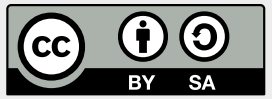
TD3, Monte Carlo Tree Search

Milan Straka

December 17, 2018

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics



Combining continuous actions and Deep Q Networks is not straightforward. In order to do so, we need a different variant of the policy gradient theorem.

Recall that in policy gradient theorem,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \sum_{s \in \mathcal{S}} \mu(s) \sum_{a \in \mathcal{A}} q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s;\boldsymbol{\theta}).$$

Deterministic Policy Gradient Theorem

Assume that the policy $\pi(s;\boldsymbol{\theta})$ is deterministic and computes an action $a \in \mathbb{R}$. Then under several assumptions about continuousness, the following holds:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \mathbb{E}_{s \sim \mu(s)} \left[\nabla_{\boldsymbol{\theta}} \pi(s;\boldsymbol{\theta}) \nabla_a q_\pi(s,a) \big|_{a=\pi(s;\boldsymbol{\theta})} \right].$$

The theorem was first proven in the paper Deterministic Policy Gradient Algorithms by David Silver et al.

Deep Deterministic Policy Gradients

Note that the formulation of deterministic policy gradient theorem allows an off-policy algorithm, because the loss functions no longer depends on actions (similarly to how expected Sarsa is also an off-policy algorithm).

We therefore train function approximation for both $\pi(s;\boldsymbol{\theta})$ and $q(s,a;\boldsymbol{\theta})$, training $q(s,a;\boldsymbol{\theta})$ using a deterministic variant of the Bellman equation:

$$q(S_t,A_t;\boldsymbol{\theta}) = \mathbb{E}_{R_{t+1},S_{t+1}}\left[R_{t+1} + \gamma q(S_{t+1},\pi(S_{t+1};\boldsymbol{\theta}))\right]$$

and $\pi(s;\boldsymbol{\theta})$ according to the deterministic policy gradient theorem.

The algorithm was first described in the paper Continuous Control with Deep Reinforcement Learning by Timothy P. Lillicrap et al. (2015).

The authors utilize a replay buffer, a target network (updated by exponential moving average with $\tau = 0.001$), batch normalization for CNNs, and perform exploration by adding a normal-distributed noise to predicted actions. Training is performed by Adam with learning rates of 1e-4 and 1e-3 for the policy and critic network, respectively.

Deep Deterministic Policy Gradients

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Algorithm 1 of the paper "Continuous Control with Deep Reinforcement Learning" by Timothy P. Lillicrap et al.

Twin Delayed Deep Deterministic Policy Gradient

The paper Addressing Function Approximation Error in Actor-Critic Methods by Scott Fujimoto et al. from February 2018 proposes improvements to DDPG which

- decrease maximization bias by training two critics and choosing minimum of their predictions;
- introduce several variance-lowering optimizations:
  - delayed policy updates;
  - target policy smoothing.

TD3 – Maximization Bias

Similarly to Q-learning, the DDPG algorithm suffers from maximization bias. In Q-learning, the maximization bias was caused by the explicit $\max$ operator. For DDPG methods, it can be caused by the gradient descent itself. Let $\boldsymbol{\theta}_{approx}$ be the parameters maximizing the $q_{\boldsymbol{\theta}}$ and let $\boldsymbol{\theta}_{true}$ be the hypothetical parameters which maximise true q_π, and let π_{approx} and π_{true} denote the corresponding policies.

Because the gradient direction is a local maximizer, for sufficiently small $\alpha < \varepsilon_1$ we have

$$\mathbb{E}\left[q_{\boldsymbol{\theta}}(S, \pi_{approx})\right] \geq \mathbb{E}\left[q_{\boldsymbol{\theta}}(S, \pi_{true})\right].$$

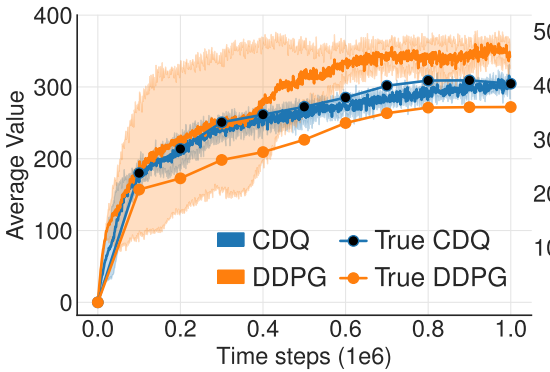
However, for real q_π and for sufficiently small $\alpha < \varepsilon_2$ it holds that

$$\mathbb{E}\left[q_{\pi}(S, \pi_{true})\right] \geq \mathbb{E}\left[q_{\pi}(S, \pi_{approx})\right].$$

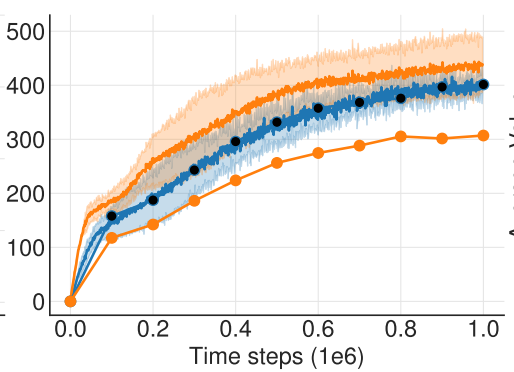
Therefore, if $\mathbb{E}\left[q_{\boldsymbol{\theta}}(S, \pi_{true})\right] \geq \mathbb{E}\left[q_{\pi}(S, \pi_{true})\right]$, for $\alpha < \min(\varepsilon_1, \varepsilon_2)$

$$\mathbb{E}\left[q_{\boldsymbol{\theta}}(S, \pi_{approx})\right] \geq \mathbb{E}\left[q_{\pi}(S, \pi_{approx})\right].$$

TD3 – Maximization Bias

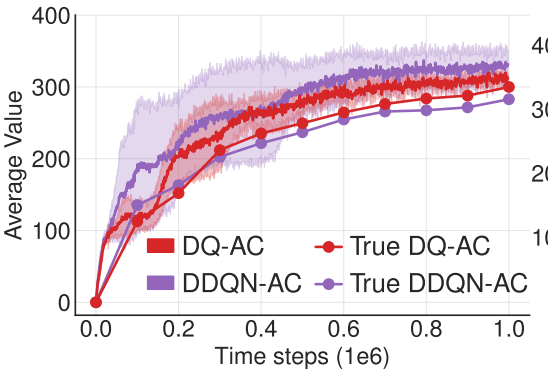


(a) Hopper-v1

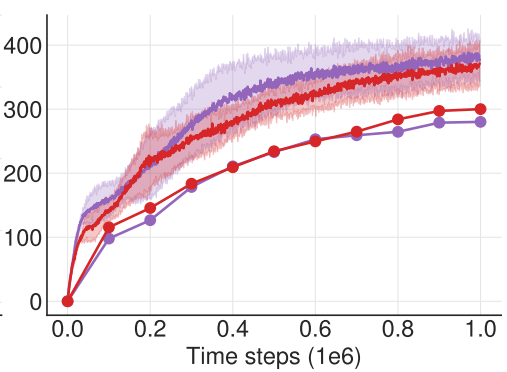


(b) Walker2d-v1

Figure 1 of the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.



(a) Hopper-v1



(b) Walker2d-v1

Figure 2 of the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

Analogously to Double DQN we could compute the learning targets using the current policy and the target critic, i.e., $r + \gamma q_{\boldsymbol{\theta}'}(s', \pi_{\boldsymbol{\theta}}(s'))$ (instead of using target policy and target critic as in DDPG), obtaining DDQN-AC algorithm. However, the authors found out that the policy changes too slowly and the target and current networks are too similar.

Using the original Double Q-learning, two pairs of actors and critics could be used, with the learning targets computed by the opposite critic, i.e., $r + \gamma q_{\boldsymbol{\theta}'_2}(s', \pi_{\boldsymbol{\theta}_1}(s))$ for updating $q_{\boldsymbol{\theta}_1}$.

The resulting DQ-AC algorithm is slightly better, but still suffering from oversetimation.

TD3 – Algorithm

The authors instead suggest to employ two critics and one actor. The actor is trained using one of the critics, and both critics are trained using the same target computed using the *minimum* value of both critics as

$$r + \gamma \min_{i=1,2} q_{\boldsymbol{\theta}'_i}(s', \pi_{\boldsymbol{\theta}}(s')).$$

Furthermore, the authors suggest two additional improvements for variance reduction.

- For obtaining higher quality target values, the authors propose to train the critics more often. Therefore, critics are updated each step, but the actor and the target networks are updated only every d-th step ($d = 2$ is used in the paper).
- To explictly model that similar actions should lead to similar results, a small random noise is added to performed actions when computing the target value:

$$r + \gamma \min_{i=1,2} q_{\boldsymbol{\theta}'_i}(s', \pi_{\boldsymbol{\theta}}(s') + \varepsilon) \quad \text{for} \quad \varepsilon \sim \mathrm{clip}(\mathcal{N}(0, \sigma), -c, c).$$

TD3 – Algorithm

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ
with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
  $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

Algorithm 1 of the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

TD3 – Algorithm

Hyper-parameter	Ours	DDPG
Critic Learning Rate	10^{-3}	10^{-3}
Critic Regularization	None	$10^{-2} \cdot \|\|\theta\|\|^2$
Actor Learning Rate	10^{-3}	10^{-4}
Actor Regularization	None	None
Optimizer	Adam	Adam
Target Update Rate (τ)	$5 \cdot 10^{-3}$	10^{-3}
Batch Size	100	64
Iterations per time step	1	1
Discount Factor	0.99	0.99
Reward Scaling	1.0	1.0
Normalized Observations	False	True
Gradient Clipping	False	False
Exploration Policy	$\mathcal{N}(0, 0.1)$	OU, $\theta = 0.15, \mu = 0, \sigma = 0.2$

Table 3 of the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

TD3 – Results

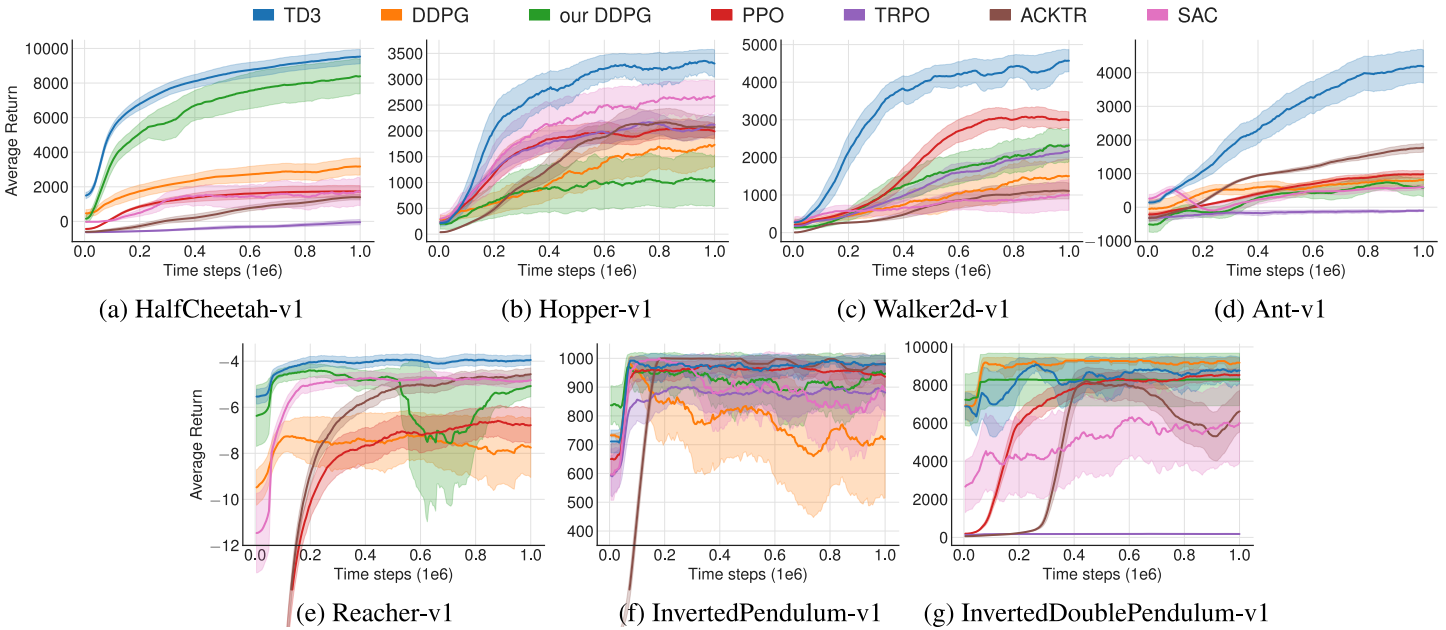
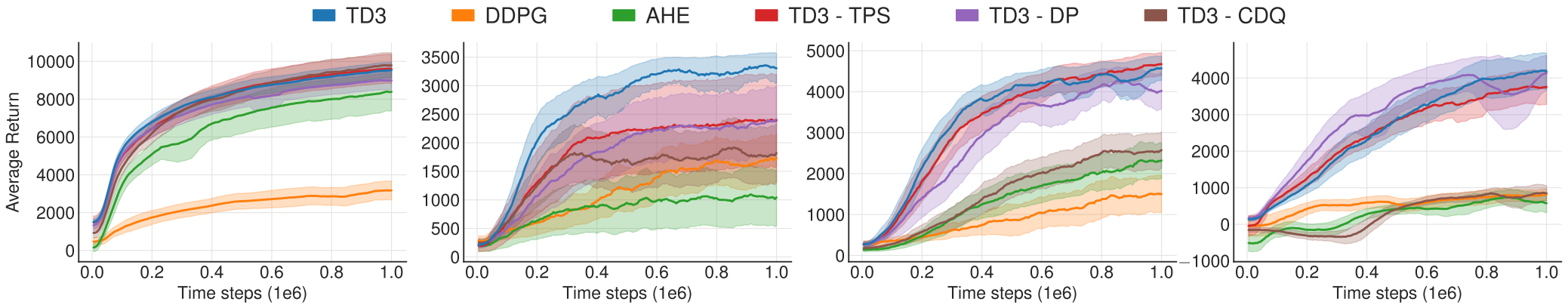


Figure 5 of the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

Environment	TD3	DDPG	Our DDPG	PPO	TRPO	ACKTR	SAC
HalfCheetah	**9636.95 ± 859.065**	3305.60	8577.29	1795.43	-15.57	1450.46	2347.19
Hopper	**3564.07 ± 114.74**	2020.46	1860.02	2164.70	2471.30	2428.39	2996.66
Walker2d	**4682.82 ± 539.64**	1843.85	3098.11	3317.69	2321.47	1216.70	1283.67
Ant	**4372.44 ± 1000.33**	1005.30	888.77	1083.20	-75.85	1821.94	655.35
Reacher	**-3.60 ± 0.56**	-6.51	**-4.01**	-6.18	-111.43	-4.26	-4.44
InvPendulum	**1000.00 ± 0.00**	**1000.00**	**1000.00**	**1000.00**	985.40	**1000.00**	**1000.00**
InvDoublePendulum	**9337.47 ± 14.96**	**9355.52**	8369.95	8977.94	205.85	9081.92	8487.15

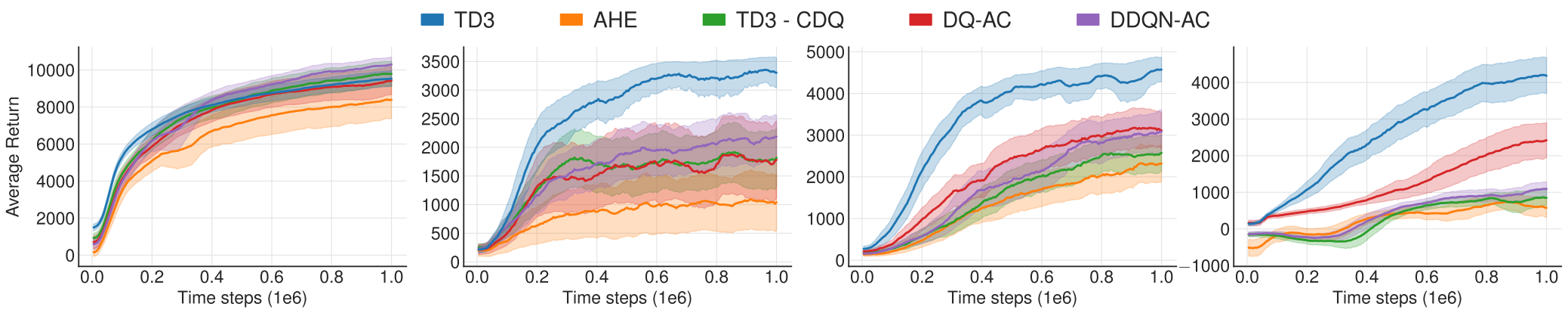
Table 1 of the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

TD3 – Ablations



(a) HalfCheetah-v1 (b) Hopper-v1 (c) Walker2d-v1 (d) Ant-v1

Figure 7 of the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.



(a) HalfCheetah-v1 (b) Hopper-v1 (c) Walker2d-v1 (d) Ant-v1

Figure 8 of the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

TD3 – Ablations

Method	HCheetah	Hopper	Walker2d	Ant
TD3	9532.99	**3304.75**	**4565.24**	**4185.06**
DDPG	3162.50	1731.94	1520.90	816.35
AHE	8401.02	1061.77	2362.13	564.07
AHE + DP	7588.64	1465.11	2459.53	896.13
AHE + TPS	9023.40	907.56	2961.36	872.17
AHE + CDQ	6470.20	1134.14	3979.21	3818.71
TD3 - DP	9590.65	2407.42	**4695.50**	3754.26
TD3 - TPS	8987.69	2392.59	4033.67	**4155.24**
TD3 - CDQ	9792.80	1837.32	2579.39	849.75
DQ-AC	9433.87	1773.71	3100.45	2445.97
DDQN-AC	**10306.90**	2155.75	3116.81	1092.18

Table 2 of the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

On 7 December 2018, the AlphaZero paper came out in Science journal. It demonstrates learning chess, shogi and go, *tabula rasa* – without any domain-specific human knowledge or data, only using self-play. The evaluation is performed against strongest programs available.

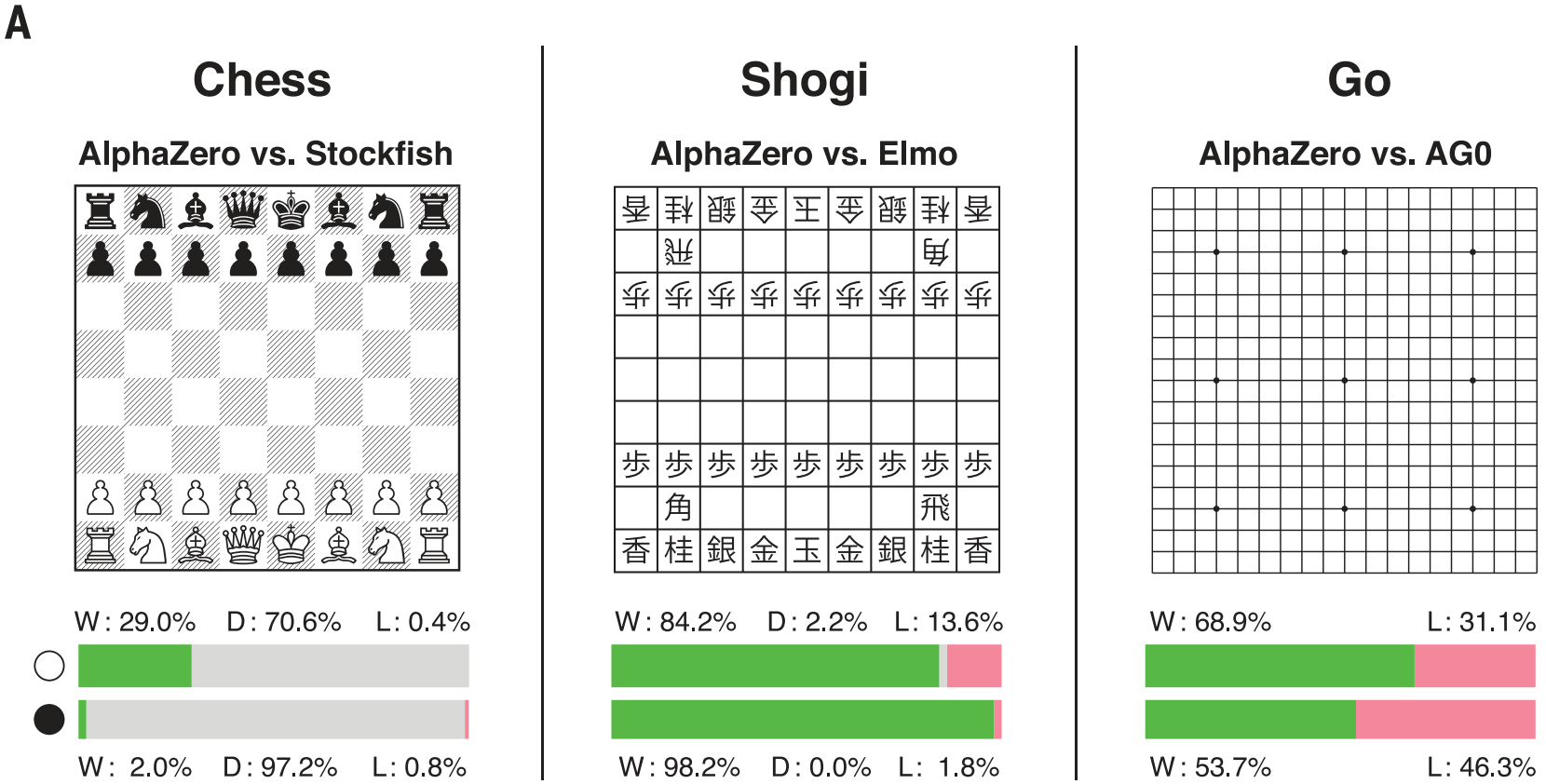


Figure 2 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Overview

AlphaZero uses a neural network which using the current state s predicts $(\boldsymbol{p}, v) = f(s; \boldsymbol{\theta})$, where:

- $\boldsymbol{p}$ is a vector of move probabilities, and
- v is expected outcome of the game in range $[-1, 1]$.

Instead of usual alpha-beta search used by classical game playing programs, AlphaZero uses Monte Carlo Tree Search (MCTS). By a sequence of simulated self-play games, the search can improve the estimate of $\boldsymbol{p}$ and v, and can be considered a powerful policy evaluation operator.

The network is trained from self-play games. The game is played by repeatedly running MCTS from the state s_t and choosing a move $a_t \sim \boldsymbol{\pi}_t$, until a terminal position s_T is encountered, which is scored according to game rules as $z \in \{-1, 0, 1\}$. Finally, the network parameters are trained to minimize the error between the predicted outcome v and simulated outcome z, and maximize the similarity of the policy vector $\boldsymbol{p}_t$ and the search probabilities $\boldsymbol{\pi}_t$:

$$L \stackrel{\text{def}}{=} (z - v)^2 + \boldsymbol{\pi}^T \log \boldsymbol{p} + c||\boldsymbol{\theta}||^2.$$

AlphaZero – Monte Carlo Tree Search

MCTS keeps a tree of currently explored states from a fixed root state. Each node corresponds to a game state. Each state-action pair (s, a) stores the following set of statistics:

- visit count $N(s, a)$,
- total action-value $W(s, a)$,
- mean action value $Q(s, a) \stackrel{\text{def}}{=} W(s, a)/N(s, a)$,
- prior probability $P(s, a)$ of selecting action a in state s.

Each simulation starts in the root node and finishes in a leaf node s_L. In a state s_t, an action is selected using a variant of PUCT algorithm as $a_t = \arg\max_a(Q(s_t, a) + U(s_t, a))$, where

$$U(s, a) \stackrel{\text{def}}{=} C(s)P(s, a)\frac{\sqrt{N(s)}}{1 + N(s, a)}$$

with $C(s) = \log((1 + N(s) + c_{\text{base}})/c_{\text{base}}) + c_{\text{init}}$ being slightly time-increasing exploration rate. Additionally, exploration in s_{root} is supported by $P(s_{\text{root}}, a) = (1 - \varepsilon)p_a + \varepsilon \operatorname{Dir}(\alpha)$, with $\varepsilon = 0.25$ and $\alpha = 0.3, 0.15, 0.03$ for chess, shogi and go, respectively.

AlphaZero – Monte Carlo Tree Search

When reaching a leaf node, it is evaluated by the network producing $(\boldsymbol{p}, v)$ and all its children are initialized to $N = W = Q = 0$, $P = \boldsymbol{p}$, and in the backward pass for all $t \leq L$ the statistics are updates using $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$ and $W(s_t, a_t) \leftarrow W(s_t, a_t) + v$.

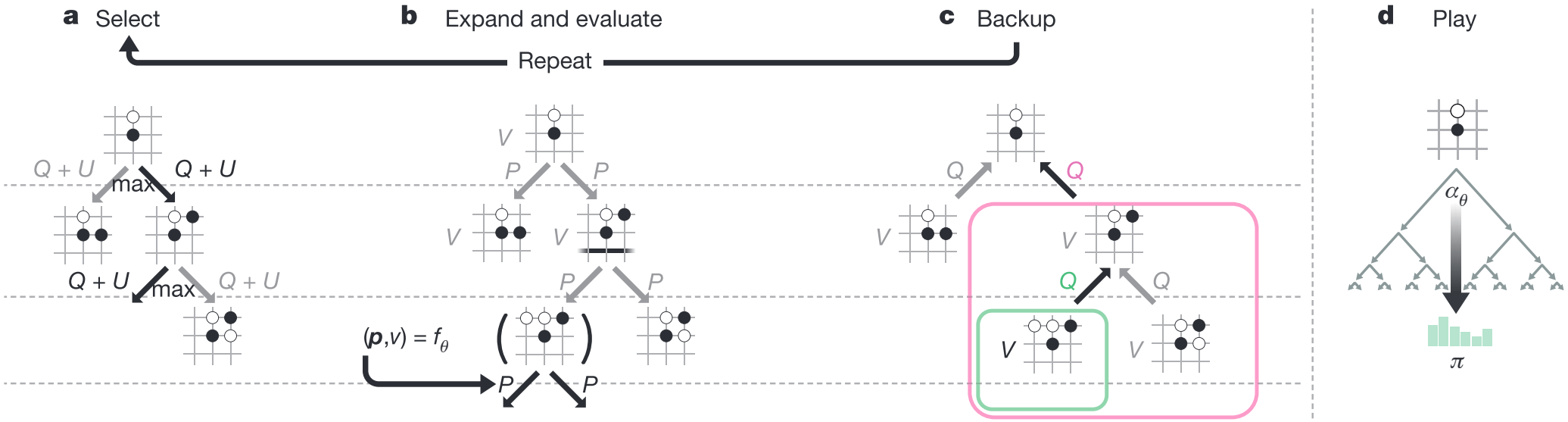


Figure 2 of the paper "Mastering the game of Go without human knowledge" by David Silver et al.

Finally, the search probabilities in the root are defined as $\boldsymbol{\pi}_{\text{root}} \propto N(s_{\text{root}}, \cdot)$.

AlphaZero – Network Architecture

The network processes game-specific input, which consists of a history of 8 board positions encoded by several $N \times N$ planes, and some number of constant-valued inputs.

Output is considered to be a categorical distribution of possible moves. For chess and shogi, for each piece we consider all possible moves (56 queen moves, 8 knight moves and 9 underpromotions for chess).

The input is processed by:

- initial convolution block with CNN with 256 3×3 kernels with stride 1, batch normalization and ReLU activation,
- 19 residual blocks, each consisting of two CNN with 256 3×3 kernels with stride 1, batch normalization and ReLU activation, and a residual connection around them,
- *policy head*, which applies another CNN with batch normalization, followed by a convolution with 73/139 filters for chess/shogi, or a linear layer of size 362 for go,
- *value head*, which applies another CNN with 1 1×1 kernel with stride 1, followed by a ReLU layer of size 256 and final $\tanh$ layer of size 1.

AlphaZero – Network Inputs

Go		Chess		Shogi	
Feature	Planes	Feature	Planes	Feature	Planes
P1 stone	1	P1 piece	6	P1 piece	14
P2 stone	1	P2 piece	6	P2 piece	14
		Repetitions	2	Repetitions	3
				P1 prisoner count	7
				P2 prisoner count	7
Colour	1	Colour	1	Colour	1
		Total move count	1	Total move count	1
		P1 castling	2		
		P2 castling	2		
		No-progress count	1		
Total	17	Total	119	Total	362

Table S1 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Network Outputs

Chess		Shogi	
Feature	Planes	Feature	Planes
Queen moves	56	Queen moves	64
Knight moves	8	Knight moves	2
Underpromotions	9	Promoting queen moves	64
		Promoting knight moves	2
		Drop	7
Total	73	Total	139

Table S2 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Training

Training is performed by running self-play games of the network with itself. Each MCTS uses 800 simulations. A replay buffer of one million most recent games is kept.

During training, 5000 first-generation TPUs are used to generate self-play games. Simultaneously, network is trained using SGD with momentum of 0.9 on batches of size 4096, utilizing 16 second-generation TPUs. Training takes approximately 9 hours for chess, 12 hours for shogi and 13 days for go.

AlphaZero – Training

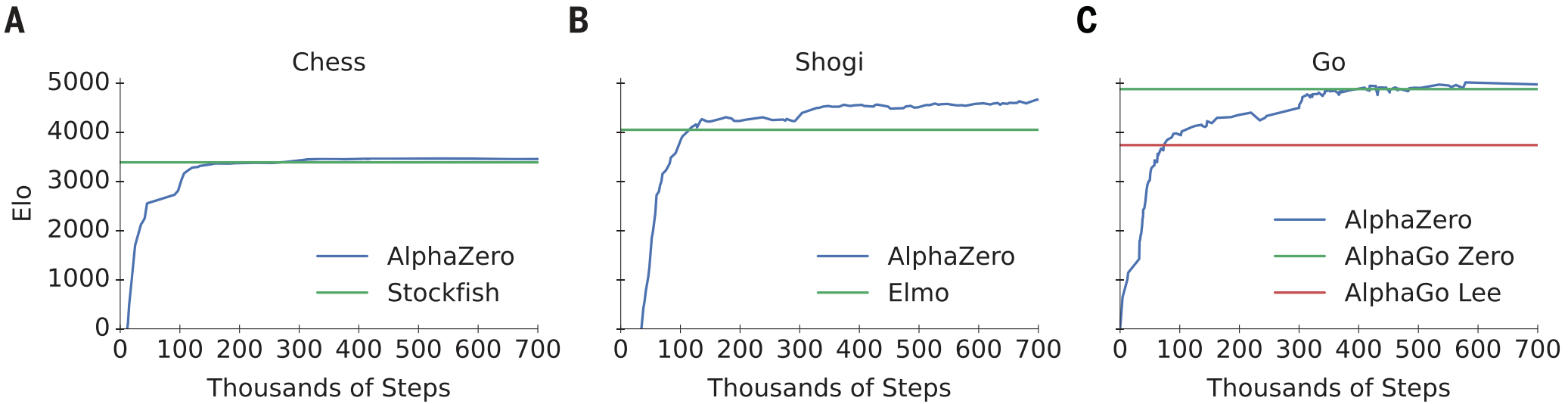


Figure 1 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

	Chess	Shogi	Go
Mini-batches	700k	700k	700k
Training Time	9h	12h	13d
Training Games	44 million	24 million	140 million
Thinking Time	800 sims	800 sims	800 sims
	$\sim$ 40 ms	$\sim$ 80 ms	$\sim$ 200 ms

Table S3 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Training

According to the authors, training is highly repeatable.

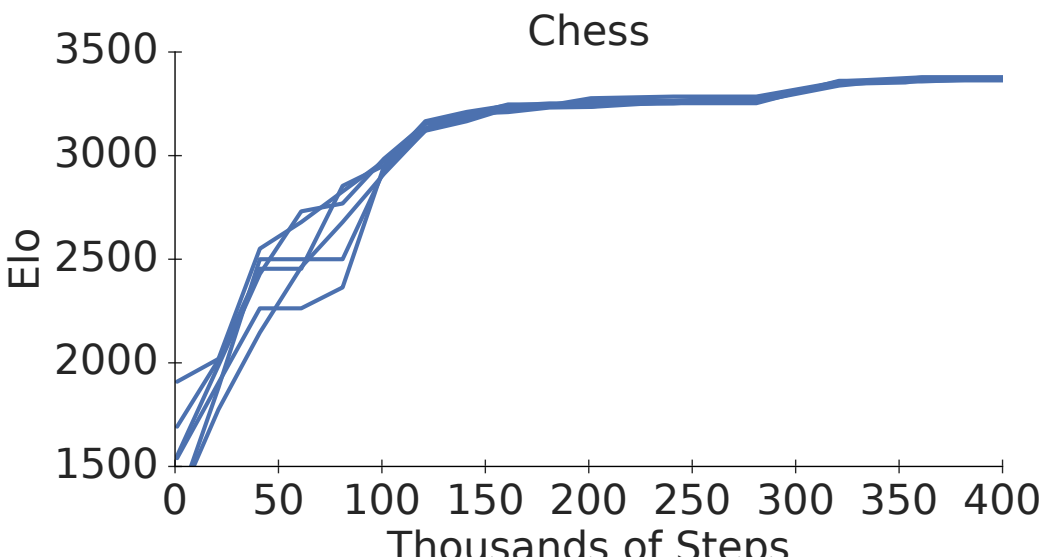


Figure S3 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Symmetries

In the original AlphaGo Zero, symmetries were explicitly utilized, by

- randomly sampling a symmetry during training,
- randomly sampling a symmetry during evaluation.

However, AlphaZero does not utilize symmetries in any way (because chess and shogi do not have them).

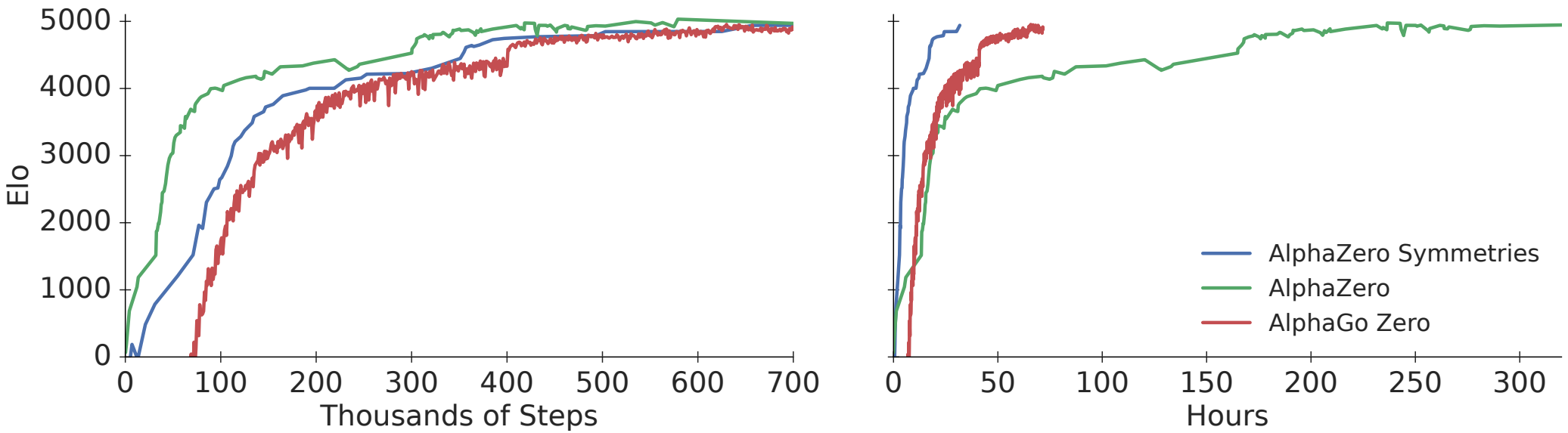


Figure S1 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Inference

During inference, AlphaZero utilizes much less evaluations than classical game playing programs.

Program	Chess	Shogi	Go
AlphaZero	63k (13k)	58k (12k)	16k (0.6k)
Stockfish	58,100k (24,000k)		
Elmo		25,100k (4,600k)	
AlphaZero	1.5 GFlop	1.9 GFlop	8.5 GFlop

Table S4 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Ablations

Fig.	Match	Start Position	AlphaZero Book	AlphaZero Main	AlphaZero Inc	Opponent Book	Opponent Main	Opponent Inc	Opponent Program
2A	Main	Initial Board	No	3h	15s	No	3h	15s	Stockfish 8
2B	1/100 time	Initial Board	No	108s	0.15s	No	3h	15s	Stockfish 8
2B	1/30 time	Initial Board	No	6min	0.5s	No	3h	15s	Stockfish 8
2B	1/10 time	Initial Board	No	18min	1.5s	No	3h	15s	Stockfish 8
2B	1/3 time	Initial Board	No	1h	5s	No	3h	15s	Stockfish 8
2C	latest Stockfish	Initial Board	No	3h	15s	No	3h	15s	Stockfish 2018.01.13
2C	Opening Book	Initial Board	No	3h	15s	Yes	3h	15s	Stockfish 8
2D	Human Openings	Figure 3A	No	3h	15s	No	3h	15s	Stockfish 8
2D	TCEC Openings	Figure S4	No	3h	15s	No	3h	15s	Stockfish 8

Table S8 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

Fig.	Match	Start Position	AlphaZero Book	AlphaZero Main	AlphaZero Inc	Opponent Book	Opponent Main	Opponent Inc	Opponent Program
2A	Main	Initial Board	No	3h	15s	Yes	3h	15s	Elmo
2B	1/100 time	Initial Board	No	108s	0.15s	Yes	3h	15s	Elmo
2B	1/30 time	Initial Board	No	6min	0.5s	Yes	3h	15s	Elmo
2B	1/10 time	Initial Board	No	18min	1.5s	Yes	3h	15s	Elmo
2B	1/3 time	Initial Board	No	1h	5s	Yes	3h	15s	Elmo
2C	Aperyqhapaq	Initial Board	No	3h	15s	No	3h	15s	Aperyqhapaq
2C	CSA time control	Initial Board	No	10min	10s	Yes	10min	10s	Elmo
2D	Human Openings	Figure 3B	No	3h	15s	Yes	3h	15s	Elmo

Table S9 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Ablations

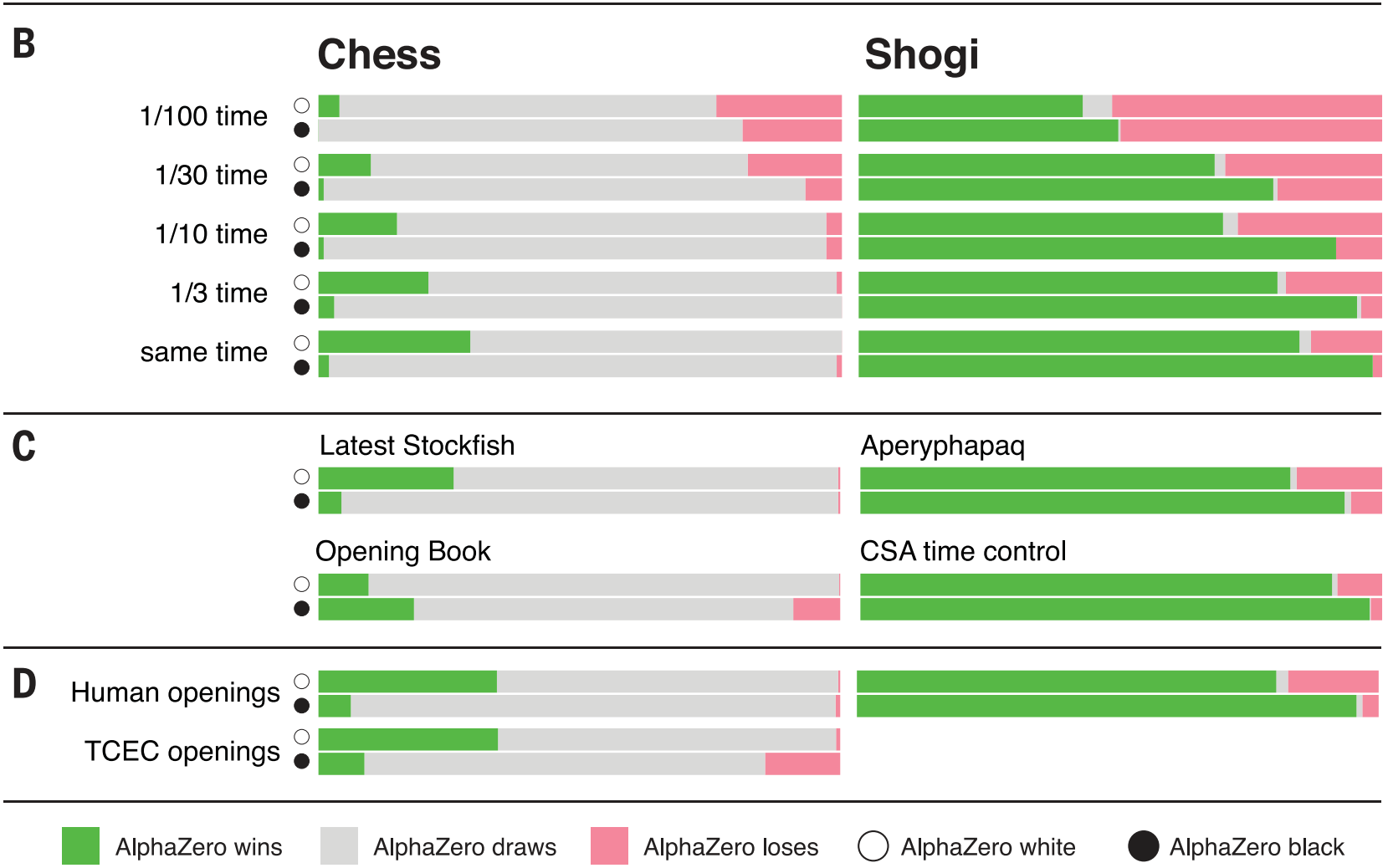


Figure 2 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Ablations

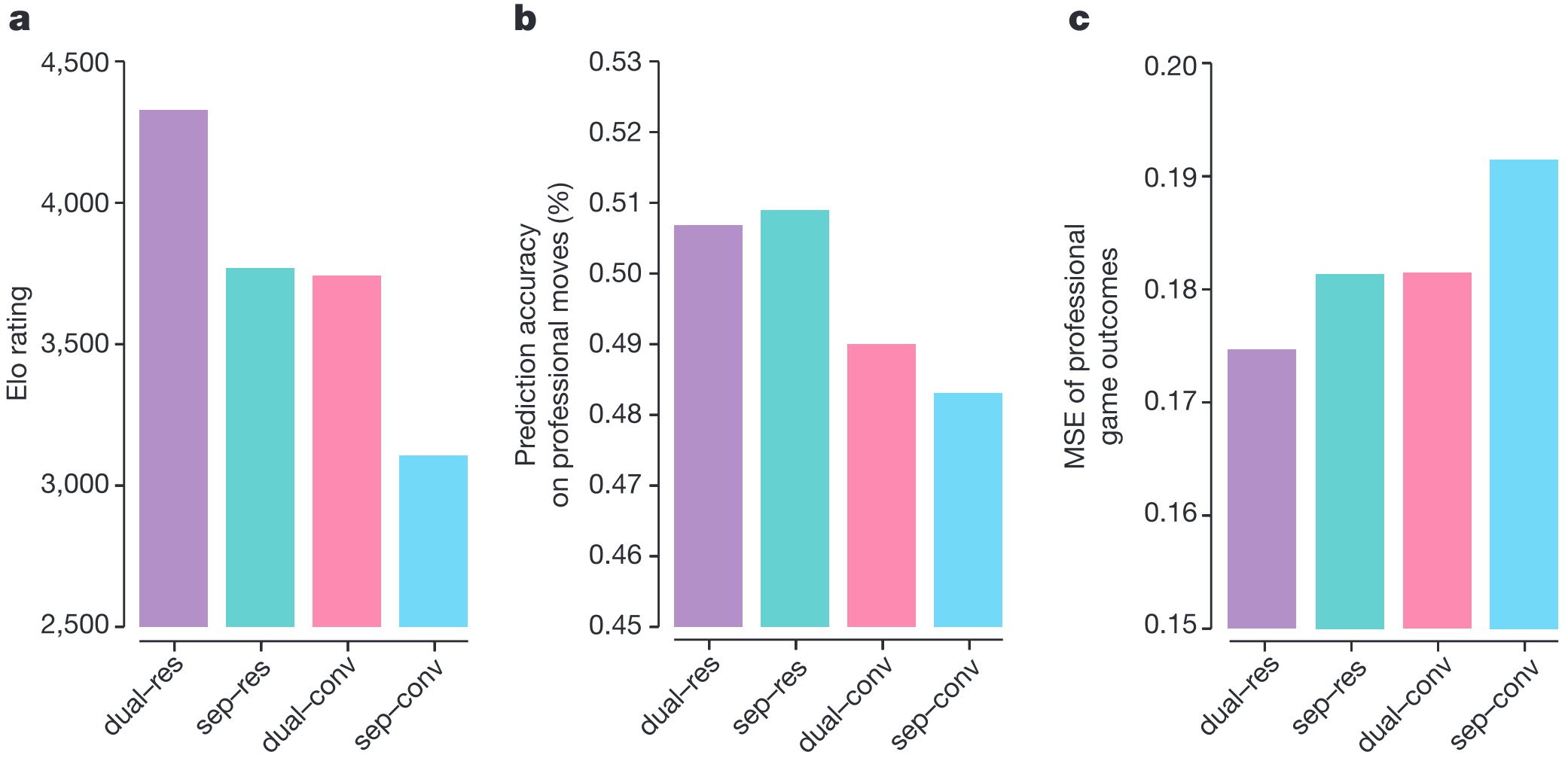


Figure 4 of the paper "Mastering the game of Go without human knowledge" by David Silver et al.