at position } (r,c) \\ 1 & \text{if the piece at } (r,c \text{ is known to have type } t \\ \frac{num\ of\ unrevealed(t)}{\Sigma_{k \neq B,F} num\ of\ unrevealed\ (k)} & \text{if the piece at } (r,c) \text{ has ever moved and } t \neq \text{B, F} \\ \frac{num\ of\ unrevealed(t)}{\Sigma_k num\ of\ unrevealed(k)} & \text{if the piece at } (r,c) \text{ has never moved} \end{cases}$$
    - Move history
      - Encoding of the last 40 moves
  - Heuristic information
    - Players perceive own team(Red/Blue), maximum length of game, previous behavior
    - Players could avoid the lake

observation component	shape
The lakes on the board, where a square that is a lake has value 1, otherwise 0.	10×10
The player's own private information $\mathbf{Prv}_i$	$10 \times 10 \times 12$
The opponent's public information $\mathbf{Pub}_{-i}$. Contains all 0's during the deployment phase.	$10 \times 10 \times 12$
The player's own public information $\mathbf{Pub}_i$: this informs i on the information $-i$ has on i's pieces. Contains all 0's during the deployment phase.	$10 \times 10 \times 12$
An encoding of the last 40 moves: $\mathbf{Mov}_i^m$ for each move made up to 40 steps ago.	$10 \times 10 \times 40$
The ratio of the game length to the maximum length before the game is considered a draw.	scalar
The ratio of the number of moves since the last attack to the maximum number of moves without attack before the game is considered a draw.	scalar
The phase of the game: either deployment (1) or play (0).	scalar
An indication of whether the agent needs to select a piece (0) or target square (1) for an already selected piece. 0 during deployment phase.	scalar
The piece selected in the previous step (1 for the selected piece, 0 elsewhere), if applicable, otherwise all 0's.	10×10

The components of the observation. (10x10x82 tensor)

Methodology (VI) – Whole network architecture

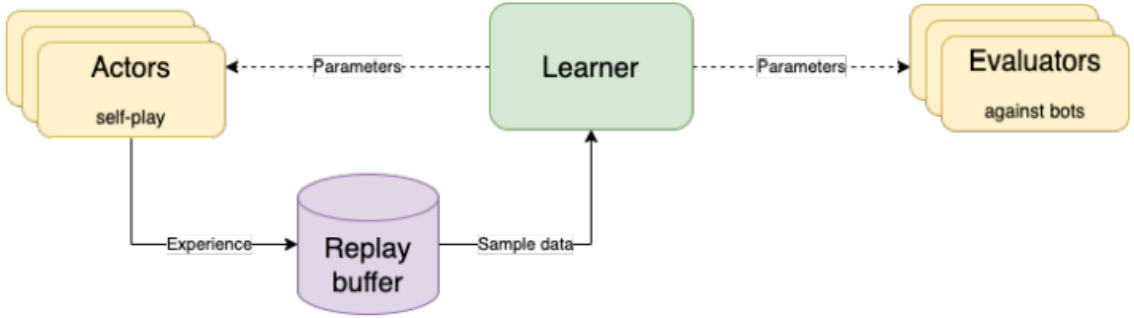
- **Whole network includes a torso module and head modules**
  - The U-Net torso module first processes the observation to produce a board game embedding
  - Processed embedding vectors provided to three policy head modules specialized by game phase and value-head module
    - The **deployment head** is used during the deployment phase.
    - The **piece-selection head** is used during the first stag e of the game phase
    - The **piece-displacement h ead** is used during the seco nd stage of the game phase
    - The **value head** is used du ring the training to compute the value function of the ag ent



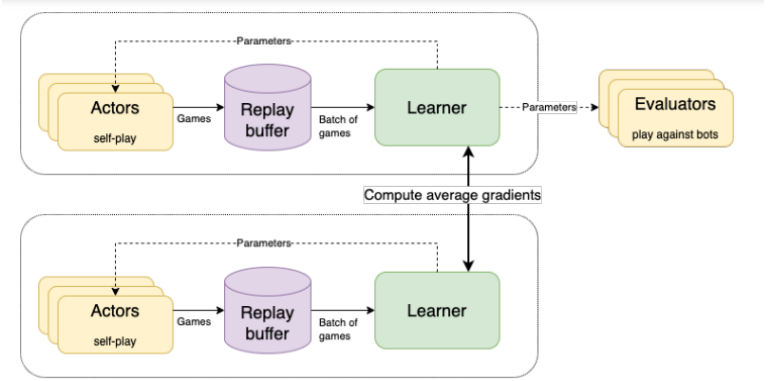
DeepNash's whole network architecture

Methodology (VII) – Infrastructure and setup

- **Several machines were participated in training based on Sebulba Podracer architecture**
  - The Actors self-play and write full games into replay buffer
    - In Actors, environment-agent loop is implemented in C++ using fibers and the native open spiel interfaces
  - The Replay Buffer, as a queue, stores games from Actors until the Learner reads it
  - The Learner, extracts a batch of games from Replay Buffer, improves the network weights, and periodically sends those to Actors/Evaluators
    - Computing the average of the gradients across all machines, and each applies those average gradients as updates, to network weights
  - The Evaluators play against fixed opponent bots, then plot the aggregated statistics
  - To train the final agent the authors used 768 TPU nodes for Learners and 256 TPU nodes for Actors



Sebulba Podracer architecture



Architecture of DeepNash

Methodology (VIII) – Test time improvements

- **The authors applied some adjustments to the policy to improve the strength of the agent further and remove some characteristics of its play which are annoying to human opponents**
  - Post-processing of policy
    - To eliminate very-low probability moves, without making the policy of the agent too deterministic, two processing methods are applied
      - Thresholding: all actions with probability lower than a fixed threshold ϵ_{tres} are dropped and the policy is renormalized
      - Discretization: sorted from high to low, probabilities are rounded up to the nearest multiple of $1/n_{disc}$ and remaining weights are discarded once a sum of 1 reached
  - Avoidance of repetitive play
    - Avoid pointless threats: if the opposing piece retreats as before, the agent cannot immediately move to re-threaten it unless the policy has no alternative moves
    - Eagerness: in order to induce the agent to be more eager in its play, at test time the authors modify this to $r' = 1 - (1 - r)^{\alpha_{reg}}$ with parameter α_{reg}

Results (I) – Comparison with humans and existing bots

- **Evaluation results of DeepNash against both human expert players and current SOTA Stratego bots**
  - Versus human experts
    - Evaluation against human experts have worked with the Gravon platform(popular online game server)
    - DeepNash was tested against top human player over the course of two weeks, resulting **42 win in 50 ranked match** and **3rd placed** of all ranked Gravon Stratego players
  - Versus previous Stratego bots
    - Several existing Stratego computer programs were selected for evaluation
      - Probe: Winner of Computer Stratego World Championships (2007, 2008, 2010)
      - Master of the Flag: Winner of championship in 2009
      - Demon of Ignorance: opensource implementation of Stratego
      - PeternLewis: Winner of Austrailian university programming competition in 2012
      - Asmodeus, Celsius, Celsius 1l1, Vixen: contestants of the Austrailian university programming competition in 2012
    - DeepNash **wins the overwhelming majority of games** against all of these bots

Opponent	Number of Games	Wins	Draws	Losses
Probe	30	100.0%	0.0%	0.0%
Master of the Flag	30	100.0%	0.0%	0.0%
Demon of Ignorance	800	97.1%	1.8%	1.1%
Asmodeus	800	99.7%	0.0%	0.3%
Celsius	800	98.2%	0.0%	1.8%
Celsius1.1	800	97.9%	0.0%	2.1%
PeternLewis	800	99.9%	0.0%	0.1%
Vixen	800	100.0%	0.0%	0.0%

Evaluation results of DeepNash against existing bots

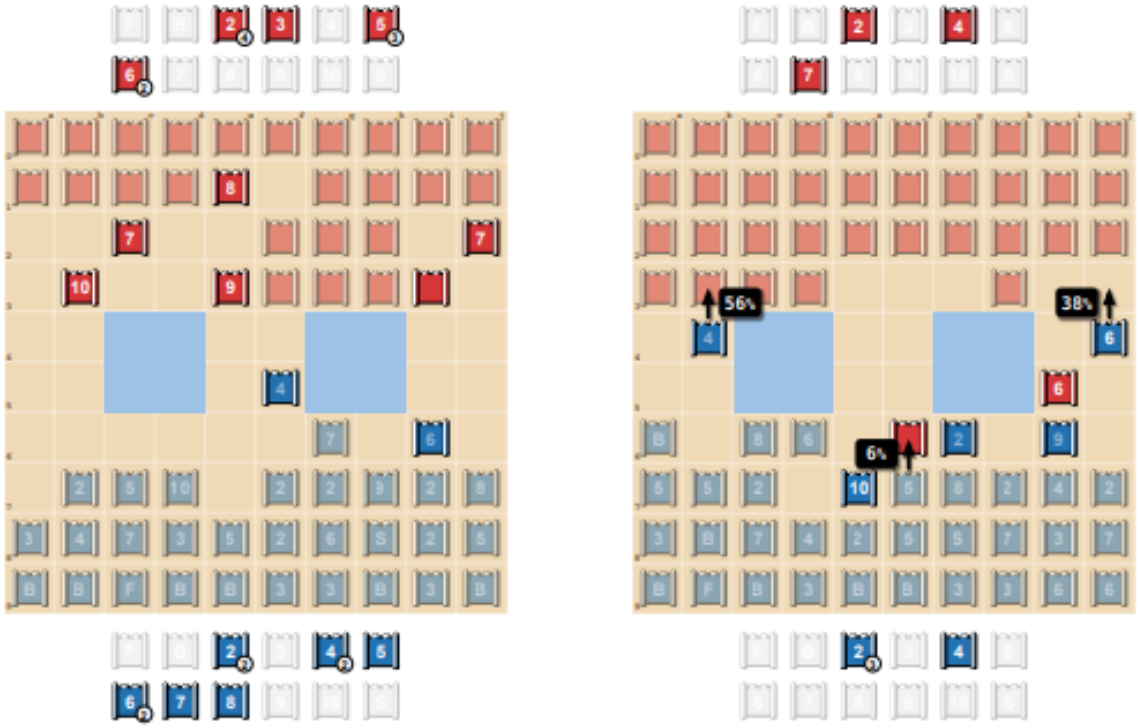
Results (II) – Illustration of DeepNash's abilities

- **Piece deployment**
  - The imperfect information in Stratego arises during the deployment phase of the game where both players lay in their 40 pieces on the board in a hidden configuration
  - DeepNash learns this deployment strategy simultaneously with the regular game-play and has indeed capacity of generating billions of unique deployments
    - The Flag is almost always put on the back row, and often protected by Bombs. (unpredictable)
      - (b): Occasionally, DeepNash didn't use the deployments of surrounding the flag with a Bomb
    - The highest pieces, the 10 and 9, are often deployed on different sides of the board
    - The Spy is quite often located not too far away from 9 (or 8), which protects it against the opponent's 10
    - DeepNash does not often deploy Bombs on the front row, which complies with the behavior seen from strong human players

Four example deployments DeepNash played on Gravon (a)~(d)

Results (III) – Illustration of DeepNash's abilities

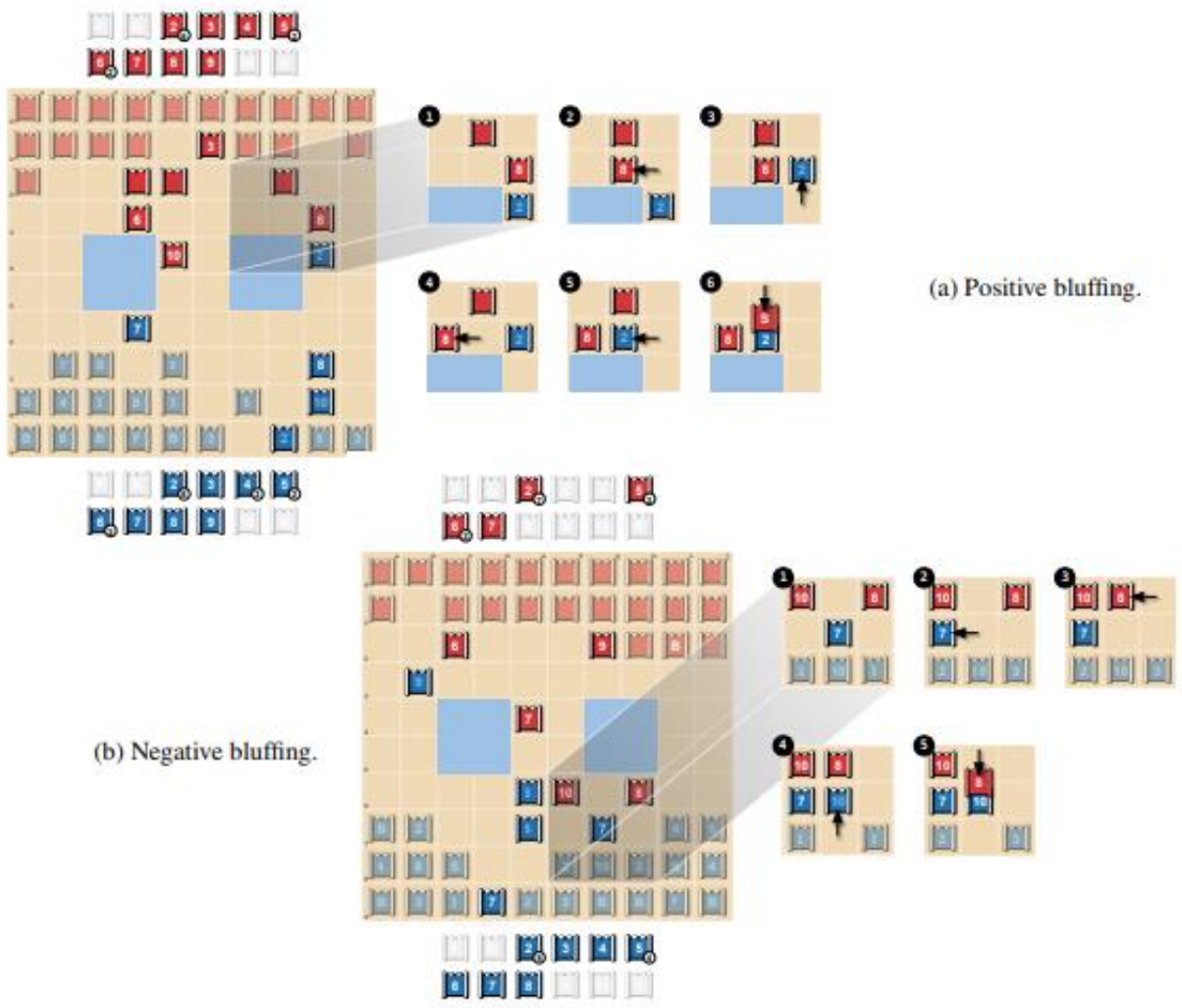
- **Trade-off between information and material**
  - DeepNash learned the important tactic in Stratego which is to keep as much information as possible hidden from an opponent in order to gain an advantage
    - (a): while blue is behind a 7 and 8, non of its pieces are revealed and only two pieces moved.
      - As a result DeepNash assess its chance of winning to be still around 70%
    - (b): DeepNash's policy supports three moves at described state, with the indicated probabilities.
      - While Blue has the opportunity to capture the opponent's 6 with its 9, this move is not considered by DeepNash, likely because the protection of 9's identity is assed to be more important than the material gain



Two playing snapshots DeepNash played on Gravon (a), (b)

Results (IV) – Illustration of DeepNash's abilities

- **Deceptive behavior and bluffing**
  - For use of value and asymmetry of information, DeepNash utilized bluffing in order to deceive its opponent and potentially gain an advantage
    - (a): **Positive bluffing**, in which a player pretends a piece to be of a higher value than it actually is.
      - DeepNash chases the opponent's 8 with an unknown piece as Scout, pretending it to be the 10.
      - The opponent believes this piece has a high chance of being the 10 and guides it next to its Spy.
    - (b): **Negative bluffing**, which means that one pretends to be a lower pieces as opposed to a positive bluff
      - Here the movement of the unknown 10 of DeepNash is interpreted by the opponent as a positive bluff as they try to capture it with a known 8 assuming DeepNash is moving a lower-ranked piece, potentially the Spy.



Two playing snapshots DeepNash played on Gravon (a), (b)

Thank you!

Q&A