StarCraft II Unplugged: Large Scale Offline Reinforcement Learning

Michaël Mathieu* Sherjil Ozair* Srivatsan Srinivasan Caglar Gulcehre

Shangtong Zhang Ray Jiang Tom Le Paine Konrad Żołna Richard Powell

Julian Schrittwieser David Choi Petko Georgiev Daniel Toyama Aja Huang

Roman Ring Igor Babuschkin Timo Ewalds Mahyar Bordbar Sarah Henderson

Sergio Gómez Colmenarejo Aäron van den Oord Wojciech Marian Czarnecki

Nando de Freitas Oriol Vinyals

DeepMind

Abstract

StarCraft II is one of the most challenging reinforcement learning (RL) environments; it is partially observable, stochastic, and multi-agent, and mastering StarCraft II requires strategic planning over long-time horizons with real-time low-level execution. It also has an active human competitive scene. StarCraft II is uniquely suited for advancing offline RL algorithms, both because of its challenging nature and because Blizzard has released a massive dataset of millions of StarCraft II games played by human players. This paper leverages that and establishes a benchmark, which we call *StarCraft II Unplugged*, that introduces unprecedented challenges for offline reinforcement learning. We define a dataset (a subset of Blizzard's release), tools standardising an API for ML methods, and an evaluation protocol. We also present baseline agents, including behaviour cloning, and offline variants of V-trace actor-critic and *MuZero*. We find that the variants of those algorithms with behaviour value estimation and single-step policy improvement work best and exceed 90% win rate against previously published AlphaStar behaviour cloning agents.

1 Introduction

Deep Reinforcement Learning (RL) is dominated by online RL algorithms, where agents must interact with the environment to explore and learn. This online RL paradigm achieved considerable success on Atari [34], Go [44], StarCraft II [51], DOTA 2 [7], and robotics [2]. However, the requirements of extensive interaction and exploration make these algorithms unsuitable and unsafe for many real-world applications. In contrast, in the offline setting [14, 15, 17], agents learn from a fixed dataset previously logged by humans or other agents. While the offline setting would enable RL in real-world applications, most offline RL benchmarks such as D4RL [14] and RL Unplugged [17]

*Indicates joint first authors.

have mostly focused on simple environments with data produced by RL agents. More challenging benchmarks are needed to make progress towards more ambitious real-world applications.

To rise to this challenge, we introduce *StarCraft II Unplugged*, an offline RL benchmark, which uses a dataset derived from replays of millions of humans playing the multi-player competitive game of StarCraft II. StarCraft II continues to be one of the most complex simulated environments, with partial observability, stochasticity, large action and observation spaces, delayed rewards, and multi-agent dynamics. Additionally, mastering the game requires strategic planning over long time horizons, and real-time low-level execution. Given these difficulties, breakthroughs in StarCraft II Unplugged will likely translate to many other offline RL settings, potentially transforming the field.

Additionally, unlike most RL domains, StarCraft II has an independent leaderboard of competitive human players over a wide range of skills. It also constitutes a rich and abundant source of data to train and evaluate offline RL agents.

With this paper, we release the most challenging large-scale offline RL benchmark to date, including the code of a canonical agent and data processing software. We note that removing the environment interactions from the training loop significantly lowers the compute demands of StarCraft II, making this environment accessible to far more researchers in the AI community.

The paper introduces several offline RL agents that can learn competitive policies purely from human replays. This has been made possible by innovations in architectures and algorithms. Extensive experimentation and evaluations, with proposed metrics, have taught us that many existing offline RL algorithms fail in this benchmark. However, they have also provided us with insights on how to design successful agents. Chiefly among these insights is the following recipe for constructing successful agents: first train a policy and value function network to estimate the behavior policy and value function. Then, during evaluation, either perform a single-step of policy improvement or improve the policy with the fixed and pretrained behavior value function. We believe sharing these insights will be valuable to anyone interested in offline RL, especially at large scale.

2 Background

The underlying system dynamics of StarCraft II can be described by a *Markov decision process* (MDP; [6]).[2] An MDP, $\mathcal{M} \overset{\text{def}}{=} (\mathcal{S}, \mathcal{A}, P, r, d)$, consists of finite sets of states $\mathcal{S}$ and actions $\mathcal{A}$, a transition distribution $P(s'|s,a), s, s' \in \mathcal{S}, a \in \mathcal{A}$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and an initial state distribution $d : \mathcal{S} \to [0,1]$. In the offline setting, the agent does not interact with the MDP but learns only from a dataset $\mathcal{D}$ containing sequences $(s_t, a_t, r_{t+1})_{t=0}^{N}$. The dataset $\mathcal{D}$ is assumed to have been generated by following an unknown *behaviour policy* μ, a distribution over actions conditioned on the state: $\mu(a|s)$.

The goal of offline RL is to find a policy π that maximizes the expected discounted return

$$v^{\pi}(s_0) \overset{\text{def}}{=} \mathbb{E}\left[\sum_{t\geq 0}(\prod_{i=1}^{t}\gamma_t) r(s_t, a_t) \middle| s_t \sim P(\cdot|s_{t-1}, a_{t-1}), a_t \sim \pi(\cdot|s_t)\right], \quad (1)$$

where $s_0 \sim d$ and γ_t is a discount factor at time step t. We refer to the policy π as the *target policy*. We use V^{μ} and V^{π} to denote the estimated value function for the behaviour and target policies μ and π, respectively.

3 StarCraft II Unplugged: An Offline RL Benchmark

This work builds on top of the StarCraft II Learning Environment and associated replay dataset [52], and the agents described in [51], by providing a few key components necessary for an offline RL benchmark:

- Evaluation metric. Built-in rule-based agents provide a measure of performance, but this metric can quickly saturate. In this work we present alternative metrics.
- Baseline agents. We provide a number of well tuned baseline agents.

[2]Strictly speaking, we have a partially observable MDP, but we simplify this for ease of presentation.

- Open source code. Building an agent that performs well on StarCraft II is a large engineering endeavor. We provide a well tuned behavior cloning agent which forms the backbone for all agents presented in this paper[3].

3.1 Challenging Properties for Offline RL

When learning from offline data, the performance of algorithms depends greatly on the availability of different state-action pairs in the data. We call this *coverage* — the more state-action pairs are absent, *i.e.* the lower the coverage, the more challenging the problem is. Here we highlight a few properties of StarCraft II that make it particularly challenging from a coverage perspective.

Action space. StarCraft II has a highly structured action space. As discussed in [51], the agent must select an action type, select a subset of its units to apply the action to, select a target for the action (either a map location or a visible unit), and decide when to observe and act next. Expanding, there are approximately 10^{26} possible actions per step. For comparison, Atari has only 18 possible actions per step. This makes it much more challenging to attain high state-action coverage for StarCraft II.

Stochastic environment. Stochastic environments may need many more trajectories to obtain high state-action coverage. The main source of stochasticity is an unknown opponent policy, which itself may be stochastic. To a small extent, the dynamics of the game engine are stochastic, for example some units have random movement, or random spawn points, and some commands can have a random delay. For comparison, in the Atari environment stochasticity arises only from sticky actions [33].

Partial Observability. StarCraft II is an imperfect information game. Players only have information about opponent units that are within the field of view of the player's own units. As a result, players need to scout, *i.e.* send their units around the map to gather information about the current state of the game, and may need it at a later point in the game. In comparison, a memory of the 3 most recent frames is usually considered sufficient for Atari.

Data source. For StarCraft II, we have access to a dataset of millions of human replays. These replays display a wide and diverse range of exploration and exploitation strategies. In contrast, the existing benchmarks [17, 1] have a bias toward datasets generated by RL agents. Replays are also labelled with the MMR of the player, a valuable data about their skill.

These properties create challenges for existing algorithms, and are likely to emphasize any differences between behaviour cloning, online RL algorithms, and offline RL algorithms.

3.2 Dataset

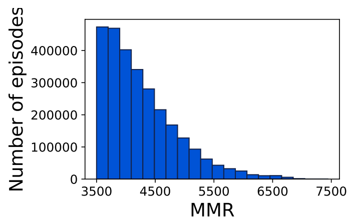

Figure 1: Histogram of player MMR from replays used for training.

About 20 million StarCraft II games are publicly available through the replay packs[4]. For technical reasons, we restrict the data to StarCraft II versions 4.8.2 to 4.9.2 which leaves nearly 5 million games. They come from the StarCraft II *ladder*, the official matchmaking mechanism. Each player is rated by their *MMR*, a ranking mechanism similar to *Elo* [12]. The MMR ranges from 0 to around 7000. Figure 1 shows the distribution of MMR among the episodes. In order to get quality training data, we only use games played by players with MMR greater than 3500, which corresponds to the top 22% of players. This leaves us with approximately 1.4 million games. Since we consider two-player games, from a machine learning point of view each game forms two episodes, one for each side, so there are 2.8 million episodes in the dataset. This represents a total of more than 30 years of game played.

Episodes can span many steps. StarCraft II is a real-time strategy game with a fast internal game clock. In order to make trajectories shorter, we only observe the steps when the player took an action. Consequently, each time step t of the dataset contain a pair (s_t, a_t), but also the *delay*, the number of internal game steps since the last action a_{t-1}. The delay is encoded as part of s_t. Each episode also

[3]Open source code will be released along with an ArXiv paper release.

[4]https://github.com/Blizzard/s2client-proto/tree/master/samples/replay-api

contains metadata, the most important ones being the outcome R which can be 1 for a victory, -1 for a defeat and 0 for a draw (draws are extremely rare in StarCraft II), as well as the MMR of each player. Since the games were played online using Blizzard's matchmaking system, both players have similar MMR in the vast majority of games.

4 Methodology

4.1 Training

StarCraft II Unplugged is a large-scale benchmark for data-driven approaches. Algorithms may only use data from the dataset described in Appendix A.2 to learn to play. Algorithms may not collect more data by interacting with the environment. However, we allow for online policy evaluation, *i.e.* policies can be run in the environment to measure how well they perform. This evaluation may be useful for hyperparameter tuning.

Unlike the original AlphaStar agents [51], the agents here are trained to play all three races of StarCraft 2. This is more challenging, as agents are typically better when they are trained on a single race. They are also trained to play on all 10 maps available in the dataset.

We typically train the agent on *rollouts*, *i.e.* sequences of K consecutive timesteps $(s_0, a_0, r_1, \ldots s_{K-1}, a_{K-1}, r_K)$, assembled in a minibatch of M independent rollouts. Unless specified otherwise, the minibatches are independent from each other, such that two consecutive minibatches are not correlated. The rollout length K differs between methods, as explained in Section 5.

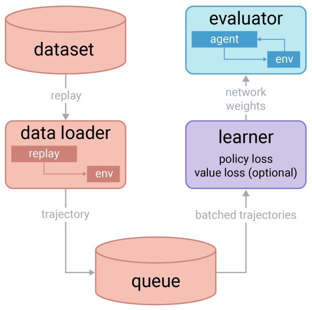

Figure 2: Training procedure.

4.2 Reference Agents

In this paper, we present six agents trained on this dataset as reference agents. We propose to use them for evaluation of future works, by measuring the win-rate of the new agent against them. They fall into three categories: *behaviour cloning*, *actor-critic* and *Monte-Carlo tree search* (MCTS). Details about these agents, their implementations, and ablations are given in Section 5. Their performances can be found in Table 1.

Behaviour Cloning (BC). Our behaviour cloning baseline [37, 45] is an imitation learning method that trains an agent to predict what would be the next action in the dataset given the observation. As such it uses a supervised loss which is the cross-entropy for StarCraft II. A BC agent learns an estimate of the behavior policy $\hat{\mu}$ using only states and actions from the dataset.

Fine-Tuned Behaviour Cloning (FT-BC). The FT-BC agent restarts training from a pretrained BC agent, and fine-tunes the pretrained parameters on the winning games with the highest MMR only, similar to the fine-tuning used in [51].

Offline Actor-Critic (OAC). The OAC agent adapts the V-trace algorithm from Impala [13] to offline RL in the i.i.d. setting. To compute importance sampling ratios, it uses the behaviour policy $\hat{\mu}$ estimated by the BC agent. The actor-critic method starts from $\hat{\mu}$ and learns an improved policy π. Unlike the online version of actor-critic, this agent uses V^{μ}, the value function of the behaviour policy, as the critic (instead of V^{π}).

Emphatic Offline Actor-Critic (E-OAC). The emphatic method is a theoretical tool to improve stability in off-policy learning [47, 25]. This agent applies *N-steps Emphatic Traces* (NETD) [26] to the Offline Actor-Critic agent, however using consecutive data sequences.

MuZero Supervised MCTS (MZS-MCTS). We apply *MuZero Unplugged* [40] for StarCraft II but with a key difference. While *MuZero Unplugged* uses *Monte Carlo tree search* (MCTS; [9]) at training time, *MuZero Supervised* is trained with supervised learning and thus does estimation of

Table 1: Evaluation of the 6 reference agents with the proposed metrics. In the bottom part of the table, we report the performance of the original AlphaStar agents[51]. Note that all the AlphaStar agents use models specific to each home race; see Section 4.1. All agents use models which train on and play all races, evaluated with softmax temperature 0.8.

Agent	Robustness	Elo	vs `very_hard`
MZS-MCTS	50%	1578	95%
E-OAC	43%	1563	97%
OAC	41%	1548	97%
FT-BC	37%	1485	95%
MZS-P	30%	1425	92%
BC	25%	1380	88%
`very_hard`	3%	1000	50%
AlphaStar Final	100%	2968	100%
AlphaStar Supervised	44%	1545	94%
AlphaStar Supervised no FT	17%	1280	82%
AlphaStar Supervised no FT All Races	8%	1171	75%

the behvaioural policy $\hat{\mu}$ and behavioural value V^{μ}. At inference time, the policy, value, and latent model is used to perform MCTS for policy improvement and action selection.

MuZero Supervised Policy (MZS-P). For completeness, we also evaluate the performance of the same *MuZero Supervised* agent, but without MCTS at inference time, and instead directly using the policy network. This is similar to a behaviour cloning agent, but can often perform better because of the additional regularisation effects from training a value function and the latent model which are not used during inference (as also noted in [18, 40, 22]).

4.3 Evaluation and Metrics

Numerous metrics can be used to evaluate the agents. On one hand, the easiest to compute (and least informative) is simply to look at the value of the loss functions. On the other hand, perhaps the most informative (and most difficult to compute) metric would be to evaluate the agent against a wide panel of human players, including professional players. In this paper, we propose a compromise between these two extremes. We evaluate our agents by playing repeated games against a fixed selection of 7 opponents: the `very_hard` built-in bot[5], as well as the 6 reference agents presented above.

During training, we only evaluate the agents against the `very_hard` bot, since it is significantly less expensive, and we mostly use that as a validation metric, to tune hyper-parameters and discard non-promising runs.

Fully trained agents are evaluated against the full set of opponents presented above. We combine these win-rates into two aggregated metrics while uniformly sampling the races of any pair of these agents: *robustness*, computed as one minus the minimum win-rate over the set of reference agents, and *Elo rating* [12]. See details of the metrics computation in Appendix A.5. These metrics are presented in Table 1.

5 Deep Dive into Reference Agents

The performance of our six reference agents is shown in Table 1. In the rest of this section, we provide details of these agents, as well as ablation studies.

Results in Table 1 are evaluated on agents with a *softmax temperature* of 0.8. This means that at evaluation time, the softmax logits are multiplied by 0.8 before being converted to probabilities. This makes the agents more deterministic, but also stronger.

[5]The `very_hard` bot is not the strongest built-in bot in StarCraft II, but it is the strongest whose strength does not come from unfair advantages which break the game rules.

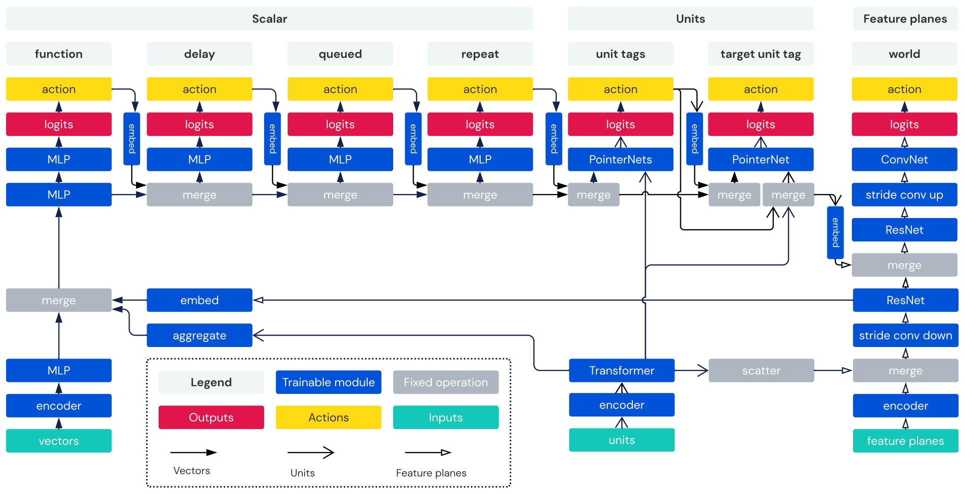

Figure 3: Illustration of the architecture that we used for our reference agents. Different types of data are denoted by different types of data (vectors, units or feature planes).

5.1 Architecture

We used the same architecture for all the agents presented in this paper. It is derived from the model used in [51], with some improvements. Inputs and outputs of the StarCraft II API are structured around three types of data: *vectors*, *units* and *feature planes* (more details can be found in Sections A.1 and A.3). We structure the architecture around them, as shown on Figure 3. Unlike the model in [51], we do not use an LSTM module or any form of memory, which is discussed in Section 5.2. More details on the architecture can be found in Appendix A.4.

5.2 Behaviour Cloning

The BC and FT-BC agents are trained using a supervised loss, where given a trajectory $(s_0, a_0, ..., s_{t-1}, a_{t-1}, s_t)$, the model predicts a_t. This is similar to training a language model. The procedure is detailed in Algorithm 1 in the Appendix. It is the same procedure that was used by the AlphaStar Supervised agent to create the subsequent Grandmaster level AlphaStar Final agent [51], with the difference that we train a single agent on all three races of StarCraft, instead of three separate agents. This makes the problem more challenging. The input of the agent is the observation s_t, the past action a_{t-1}, the MMR of the player and the previous LSTM state (for the model using a LSTM, see below). During inference, we can then control the quality of the game played by the agent by changing the MMR input. In practice, we set the MMR to the highest value to ensure the agent plays its best. We used the Adam optimizer [32] with a cosine learning rate schedule [30]. Implementation details can be found in Appendix B.1.1.

Memory. The AlphaStar agent of [51] uses a LSTM module. Surprisingly, our experiments show that removing the LSTM entirely result in a better performing model, although the final values of the losses are higher. The LSTM-based agent reaches a win-rate of 70% against the `very_hard` bot (vs. 84% for the memory-less agent). Therefore all agents presented in this paper are memory-less.

Data filtering. Since behaviour cloning just mimics the training data, a higher quality data can lead to better performance. We ran experiments to filter the data to include only top-tier MMR and/or only winning games. Unfortunately, since filtering the data also decreases the number of episodes, the overall performance of the agent is deteriorated. The number of episodes after filtering and the final performance are shown in Table 3.

Minibatch size and rollout length. The minibatch size M and rollout size K influence the final behaviour cloning performance, as shown in Table 2. We found that larger total amount of data per learner step ($M \times K$) leads to better final performance. We use $K = 1$ and $M = 32,768$ for the BC and FT-BC agents.

Fine-tuning. The BC agent achieves better results when a secondary training phase is added. We call the resulting method FT-BC. We first train the agent on all the data (with MMR>3500), then, after reducing the learning rate, we further train the agent on the very best data (MMR>6200) and only winning games. This scheme leads to the best behaviour cloning results, as shown in Table 3. Implementation details of the FT-BC agent can be found in Appendix B.1.1.

5.3 Offline Actor-Critic

Actor-critic [55, 5] algorithms learn a policy π and a value function V^π. In off-policy settings, where π differs from the behaviour policy μ, it computes Importance Sampling (IS) ratios $\rho_t(a_t|s_t) \overset{\text{def}}{=} \pi(a_t|s_t)/\mu(a_t|s_t)$ where (s_t, a_t) are sampled from the behaviour policy, *i.e.* come from the data. We use V-trace [13] to reduce variance.

Our setup differs from the classical off-policy actor-critic. Indeed, we do not have access to the behaviour policy μ used by the players, we can only observe their actions. Instead, we use the BC agent to provide an estimate of the behaviour policy $\hat{\mu}$ and use the estimated $\hat{\rho} \overset{\text{def}}{=} \pi/\hat{\mu}$ in place of the ground truth IS ratios. The BC agent is also used as the starting point for OAC (*i.e.* it is used to initialize the weights).

Learning V^μ. We train our network to produce the estimated behaviour value V^μ using a secondary output, *via* a MLP conditioned on the features. It is learned using a *Mean-Squared Error* (MSE) loss: $||V^\mu(s_t) - R||_2^2$ where s_t comes from a trajectory which leads to the game outcome R (-1 for a loss, 1 for a win). We have tried both training it at the same time as behaviour cloning, or afterwards on the pre-trained model, and have observed no significant difference in performance. The value function trained for the OAC and E-OAC models is trained afterwards. It reaches 72% accuracy, which is computed as the fraction of steps where the sign of V^μ is the same as R.

Divergence of V^π. The standard actor-critic method learns π, an improvement of μ, as well as V^π, the corresponding value function. This is usually done with temporal differences (TD) learning. We observe that when doing so, using the value function V^π leads to divergence during training, as shown on Figure 5 in the Appendix. We explored several ways to mitigate this problem:

- Early stopping. The policy π improves at first, before deteriorating. Therefore we could stop training early to obtain an improved policy π (shown in Figure 5). We do not use this method since it requires using the environment to detect when to stop.
- Alternatively, we can use V^μ as a critic, and keep it fixed, instead of estimating V^π iteratively. This is similar to the behavior value estimation of [16]. It is implemented for the OAC agent.
- The Emphatic Offline Actor-Critic (E-OAC) agent uses emphatic traces to weight each value and policy update in OAC. *N-step Emphatic Traces* (NETD) [26] avoids deadly triads in off-policy learning with linear value function approximation. Using estimated importance sampling ratios $\hat{\rho}$, the NETD trace F can be computed as in [26].

Training parameters. Since actor-critic uses temporal differences, the rollout length K has to be larger than 1, we use $K = 64$. Because of the way emphatic traces are computed, the E-OAC

Table 2: Comparison of the performance of behaviour cloning with different minibatch sizes and rollout lengths. The total number of observations seen by the agent is the same (10^{10}) for all agents.

Minibatch size M	Rollout length K	$M \times K$	Win-rate vs. `very_hard`
8,192	1	8,192	70%
16,384	1	16,384	79%
256	64	16,384	79%
32,768	1	32,768	84%

Table 3: Performance agents trained with different MMR filtering schemes against the `very_hard` bot, along with the number of episodes available. Training on higher quality data leads to worse performance, since the number of episodes is decreased. Training on the full dataset, followed by a fine-tuning phase on higher quality, is best.

Main training			Fine-tuning			
MMR	filter	#episodes	MMR	filter	#episodes	final performance
>3500	win+loss	2,776,466				84%
>6000	win+loss	64,894				65%
>6000	win	32,447				51%
>3500	win+loss	2,776,466	>6200	win	21,836	**89%**

agent requires learning from consecutive minibatches[6]. Details can be found in Appendices B.1.2 and B.1.3. As explained in Appendix A.3, we only apply policy improvement to the `function` and `delay` arguments of the action for simplicity.

5.4 MuZero

MuZero Unplugged [40] is a recently introduced offline RL approach which achieves state-of-the-art performance on a variety of offline RL benchmarks [17, 11]. Inspired by recent extensions to handle stochastic environments [35], we evaluate *MuZero* as an offline RL approach for StarCraft II.

We use the approach introduced in *Sampled MuZero* [24] to handle the large action space of StarCraft II. *Sampled MuZero* samples multiple actions from the policy and then restricts the search to only the sampled actions. This allows us to scale to the large action space of StarCraft II. We provide further implementation details in Appendix B.1.4.

MuZero Supervised. Our first key result is that MCTS is a strong policy improvement for StarCraft II when training the model with supervised learning. Throughout the training, and thus even for relatively weak policies and value functions, we found that MCTS almost always performs better than the neural network policy when training the policy and value function via supervised learning.

MuZero Unplugged. Our preliminary experiments on using the full *MuZero Unplugged* algorithm, *i.e.* training with MCTS targets, were not successful. We found that the policy would collapse quickly to a few actions with high (over-)estimated value. While MCTS at inference time improves performance, using MCTS at training time leads to a collapsed policy. To investigate this further, we evaluate the performance of repeated applications of MCTS policy improvement on the behavioural policy $\hat{\mu}$ and value V^μ. We do this by training a new MuZero model using MCTS actions of a behavioural policy, i.e. $\hat{\nu} = MCTS(\hat{\mu}, V^\mu)$. We found that the MCTS performance of this policy $MCTS(\hat{\nu}, V^\mu)$ is worse than the performance of $\hat{\nu}$ or $MCTS(\hat{\mu}, V^\mu)$. Thus, repeated applications of MCTS do not continue to improve the policy. We believe this is likely due to MCTS policy distribution generating out of distribution action samples with over-estimated value estimates.

6 Discussion

The behaviour cloning agent is the base of all agents in this work. It is conditioned on the MMR during training, which helps filter out lower quality trajectories. Nevertheless, the behavior cloning agent is still fundamentally limited to estimating the behavior policy, ignorant about rewards. As a result, the policy it learns is a smoothed version of all the policies that generated the dataset. We experimented with training the BC agent conditioning on rewards, and evaluated it conditioning on the winning game outcomes, without observing any gain in performance. In contrast, offline RL methods use rewards to improve learned policies in different ways. MZS-P uses rewards in auxiliary losses, FT-BC uses rewards when selecting episodes for finetuning, and OAC, E-OAC, and

[6]Such that the first element of each rollout of a minibatch are adjacent to the last element of each rollouts of the previous minibatch

MZS-MCTS use rewards to perform reinforcement learning. Final results show that agents that use rewards *via* reinforcement learning perform the best.

We have observed that algorithms originally designed for online learning — even with off-policy corrections — do not work well when applied directly to the full offline RL setting. We attribute this in part to the problem of Deadly Triads [49, 46, 50]. However, many recent works have found these algorithms can be made more effective simply by making modifications that ensure the learned policy stays close to the behavior policy μ, that the learned value stays close to V^{μ}, or both. Our results with Actor-Critic and MuZero are in accordance with these findings.

The reference agents are the methods we attempted that resulted improving the policy. However, we have tried several other methods without success, including Advantage-Weighted Regression (AWR) [36], PPO [42], and state-action value based methods like SARSA [38], CRR [54] and R-BVE [16]. To this day, we do not know if there is a fundamental reason for this, or if we simply failed to find the necessary changes these methods would require to perform well on this dataset.

7 Related Work

Online RL has been very impactful for building agents to play computer games. RL agents can outperform professional human players in many games such as StarCraft II [51], DOTA [7] or Atari [34, 4]. Similar levels of progression have been observed on board games, including chess and Go [43, 44]. Although offline RL approaches have shown promising results on Atari recently [41], it has not been previously explored how they would perform on complex partially observable games using data derived from human experts.

RL Unplugged [17] introduces a suite of benchmarks for Offline RL with a diverse set of task domains with a unified API and evaluation protocol. D4RL [14] is an offline RL benchmark suite focusing only on mixed data sources. However, both RL Unplugged and D4RL lack high-dimensional, partially observable tasks. This paper fills that gap by introducing a benchmark for StarCraft II.

Offline RL has become an active research area, as it enables us to leverage fixed datasets to learn policies to deploy in the real-world. Offline RL methods include 1) policy-constraint approaches that regularize the learned policy to stay close to the behaviour policy [54, 15], 2) value-based approaches that encourage more conservative value estimates, either through a pessimistic regularization or uncertainty [29, 16], 3) model-based approaches [56, 27, 41], and 4) adaptations of standard off-policy RL methods such as DQN [1] or D4PG [54]. Recently methods using only one-step of policy improvement has been proven to be very effective on offline reinforcement learning [16, 8].

8 Conclusions

Offline RL has enabled the deployment of RL ideas to the real-world. Academic interest in this area has grown and several benchmarks have been proposed, including RL-Unplugged [17], D4RL [14], and RWRL [11]. However, because of the relatively small-scale and synthetic nature of these benchmarks, they don't capture the challenges of real-world offline RL.

In this paper, we introduced StarCraft II Unplugged, a benchmark to evaluate agents which play StarCraft II by learning only from *offline* data. This data is comprised of over a million games games mostly played by amateur human StarCraft II players on Blizzard's Battle.Net[7] Thus, the benchmark more accurately captures the challenges of offline RL where an agent must learn from logged data, generated by a diverse group of weak experts, and where the data doesn't exhaust the full state and action space of the environment.

We showed that offline RL algorithms can exceed 90% win-rate against the previously published AlphaStar Supervised agent (trained using behaviour cloning). However, the gap between online and offline methods still exists and we hope the benchmark will serve as a testbed to advance the state of art in offline RL algorithms.

[7] https://en.wikipedia.org/wiki/Battle.net

Acknowledgments and Disclosure of Funding

We would like to thank Scott Reed and David Silver for reviewing the manuscript, the Alphastar team [51] for sharing their knowledge and experience about the game. We would like to thank the authors of *MuZero Unplugged* [40] and *Sampled MuZero* [24] for advising on the development of the *MuZero Supervised* agent. We also thank the wider DeepMind research, engineering, and environment teams for the technical and intellectual infrastructure upon which this work is built. We are grateful to the developers of tools and frameworks like Jax [3], Haiku [21] and Acme [23] that enabled this research.

References

[1] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[2] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[3] Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Claudio Fantacci, Jonathan Godwin, Chris Jones, Tom Hennigan, Matteo Hessel, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Lena Martens, Vladimir Mikulik, Tamara Norman, John Quan, George Papamakarios, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Wojciech Stokowiec, and Fabio Viola. The DeepMind JAX Ecosystem, 2020.

[4] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pages 507–517. PMLR, 2020.

[5] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[6] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 1957.

[7] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[8] David Brandfonbrener, William F Whitney, Rajesh Ranganath, and Joan Bruna. Offline rl without off-policy evaluation. *arXiv preprint arXiv:2106.08909*, 2021.

[9] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[10] Trevor Davis, Neil Burch, and Michael Bowling. Using response functions to measure strategy strength. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[11] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[12] Arpad E Elo. *The rating of chessplayers, past and present*. Arco Pub., 1978.

[13] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018.

[14] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[15] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[16] Caglar Gulcehre, Sergio Gómez Colmenarejo, Ziyu Wang, Jakub Sygnowski, Thomas Paine, Konrad Zolna, Yutian Chen, Matthew Hoffman, Razvan Pascanu, and Nando de Freitas. Regularized behavior value estimation. *arXiv preprint arXiv:2103.09575*, 2021.

[17] Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gomez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, et al. Rl unplugged: Benchmarks for offline reinforcement learning. *arXiv e-prints*, pages arXiv–2006, 2020.

[18] Jessica B Hamrick, Abram L Friesen, Feryal Behbahani, Arthur Guez, Fabio Viola, Sims Witherspoon, Thomas Anthony, Lars Buesing, Petar Veličković, and Théophane Weber. On the role of planning in model-based deep reinforcement learning. *arXiv preprint arXiv:2011.04021*, 2020.

[19] Lei Han, Jiechao Xiong, Peng Sun, Xinghai Sun, Meng Fang, Qingwei Guo, Qiaobo Chen, Tengfei Shi, Hongsheng Yu, Xipeng Wu, and Zhengyou Zhang. Tstarbot-x: An open-sourced and comprehensive study for efficient league training in starcraft ii full game, 2021.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[21] Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020.

[22] Matteo Hessel, Ivo Danihelka, Fabio Viola, Arthur Guez, Simon Schmitt, Laurent Sifre, Theophane Weber, David Silver, and Hado van Hasselt. Muesli: Combining improvements in policy optimization. *arXiv preprint arXiv:2104.06159*, 2021.

[23] Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

[24] Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mohammadamin Barekatain, Simon Schmitt, and David Silver. Learning and planning in complex action spaces. *arXiv preprint arXiv:2104.06303*, 2021.

[25] Ehsan Imani, Eric Graves, and Martha White. An off-policy policy gradient theorem using emphatic weightings. *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS 2018)*, 2018.

[26] Ray Jiang, Tom Zahavy, Adam White, Zhongwen Xu, Matteo Hessel, Charles Blundell, and Hado van Hasselt. Emphatic algorithms for deep reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2021.

[27] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[30] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[31] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[32] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[33] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[34] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[35] Sherjil Ozair, Yazhe Li, Ali Razavi, Ioannis Antonoglou, Aäron van den Oord, and Oriol Vinyals. Vector quantized models for planning. *arXiv preprint arXiv:2106.04615*, 2021.

[36] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *CoRR*, abs/1910.00177, 2019.

[37] Dean A Pomerleau. ALVINN: An autonomous land vehicle in a neural network. pages 305–313, 1989.

[38] G. Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

[39] Mikayel Samvelyan, Tabish Rashid, Christian Schröder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob N. Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *CoRR*, abs/1902.04043, 2019.

[40] Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model. *arXiv preprint arXiv:2104.06294*, 2021.

[41] Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model. *arXiv preprint arXiv:2104.06294*, 2021.

[42] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. abs/1707.06347, 2017.

[43] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[44] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[45] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. pages 3104–3112, 2014.

[46] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 2018.

[47] Richard S Sutton, A Rupam Mahmood, and Martha White. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, 17(1):2603–2631, 2016.

[48] Open-Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

[49] John N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[50] Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *CoRR*, abs/1812.02648, 2018.

[51] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[52] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[53] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks, 2017.

[54] Ziyu Wang, Alexander Novikov, Konrad Zolna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *arXiv preprint arXiv:2006.15134*, 2020.

[55] Ian H. Witten. An adaptive optimal controller for discrete-time markov environments. *Information and Control*, 34:286–295, 1977.

[56] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.