Boulder

Reinforcement Learning; Games

YouTube Playlist

Maziar Raissi

Assistant Professor

Department of Applied Mathematics

University of Colorado Boulder

maziar.raissi@colorado.edu

Playing Atari with Deep Reinforcement Learning

YouTube Video

$\mathcal{E} \rightarrow$ environment
actions, observations, rewards
$a_t \rightarrow$ action
$\mathcal{A} = \{1, \ldots, K\} \rightarrow$ set of legal actions
$x_t \in \mathbb{R}^d \rightarrow$ image
$r_t \rightarrow$ reward
partially observed task
$s_t = x_1, a_1, x_2, a_2, \ldots, a_{t-1}, x_t \rightarrow$ state
MDP: Markov Decision Process

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \rightarrow \text{future discounted return at time } t$$

$Q^*(s,a) \rightarrow$ optimal action-value function

$\pi \rightarrow$ policy mapping sequences s to actions
(distribution over actions)

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$$

Bellman Equation

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q^*(s',a') | s, a]$$

Value Iteration Algorithm

$$Q_{i+1}(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q_i(s',a') | s, a]$$

$Q(s,a;\theta) \approx Q^*(s,a) \rightarrow$ Q-Network

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.)}[(y_i - Q(s,a;\theta_i))^2]$$

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s, a]$$

$y_i \rightarrow$ target for iteration i

$\rho(s,a) \rightarrow$ prob. distribution over sequences & actions
(behavior distribution)

model free: no explicit construction of an estimator for $\mathcal{E}$

off-policy: greedy strategy $a^* = \max_a Q(s,a;\theta)$

Experience Replay

$e_t = (s_t, a_t, r_t, s_{t+1}) \rightarrow$ agent's experience
$\mathcal{D} = e_1, e_2, \ldots, e_N \rightarrow$ replay memory
Input to Q-Network: state representation
Output of DQN: predicted Q-values of individual actions

Arcade Learning Environment (ALE)

Algorithm 1 Deep Q-learning with Experience Replay

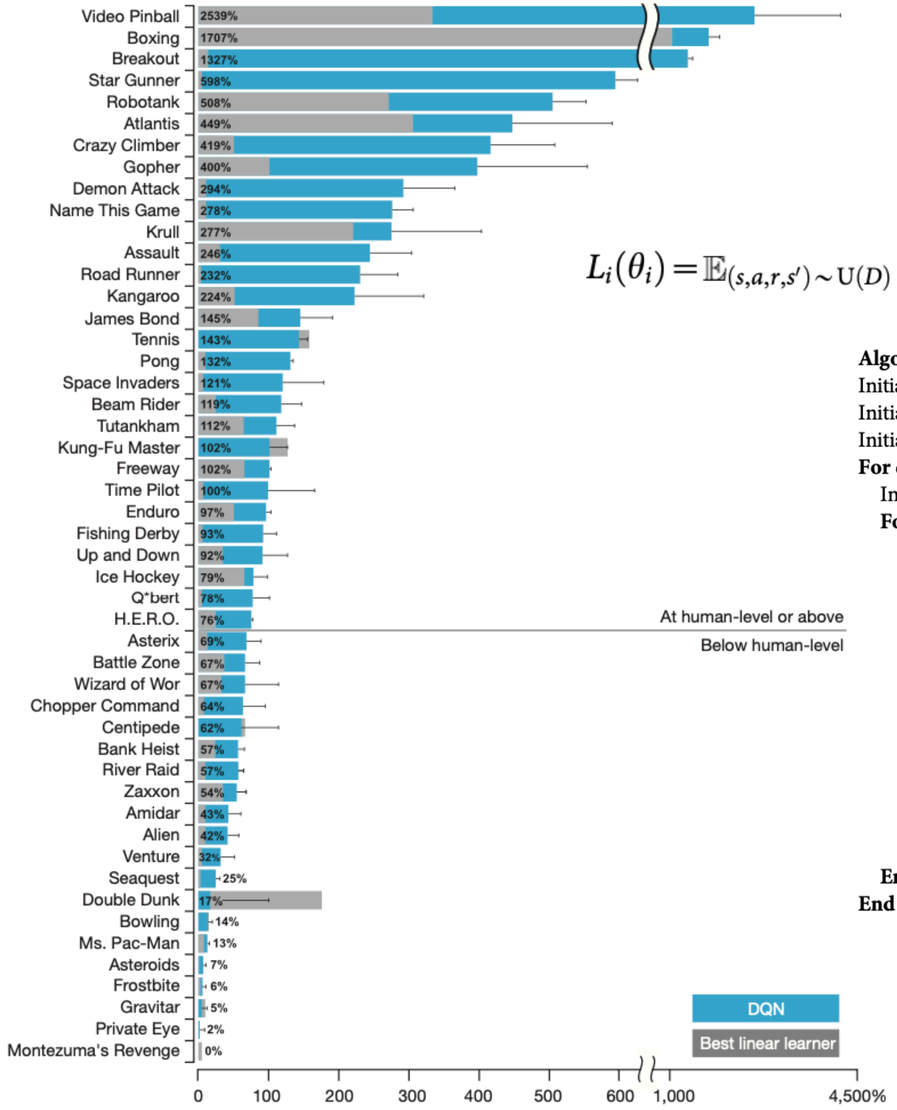
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    ϵ-greedy $\rightarrow$ { greedy action with prob. $1-\epsilon$; random action with prob. ϵ
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$ $\rightarrow$ last four frames of a history (stacked)
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

Mnih, Volodymyr, et al. "Playing Atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).

Human-level Control through Deep Reinforcement Learning

Boulder

YouTube Video



$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t),a;\theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

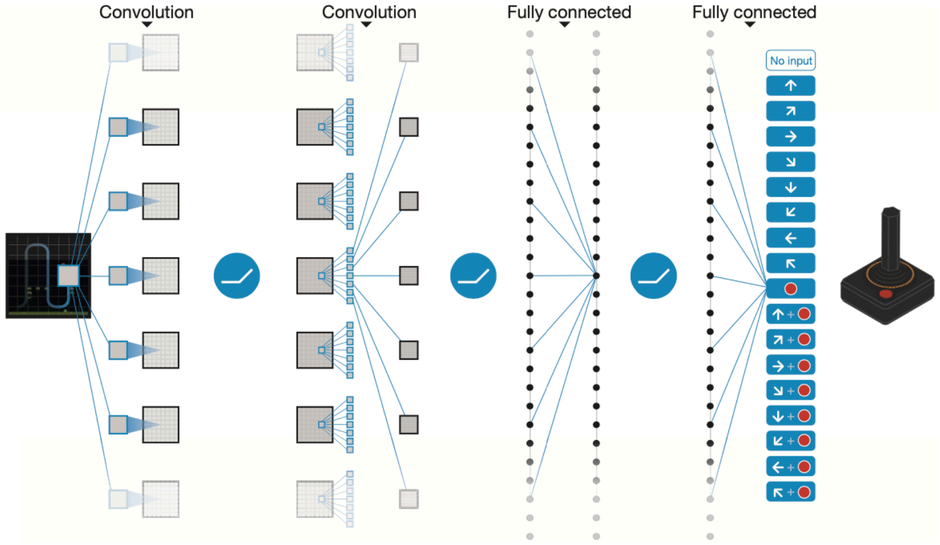
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

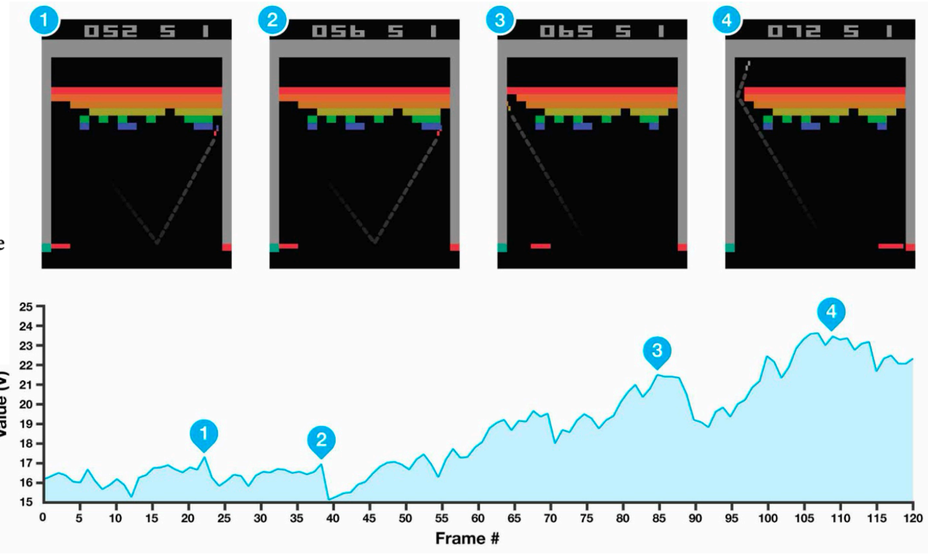
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For





Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *nature* 518.7540 (2015): 529-533.

Deep Reinforcement Learning with Double Q-Learning

YouTube Video

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

Bellman Optimality Equation

$$V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$

Value Iteration Algorithm

$$Q_{k+1}(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{s'} Q_k(s',a')]$$

Q-Value Iteration Algorithm

$$V_{k+1}(s) = (1-\alpha)V_k(s) + \alpha(r + \gamma V_k(s'))$$
$$= V_k(s) + \alpha \underbrace{(r + \gamma V_k(s') - V_k(s))}_{\delta_k(s,r,s')}$$

Temporal Difference Learning

$$Q_{k+1}(s,a) = (1-\alpha)Q_k(s,a) + \alpha(r + \gamma \max_{a'} Q_k(s',a'))$$
$$= Q_k(s,a) + \alpha(r + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a))$$

Q-Learning

Q-Learning is a form of Temporal Difference Learning

Q-Learning overestimates action values

$$Q_\pi(s,a) \equiv \mathbb{E}[R_1 + \gamma R_2 + \ldots | S_0 = s, A_0 = a, \pi]$$

$$Q_*(s,a) = \max_\pi Q_\pi(s,a)$$

$$Q(s,a;\theta_t)$$

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t,A_t;\theta_t))\nabla_{\theta_t} Q(S_t,A_t;\theta_t)$$

$$Y_t^Q \equiv R_{t+1} + \gamma \max_a Q(S_{t+1},a;\theta_t)$$

$$Q(S_t,A_t;\theta_{t+1}) \approx Q(S_t,A_t,\theta_t) + \alpha(Y_t^Q - Q(S_t,A_t;\theta_t))\|\nabla_{\theta_t} Q(S_t,A_t;\theta_t)\|^2$$

Taylor Series Expansion

Deep Q Networks

⋆ target network

⋆ experience replay

$$Y_t^{DQN} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1},a;\theta_t^-)$$

Double Q-Learning

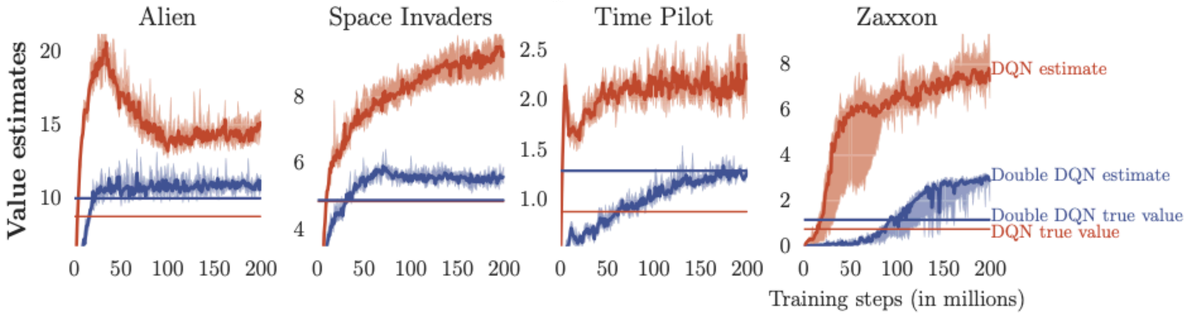
⋆ decouple selection from evaluation

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1},a;\theta_t);\theta_t)$$

$$Y_t^{DoubleQ} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1},a;\theta_t);\theta_t') \rightarrow \text{two value functions}$$

Double DQN

$$Y_t^{DoubleDQN} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1},a;\theta_t);\theta_t^-) \rightarrow \text{target network}$$



Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." *arXiv preprint arXiv:1509.06461* (2015).

Prioritized Experience Replay

Experience replay deals with the following two issues: (a) strongly correlated updates that break the i.i.d. assumption of many popular stochastic gradient-based algorithms, and (b) the rapid forgetting of possibly rare experiences that would be useful later on.

$e = (s, a, r, s') \rightarrow$ experience (transition)

$\mathcal{E} = \{e_i : i = 1, \ldots, N\} \rightarrow$ replay memory

Prioritizing Experiences with TD-Error

$$\delta = r + \gamma Q_{\text{target}}(s', \arg\max_{a'} Q(s', a')) - Q(s, a)$$

$\delta \rightarrow$ temporal difference error (TD-Error)

"importance" of transition e

how "surprising" or unexpected the transition is

δ keeps getting updated each time the corresponding experience is sampled from the buffer

$p_i > 0 \rightarrow$ priority of transition i

$p_i = |\delta_i| + \varepsilon \rightarrow$ proportional prioritization (direct)

$\varepsilon \rightarrow$ prevent edge-cases (e.g., transitions not being revisited once their error is zero)

$p_i = \frac{1}{\text{rank}(i)} \rightarrow$ rank-based prioritization (indirect)

$\text{rank}(i) \rightarrow$ rank of transition i when the replay memory is sorted according to $|\delta_i|$

Stochastic Prioritization

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha} \rightarrow \text{probability of sampling transition } i$$

$\alpha = 0 \rightarrow$ uniform sampling

Annealing the Bias

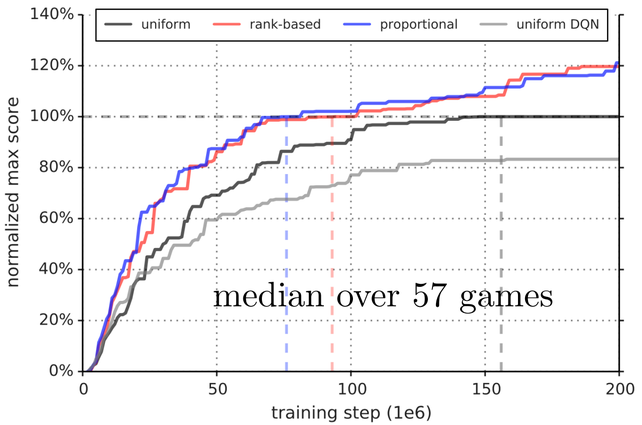
$w_i = (\frac{1}{N}\frac{1}{P(i)})^\beta \rightarrow$ importance sampling weights

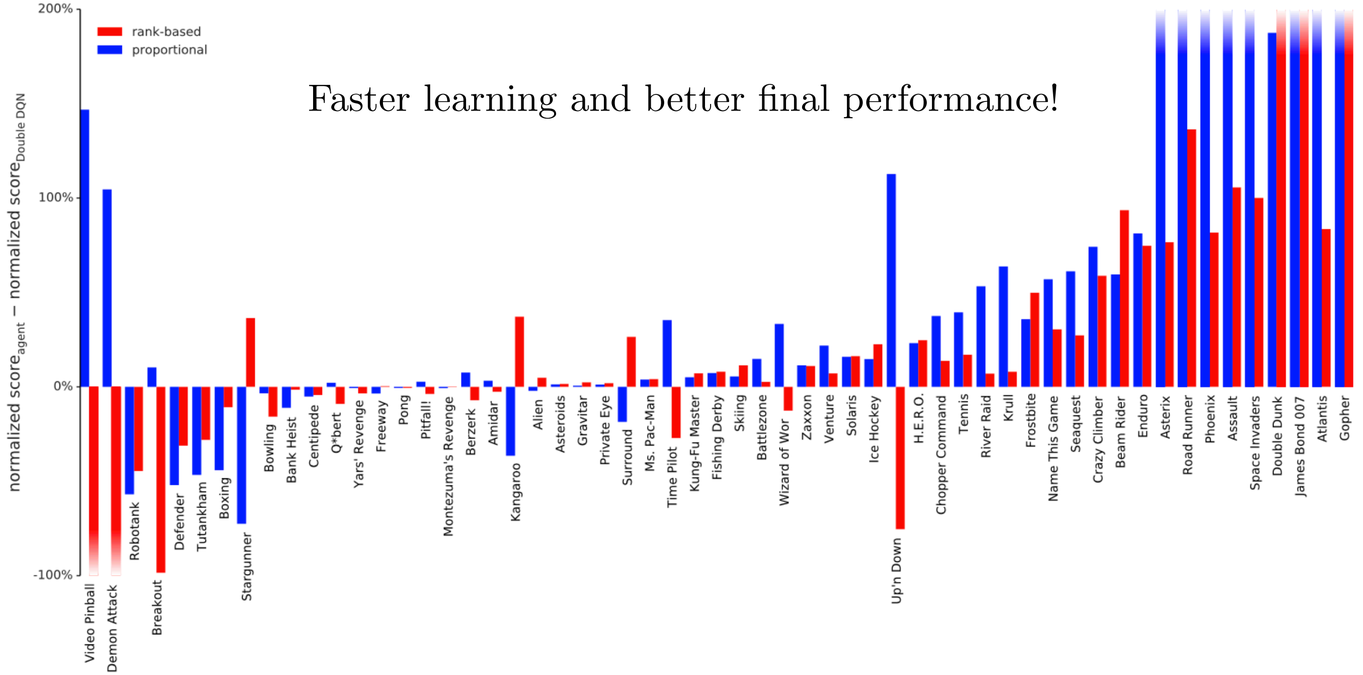
$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{N} w_i |\delta_i|^2 \rightarrow \text{loss function}$$

$\beta = 1 \rightarrow$ fully compensates for non-uniform sampling

$\beta \rightarrow$ annealed to start from β_0 and reach 1 at the end of training

normalize the weights by $\max_j w_j$ to always scale the gradient updates downwards

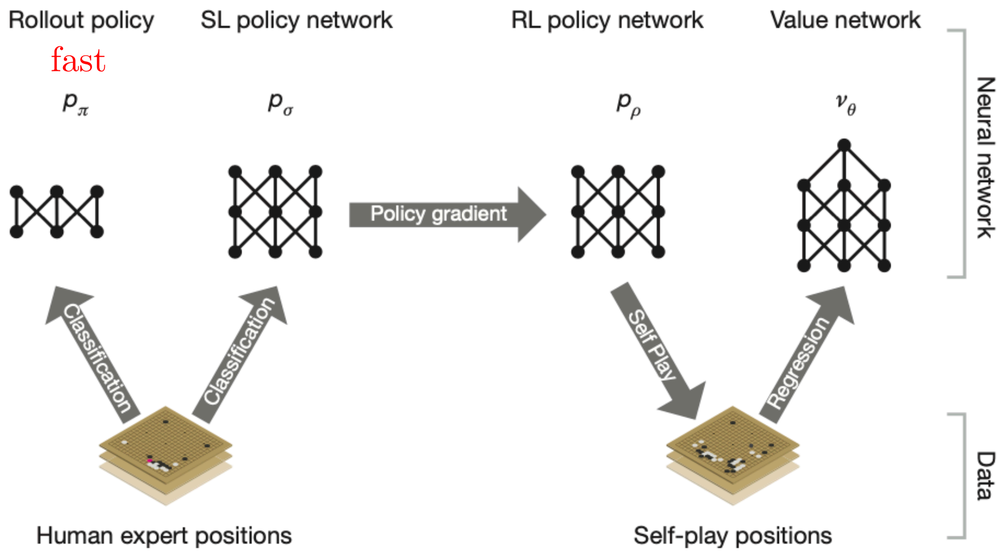


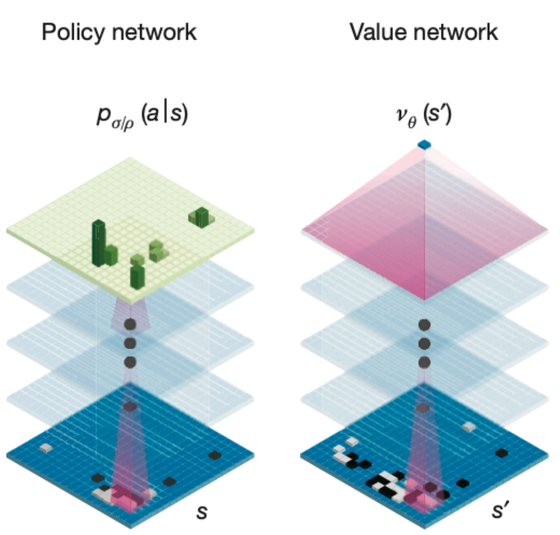


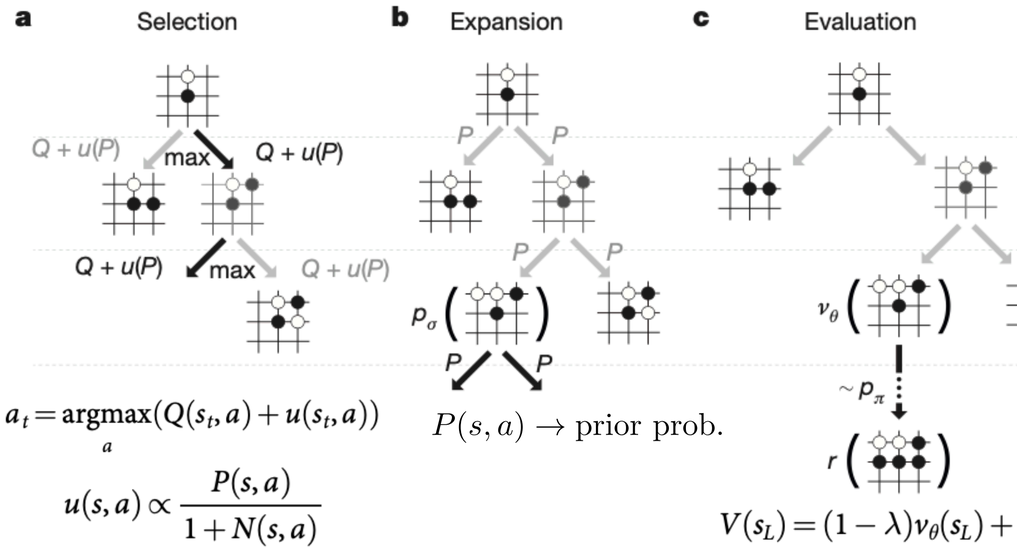
Schaul, Tom, et al. "Prioritized experience replay." *arXiv preprint arXiv:1511.05952* (2015).

Mastering the game of Go with deep neural networks and tree search

YouTube Video







$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t,a) + u(s_t,a))$$

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

$P(s,a) \rightarrow$ prior prob.

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

Supervised learning of policy networks

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma} \rightarrow \text{stochastic gradient ascent (likelihood)}$$

Reinforcement learning of policy networks

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial\rho} z_t \rightarrow \text{stochastic gradient ascent (expected outcome)}$$

policy gradient reinforcement learning

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial\rho}(z_t - v(s_t)) \rightarrow \text{REINFORCE}$$

variance reduction

$$z_t = \sum_{t'=t+1}^{T} r_{t'} = r_T \text{ since } r_{t'} = 0 \text{ for } t' < T$$

https://bit.ly/31OImVQ

$$r_T = \begin{cases} +1 & \text{winning} \\ -1 & \text{losing} \end{cases}$$

https://bit.ly/2RCWHn1

Reinforcement learning of value networks

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s)) \rightarrow \text{stochastic gradient descent (MSE: mean squared error)}$$

Monte Carlo tree search (MCTS)

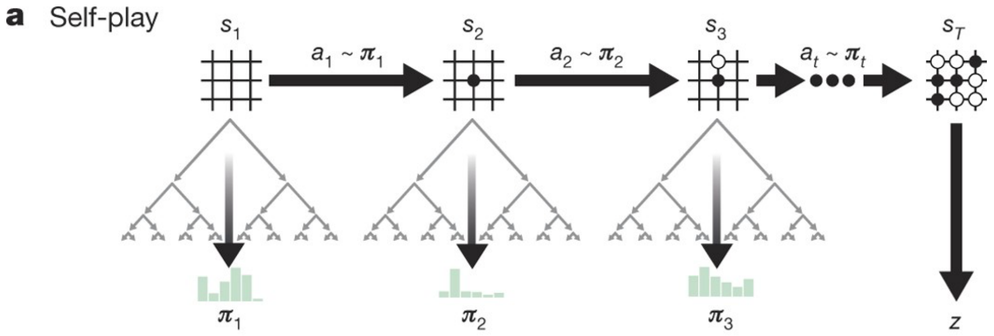
$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i) \rightarrow \text{visit count}$$

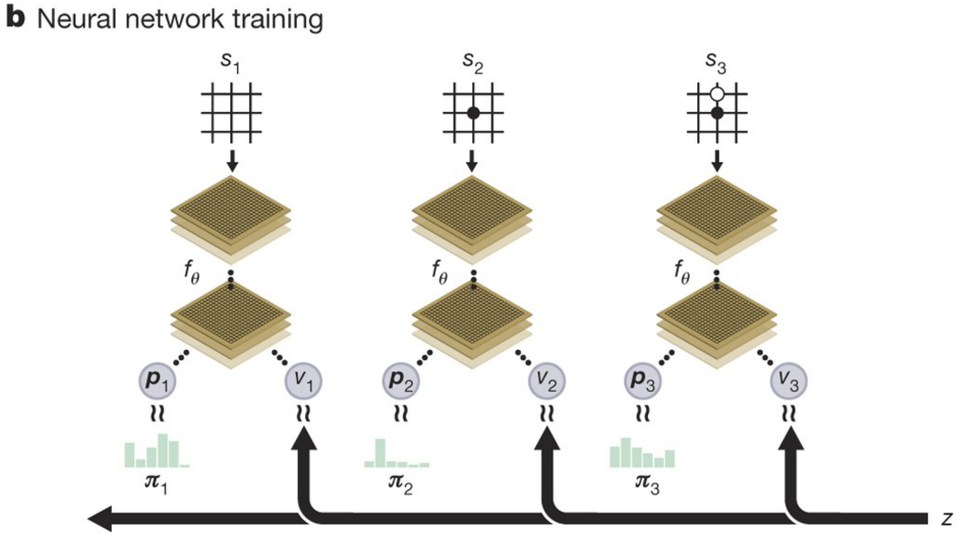
$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i) \rightarrow \text{action value}$$

Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." *nature* 529.7587 (2016): 484-489.

Mastering the game of Go without human knowledge

YouTube Video

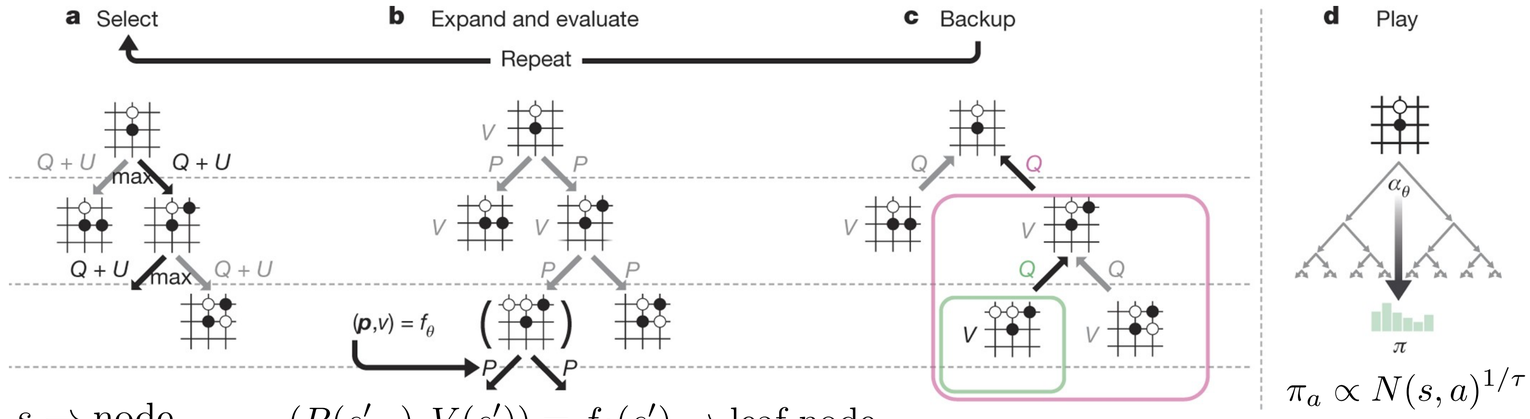




$(\mathbf{p}, v) = f_\theta(s) \rightarrow$ policy & value networks

$$\ell = \underbrace{(z-v)^2}_{\text{MSE}} - \underbrace{\pi^T \log \mathbf{p}}_{\text{cross-entropy}} + \underbrace{c\|\theta\|^2}_{\text{L2 regularization}} \rightarrow \text{loss}$$

$z \rightarrow$ winning or losing



$s \rightarrow$ node $\quad (P(s',.), V(s')) = f_\theta(s') \rightarrow$ leaf node

$(s,a) \rightarrow$ edge

$P(s,a) \rightarrow$ prior prob.

$N(s,a) \rightarrow$ visit count

$Q(s,a) \rightarrow$ action value

$Q(s,a) + U(s,a) \rightarrow$ upper confidence bound

$$U(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{s'|(s,a)\rightarrow s'} V(s')$$

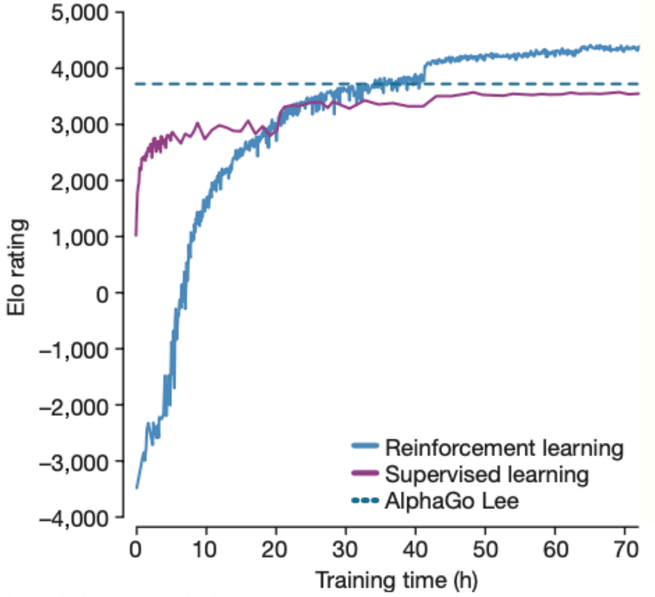
Evaluation

Bayesian Logistic Regression

$$p(a \text{ defeats } b) = \frac{1}{1+\exp(c_{\text{elo}}(e(b)-e(a))}$$

$e(a) \rightarrow$ Elo rating of player a

$c_{\text{elo}} = 1/400$



Silver, David, et al. "Mastering the game of go without human knowledge." *nature* 550.7676 (2017): 354-359.

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

YouTube Video

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

$s \rightarrow$ board position

$\mathbf{p} \rightarrow$ move probability

$a \rightarrow$ action

$p_a = Pr(a|s)$

$v \rightarrow$ expected outcome z of the game from position s

$v \approx \mathbb{E}[z|s]$

$z = +1\ (\text{win}), 0\ (\text{draw}), -1\ (\text{lose})$

Monte-Carlo Tree Search (MCTS)

$s_{\text{root}} \rightarrow$ root state

$\pi_a = Pr(a|s_{\text{root}})$

$\ell = (z - v)^2 - \pi^T \log \mathbf{p} + c\|\theta\|^2 \rightarrow$ loss

Search

$\{ \underbrace{N(s,a)}_{\text{visit count}}, \underbrace{W(s,a)}_{\text{total action-value}}, \underbrace{Q(s,a)}_{\text{mean action-value}}, \underbrace{P(s,a)}_{\text{prior prob.}} \}$

set of statistics stored at each state-action pair (s, a)

$s_0 \rightarrow$ root node of the tree search

$s_L \rightarrow$ leaf node at time-step L

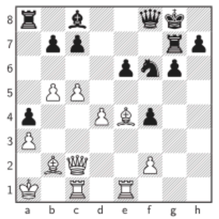
$a_t = \arg\max_a (Q(s_t, a) + U(s_t, a)), \text{for } t < L$

$U(s,a) = C(s)P(s,a)\sqrt{N(s)}/(1 + N(s,a))$

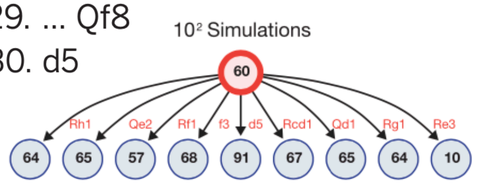
parent visit count

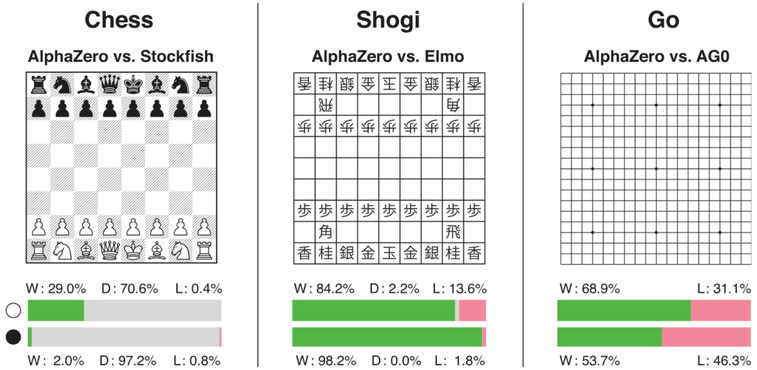
$C(s) = \log\left((1 + N(s) + c_{\text{base}})/c_{\text{base}}\right) + c_{\text{init}}$

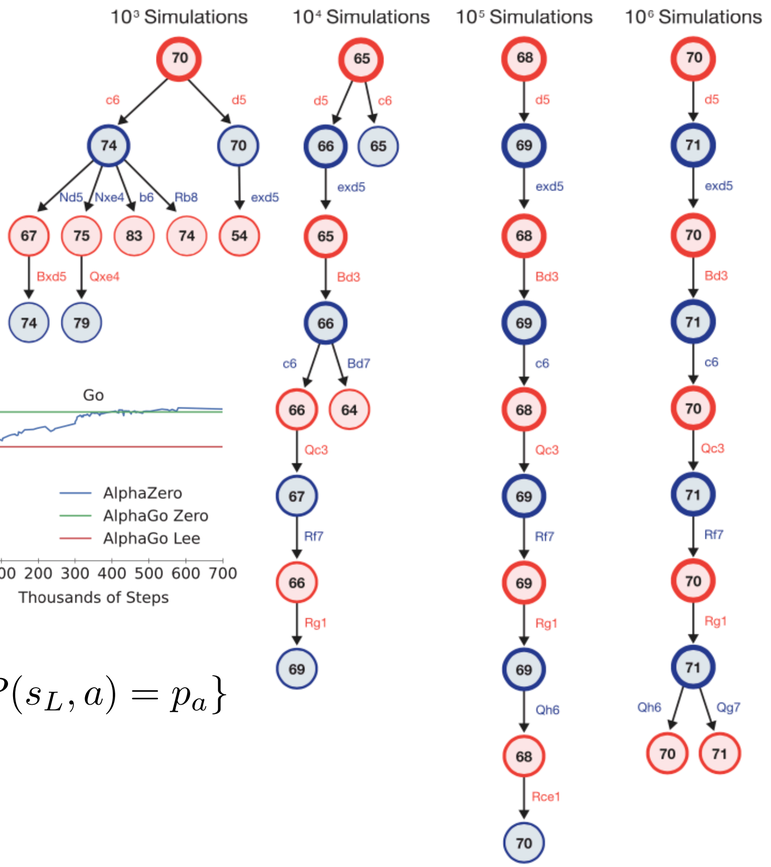
exploration rate

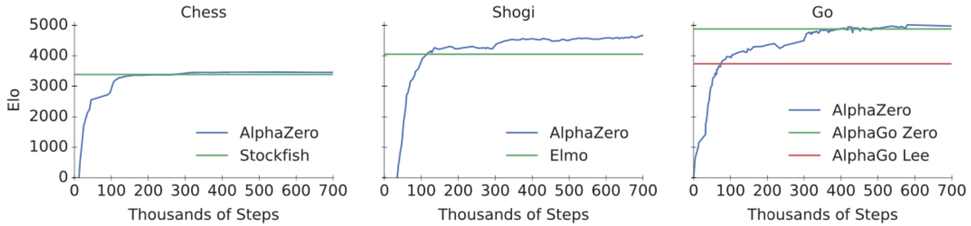


29. ... Qf8
30. d5









$(\mathbf{p}, v) = f_\theta(s_L)$

$\{N(s_L, a) = 0, W(s_L, a) = 0, Q(s_L, a) = 0, P(s_L, a) = p_a\}$

$$\left.\begin{aligned} N(s_t, a_t) &= N(s_t, a_t) + 1 \\ W(s_t, a_t) &= W(s_t, a_t) + v \\ Q(s_t, a_t) &= W(s_t, a_t)/N(s_t, a_t) \end{aligned}\right\} \text{for } t \leq L$$

Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." *Science* 362.6419 (2018): 1140-1144.

Grandmaster level in StarCraft II using multi-agent reinforcement learning

Boulder

StarCraft II is a real-time strategy (RTS) game!
Three races: Terran, Protoss, and Zerg
https://tinyurl.com/3cmj8h8c
https://github.com/Blizzard/s2client-proto
https://github.com/deepmind/pysc2

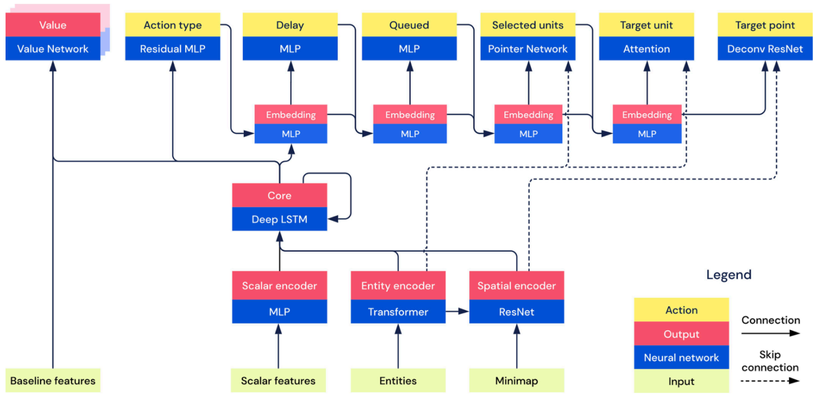
Agent action space

Field	Description
Action type	Which action to execute. Some examples of actions are moving a unit, training a unit from a building, moving the camera, or no-op. See PySC2 for a full list[7]
Selected units	Entities that will execute the action
Target	An entity or location in the map discretised to 256x256 targeted by the action
Queued	Whether to queue this action or execute it immediately
Repeat	Whether or not to issue this action multiple times
Delay	The number of game time-steps to wait until receiving the next observation

approximately 10^{26} possible choices at each step

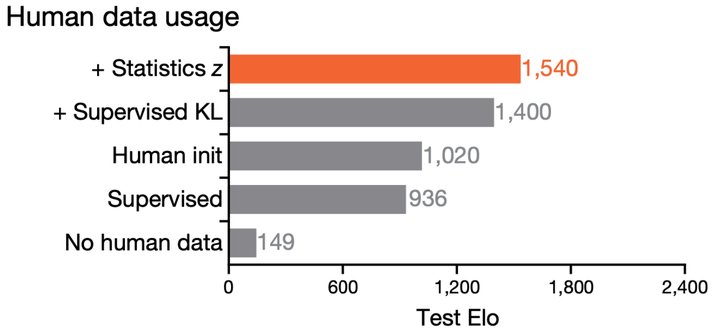
Agent input space

Category	Field	Description
Entities: up to 512	Unit type	E.g. Drone or Forcefield
	Owner	Agent, opponent, or neutral
	Status	Current health, shields, energy
	Display type	E.g. Snapshot, for opponent buildings in the fog of war
	Position	Entity position
	Number of workers	For resource collecting base buildings
	Cooldowns	Attack cooldown
	Attributes	Invisible, powered, hallucination, active, in cargo, and/or on the screen
	Unit attributes	E.g. Biological or Armored
	Cargo status	Current and maximum amount of cargo space
	Building status	Build progress, build queue, and add-on type
	Resource status	Remaining resource contents
	Order status	Order queue and order progress
	Buff status	Buffs and buff durations
Map: 128x128 grid	Height	Heights of map locations
	Visibility	Whether map locations are currently visible
	Creep	Whether there is creep at a specific location
	Entity owners	Which player owns entities
	Alerts	Whether units are under attack
	Pathable	Which areas can be navigated over
	Buildable	Which areas can be built on
Player data	Race	Agent and opponent requested race, and agent actual race
	Upgrades	Agent upgrades and opponent upgrades, if they would be known to humans
	Agent statistics	Agent current resources, supply, army supply, worker supply, maximum supply, number of idle workers, number of Warp Gates, and number of Larva
Game statistics	Camera	Current camera position. The camera is a 32x20 game-unit sized rectangle
	Time	Current time in game



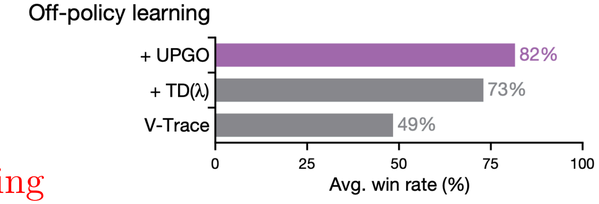
$\pi_\theta(a_t|s_t, z) \rightarrow$ policy of AlphaStar
$s_t = o_{1:t}, a_{1:t-1} \rightarrow$ all previous observations and actions
imperfect information game (limited camera view)

Actor-Critic paradigm (A2C)
$V_\theta(s_t, z) \rightarrow$ value function
trained to predict r_t ($r_T = -1, 0, +1$) and used to update policy!

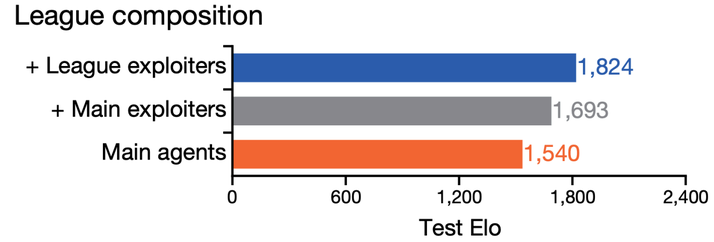


Pseudo-reward: Edit & Hamming distances btw sampled and executed build orders & cumulative statistics (i.e., z)
The first 20 constructed buildings and units
cumulative statistics: units, buildings, effects, and upgrades that were present during a game
z is randomly sampled from human data

Off-policy corrections
off-policy learning: updating the current policy from experience generated by a previous policy
– update policy using V-trace
– update value estimate using $TD(\lambda)$ } temporal difference learning
– UPGO (upgoing policy update): moves the policy towards trajectories with better than average reward



League Training
Self-play (SP): games btw latest agents for all three races
StarCraft is a non-transitive game!
(i.e., A defeats B, and B defeats C, but A loses to C)
Prioritized fictitious self-play (PFSP):
play against all previous players in the league
(focus on hardest players or opponents at similar level)



Vinyals, Oriol, et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning." *Nature* 575.7782 (2019): 350-354.

Boulder

Questions?

YouTube Playlist