Averaged-DQN: Variance Reduction and Stabilization for Deep Reinforcement Learning

Oron Anschel[1] **Nir Baram**[1] **Nahum Shimkin**[1]

Abstract

Instability and variability of Deep Reinforcement Learning (DRL) algorithms tend to adversely affect their performance. Averaged-DQN is a simple extension to the DQN algorithm, based on averaging previously learned Q-values estimates, which leads to a more stable training procedure and improved performance by reducing approximation error variance in the target values. To understand the effect of the algorithm, we examine the source of value function estimation errors and provide an analytical comparison within a simplified model. We further present experiments on the Arcade Learning Environment benchmark that demonstrate significantly improved stability and performance due to the proposed extension.

1. Introduction

In Reinforcement Learning (RL) an agent seeks an optimal policy for a sequential decision making problem (Sutton & Barto, 1998). It does so by learning which action is optimal for each environment state. Over the course of time, many algorithms have been introduced for solving RL problems including Q-learning (Watkins & Dayan, 1992), SARSA (Rummery & Niranjan, 1994; Sutton & Barto, 1998), and policy gradient methods (Sutton et al., 1999). These methods are often analyzed in the setup of linear function approximation, where convergence is guaranteed under mild assumptions (Tsitsiklis, 1994; Jaakkola et al., 1994; Tsitsiklis & Van Roy, 1997; Even-Dar & Mansour, 2003). In practice, real-world problems usually involve high-dimensional inputs forcing linear function approximation methods to rely upon hand engineered features for problem-specific state representation. These problem-specific features diminish the agent flexibility, and so the need of an expressive and flexible non-linear function approximation emerges. Except for few successful attempts (e.g., TD-gammon, Tesauro (1995)), the combination of non-linear function approximation and RL was considered unstable and was shown to diverge even in simple domains (Boyan & Moore, 1995).

The recent Deep Q-Network (DQN) algorithm (Mnih et al., 2013), was the first to successfully combine a powerful non-linear function approximation technique known as Deep Neural Network (DNN) (LeCun et al., 1998; Krizhevsky et al., 2012) together with the Q-learning algorithm. DQN presented a remarkably flexible and stable algorithm, showing success in the majority of games within the Arcade Learning Environment (ALE) (Bellemare et al., 2013). DQN increased the training stability by breaking the RL problem into sequential supervised learning tasks. To do so, DQN introduces the concept of a target network and uses an Experience Replay buffer (ER) (Lin, 1993).

Following the DQN work, additional modifications and extensions to the basic algorithm further increased training stability. Schaul et al. (2015) suggested sophisticated ER sampling strategy. Several works extended standard RL exploration techniques to deal with high-dimensional input (Bellemare et al., 2016; Tang et al., 2016; Osband et al., 2016). Mnih et al. (2016) showed that sampling from ER could be replaced with asynchronous updates from parallel environments (which enables the use of on-policy methods). Wang et al. (2015) suggested a network architecture base on the advantage function decomposition (Baird III, 1993).

In this work we address issues that arise from the combination of Q-learning and function approximation. Thrun & Schwartz (1993) were first to investigate one of these issues which they have termed as the *overestimation phenomena*. The max operator in Q-learning can lead to overestimation of state-action values in the presence of noise. Van Hasselt et al. (2015) suggest the Double-DQN that uses the Double Q-learning estimator (Van Hasselt, 2010) method as a solution to the problem. Additionally, Van Hasselt et al. (2015) showed that Q-learning overestimation do occur in practice

[1]Department of Electrical Engineering, Haifa 32000, Israel. Correspondence to: Oron Anschel <oronanschel@campus.technion.ac.il>, Nir Baram <nirb@campus.technion.ac.il>, Nahum Shimkin <shimkin@ee.technion.ac.il>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

(at least in the ALE).

This work suggests a different solution to the overestimation phenomena, named Averaged-DQN (Section 3), based on averaging previously learned Q-values estimates. The averaging reduces the target approximation error variance (Sections 4 and 5) which leads to stability and improved results. Additionally, we provide experimental results on selected games of the Arcade Learning Environment.

We summarize the main contributions of this paper as follows:

- A novel extension to the DQN algorithm which stabilizes training, and improves the attained performance, by averaging over previously learned Q-values.
- Variance analysis that explains some of the DQN problems, and how the proposed extension addresses them.
- Experiments with several ALE games demonstrating the favorable effect of the proposed scheme.

2. Background

In this section we elaborate on relevant RL background, and specifically on the Q-learning algorithm.

2.1. Reinforcement Learning

We consider the usual RL learning framework (Sutton & Barto, 1998). An agent is faced with a sequential decision making problem, where interaction with the environment takes place at discrete time steps ($t = 0, 1, \ldots$). At time t the agent observes state $s_t \in S$, selects an action $a_t \in A$, which results in a scalar reward $r_t \in \mathbb{R}$, and a transition to a next state $s_{t+1} \in S$. We consider infinite horizon problems with a discounted cumulative reward objective $R_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$, where $\gamma \in [0, 1]$ is the discount factor. The goal of the agent is to find an optimal policy $\pi : S \to A$ that maximize its expected discounted cumulative reward.

Value-based methods for solving RL problems encode policies through the use of value functions, which denote the expected discounted cumulative reward from a given state s, following a policy π. Specifically we are interested in state-action value functions:

$$Q^{\pi}(s, a) = \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \;|s_0 = s, a_0 = a\right].$$

The optimal value function is denoted as $Q^*(s, a) = \max_\pi Q^\pi(s, a)$, and an optimal policy π^* can be easily derived by $\pi^*(s) \in \text{argmax}_a Q^*(s, a)$.

2.2. Q-learning

One of the most popular RL algorithms is the Q-learning algorithm (Watkins & Dayan, 1992). This algorithm is based on a simple value iteration update (Bellman, 1957), directly estimating the optimal value function Q^*. Tabular Q-learning assumes a table that contains old action-value function estimates and preform updates using the following update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)), \tag{1}$$

where s' is the resulting state after applying action a in the state s, r is the immediate reward observed for action a at state s, γ is the discount factor, and α is a learning rate.

When the number of states is large, maintaining a look-up table with all possible state-action pairs values in memory is impractical. A common solution to this issue is to use function approximation parametrized by θ, such that $Q(s, a) \approx Q(s, a; \theta)$.

2.3. Deep Q Networks (DQN)

We present in Algorithm 1 a slightly different formulation of the DQN algorithm (Mnih et al., 2013). In iteration i the DQN algorithm solves a supervised learning problem to approximate the action-value function $Q(s, a; \theta)$ (line 6). This is an extension of implementing (1) in its function approximation form (Riedmiller, 2005).

Algorithm 1 DQN

```
1: Initialize $Q(s, a; \theta)$ with random weights $\theta_0$
2: Initialize Experience Replay (ER) buffer $\mathcal{B}$
3: Initialize exploration procedure *Explore(·)*
4: **for** $i = 1, 2, \ldots, N$ **do**
5:     $y^i_{s,a} = \mathbb{E}_{\mathcal{B}}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | \, s, a\right]$
6:     $\theta_i \approx \text{argmin}_\theta \, \mathbb{E}_{\mathcal{B}}\left[(y^i_{s,a} - Q(s, a; \theta))^2\right]$
7:     *Explore(·)*, update $\mathcal{B}$
8: **end for**
**output** $Q^{\text{DQN}}(s, a; \theta_N)$
```

The target values $y^i_{s,a}$ (line 5) are constructed using a designated *target-network* $Q(s, a; \theta_{i-1})$ (using the previous iteration parameters θ_{i-1}), where the expectation ($\mathbb{E}_{\mathcal{B}}$) is taken w.r.t. the sample distribution of experience transitions in the ER buffer $(s, a, r, s') \sim \mathcal{B}$. The DQN loss (line 6) is minimized using a Stochastic Gradient Descent (SGD) variant, sampling mini-batches from the ER buffer. Additionally, DQN requires an exploration procedure (which we denote as *Explore(·)*) to interact with the environment (e.g., an ϵ-greedy exploration procedure). The number of new experience transitions (s, a, r, s') added by exploration to the ER buffer in each iteration is small, relatively to the size of the ER buffer. Thereby, θ_{i-1} can be used as a good initialization for θ in iteration i.

Note that in the original implementation (Mnih et al., 2013; 2015), transitions are added to the ER buffer simultaneously with the minimization of the DQN loss (line 6). Using the hyperparameters employed by Mnih et al. (2013; 2015) (detailed for completeness in Appendix E), 1% of the experience transitions in ER buffer are replaced between target network parameter updates, and 8% are sampled for minimization.

3. Averaged DQN

The Averaged-DQN algorithm (Algorithm 2) is an extension of the DQN algorithm. Averaged-DQN uses the K previously learned Q-values estimates to produce the current action-value estimate (line 5). The Averaged-DQN algorithm stabilizes the training process (see Figure 1), by reducing the variance of *target approximation error* as we elaborate in Section 5. The computational effort compared to DQN is, K-fold more forward passes through a Q-network while minimizing the DQN loss (line 7). The number of back-propagation updates remains the same as in DQN. Computational cost experiments are provided in Appedix D. The output of the algorithm is the average over the last K previously learned Q-networks.

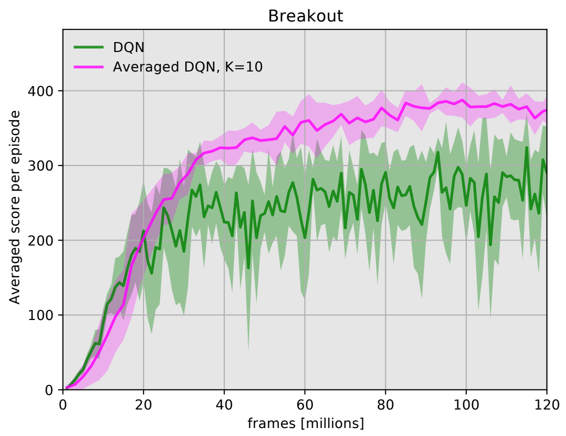

Figure 1. DQN and Averaged-DQN performance in the Atari game of BREAKOUT. The bold lines are averages over seven independent learning trials. Every 1M frames, a performance test using ϵ-greedy policy with $\epsilon = 0.05$ for 500000 frames was conducted. The shaded area presents one standard deviation. For both DQN and Averaged-DQN the hyperparameters used were taken from Mnih et al. (2015).

In Figures 1 and 2 we can see the performance of Averaged-DQN compared to DQN (and Double-DQN), further experimental results are given in Section 6.

Algorithm 2 Averaged DQN

```
1: Initialize $Q(s,a;\theta)$ with random weights $\theta_0$
2: Initialize Experience Replay (ER) buffer $\mathcal{B}$
3: Initialize exploration procedure *Explore*$(\cdot)$
4: for $i = 1, 2, \dots, N$ do
5:     $Q^A_{i-1}(s,a) = \frac{1}{K}\sum_{k=1}^{K} Q(s,a;\theta_{i-k})$
6:     $y^i_{s,a} = \mathbb{E}_{\mathcal{B}}\left[r + \gamma \max_{a'} Q^A_{i-1}(s',a') \middle| s,a\right]$
7:     $\theta_i \approx \mathrm{argmin}_\theta\, \mathbb{E}_{\mathcal{B}}\left[(y^i_{s,a} - Q(s,a;\theta))^2\right]$
8:     *Explore*$(\cdot)$, update $\mathcal{B}$
9: end for
output $Q^A_N(s,a) = \frac{1}{K}\sum_{k=0}^{K-1} Q(s,a;\theta_{N-k})$
```

We note that recently-learned state-action value estimates are likely to be better than older ones, therefore we have also considered a recency-weighted average. In practice, a weighted average scheme did not improve performance and therefore is not presented here.

4. Overestimation and Approximation Errors

Next, we discuss the various types of errors that arise due to the combination of Q-learning and function approximation in the DQN algorithm, and their effect on training stability. We refer to DQN's performance in the BREAKOUT game in Figure 1. The source of the learning curve variance in DQN's performance is an occasional sudden drop in the average score that is usually recovered in the next evaluation phase (for another illustration of the variance source see Appendix A). Another phenomenon can be observed in Figure 2, where DQN initially reaches a steady state (after 20 million frames), followed by a gradual deterioration in performance.

For the rest of this section, we list the above mentioned errors, and discuss our hypothesis as to the relations between each error and the instability phenomena depicted in Figures 1 and 2.

We follow terminology from Thrun & Schwartz (1993), and define some additional relevant quantities. Letting $Q(s,a;\theta_i)$ be the value function of DQN at iteration i, we denote $\Delta_i = Q(s,a;\theta_i) - Q^*(s,a)$ and decompose it as follows:

$$\begin{aligned}\Delta_i &= Q(s,a;\theta_i) - Q^*(s,a)\\ &= \underbrace{Q(s,a;\theta_i) - y^i_{s,a}}_{\substack{\textit{Target Approximation}\\ \textit{Error}}} + \underbrace{y^i_{s,a} - \hat{y}^i_{s,a}}_{\substack{\textit{Overestimation}\\ \textit{Error}}} + \underbrace{\hat{y}^i_{s,a} - Q^*(s,a)}_{\substack{\textit{Optimality}\\ \textit{Difference}}}.\end{aligned}$$

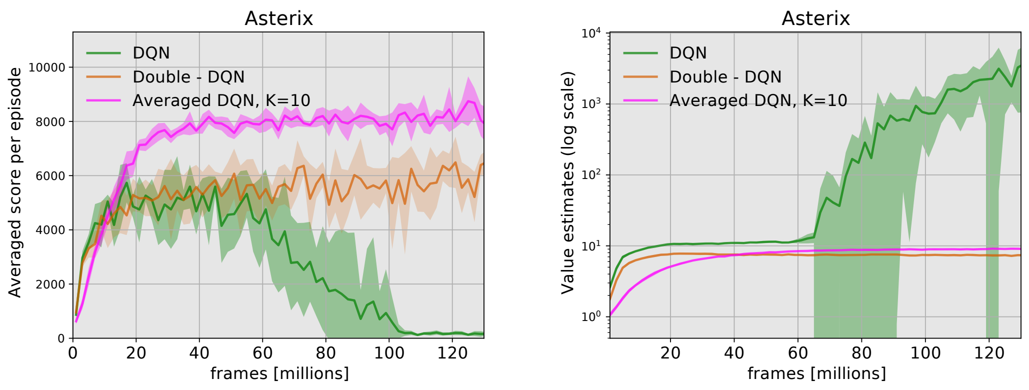

Figure 2. DQN, Double-DQN, and Averaged-DQN performance (left), and average value estimates (right) in the Atari game of ASTERIX. The bold lines are averages over seven independent learning trials. The shaded area presents one standard deviation. Every 2M frames, a performance test using ϵ-greedy policy with $\epsilon = 0.05$ for 500000 frames was conducted. The hyperparameters used were taken from Mnih et al. (2015).

Here $y^i_{s,a}$ is the *DQN target*, and $\hat{y}^i_{s,a}$ is the *true target*:

$$\begin{aligned} y^i_{s,a} &= \mathbb{E}_{\mathcal{B}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) |\, s,a\right], \\ \hat{y}^i_{s,a} &= \mathbb{E}_{\mathcal{B}}\left[r + \gamma \max_{a'} (y^{i-1}_{s',a'}) |\, s,a\right]. \end{aligned}$$

Let us denote by $Z^i_{s,a}$ the target approximation error, and by $R^i_{s,a}$ the overestimation error, namely

$$\begin{aligned} Z^i_{s,a} &= Q(s,a;\theta_i) - y^i_{s,a}, \\ R^i_{s,a} &= y^i_{s,a} - \hat{y}^i_{s,a}. \end{aligned}$$

The optimality difference can be seen as the error of a standard tabular Q-learning, here we address the other errors. We next discuss each error in turn.

4.1. Target Approximation Error (TAE)

The TAE ($Z^i_{s,a}$), is the error in the learned $Q(s,a;\theta_i)$ relative to $y^i_{s,a}$, which is determined after minimizing the DQN loss (Algorithm 1 line 6, Algorithm 2 line 7). The TAE is a result of several factors: Firstly, the sub-optimality of θ_i due to inexact minimization. Secondly, the limited representation power of a neural net (model error). Lastly, the generalization error for unseen state-action pairs due to the finite size of the ER buffer.

The TAE can cause a deviations from a policy to a worse one. For example, such deviation to a sub-optimal policy occurs in case $y^i_{s,a} = \hat{y}^i_{s,a} = Q^*(s,a)$ and,

$$\begin{aligned} \text{argmax}_a[Q(s,a;\theta_i)] &\neq \text{argmax}_a[Q(s,a;\theta_i) - Z^i_{s,a}] \\ &= \text{argmax}_a[y^i_{s,a}]. \end{aligned}$$

We hypothesize that the variability in DQN's performance in Figure 1, that was discussed at the start of this section, is related to deviating from a steady-state policy induced by the TAE.

4.2. Overestimation Error

The Q-learning *overestimation phenomena* were first investigated by Thrun & Schwartz (1993). In their work, Thrun and Schwartz considered the TAE $Z^i_{s,a}$ as a random variable uniformly distributed in the interval $[-\epsilon, \epsilon]$. Due to the $\max$ operator in the DQN target $y^i_{s,a}$, the expected overestimation errors $\mathbb{E}_z[R^i_{s,a}]$ are upper bounded by $\gamma\epsilon\frac{n-1}{n+1}$ (where n is the number of applicable actions in state s). The intuition for this upper bound is that in the worst case, all Q values are equal, and we get equality to the upper bound:

$$\mathbb{E}_z[R^i_{s,a}] = \gamma \mathbb{E}_z[\max_{a'}[Z^{i-1}_{s',a'}]] = \gamma\epsilon\frac{n-1}{n+1}.$$

The overestimation error is different in its nature from the TAE since it presents a positive bias that can cause asymptotically sub-optimal policies, as was shown by Thrun & Schwartz (1993), and later by Van Hasselt et al. (2015) in the ALE environment. Note that a uniform bias in the action-value function will not cause a change in the induced policy. Unfortunately, the overestimation bias is uneven and is bigger in states where the Q-values are similar for the different actions, or in states which are the start of a long trajectory (as we discuss in Section 5 on accumulation of TAE variance).

Following from the above mentioned overestimation upper bound, the magnitude of the bias is controlled by the variance of the TAE.

The Double Q-learning and its DQN implementation (Double-DQN) (Van Hasselt et al., 2015; Van Hasselt, 2010) is one possible approach to tackle the overestimation problem, which replaces the positive bias with a negative one. Another possible remedy to the adverse effects of this error is to directly reduce the variance of the TAE, as in our proposed scheme (Section 5).

In Figure 2 we repeated the experiment presented in Van Hasselt et al. (2015) (along with the application of Averaged-DQN). This experiment is discussed in Van Hasselt et al. (2015) as an example of overestimation that leads to asymptotically sub-optimal policies. Since Averaged-DQN reduces the TAE variance, this experiment supports an hypothesis that the main cause for overestimation in DQN is the TAE variance.

5. TAE Variance Reduction

To analyse the TAE variance we first must assume a statistical model on the TAE, and we do so in a similar way to Thrun & Schwartz (1993). Suppose that the TAE $Z^i_{s,a}$ is a random process such that $\mathbb{E}[Z^i_{s,a}] = 0$, $\text{Var}[Z^i_{s,a}] = \sigma^2_s$, and for $i \neq j$: $\text{Cov}[Z^i_{s,a}, Z^j_{s',a'}] = 0$. Furthermore, to focus only on the TAE we eliminate the overestimation error by considering a fixed policy for updating the target values. Also, we can conveniently consider a zero reward $r = 0$ everywhere since it has no effect on variance calculations.

Denote by $Q_i \triangleq Q(s;\theta_i)_{s\in S}$ the vector of value estimates in iteration i (where the fixed action a is suppressed), and by Z_i the vector of corresponding TAEs. For Averaged-DQN we get:

$$Q_i = Z_i + \gamma P \frac{1}{K}\sum_{k=1}^{K} Q_{i-k},$$

where $P \in \mathbb{R}^{S\times S}_{+}$ is the transition probabilities matrix for the given policy. The covariance of the above Vector Autoregressive (VAR) model is given by the discrete-time Lyapunov equation, and can be solved directly or by specialized numerical algorithms (Arthur E Bryson, 1975). However, to obtain an explicit comparison, we further specialize the model to an M-state unidirectional MDP as in Figure 3

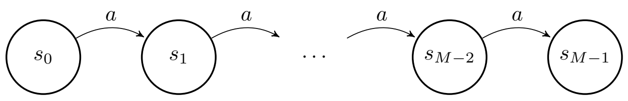

Figure 3. M states unidirectional MDP, The process starts at state s_0, then in each time step moves to the right, until the terminal state s_{M-1} is reached. A zero reward is obtained in any state.

5.1. DQN Variance

We assume the statistical model mentioned at the start of this section. Consider a unidirectional Markov Decision Process (MDP) as in Figure 3, where the agent starts at state s_0, state s_{M-1} is a terminal state, and the reward in any state is equal to zero.

Employing DQN on this MDP model, we get that for $i > M$:

$$\begin{aligned} Q^{\text{DQN}}(s_0,a;\theta_i) &= Z^i_{s_0,a} + y^i_{s_0,a} \\ &= Z^i_{s_0,a} + \gamma Q(s_1,a;\theta_{i-1}) \\ &= Z^i_{s_0,a} + \gamma[Z^{i-1}_{s_1,a} + y^{i-1}_{s_1,a}] = \cdots = \\ &= Z^i_{s_0,a} + \gamma Z^{i-1}_{s_1,a} + \cdots + \gamma^{(M-1)} Z^{i-(M-1)}_{s_{M-1},a}, \end{aligned}$$

where in the last equality we have used the fact $y^j_{M-1,a} = 0$ for all j (terminal state). Therefore,

$$\text{Var}[Q^{\text{DQN}}(s_0,a;\theta_i)] = \sum_{m=0}^{M-1} \gamma^{2m}\sigma^2_{s_m}.$$

The above example gives intuition about the behavior of the TAE variance in DQN. The TAE is accumulated over the past DQN iterations on the updates trajectory. Accumulation of TAE errors results in bigger variance with its associated adverse effect, as was discussed in Section 4.

Algorithm 3 Ensemble DQN

```
1: Initialize $K$ Q-networks $Q(s,a;\theta^k)$ with random weights $\theta^k_0$ for $k \in \{1,\dots,K\}$
2: Initialize Experience Replay (ER) buffer $\mathcal{B}$
3: Initialize exploration procedure *Explore(·)*
4: for $i = 1,2,\dots,N$ do
5:     $Q^E_{i-1}(s,a) = \frac{1}{K}\sum_{k=1}^{K} Q(s,a;\theta^k_{i-1})$
6:     $y^i_{s,a} = \mathbb{E}_{\mathcal{B}}\left[r + \gamma \max_{a'} Q^E_{i-1}(s',a')) \middle| s,a\right]$
7:     for $k = 1,2,\dots,K$ do
8:         $\theta^k_i \approx \text{argmin}_\theta \, \mathbb{E}_{\mathcal{B}}\left[(y^i_{s,a} - Q(s,a;\theta))^2\right]$
9:     end for
10:    *Explore(·)*, update $\mathcal{B}$
11: end for
output $Q^E_N(s,a) = \frac{1}{K}\sum_{k=1}^{K} Q(s,a;\theta^k_i)$
```

5.2. Ensemble DQN Variance

We consider two approaches for TAE variance reduction. The first one is the Averaged-DQN and the second we term *Ensemble-DQN*. We start with Ensemble-DQN which is a straightforward way to obtain a $1/K$ variance reduction, with a computational effort of K-fold learning problems, compared to DQN. Ensemble-DQN (Algorithm 3) solves K DQN losses in parallel, then averages over the resulted Q-values estimates.

For Ensemble-DQN on the unidirectional MDP in Figure 3, we get for $i > M$:

$$
\begin{aligned}
Q_i^E(s_0, a) &= \sum_{m=0}^{M-1} \gamma^m \frac{1}{K} \sum_{k=1}^{K} Z_{s_m,a}^{k,i-m}, \\
\text{Var}[Q_i^E(s_0, a)] &= \sum_{m=0}^{M-1} \frac{1}{K} \gamma^{2m} \sigma_{s_m}^2 \\
&= \frac{1}{K} \text{Var}[Q^{\text{DQN}}(s_0, a; \theta_i)],
\end{aligned}
$$

where for $k \neq k'$: $Z_{s,a}^{k,i}$ and $Z_{s',a'}^{k',j}$ are two uncorrelated TAEs. The calculations of $Q^E(s_0, a)$ are detailed in Appendix B.

5.3. Averaged DQN Variance

We continue with Averaged-DQN, and calculate the variance in state s_0 for the unidirectional MDP in Figure 3. We get that for $i > KM$:

$$
\text{Var}[Q_i^A(s_0, a)] = \sum_{m=0}^{M-1} D_{K,m} \gamma^{2m} \sigma_{s_m}^2,
$$

where $D_{K,m} = \frac{1}{N} \sum_{n=0}^{N-1} |U_n/K|^{2(m+1)}$, with $U = (U_n)_{n=0}^{N-1}$ denoting a Discrete Fourier Transform (DFT) of a rectangle pulse, and $|U_n/K| \leq 1$. The calculations of $Q^A(s_0, a)$ and $D_{K,m}$ are more involved and are detailed in Appendix C.

Furthermore, for $K > 1, m > 0$ we have that $D_{K,m} < 1/K$ (Appendix C) and therefore the following holds

$$
\begin{aligned}
\text{Var}[Q_i^A(s_0, a)] &< \text{Var}[Q_i^E(s_0, a)] \\
&= \frac{1}{K} \text{Var}[Q^{\text{DQN}}(s_0, a; \theta_i)],
\end{aligned}
$$

meaning that Averaged-DQN is theoretically more efficient in TAE variance reduction than Ensemble-DQN, and at least K times better than DQN. The intuition here is that Averaged-DQN averages over TAEs averages, which are the value estimates of the next states.

6. Experiments

The experiments were designed to address the following questions:

- How does the number K of averaged target networks affect the error in value estimates, and in particular the overestimation error.
- How does the averaging affect the learned polices quality.

To that end, we ran Averaged-DQN and DQN on the ALE benchmark. Additionally, we ran Averaged-DQN, Ensemble-DQN, and DQN on a Gridworld toy problem where the optimal value function can be computed exactly.

6.1. Arcade Learning Environment (ALE)

To evaluate Averaged-DQN, we adopt the typical RL methodology where agent performance is measured at the end of training. We refer the reader to Liang et al. (2016) for further discussion about DQN evaluation methods on the ALE benchmark. The hyperparameters used were taken from Mnih et al. (2015), and are presented for completeness in Appendix E. DQN code was taken from McGill University RLLAB, and is available online[1] (together with Averaged-DQN implementation).

We have evaluated the Averaged-DQN algorithm on three Atari games from the Arcade Learning Environment (Bellemare et al., 2013). The game of BREAKOUT was selected due to its popularity and the relative ease of the DQN to reach a steady state policy. In contrast, the game of SEAQUEST was selected due to its relative complexity, and the significant improvement in performance obtained by other DQN variants (e.g., Schaul et al. (2015); Wang et al. (2015)). Finally, the game of ASTERIX was presented in Van Hasselt et al. (2015) as an example to overestimation in DQN that leads to divergence.

As can be seen in Figure 4 and in Table 1 for all three games, increasing the number of averaged networks in Averaged-DQN results in lower average values estimates, better-preforming policies, and less variability between the runs of independent learning trials. For the game of ASTERIX, we see similarly to Van Hasselt et al. (2015) that the divergence of DQN can be prevented by averaging.

Overall, the results suggest that in practice Averaged-DQN reduces the TAE variance, which leads to smaller overestimation, stabilized learning curves and significantly improved performance.

6.2. Gridworld

The Gridworld problem (Figure 5) is a common RL benchmark (e.g., Boyan & Moore (1995)). As opposed to the ALE, Gridworld has a smaller state space that allows the ER buffer to contain all possible state-action pairs. Additionally, it allows the optimal value function Q^* to be ac-

[1]McGill University RLLAB DQN Atari code: `https://bitbucket.org/rllabmcgill/atari_release`. Averaged-DQN code `https://bitbucket.org/oronanschel/atari_release_averaged_dqn`

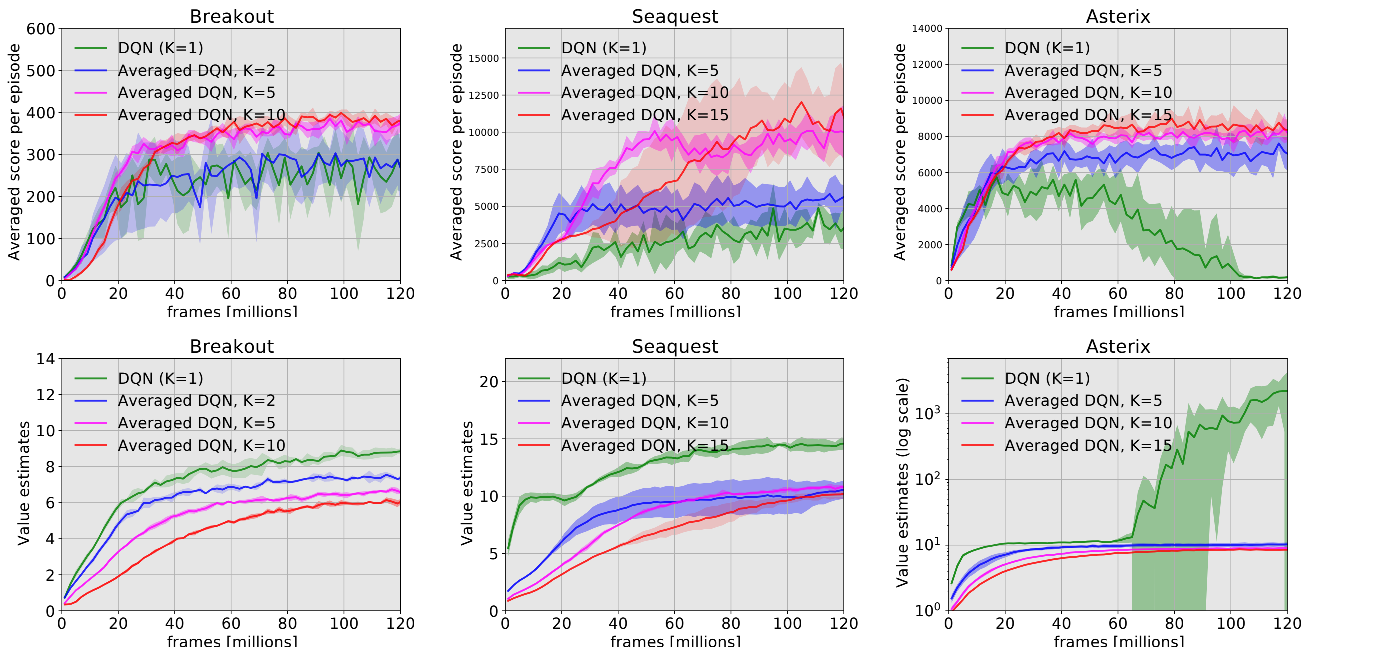

Figure 4. The **top** row shows Averaged-DQN performance for the different number K of averaged networks on three Atari games. For $K = 1$ Averaged-DQN is reduced to DQN. The bold lines are averaged over seven independent learning trials. Every 2M frames, a performance test using ϵ-greedy policy with $\epsilon = 0.05$ for 500000 frames was conducted. The shaded area presents one standard deviation. The **bottom** row shows the average value estimates for the three games. It can be seen that as the number of averaged networks is increased, overestimation of the values is reduced, performance improves, and less variability is observed. The hyperparameters used were taken from Mnih et al. (2015).

curately computed.

For the experiments, we have used Averaged-DQN, and Ensemble-DQN with ER buffer containing all possible state-action pairs. The network architecture that was used composed of a small fully connected neural network with one hidden layer of 80 neurons. For minimization of the DQN loss, the ADAM optimizer (Kingma & Ba, 2014) was used on 100 mini-batches of 32 samples per target network parameters update in the first experiment, and 300 mini-batches in the second.

6.2.1. Environment Setup

In this experiment on the problem of Gridworld (Figure 5), the state space contains pairs of points from a 2D discrete grid ($S = \{(x, y)\}_{x,y \in 1,\dots,20}$). The algorithm interacts with the environment through raw pixel features with a one-hot feature map $\phi(s_t) := (\mathbb{1}\{s_t = (x, y)\})_{x,y \in 1,\dots,20}$. There are four actions corresponding to steps in each compass direction, a reward of $r = +1$ in state $s_t = (20, 20)$, and $r = 0$ otherwise. We consider the discounted return problem with a discount factor of $\gamma = 0.9$.

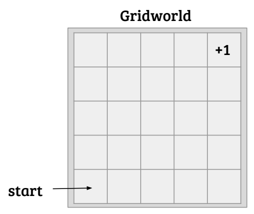

Figure 5. Gridworld problem. The agent starts at the left-bottom of the grid. In the upper-right corner, a reward of +1 is obtained.

6.2.2. Overestimation

In Figure 6 it can be seen that increasing the number K of averaged target networks leads to reduced overestimation eventually. Also, more averaged target networks seem to reduces the overshoot of the values, and leads to smoother and less inconsistent convergence.

6.2.3. Averaged versus Ensemble DQN

In Figure 7, it can be seen that as was predicted by the analysis in Section 5, Ensemble-DQN is also inferior to Averaged-DQN regarding variance reduction, and as a con-

Table 1. The columns present the average performance of DQN and Averaged-DQN after 120M frames, using ϵ-greedy policy with $\epsilon = 0.05$ for 500000 frames. The standard variation represents the variability over seven independent trials. Average performance improved with the number of averaged networks. Human and random performance were taken from Mnih et al. (2015).

GAME	DQN AVG. (STD. DEV.)	AVERAGED-DQN (K=5)	AVERAGED-DQN (K=10)	AVERAGED-DQN (K=15)	HUMAN	RANDOM
BREAKOUT	245.1 (124.5)	381.5 (20.2)	381.8 (24.2)	- -	31.8	1.7
SEAQUEST	3775.2 (1575.6)	5740.2 (664.79)	9961.7 (1946.9)	10475.1 (2926.6)	20182.0	68.4
ASTERIX	195.6 (80.4)	6960.0 (999.2)	8008.3 (243.6)	8364.9 (618.6)	8503.0	210.0

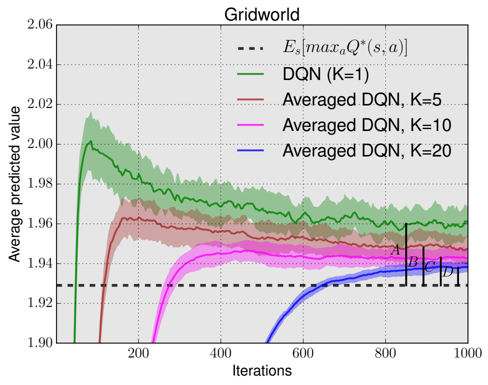

Figure 6. Averaged-DQN average predicted value in Gridworld. Increasing the number K of averaged target networks leads to a faster convergence with less overestimation (positive-bias). The bold lines are averages over 40 independent learning trials, and the shaded area presents one standard deviation. In the figure, **A,B,C,D** present DQN, and Averaged-DQN for K=5,10,20 average overestimation.

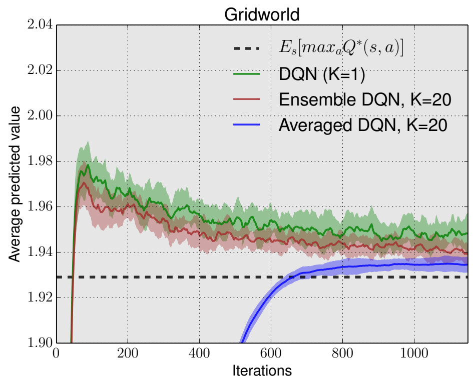

Figure 7. Averaged-DQN and Ensemble-DQN predicted value in Gridworld. Averaging of past learned value is more beneficial than learning in parallel. The bold lines are averages over 20 independent learning trials, where the shaded area presents one standard deviation.

sequence far more overestimates the values. We note that Ensemble-DQN was not implemented for the ALE experiments due to its demanding computational effort, and the empirical evidence that was already obtained in this simple Gridworld domain.

7. Discussion and Future Directions

In this work, we have presented the Averaged-DQN algorithm, an extension to DQN that stabilizes training, and improves performance by efficient TAE variance reduction. We have shown both in theory and in practice that the proposed scheme is superior in TAE variance reduction, compared to a straightforward but computationally demanding approach such as Ensemble-DQN (Algorithm 3). We have demonstrated in several games of Atari that increasing the number K of averaged target networks leads to better policies while reducing overestimation. Averaged-DQN is a simple extension that can be easily integrated with other DQN variants such as Schaul et al. (2015); Van Hasselt et al. (2015); Wang et al. (2015); Bellemare et al. (2016); He et al. (2016). Indeed, it would be of interest to study the added value of averaging when combined with these variants. Also, since Averaged-DQN has variance reduction effect on the learning curve, a more systematic comparison between the different variants can be facilitated as discussed in (Liang et al., 2016).

In future work, we may dynamically learn when and how many networks to average for best results. One simple suggestion may be to correlate the number of networks with the state TD-error, similarly to Schaul et al. (2015). Finally, incorporating averaging techniques similar to Averaged-DQN within on-policy methods such as SARSA and Actor-Critic methods (Mnih et al., 2016) can further stabilize these algorithms.

References

Arthur E Bryson, Yu Chi Ho. *Applied Optimal Control: Optimization Estimation and Control*. Hemisphere Publishing, 1975.

Baird III, Leemon C. Advantage updating. Technical report, DTIC Document, 1993.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Bellemare, Marc G, Srinivasan, Sriram, Ostrovski, Georg, Schaul, Tom, Saxton, David, and Munos, Remi. Unifying count-based exploration and intrinsic motivation. *arXiv preprint arXiv:1606.01868*, 2016.

Bellman, Richard. A Markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

Boyan, Justin and Moore, Andrew W. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, pp. 369–376, 1995.

Even-Dar, Eyal and Mansour, Yishay. Learning rates for q-learning. *Journal of Machine Learning Research*, 5 (Dec):1–25, 2003.

He, Frank S., Yang Liu, Alexander G. Schwing, and Peng, Jian. Learning to play in a day: Faster deep reinforcement learning by optimality tightening. *arXiv preprint arXiv:1611.01606*, 2016.

Jaakkola, Tommi, Jordan, Michael I, and Singh, Satinder P. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.

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

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in NIPS*, pp. 1097–1105, 2012.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Liang, Yitao, Machado, Marlos C, Talvitie, Erik, and Bowling, Michael. State of the art control of Atari games using shallow reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pp. 485–493, 2016.

Lin, Long-Ji. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

Mnih, Volodymyr, Badia, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy P, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

Osband, Ian, Blundell, Charles, Pritzel, Alexander, and Van Roy, Benjamin. Deep exploration via bootstrapped DQN. *arXiv preprint arXiv:1602.04621*, 2016.

Riedmiller, Martin. Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pp. 317–328. Springer, 2005.

Rummery, Gavin A and Niranjan, Mahesan. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, 1994.

Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Sutton, Richard S and Barto, Andrew G. *Reinforcement Learning: An Introduction*. MIT Press Cambridge, 1998.

Sutton, Richard S, McAllester, David A, Singh, Satinder P, and Mansour, Yishay. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pp. 1057–1063, 1999.

Tang, Haoran, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, and Filip De Turck, Pieter Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning. *arXiv preprint arXiv:1611.04717*, 2016.

Tesauro, Gerald. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

Thrun, Sebastian and Schwartz, Anton. Issues in using function approximation for reinforcement learning. In

Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum, 1993.

Tsitsiklis, John N. Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16(3):185–202, 1994.

Tsitsiklis, John N and Van Roy, Benjamin. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5): 674–690, 1997.

Van Hasselt, Hado. Double Q-learning. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems 23*, pp. 2613–2621. 2010.

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

Wang, Ziyu, de Freitas, Nando, and Lanctot, Marc. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv: 1511.06581*, 2015.

Watkins, Christopher JCH and Dayan, Peter. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

A. DQN Variance Source Example

Figure 8 presents a single learning trial of DQN compared to Averaged-DQN, which emphasizes that the source of variability in DQN between learning trials is due to occasions drops in average score within the learning trial. As suggested in Section 4, this effect can be related to the TAE causing to a deviation from the steady state policy.

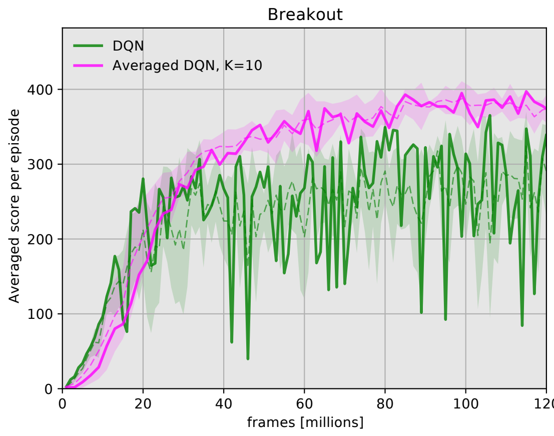

Figure 8. DQN and Averaged-DQN performance in the Atari game of BREAKOUT. The **bold lines** are **single learning trials** of the DQN and Averaged-DQN algorithm. The dashed lines present average of 7 independent learning trials. Every 1M frames, a performance test using ϵ-greedy policy with $\epsilon = 0.05$ for 500000 frames was conducted. The shaded area presents one standard deviation (from the average). For both DQN and Averaged-DQN the hyperparameters used, were taken from Mnih et al. (2015).

B. Ensemble DQN TAE Variance Calculation in a unidirectional MDP (Section 5.2)

Recall that $\mathbb{E}[Z^{k,i}_{s,a}] = 0$, $\text{Var}[Z^{k,i}_{s,a}] = \sigma_s^2$, for all $i \neq j$: $\text{Cov}[Z^{k,i}_{s,a}, Z^{k',j}_{s',a}] = 0$, and for all $k \neq k'$: $\text{Cov}[Z^{k,i}_{s,a}, Z^{k',j}_{s',a}] = 0$. Following the Ensemble-DQN update equations in Algorithm 3:

$$\begin{aligned}
&Q^E_i(s_0,a) = \\
&= \frac{1}{K}\sum_{k=1}^{K} Q(s_0,a;\theta^k_i) \\
&= \frac{1}{K}\sum_{k=1}^{K} [Z^{k,i}_{s_0,a} + y^i_{s_0,a}] \\
&= \frac{1}{K}\sum_{k=1}^{K} [Z^{k,i}_{s_0,a}] + y^i_{s_0,a} \\
&= \frac{1}{K}\sum_{k=1}^{K} [Z^{k,i}_{s_0,a}] + \gamma Q^E_{i-1}(s_1,a) \\
&= \frac{1}{K}\sum_{k=1}^{K} [Z^{k,i}_{s_0,a}] + \frac{\gamma}{K}\sum_{k=1}^{K} [Z^{k,i-1}_{s_1,a}] + \gamma y^{i-1}_{s_2,a}.
\end{aligned}$$

By iteratively expanding $y^{i-1}_{s_2,a}$ as above, and noting that $y^j_{s_{M-1},a} = 0$ for all times (terminal state), we obtain,

$$Q^E_i(s_0,a) = \sum_{m=0}^{M-1} \gamma^m \frac{1}{K}\sum_{k=1}^{K} Z^{k,i-m}_{s_m,a}.$$

Since the TAEs are uncorrelated by assumption, we get

$$\text{Var}[Q^E_i(s_0,a)] = \sum_{m=0}^{M-1} \frac{1}{K}\gamma^{2m}\sigma^2_{s_m}.$$

C. Averaged DQN TAE Variance Calculation in a unidirectional MDP (Section 5.3)

Recall that $\mathbb{E}[Z^i_{s,a}] = 0$, $\text{Var}[Z^i_{s,a}] = \sigma_s^2$, and for $i \neq j$: $\text{Cov}[Z^i_{s,a}, Z^j_{s',a'}] = 0$. Further assume that for all $s \neq s'$: $\text{Cov}[Z^i_{s,a}, Z^i_{s',a}] = 0$. Following the Averaged-DQN update equations in Algorithm 2:

$$\begin{aligned}
&Q^A_i(s_0,a) = \\
&= \frac{1}{K}\sum_{k=1}^{K} Q(s_0,a;\theta_{i+1-k}) \\
&= \frac{1}{K}\sum_{k=1}^{K} [Z^{i+1-k}_{s_0,a} + y^{i+1-k}_{s_0,a}] \\
&= \frac{1}{K}\sum_{k=1}^{K} [Z^{i+1-k}_{s_0,a}] + \frac{\gamma}{K}\sum_{k=1}^{K} Q^A_{i-k}(s_1,a) \\
&= \frac{1}{K}\sum_{k=1}^{K} [Z^{i+1-k}_{s_0,a}] + \frac{\gamma}{K^2}\sum_{k=1}^{K}\sum_{k'=1}^{K} Q(s_1,a;\theta_{i+1-k-k'}).
\end{aligned}$$

By iteratively expanding $Q(s_1,a;\theta_{i+1-k-k'})$ as above, and noting that $y^j_{s_{M-1},a} = 0$ for all times (terminal state), we get

$$\begin{aligned}
Q^A_i(s_0,a) &= \frac{1}{K}\sum_{k=1}^{K} Z^{i+1-k}_{s_0,a} + \frac{\gamma}{K^2}\sum_{k=1}^{K}\sum_{k'=1}^{K} Z^{i+1-k-k'}_{s_1,a} \\
&+ \cdots + \frac{\gamma^{M-1}}{K^M}\sum_{j_1=1}^{K}\sum_{j_2=1}^{K}\cdots\sum_{j_M=1}^{K} Z^{i+1-j_1-\cdots-j_M}_{s_{M-1},a}.
\end{aligned}$$

Since the TAEs in different states are uncorrelated by assumption the latter sums are uncorrelated and we may calculate the variance of each separately. For $L = 1, \ldots, M$, denote

$$V_L = \text{Var}\left[\frac{1}{K^L}\sum_{i_1=1}^{K}\sum_{i_1=2}^{K}\cdots\sum_{i_L=1}^{K} Z_{i_1+i_2+\ldots+i_L}\right],$$

where to simplify notation $Z_L, Z_{L+1}, \ldots, Z_{K\cdot L}$ are independent and identically distributed (i.i.d.) TAEs random variables, with $\mathbb{E}[Z_l] = 0$ and $\mathbb{E}[(Z_l)^2] = \sigma_z^2$.

Since the random variables are zero-mean and i.i.d. we have that:

$$\begin{aligned} V_L &= \frac{1}{K^{2L}}\mathbb{E}_z\left[\left(\sum_{j=L}^{KL} n_{j,L} Z_j\right)^2\right] \\ &= \frac{\sigma_z^2}{K^{2L}}\sum_{j=L}^{KL}(n_{j,L})^2, \end{aligned}$$

where $n_{j,L}$ is the number of times Z_j is counted in the multiple summation above. The problem is reduced now to evaluating $n_{j,L}$ for $j \in \{L, \ldots, K \cdot L\}$, where $n_{j,L}$ is the number of solutions for the following equation:

$$i_1 + i_2 + \ldots + i_L = j, \tag{2}$$

over $i_1, \ldots, i_L \in \{1, \ldots, K\}$. The calculation can be done recursively, by noting that

$$n_{j,L} = \sum_{i=1}^{K} n_{j-i,L-1}.$$

Since the final goal of this calculation is bounding the variance reduction coefficient, we will calculate the solution in the frequency domain where the bound can be easily obtained.

Denote

$$u_j^K = \begin{cases} 1 \text{ if } j \in \{1, \ldots, K\} \\ 0 \text{ otherwise} \end{cases},$$

For $L = 1$ (base case), we trivially get that for any $j \in \mathbb{Z}$:

$$n_{j,1} = u_j^K,$$

and we can rewrite our recursive formula for $n_{j,L}$ as:

$$\begin{aligned} n_{j,L} &= \sum_{i=-\infty}^{\infty} n_{j-i,L-1} \cdot u_i^K \\ &\equiv (n_{j-1,L-1} * u^K)_j \\ &= (\underbrace{u^K * u^K \ldots * u^K}_{L \text{ times}})_j, \end{aligned}$$

where $*$ is the discrete convolution.

To continue, we denote the Discrete Fourier Transform (DFT) of $u^K = (u_n^K)_{n=0}^{N-1}$ as $U = (U_n)_{n=0}^{N-1}$, and by using Parseval's theorem we have that

$$\begin{aligned} V_L &= \frac{\sigma_z^2}{K^{2L}}\sum_{n=0}^{N-1}|(u^K * u^K \ldots * u^K)_n|^2 \\ &= \frac{\sigma_z^2}{K^{2L}}\frac{1}{N}\sum_{n=0}^{N-1}|U_n|^{2L}, \end{aligned}$$

where N is the length of the vectors u and U and is taken large enough so that the sum includes all non-zero elements of the convolution. We denote $D_{K,m} = \frac{1}{K^{2(m+1)}}\frac{1}{N}\sum_{n=0}^{N-1}|U_n|^{2(m+1)}$, and now we can write Averaged-DQN variance as:

$$\text{Var}[Q_i^A(s_0, a)] = \sum_{m=0}^{M-1} D_{K,m}\gamma^{2m}\sigma_{s_m}^2.$$

Next we bound $D_{K,m}$ in order to compare Averaged-DQN to Ensemble-DQN, and to DQN.

For $K > 1, m > 0$:

$$\begin{aligned} D_{K,m} &= \frac{1}{K^{2(m+1)}}\frac{1}{N}\sum_{n=0}^{N-1}|U_n|^{2(m+1)} \\ &= \frac{1}{N}\sum_{n=0}^{N-1}|U_n/K|^{2(m+1)} \\ &< \frac{1}{N}\sum_{n=0}^{N-1}|U_n/K|^2 \\ &= \frac{1}{K^2}\sum_{n=0}^{N-1}|u_n^K|^2 \\ &= \frac{1}{K} \end{aligned}$$

where we have used the easily verified facts that $\frac{1}{K}|U_n| \leq 1$, $\frac{1}{K}|U_n| = 1$ only if $n = 0$, and Parseval's theorem again.

D. Computational Cost Experiments

We have experimented with different number of averaged networks on the ALE to show the added computational cost of in both training and testing. The results are summarized in Figure 9. During testing, we have only forwards passes of the states through the Q-networks (no backpropogations). A naive evaluation of the added cost of $K - 1$ Q-networks in testing would predicts a K times slower testing time (compared to DQN ($K = 1$)). In Figure 9 we see this is not the case. The faster testing speeds are since same state is being forward in the K Q-networks, the

GPU carries out the forward-passes simultaneously. During training time the added computational cost is again $k-1$ more forward-passes through the Q-networks, this time only while learning. For the reason the $k-1$ network are being used only during learning, the effect of averaging more networks is less significant.

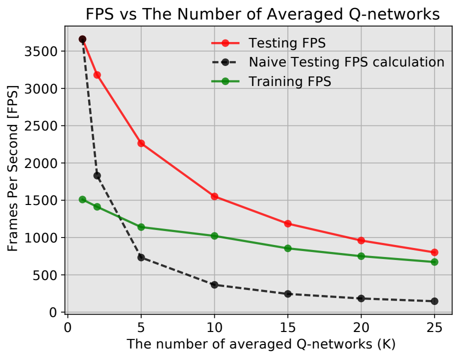

Figure 9. Testing and training speeds. As the number of averaged-Q networks increases speed get slower. However, due to implicit parallel execution of forward passes in the GPU not as slow as a navie prediction.

E. Experimental Details for the Arcade Learning Environment Domain

We have selected three popular games of Atari to experiment with Averaged-DQN. We have used the exact setup proposed by Mnih et al. (2015) for the experiments, we only provide the details here for completeness. The full implementation is available at `https://bitbucket.org/oronanschel/atari_release_averaged_dqn`.

Each episode starts by executing a no-op action for one up to 30 times uniformly. We have used a frame skipping where each agent action is repeated four times before the next frame is observed. The rewards obtained from the environment are clipped between -1 and 1.

E.1. Network Architecture

The network input is a 84x84x4 tensor. It contains a concatenation of the last four observed frames. Each frame is rescaled (to a 84x84 image), and gray-scale. We have used three convolutional layers followed by a fully-connected hidden layer of 512 units. The first convolution layer convolves the input with 32 filters of size 8 (stride 4), the second, has 64 layers of size 4 (stride 2), the final one has 64 filters of size 3 (stride 1). The activation unit for all of the layers a was Rectifier Linear Units (ReLu). The fully connected layer output is the different Q-values (one for each action). For minimization of the DQN loss function RMSProp (with momentum parameter 0.95) was used.

E.2. Hyper-parameters

The discount factor was set to $\gamma = 0.99$, and the optimizer learning rate to $\alpha = 0.00025$. The steps between target network updates were 10,000. Training is done over 120M frames. The agent is evaluated every 1M/2M steps (according to the figure). The size of the experience replay memory is 1M tuples. The ER buffer gets sampled to update the network every 4 steps with mini batches of size 32. The exploration policy used is an ϵ-greedy policy with ϵ decreasing linearly from 1 to 0.1 over 1M steps.

F. Comparison to Double-DQN

As can be seen in Figures 10, **??**, 11, for all three games, the Averaged-DQN results outperform both DQN and Double-DQN. Interestingly, Double-DQN predicts lower values estimates than Averaged-DQN. Combined with the fact that Double-DQN outputs have polices that are inferior to Averaged-DQN, this supports the hypothesis that Double-DQN underestimates the values.

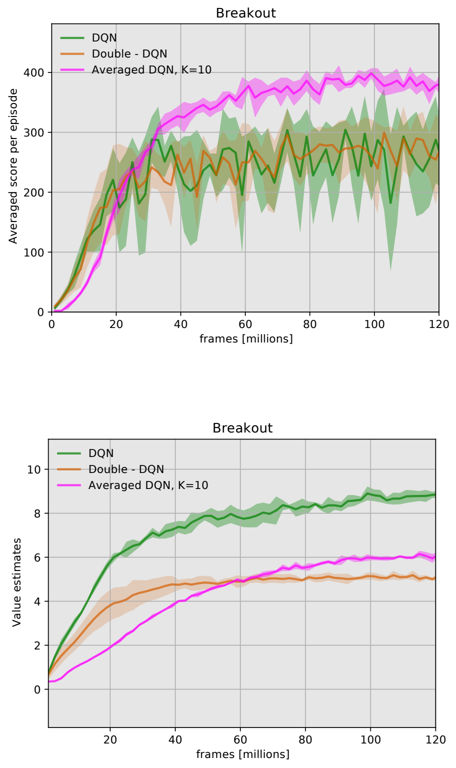

Figure 10. DQN, Double-DQN, and Averaged-DQN ($K = 10$) on the Atari game of Breakout. We see that Average-DQN predicts a higher estimated value than Double-DQN, while demonstrating better policies.

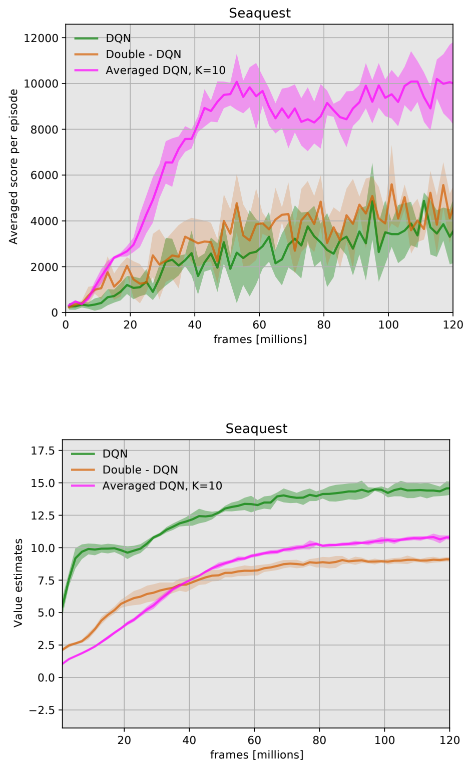

Figure 11. DQN, Double-DQN, and Averaged-DQN ($K = 10$) performance on the Atari game of Seaquest. Averaged-DQN provides significantly improved results over DQN while Double-DQN does not.