# Iterated Deep Q-Network: Efficient Learning of Bellman Iterations for Deep Reinforcement Learning

Théo Vincent[1] * Boris Belousov[1] Carlo D'Eramo[2,3] Jan Peters[1,2,3]

[1]DFKI GmbH, SAIROL, Germany [2]University of Würzburg, Germany
[3]Hessian.ai, Germany [4]TU Darmstadt, Germany

## Abstract

Value-based reinforcement learning methods strive to obtain accurate approximations of optimal action-value functions. Notoriously, these methods heavily rely on the application of the optimal Bellman operator, which needs to be approximated from samples. Most approaches consider only a single Bellman iteration, which limits their power. In this paper, we introduce iterated Deep Q-Network (iDQN), a new DQN-based algorithm that incorporates several consecutive Bellman iterations into the training loss. iDQN leverages the online network of DQN to build a target for a second online network, which in turn serves as a target for a third online network, and so forth, thereby taking into account future Bellman iterations. While using the same number of gradient steps, iDQN allows for better learning of the Bellman iterations than DQN. After providing some theoretical guarantees, we evaluate iDQN against relevant baselines on $54$ Atari 2600 games to showcase its benefit in terms of approximation error and performance. iDQN outperforms DQN while being orthogonal to more advanced DQN-based approaches.

## 1 Introduction

Deep value-based Reinforcement Learning algorithms have achieved remarkable success in various fields, from nuclear physics (Degrave et al., 2022) to construction assembly tasks (Funk et al., 2022). These algorithms aim at learning a function as close as possible to the optimal action-value function, on which they can build a policy to solve the task at hand. To obtain an accurate estimate of the optimal action-value function, the optimal Bellman operator is used to guide the learning procedure in the space of $Q$-functions (Bertsekas, 2019) through successive iterations, starting from any $Q$-function to the optimal action-value function. In Reinforcement Learning (RL), as opposed to Dynamic Programing, the reward function and system dynamics are not assumed to be known (Bertsekas, 2015). This forces us to approximate the optimal Bellman operator with an empirical Bellman operator. This problem has received a lot of attention from the community (Fellows et al. (2021), Van Hasselt et al. (2016)). On top of that, the use of function approximation results in the necessity of learning the projection of the empirical Bellman operator's iteration on the space of approximators. In this work, we focus on the projection step.

We propose a way to improve the accuracy of the learned projection by increasing the number of gradient steps and samples that each $Q$-function estimate has been trained on. This idea, implemented in the training loss function, uses the same total number of gradient steps and samples than the classical approaches. At a given timestep of the learning process, this new loss is composed of the consecutive temporal differences corresponding to the following Bellman iterations needed to be

*Correspondance to: `theo.vincent@dfki.de` learned, as opposed to DQN (Mnih et al., 2015), where only one temporal difference related to the first projection step is considered. Each temporal difference is learned by a different neural network, making this method part of the DQN variants using multiple $Q$ estimates during learning. Those consecutive temporal differences are computed in a telescopic manner, where the online network of the first temporal difference is used to build a target for the second temporal difference and so on. This loss implicitly incurs a hierarchical order between the $Q$ estimates by forcing each $Q$ estimate to be the projection of the Bellman iteration corresponding to the previous $Q$ estimate, hence the name *iterated Deep Q-Network* (iDQN). In the following, we start by reviewing algorithms built on top of DQN, highlighting their behavior in the space of $Q$-functions. We then introduce a new approach to Q-learning that emerges naturally from a graphical representation of DQN. We provide a theoretical grounding for this new approach. In Section 5, we show the benefit of our method on the Arcade Learning Environment benchmark (Bellemare et al., 2013). Our approach outperforms DQN while being orthogonal to other variants, establishing iDQN as a relevant method to consider when aggregating significant advances to design a powerful value-based agent such as Rainbow (Hessel et al., 2018). We also perform further experimental studies to bring evidence of the intuition on which iDQN is built. We conclude the paper by discussing the limits of iDQN and pointing at some promising follow-up ideas.

## 2 Preliminaries

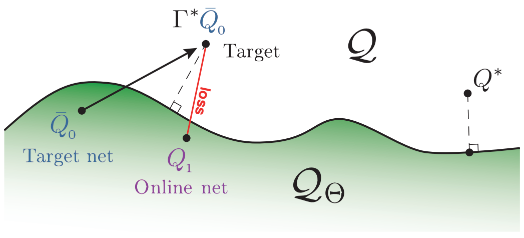
We consider discounted Markov decision processes (MDPs) defined as $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition kernel of the dynamics of the system, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, and $\gamma \in [0, 1)$ is a discount factor (Puterman, 1990). A policy $\pi : \mathcal{S} \to \mathcal{A}$ is a function mapping a state to an action, inducing a value function $V^\pi(s) \triangleq \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t \mathcal{R}(S_t, \pi(S_t)) | S_0 = s\right]$ representing the expected cumulative discounted reward starting in state $s$ and following policy $\pi$ thereafter. Similarly, the action-value function $Q^\pi(s, a) \triangleq \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t \mathcal{R}(S_t, A_t) | S_0 = s, A_0 = a, A_t = \pi(S_t)\right]$ is the expected discounted cumulative reward executing action $a$ in state $s$, following policy $\pi$ thereafter. Q-learning aims to find a function $Q$ from which the greedy policy $\pi^Q(s) = \arg\max_a Q(\cdot, a)$ yields the optimal value function $V^*(\cdot) \triangleq \max_{\pi:\mathcal{S}\to\mathcal{A}} V^\pi(\cdot)$ (Puterman, 1990). The optimal Bellman operator $\Gamma^*$ is a fundamental tool in RL for obtaining optimal policies, and it is defined as:

$$(\Gamma^* Q)(s, a) \triangleq \mathcal{R}(s, a) + \gamma \int_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') \max_{a' \in \mathcal{A}} Q(s', a') \mathrm{d}s', \tag{1}$$

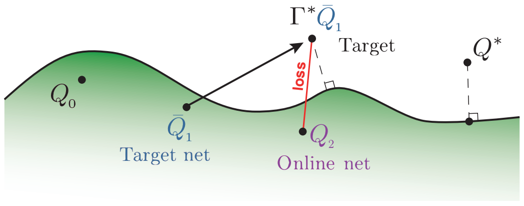
for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. It is well-known that Bellman operators are contraction mappings in $L_\infty$-norm, such that their iterative application leads to the fixed point $\Gamma^* Q^* = Q^*$ in the limit (Bertsekas, 2015). We consider using function approximation to represent value functions and denote $\Theta$ the space of their parameters. Thus, we define $\mathcal{Q}_\Theta = \{Q(\cdot|\theta) : \mathcal{S} \times \mathcal{A} \to \mathbb{R} | \theta \in \Theta\}$ as the set of value functions representable by parameters of $\Theta$.

## 3 Related Work

To provide an overview of the related work, we propose to view the related algorithms from the perspective of their behavior in the space of $Q$-functions, which we denote by $\mathcal{Q}$. Due to the curse of dimensionality, covering the whole space of $Q$-functions with function approximators is practically infeasible, as it requires a large number of parameters. Therefore, the space of *representable* $Q$-functions $\mathcal{Q}_\Theta$ only covers a small part of the whole space $\mathcal{Q}$. We illustrate this in Figure 1a by depicting the space of representable $Q$-functions $\mathcal{Q}_\Theta$ as a subspace of $\mathcal{Q}$. One can deduce two properties from this gap in dimensionality. First, the optimal $Q$-function $Q^*$ is a priori not representable by any chosen function approximator. Second, the same is true for the optimal Bellman operator $\Gamma^*$ applied to a representable $Q$-function. That is why in Figure 1a, both functions $Q^*$ and $\Gamma^* Q$ are drawn outside of $\mathcal{Q}_\Theta$. Additionally, thanks to the contracting property of the optimal Bellman operator $||\Gamma^* Q - Q^*||_\infty \le \gamma ||Q - Q^*||_\infty$, we know that the distance between the iterated $Q$ given by $\Gamma^* Q$ and the optimal $Q^*$ is shrunk by $\gamma$ (Bertsekas, 2015). The goal of most value-based methods



(a) Starting from a random $Q$-function $\bar{Q}_0$, the first Bellman iteration is learned via an online network $Q_1$.



(b) To learn a second Bellman iteration, the target network is updated to the position of the online network.

Figure 1: Graphical representation of DQN in the space of $Q$-functions $\mathcal{Q}$. DQN makes use of a target network $\bar{Q}_{k-1}$ to learn its optimal Bellman iteration $\Gamma^*\bar{Q}_{k-1}$, called target, with an online network $Q_k$. Each iteration is learned by minimizing the distance between the target and the online network.

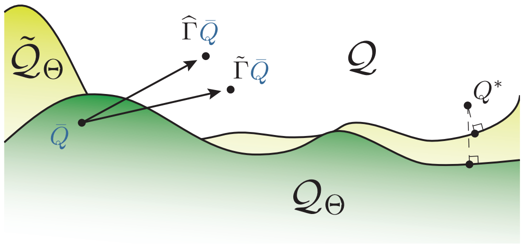
is to learn a $Q$-function that is as close as possible to the projection[2] of the optimal $Q$-function on the space of representable $Q$-functions, shown with a dotted line in Figure 1a.

This perspective allows us to represent various Q-learning algorithms proposed so far in an intuitive way in a single picture. For example, Figure 1a depicts how Deep Q-Network (DQN) by Mnih et al. (2015) works. With a target network $\bar{Q}_0$, DQN aims at learning the iterated target network $\Gamma^*\bar{Q}_0$, also called "target", using an online network $Q_1$. The loss used during training is shown in red. For each Bellman iteration, the goal is to train the online network to be as close as possible to the target computed from the target network. The equality is unlikely because, as previously discussed, the target can be located outside of $\mathcal{Q}_\Theta$, shown in green in all figures. This is why, in the optimal case, the online network is located at the projection of the target on the space of representable $Q$ functions (shown with a dotted line). This perspective also gives a way to understand the hyper-parameter related to the frequency at which the target network is updated. It is the number of training steps before learning the next Bellman iteration. When the target network is updated, it will be equal to the online network, and the next Bellman iteration will be computed from there, as shown in Figure 1b. It is important to note that in DQN, the empirical Bellman operator is used instead of the optimal Bellman operator. The term included in the loss at every gradient step is a stochastic estimation of the optimal Bellman iteration. In Figure 11 of the appendix, we practically show that DQN follows the described behavior in a toy experiment.

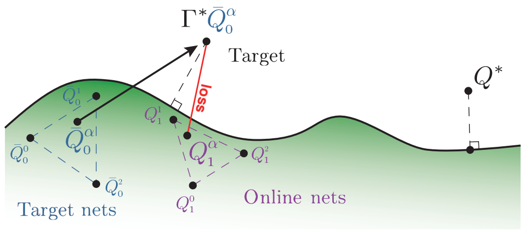
## 3.1 DQN Variants

The DQN paper has inspired the community to develop further methods which improve its efficiency. A large number of those algorithms focuses on using a better empirical Bellman operator (Van Hasselt et al. (2016), Fellows et al. (2021), Sutton (1988)). For instance, double DQN (Van Hasselt et al., 2016) uses an empirical Bellman operator designed to avoid overestimating the return. As shown in Figure 2a, this results in a different location of the Bellman iteration $\tilde{\Gamma}\bar{Q}$ compared to the classical Bellman iteration $\widehat{\Gamma}\bar{Q}(s,a) = \mathcal{R}(s,a) + \gamma \max_{a'} \bar{Q}(s',a')$ for a state $s$, an action $a$ and a next state $s'$. Likewise, $n$-step return (Sutton, 1988) is another type of empirical Bellman operator that computes the target as an interpolation between a one-step bootstrapping and a Monte-Carlo estimate. Other approaches consider changing the space of representable $Q$-functions $\mathcal{Q}_\Theta$(Fatemi & Tavakoli (2022), Wang et al. (2016), Osband et al. (2016)). The hope is that the projection of $Q^*$ on $\mathcal{Q}_\Theta$ is closer than for the classical neural network architecture chosen in DQN. It is important to note that adding a single neuron to one architecture layer can significantly change $\mathcal{Q}_\Theta$. Wang et al. (2016) showed that performance can be increased by including inductive bias in the neural network architecture. This idea can be understood as a modification of $\mathcal{Q}_\Theta$, as shown in Figure 2a where the new space of representable $Q$-function $\tilde{\mathcal{Q}}_\Omega$ is colored in yellow. Furthermore, algorithms such as Rainbow (Hessel et al., 2018) leverage both ideas. Other approaches, however, such as prioritized replay buffer (Schaul et al., 2015), cannot be represented in the picture.

[2]The question of whether a projection on $\mathcal{Q}_\Theta$ exists depends only on the choice of the function approximators. We point out that even if the projection does not exist, the presented abstraction still holds. The unicity of the projection does not play a role here.

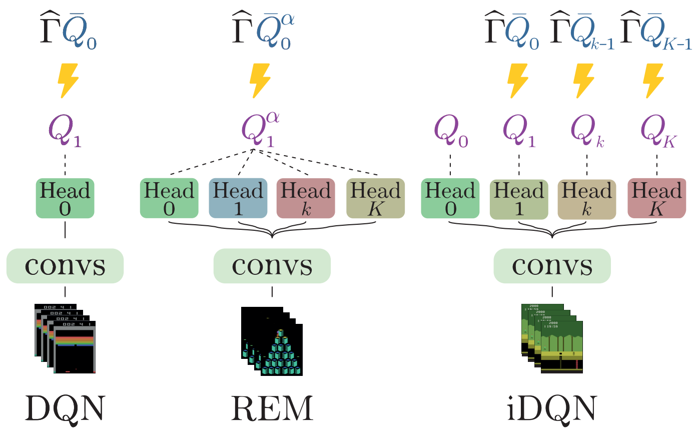


(a) Other empirical Bellman operators can be represented using another notation $\tilde{\Gamma}$ than the classical empirical Bellman operator $\widehat{\Gamma}$. Changing the class of function approximators $\mathcal{Q}_\Theta$ results in a new space $\tilde{\mathcal{Q}}_\Theta$.

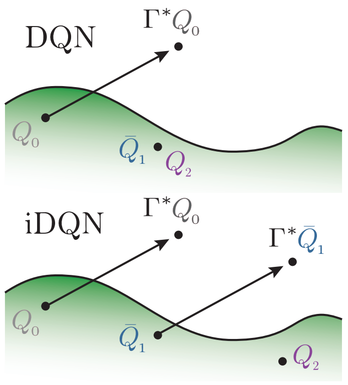


(b) In order to better explore the space $\mathcal{Q}_\Theta$, REM uses a target $Q$-function computed as a convex combinations of 3 networks $(\bar{Q}^0_i)^2_{i=0}$. The same goes for the online $Q$-function that uses 3 online networks $(Q^0_i)^2_{i=0}$.

Figure 2: Graphical representation of DQN variants in the space of $Q$-functions $\mathcal{Q}$.


$\widehat{\Gamma}\bar{Q}_0$
$\widehat{\Gamma}\bar{Q}^\alpha_0$
$\widehat{\Gamma}\bar{Q}_0$ $\widehat{\Gamma}\bar{Q}_{k-1}$ $\widehat{\Gamma}\bar{Q}_{K-1}$
$Q_1$
$Q^\alpha_1$
$Q_0$ $Q_1$ $Q_k$ $Q_K$
Head 0
Head 0
Head 1
Head $k$
Head $K$
Head 0
Head 1
Head $k$
Head $K$
convs
convs
convs
DQN
REM
iDQN

(a) Losses and neural networks architectures. The dotted lines link the outputs of the neural networks to the mathematical objects they represent. The flash signs stress how the information flows from the target(s) $\widehat{\Gamma}\bar{Q}$, considered fixed, to the online network(s) $Q$.



(b) After the same number of gradient steps, iDQN has already started to learn the second Bellman iteration, noted $Q_2$.

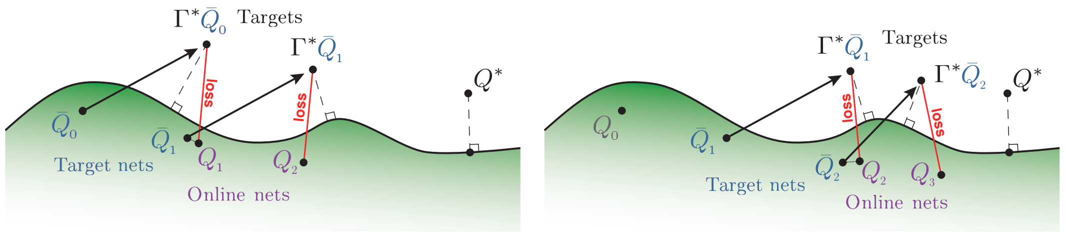
Figure 3: Understanding the loss of iDQN.

## 3.2 Random Ensemble Mixture

Among the variants of DQN, ideas involving learning several $Q$-functions (Anschel et al. (2017), Lan et al. (2020), Osband et al. (2016), An et al. (2021), Agarwal et al. (2020)) are particularly close to our method. Even if they are close, they remain orthogonal in the sense that they can be combined with our idea to create a more powerful agent. Random Ensemble Mixture (REM, Agarwal et al. (2020)) has been shown to be state-of-the-art (SOTA) for DQN variants with several $Q$-functions. Instead of exploring the space of $Q$-functions point by point as DQN does, REM moves in this space by exploring area by area, where the areas are the convex hull of the $Q$-functions stored in memory. As represented by the red line in Figure 2b, the loss used by REM is

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\mathbb{E}_{\alpha\sim\Delta}[l(\delta^\alpha(s,a,r,s'|\theta))]\right],$$
$$\text{with } \delta^\alpha(s,a,r,s'|\theta) = Q^\alpha_1(s,a|\theta) - r - \gamma \max_{a'} \bar{Q}^\alpha_0(s',a'|\bar{\theta})$$

where $\theta$ denotes the parameters of the online network and $\bar{\theta}$ the target parameters, $\mathcal{D}$ is the replay buffer, $\Delta$ is the standard simplex and $l$ is the Huber loss (Huber, 1992), and $Q^\alpha_i = \sum_k \alpha_k Q^k_i, i \in \{0,1\}$. For a Bellman iteration $i$, the $k^{\text{th}}$ learnt $Q$-function is noted $Q^k_i$. Figure 3a shows how this loss is computed with the neural network's architecture used in REM.



(a) iDQN after a few gradient steps and a few updates of the target parameters has been done.

(b) A rolling step is performed to learn the third Bellman iteration with $Q_3$.

Figure 4: Graphical representation of iDQN with $K = 2$ in the space of $Q$-functions denoted $\mathcal{Q}$. Each iteration is learned by minimizing the distance between the target $\Gamma^*\bar{Q}_{k-1}$ and the online network $Q_k$ (see the red lines). The update target frequency regulates the distance between the target network and the online network corresponding to the same Bellman iteration (shown in dotted points).

## 4 Iterated Deep Q-Networks

We propose an approach built on top of DQN. In practice, this new idea consists of changing the loss of DQN such that it is composed of a particular ensemble of $K$ one-step temporal difference instead of one:

$$\mathcal{L}(s, a, r, s'|\theta) = \sum_{k=1}^{K} \left( Q_k(s, a|\theta) - r - \gamma \max_{a'} \bar{Q}_{k-1}(s', a'|\bar{\theta}) \right)^2 \tag{2}$$

where $\theta$ is the online parameters and $\bar{\theta}$ the target parameters. The $k^{th}$ learned $Q$-function corresponds to the $k^{th}$ Bellman iteration and is denoted $Q_k$. The way the loss is computed from the neural network's architecture is presented in Figure 3a. One can see how the $Q$-functions are chained one after the other to learn the Bellman iterations.

In iDQN, updating the target networks does not bring the target parameters to the next Bellman iteration like in DQN. It simply refines their positions to be closer to the online networks to allow better estimates of the iterated $Q$-functions. To be able to go further in the Bellman iterations, we periodically consider a new online $Q$-function and discard the first target network. To learn the $K + 1^{th}$ iteration, the index $k$ in the loss would now go from 2 to $K + 1$. We call this procedure a *rolling step*. In practice, the rolling step is simple to implement. A new head to the neural network of Figure 3a is added, with the index $K + 1$, and the first head is removed (see Figure 12). The new head is initialized with the same parameters as the head with index $K$. It leads us to introduce a new hyper-parameter that indicates at which frequency the rolling step is performed. It is worth noticing that if K is set to 1 and if the rolling step frequency is synchronized with the target update frequency in DQN, then we recover DQN, i.e., iDQN with $K = 1$ is equal to DQN.

This main idea emerges naturally from the representation developed in Section 3. In DQN, Figure 1 illustrates that to learn 2 Bellman iterations, we first need to wait until the first iteration is learned, and then we need to update the target before learning the second iteration. Conversely, we propose to use a second online network that learns the second Bellman iteration while the first Bellman iteration is being learned. The target for the second online network is created from a second target network that is frequently updated to be equal to the first online network. Figure 4a shows how iDQN behaves in the space of $Q$-function. It is important to understand that in iDQN with $K = 2$, both online networks are learned at the same time. As explained earlier in this section, we can learn a following Bellman iteration by adding a new online network $Q_3$ that would use a new target network $\bar{Q}_2$ set to be equal to the last online network $Q_2$ as shown in Figure 4b. In the meantime, the target and online network, $\bar{Q}_0$ and $Q_1$, are discarded to keep the memory usage constant. In practice, the choice of $K$ can be increased until memory usage becomes an issue, as we will show in Section 4.1.

In DQN, the actions are drawn from the online network. For iDQN, one must choose from which of the multiple online networks to sample. One could stick to DQN and choose the first online network. One could also use the last online network since it is supposed to be the one that is closer to the optimal $Q$-function, or one could pick an online neural network at random as it is done in Bootstrapped DQN (Osband et al., 2016). We do not consider taking the mean as REM proposes because the online $Q$-functions are expected to follow a specific arrangement in space. Taking their mean could lead to unwanted behavior. We investigate these sampling strategies in Section 5.1. Algorithm 1 shows how iDQN remains close to DQN, having only two minor modifications on the behavioral policy and the loss.

### 4.1 Understanding the Loss of iDQN

We show that, in principle, iDQN is expected to improve upon the performance of DQN. Namely, we can invoke Theorem 3.4 from Farahmand (2011) on error propagation for Approximate Value Iteration (AVI):

**Theorem 3.4**. Let $K \in \mathbb{N}^*$, $\rho$, $\nu$ two distribution probabilities over $\mathcal{S} \times \mathcal{A}$. For any sequence $(Q_k)_{k=0}^K \subset B(\mathcal{S} \times \mathcal{A}, R_\gamma)$ where $R_\gamma$ depends on reward function and discount factor, we have

$$\|Q^* - Q^{\pi_K}\|_{1,\rho} \leq C_{K,\gamma,R_\gamma} + \inf_{r\in[0,1]} F(r; K, \rho, \gamma) \left(\sum_{k=1}^K \alpha_k^{2r} \|\Gamma^* Q_{k-1} - Q_k\|_{2,\nu}^2\right)^{\frac{1}{2}}$$

where $\alpha_k$ and $C_{K,\gamma,R_\gamma}$ do not depend on the sequence $(Q_k)_{k=0}^K$. Function $F(r; K, \rho, \gamma)$ depends on the concentrability coefficients of the greedy policies w.r.t. the value functions.

In simpler words, this theorem bounds the approximation error at each iteration by a term that includes the sum of approximation errors until the current timestep, i.e., $\sum_{k=1}^K \alpha_k^{2r} \|\Gamma^* Q_{k-1} - Q_k\|_{2,\nu}^2$.

It can be seen that at iteration $k$, the loss of DQN $(r + \gamma \max_{a'} Q_{k-1}(s', a') - Q_k(s, a))^2$ is an unbiased estimator of the approximation error $\|\Gamma^* Q_{k-1} - Q_k\|_2^2$. Likewise, the loss of iDQN $\sum_{k=1}^K (r + \gamma \max_{a'} Q_{k-1}(s', a') - Q_k(s, a))^2$ is an unbiased estimator of the sum of approximation errors $\sum_{k=1}^K \|\Gamma^* Q_{k-1} - Q_k\|_2^2$. From there, one can see that at each gradient step, DQN minimizes only one term of the bound, while iDQN minimizes the whole sum of terms we have influence over[3]. Hence, at each gradient step, iDQN can lower the approximation error bound more than DQN.

We complement this theoretical analysis with an empirical evaluation on a low-dimensional offline problem, Car-On-Hill (Ernst et al., 2005), where the agent needs to drive an underpowered car to the top of a hill. It has a continuous state space and two possible actions: moving left or right. In this problem, the optimal value function $V^*$ can be computed via brute force (Ernst et al., 2005). Figure 5 shows the distance between the optimal value function $V^*$ and $V^{\pi_i}$, i.e., the value function of the greedy policy of the current action-value function estimate obtained with iDQN. This distance is plotted according to the Bellman iterations computed during the training for several values of $K$. We recall that iDQN with $K = 1$ is equivalent to DQN or, more precisely FQI since it is an offline problem. The plot clearly shows that for higher values of $K$, iDQN performs better in the sense that it reaches lower approximation errors earlier during the training. This relates to the theorem previously described. By increasing the value of $K$, we increase the number of Bellman iterations taken into account for each gradient step. Hence, we decrease the upper bound on the distance between the optimal value function and the current estimate, which is what is happening in the plot.

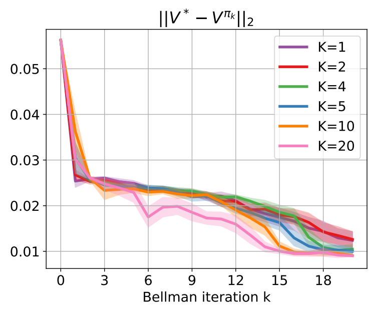


Figure 5: Distance between the optimal value function $V^*$ and $V^{\pi_i}$, the value function obtained at each iteration, for different values of $K$. For $K < 20$, we simply repeat the process until we reach 20 Bellman iterations.

In practice, with iDQN, each Bellman iteration is learned with $K$ times more gradient steps than in DQN while having the same overall number of gradient steps. This means that each selected sample is used $K$ times more or, in other words, that each network sees $K$ times more samples. As mentioned earlier, updating the target in DQN moves the learning procedure one Bellman iteration further. The same goes for the rolling step for iDQN. Since each Bellman iteration is learned with $K$ times more gradient steps than in DQN, we can allow iDQN to perform the rolling step more frequently than DQN updates the target. This means that iDQN will do more Bellman iterations at the end of the training, while still learning each iteration better than DQN. Figure 3b pinpoints the

[3]We do not consider the weights $\alpha_k$ here and leave it for future works.

advantage of iDQN with $K = 2$ over DQN. There, we assume that the update target frequency of DQN is equal to the rolling step frequency in iDQN. When DQN updates its target network $Q_0$ to $\bar{Q}_1$, the new online network $Q_2$ is located at $\bar{Q}_1$. When the rolling step is performed for iDQN, $\bar{Q}_1$ is located at the same position as the $\bar{Q}_1$ of DQN[4] but $Q_2$ has already been learnt and is closer to the optimal $Q$-function than the $Q_2$ of DQN. This phenomenon is even stronger when $K$ increases. Another way to understand iDQN is to see iDQN as a way to pre-train the next online $Q$-functions instead of taking them equal to the online network as it is done in DQN.

## 5 Experiments

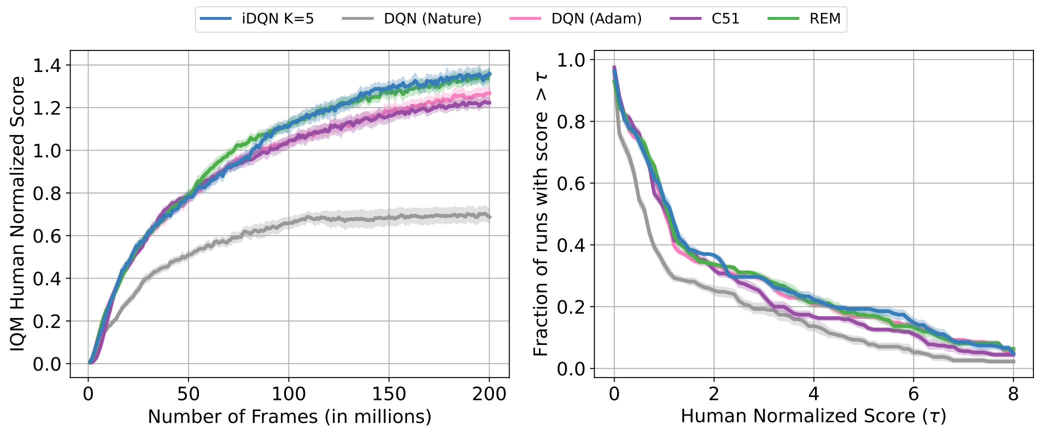
We evaluate our proposed algorithm on $54$ Atari 2600 Games (Bellemare et al., 2013). Many implementations of Atari environments along with classical baselines are available online (Castro et al. (2018), D'Eramo et al. (2021), Raffin et al. (2021), Huang et al. (2022)). We choose to mimic the implementation choices made in Dopamine (Castro et al., 2018) since it is the only one to release the evaluation metric for all relevant baselines to our work and the only one to use the evaluation metric recommended by Machado et al. (2018). Namely, we use the *game over* signal to terminate an episode instead of the *life* signal. The input given to the neural network is a concatenation of $4$ frames in gray scale of dimension $84$ by $84$. To get a new frame, we sample $4$ frames from the Gym environment (Brockman et al., 2016) configured with no frame skip, and we apply a max pooling operation on the $2$ last gray scale frames. We use sticky actions to make the environment stochastic (with $p = 0.25$). The training performance is the one obtained during learning. By choosing an identical setting as Castro et al. (2018) does, we can take the baselines' training performance of Dopamine without the need to train them again ourselves. To certify that the comparison is fair, we compared our version of DQN to their version and concluded positively (see Figure 13 of the appendix).

**Hyperparameter tuning.** The hyperparameters shared with DQN are kept untouched. The two additional hyperparameters (rolling step frequency and target update frequency) were set to follow our intuition on their impact on the performance. As a reminder, the rolling step frequency is comparable to the target update frequency in DQN. To further ensure that our code is trustworthy, Figure 13 in the appendix shows that DQN achieves similar training performances than iDQN with $K = 1$ and the rolling step frequency is set to be equal to the target update frequency of DQN. Since iDQN allows more gradient steps per iteration, we set this hyperparameter to be $25\%$ lower than the target update frequency in DQN ($6000$ compared to $8000$). It is important to note that decreasing the target parameters update results in a more stable training but also a higher delay with the online networks which can harm the overall performance. We set it to 30, allowing 200 target updates per rolling step. We choose $K = 5$. This choice is further discussed in Section 5.1. To make the experiments run faster, we designed the $Q$-functions to share the convolutional layers. Additionally, we consider the first layers of the neural network useful for extracting a feature representation of the state space. This is why this choice can potentially be beneficial to our algorithm. Further details about the hyperparameters can be found in Table 1 of the appendix.

**Performance metric.** As recommended by Agarwal et al. (2021), we choose the interquartile mean (IQM) of the human normalized score to report the results of our experiments with shaded regions showing pointwise $95\%$ percentile stratified bootstrap confidence intervals. IQM is a trade-off between the mean and the median where the tail of the score distribution is removed on both sides to consider only $50\%$ of the runs. 5 seeds are used for each game.

**Main result.** iDQN greatly outperforms DQN (Adam) on the aggregation metric, proposed in Agarwal et al. (2021). Figure 6a shows the IQM human normalized score over $54$ Atari games according to the number of frames sampled during the training. In the last millions of frames, iDQN reaches a higher IQM human normalized score than DQN (Adam). We do not consider other variants of DQN to be relevant baselines to compare with. The ideas used in Rainbow, Implicit Quantile Networks (IQN Dabney et al. (2018a)) or Munchausen DQN (Vieillard et al., 2020) can be included in iDQN algorithm to build an even more powerful agent. We further discuss it in Section 5.1. To visualize the distribution of final scores, we plot the performance profile in Figure 6b. It shows the

[4]The loss in iDQN is additive and includes the DQN loss. Thus, both $\bar{Q}_1$ are located at the same position.



(a) Training performance.  (b) Performance profile.

Figure 6: iDQN outperforms DQN (Nature), DQN (Adam), and C51. DQN (Nature) uses the RMSProp optimizer (Tieleman et al., 2012) while DQN (Adam) uses Adam (Kingma & Ba, 2015).

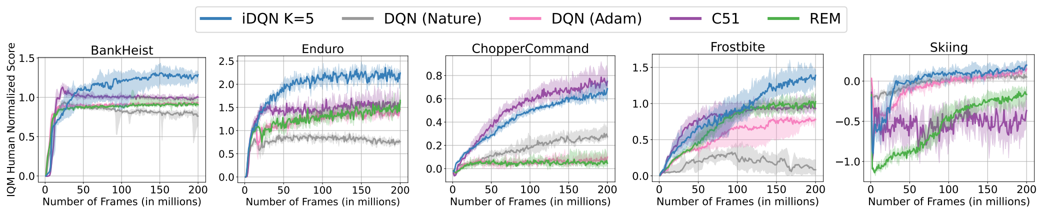


Figure 7: 5 Atari games where different behaviors can be observed.

fraction of runs with a higher final score than a certain threshold given by the $X$ axis. In some ranges of human normalized score, iDQN statistically dominates DQN. For example, there are more games in which iDQN achieves 2 times a human performance ($\tau = 2$) than DQN. iDQN performs similarly to REM on this aggregated metric. Interestingly, those two algorithms show different behaviors on individual games, as shown in Figure 17 of the appendix, where we present the training performance on all the games. This highlights the relevance of iDQN performing on par with the SOTA for DQN variants with several $Q$-functions while relying on an orthogonal idea. In Figure 7, we selected 5 games where different behaviors can be observed. On some games like *BankHeist* and *Enduro*, iDQN overtakes all its baselines. In *ChopperCommand*, DQN (Adam) and REM fail at outperforming DQN (Nature), while iDQN is comparable with C51 (Bellemare et al., 2017) in performance. This shows that efficiently learning the Bellman iterations plays an important role in some environments. In *Frostbite*, REM is more efficient than DQN (Adam). iDQN outperforms both algorithms in this game, being the only one achieving superhuman performances. Finally, in *Skiing*, REM and C51 failed to be better than a random agent, while iDQN sticks to the performance of DQN (Adam). We believe this behavior comes from the fact that iDQN is close to DQN in principle, which minimizes the risk of failing when DQN succeeds.

## 5.1 Ablation Studies

We perform several ablation studies to showcase the different behaviors of iDQN. We first investigate the importance of the number of Bellman iterations $K$ taken into account in the loss. As shown in Figure 8 for the games *Asteroids* and *Asterix*, increasing $K$ to 10 iterations could be beneficial. In *Qbert*, the gain seems not certain. We believe further tuning of hyperparameters should bring iDQN with $K = 10$ to yield better performances than iDQN with $K = 5$. We insist that no hyperparameter tuning has been performed to generate this plot. In order to have the same number of gradient steps per Bellman iteration for $K = 5$ and $K = 10$, we simply halved the rolling step frequency for $K = 10$, bringing it to 3000 since we doubled $K$. Interestingly, the performance never drops in the 3

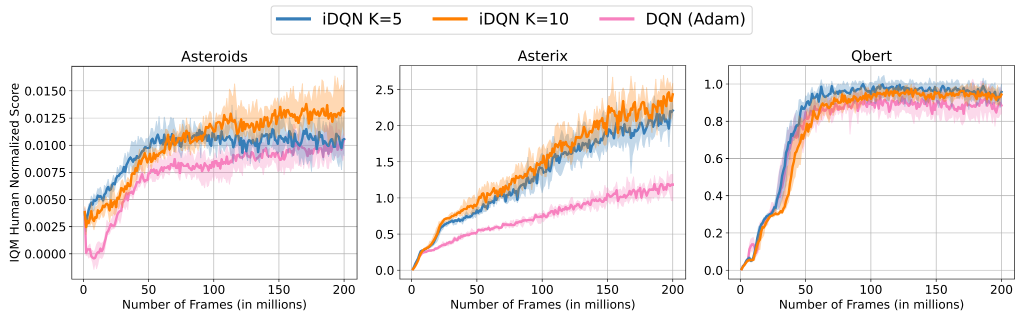


Figure 8: Ablation study on the number of Bellman iterations $K$ taken into account in the loss.

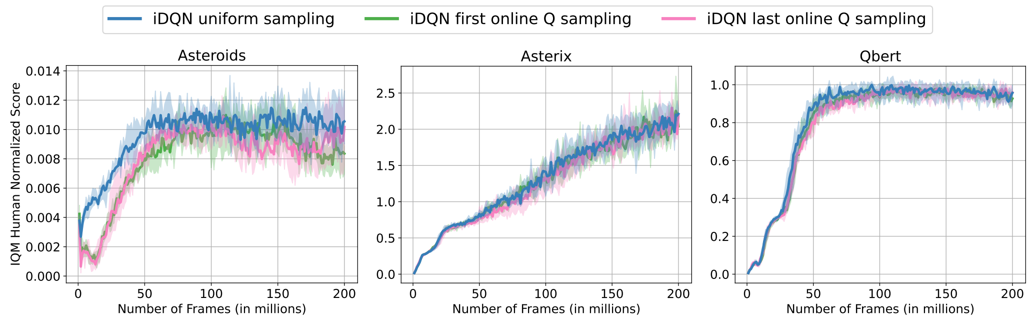


Figure 9: Ablation study on the way actions are sampled to interact with the environment. Actions can be sampled from an online $Q$-function taken at random (in blue), from the first online $Q$-function (in green), or from the last $Q$-function (in pink).

considered games. Therefore, *we recommend increasing $K$ as much as the computational resources allow it.*

In Section 4, we mentioned several sampling strategies. Figure 9 illustrates the different possibilities mentioned earlier: sampling from a uniform online $Q$-function, sampling from the first online $Q$-function like DQN does, or sampling from the last online $Q$-function, it is supposed to be the closest to the optimal $Q$-function. No significant difference exists except for the game *Asteroids*, where sampling from a uniform online $Q$-function seems to yield better performance throughout the training. We believe that the increase in performance comes from the fact that the online Q-functions generate a wider variety of samples compared to only sampling from the first or last online Q-function. *We recommend sampling actions from an online $Q$-function chosen at random.*

iDQN heavily relies on the fact that the learned Q functions are located at different areas in the space of $Q$-functions. We computed the standard deviation of the output of the learned $Q$-functions during the training in Figure 16 of the appendix to verify this assumption. The figure shows that the standard deviation among the $Q$-function is indeed greater than zero across the 3 studied games. Furthermore, the standard deviation decreases during training, suggesting they become increasingly closer. This matches the intuition that at the end of the training, the iteration of the $Q$-functions should point at directions that cannot be followed by the $Q$-functions, hence being close to each other by being stuck on the boundary of the space of representable $Q$-functions.

## 6 Discussion

In Figure 10, we compare iDQN with other powerful baselines to show the gap between those baselines and our approach, which does not use the benefit of a prioritized replay buffer and a $n$-step return. The curves for other algorithms shown in Figure 10 depict the publicly available metrics for those algorithms[5]. The training performance of IQN and Munchausen DQN without the $n$-step return would be interesting to analyze, but our limited resources do not allow us to run those baselines.

[5]Different metrics are often used for different algorithms, making the comparison not straightforward.

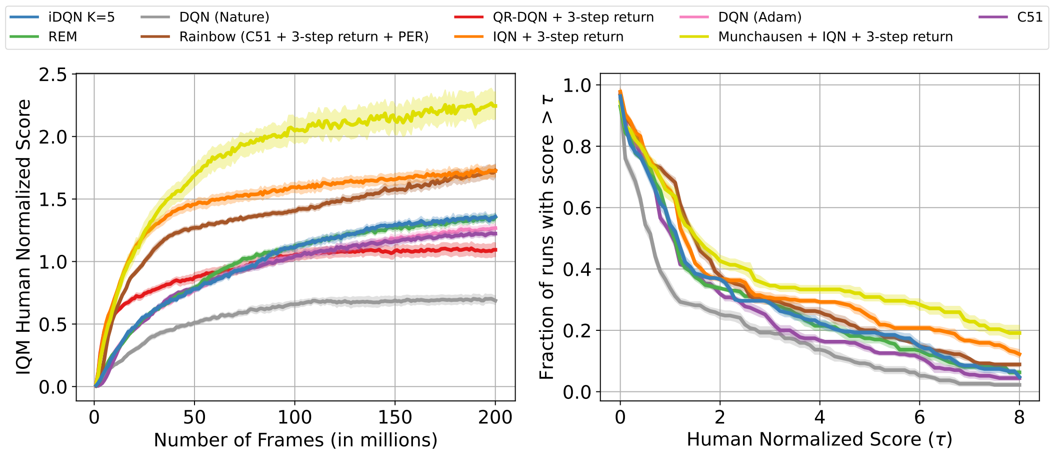


Figure 10: Training performance (left) and performance profile (right) of iDQN and other orthogonal methods. QR-DQN (Dabney et al., 2018b) and DQN (Adam) have been removed from the performance profile for clarity.

The major improvement of Rainbow over C51 is made by using a prioritized replay buffer and adding a 3-step return, which gives hope for following works to study the potential strength of any Rainbow-like iDQN approach. We provide some first encouraging results where we combine iDQN with $K = 3$ and IQN in Figure 15. In the two considered Atari games, iIQN (i.e., iDQN + IQN) outperforms iDQN and IQN, showing that not only is it technically feasible to combine those two algorithms but that it can yield better performance. The training performances of iDQN on 54 Atari games along with those more advanced methods, are available in Figure 18 of the appendix.

Our approach introduces two new hyperparameters, rolling step frequency and target parameters update frequency that need to be tuned. However, we provide a thorough understanding of their effects to mitigate this drawback. First, the rolling step frequency defines the speed at which we learn the Bellman iterations. Problems in which the environment is highly stochastic will require more gradient steps to learn a Bellman iteration hence the need to increase the rolling step frequency. Conversely, problems with sparse rewards and long horizons will be faster to learn with a high rolling step frequency because more Bellman iterations are needed to reach good performance. Second, the target update frequency indicates the speed at which the target networks follow the online networks. Once again, highly stochastic problems will benefit from having a small target update frequency since the positions of the online networks are more likely to be noisy. Conversely, problems with sparse rewards and long horizons can benefit from having the target networks closely follow online networks. Regarding the resources needed to train an iDQN agent, more computations are required to get the gradient of the loss compared to DQN. Thanks to the ability of JAX (Bradbury et al., 2018) to parallelize the computation, iDQN with $K = 5$ only requires 1 to 2 times more time to run. With the released code base, each run presented in this paper can be run under 3 days on an NVIDIA RTX 3090. A detailed analysis of spacial complexity is available in the appendix.

## 7 Conclusion

In this paper, we have presented a way to learn the Bellman iterations more efficiently than DQN. The underlying idea of iDQN comes from an intuitive understanding of DQN's behavior in the space of $Q$-functions. It allows each $Q$ estimate to be learned with more gradient steps without increasing the overall number of gradient steps. iDQN outperforms its closest baseline, DQN, on the Atari 2600 benchmark. While we proposed an approach to $Q$-learning that focuses on the projection step of the Bellman iterations, an interesting direction for future work would be to investigate which other past improvements of DQN, in combination with iDQN, would lead to a new state-of-the-art for value-based methods.

## Acknowledgement

We acknowledge the grant “Einrichtung eines Labors des Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) an der Technischen Universität Darmstadt” of the Hessisches Ministerium für Wissenschaft und Kunst. This work was funded by the German Federal Ministry of Education and Research (BMBF) (Project: 01IS22078). This work was also funded by Hessian.ai through the Connectom Fund on Lifelong Explainable Robot Learning and the project 'The Third Wave of Artificial Intelligence – 3AI' by the Ministry for Science and Arts of the state of Hessen. We thank Daniel Palenicek and Tim Schneider for their support with the experiments on the IAS group's computing cluster at TU Darmstadt. The authors gratefully acknowledge the computing time provided to them on the high-performance computer Lichtenberg at the NHR Centers NHR4CES at TU Darmstadt. This is funded by the Federal Ministry of Education and Research, and the state governments participating on the basis of the resolutions of the GWK for national high performance computing at universities.

## Carbon Impact

As recommended by Lannelongue & Inouye (2023), we used GreenAlgorithms (Lannelongue et al., 2021) and ML $CO_2$ Impact (Lacoste et al., 2019) to compute the carbon emission related to the production of the electricity used for the computations of our experiments. We only consider the energy used to generate the figures presented in this work and ignore the energy used for preliminary studies. The estimations vary between $1.80$ and $2.04$ tonnes of $CO_2$ equivalent. As a reminder, the Intergovernmental Panel on Climate Change advocates a carbon budget of 2 tonnes of $CO_2$ equivalent per year per person.

## References

Agarwal, R., Schuurmans, D., and Norouzi, M. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pp. 104–114. PMLR, 2020.

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

An, G., Moon, S., Kim, J.-H., and Song, H. O. Uncertainty-based offline reinforcement learning with diversified q-ensemble. *Advances in neural information processing systems*, 34:7436–7447, 2021.

Anschel, O., Baram, N., and Shimkin, N. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International conference on machine learning*, pp. 176–185. PMLR, 2017.

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, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pp. 449–458. PMLR, 2017.

Bertsekas, D. *Reinforcement learning and optimal control*. Athena Scientific, 2019.

Bertsekas, D. P. *Dynamic Programming and Optimal Control 4 th Edition , Volume II*. Athena Scientific, 2015.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Bradtke, S. Reinforcement learning applied to linear quadratic regulation. *Advances in neural information processing systems*, 5, 1992.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pp. 1096–1105. PMLR, 2018a.

Dabney, W., Rowland, M., Bellemare, M., and Munos, R. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018b.

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de Las Casas, D., et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

D'Eramo, C., Tateo, D., Bonarini, A., Restelli, M., and Peters, J. Mushroomrl: Simplifying reinforcement learning research. *Journal of Machine Learning Research*, 22(131):1–5, 2021.

Ernst, D., Geurts, P., and Wehenkel, L. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6, 2005.

Farahmand, A.-m. Regularization in reinforcement learning. 2011.

Fatemi, M. and Tavakoli, A. Orchestrated value mapping for reinforcement learning. *arXiv preprint arXiv:2203.07171*, 2022.

Fellows, M., Hartikainen, K., and Whiteson, S. Bayesian bellman operators. *Advances in Neural Information Processing Systems*, 34:13641–13656, 2021.

Funk, N., Chalvatzaki, G., Belousov, B., and Peters, J. Learn2assemble with structured representations and search for robotic architectural construction. In *Conference on Robot Learning*, pp. 1401–1411. PMLR, 2022.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL `http://jmlr.org/papers/v23/21-1342.html`.

Huber, P. J. Robust estimation of a location parameter. *Breakthroughs in statistics: Methodology and distribution*, pp. 492–518, 1992.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

Lan, Q., Pan, Y., Fyshe, A., and White, M. Maxmin q-learning: Controlling the estimation bias of q-learning. *arXiv preprint arXiv:2002.06487*, 2020.

Lannelongue, L. and Inouye, M. Carbon footprint estimation for computational research. *Nature Reviews Methods Primers*, 3(1):9, 2023.

Lannelongue, L., Grealey, J., and Inouye, M. Green algorithms: quantifying the carbon footprint of computation. *Advanced science*, 8(12):2100707, 2021.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

Puterman, M. L. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1): 9–44, August 1988. URL `http://www.cs.ualberta.ca/~sutton/papers/sutton-88.pdf`.

Tieleman, T., Hinton, G., et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Vieillard, N., Pietquin, O., and Geist, M. Munchausen reinforcement learning. *Advances in Neural Information Processing Systems*, 33:4235–4246, 2020.

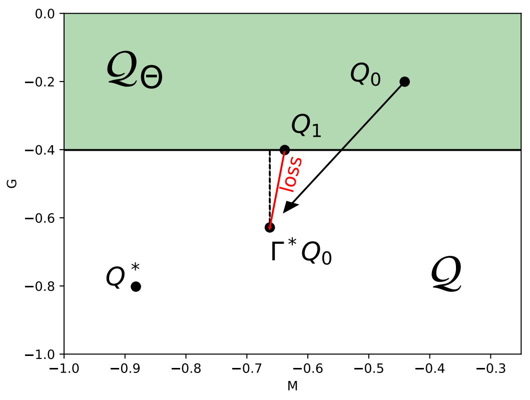
Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016.

## Spatial complexity comparison between DQN and iDQN

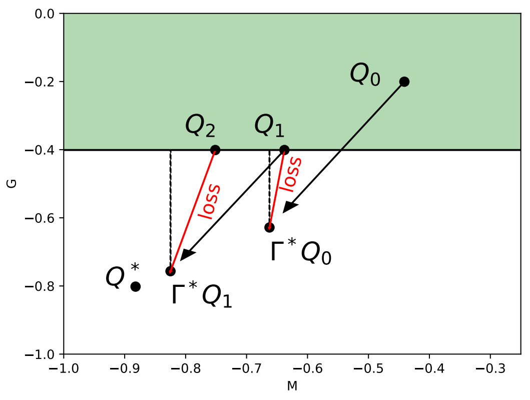
Suppose we note $C$, the memory necessary to store the parameters for the convolutional layers, and $F$, the memory used for the parameters of the fully connected layers. DQN needs $2(C+F)$; the 2 comes from the fact that there is a target and an online network. For iDQN, the memory used is $2(2C+(K+1)F)$. There is a target and an online network as well. This is why there is a 2 on the left. Then, as shown in Figure 12, 2 sets of convolutional parameters are stored along with $K+1$ heads. More precisely, the classical architecture used in Atari games requires 16 MB of memory while iDQN with $K=5$ requires 92 MB and 168 MB for $K=10$. It is worth noticing that those quantities are negligible compared to the space the replay buffer needs. It can reach several GBs even with some memory optimization tricks.

## Table of figures

- In Figure 11, a graphical representation of DQN and iDQN in the space of $Q$-function is shown. This representation has been obtained from a real experiment.
- Figure 12 provides more insights into the neural network's architecture of iDQN. It also illustrates how the rolling step is performed.
- Table 1 gathers all the hyperparameters of iDQN.
- Algorithm 1 shows the pseudo-code of iDQN.
- Figure 13 certifies that our implementation of DQN is comparable to that of DQN in Dopamine. It also shows that iDQN with $K=1$ performs similarly to DQN.
- Figure 14 certifies that our implementation of IQN is comparable to that of IQN in Dopamine. It also shows that iIQN with $K=1$ performs similarly to IQN.
- Figure 15 exhibits the training performance of iIQN along with some relevant baselines.
- Figure 16 shows the standard deviation of the considered $Q$-function during the training.
- In Figure 17, the individual scores of iDQN on the 54 Atari games are presented along with the baselines' scores.
- In Figure 18, the individual scores of iDQN on the 54 Atari games are presented along with the scores of more advanced methods.

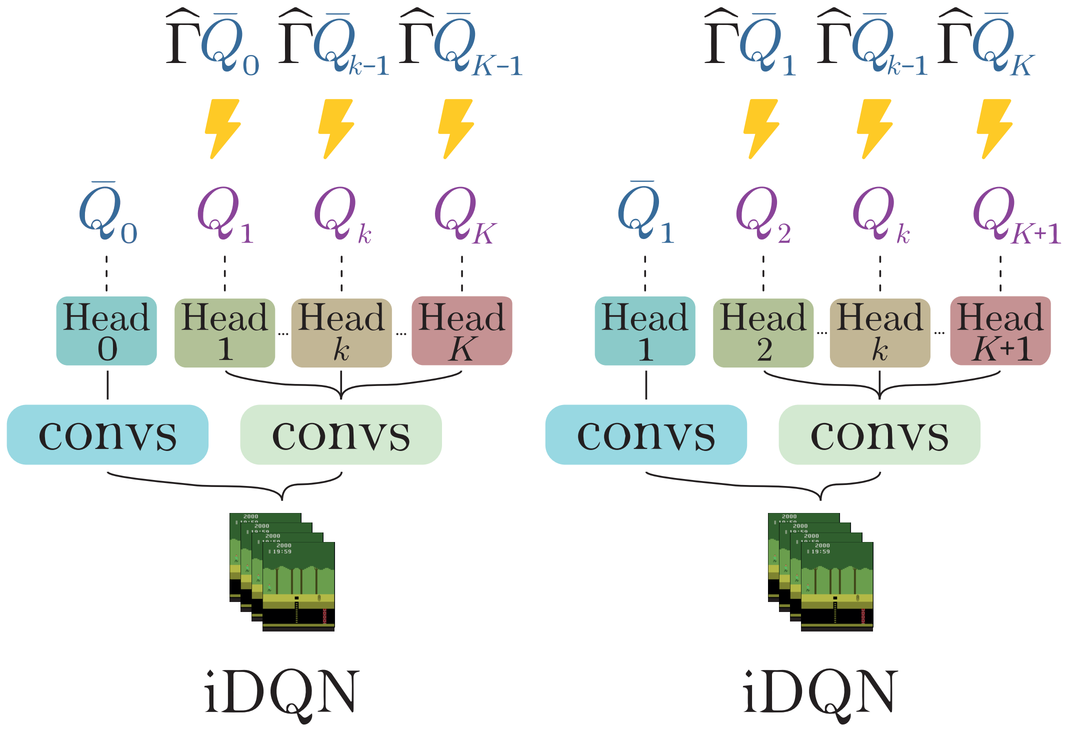


(a) Behavior of DQN.



(b) Behavior of iDQN.

Figure 11: Graphical representation of DQN and iDQN in the space of $Q$-functions $\mathcal{Q}$. This representation is computed from an experiment on a toy offline problem: Linear Quadratic Regulator (Bradtke, 1992). In this problem, the state and action spaces are continuous and one-dimensional. The dynamics are linear: for a state $s$ and an action $a$, the next state is given by $s' = 0.8s - 0.9a$, and the reward is quadratic $r(s, a) = 0.5s^2 + 0.4sa - 0.5a^2$. We choose to parametrize the space of $Q$-function with 2 parameters $(M, G)$ such that, for a state $s$ and an action $a$, $Q(s, a) = Ma^2 + Gs^2$. To avoid having a too big space of representable $Q$-functions, we constrain the parameter $M$ to be negative and parameter $G$ to be between $-0.4$ and $0.4$. Starting from some initial parameters, we perform 30 gradient steps with a learning rate of $0.05$ using the loss of DQN and iDQN. Both figures show the space of representable $Q$-function $\mathcal{Q}_\Theta$ in green, the optimal $Q$-function $Q^*$, the initial $Q$-function $Q_0$ and its optimal Bellman iteration $\Gamma^* Q_0$. The projection of the optimal Bellman iteration is also shown with a dotted line. As we claim in the main paper, iDQN manages to find a $Q$-function $Q_2$ closer to the optimal $Q$-function $Q^*$ than $Q_1$ found by DQN. The figure on the left closely resembles Figure 1a, likewise, the figure on the right looks like Figure 4a, showing that the high-level ideas presented in the paper are actually happening in practice.



$\widehat{\Gamma}\bar{Q}_0$ $\widehat{\Gamma}\bar{Q}_{k-1}$ $\widehat{\Gamma}\bar{Q}_{K-1}$
$\bar{Q}_0$ $Q_1$ $Q_k$ $Q_K$
Head 0
Head 1
Head $k$
Head $K$
convs
convs
iDQN
$\widehat{\Gamma}\bar{Q}_1$ $\widehat{\Gamma}\bar{Q}_{k-1}$ $\widehat{\Gamma}\bar{Q}_K$
$\bar{Q}_1$ $Q_2$ $Q_k$ $Q_{K+1}$
Head 1
Head 2
Head $k$
Head $K$+1
convs
convs
iDQN

Figure 12: Illustration of the rolling step. The first head of the neural network's architecture is not trained. Therefore, the convolutional layers for the first head are computed separately to ensure their outputs are still the ones the first head has been trained on.

Table 1: Summary of all hyperparameters. $\text{Conv}_{a,b}^{d}C$ is a 2D convolutional layer with $C$ filters of size $a \times b$ and of stride $d$, and FC$E$ is a fully connected layer with $E$ neurons.

| Environment | |
|---|---|
| $\gamma$ | 0.99 |
| $H$ | 27000 |
| full action space | No |
| reward clipping | clip$(-1, 1)$ |
| **DQN** | |
| number of epochs $N$ | 200 |
| number of training steps per epochs $n$ | 250000 |
| type of the replay buffer $\mathcal{D}$ | FIFO |
| initial number of samples in $\mathcal{D}$ | 20000 |
| maximum number of samples in $\mathcal{D}$ | 1000000 |
| gradient step frequency $G$ | 4 |
| target update frequency $T$ | 8000 |
| starting $\epsilon$ | 1 |
| ending $\epsilon$ | 0.01 |
| $\epsilon$ linear decay duration | 250000 |
| batch size | 32 |
| learning rate | $6.25 \times 10^{-5}$ |
| Adam $\epsilon$ | $1.5 \times 10^{-4}$ |
| torso architecture | $\text{Conv}_{8,8}^{4}32 - \text{Conv}_{4,4}^{2}64 - \text{Conv}_{3,3}^{1}64-$ |
| head architecture | $-\text{FC}512 - \text{FC}n_{\mathcal{A}}$ |
| activations | ReLU |
| initializer | Xavier uniform |
| **iDQN** | |
| rolling step frequency $R$ | 6000 |
| target update frequency $T$ | 30 |
| sampling policy $\mu$ | uniform |

**Algorithm 1** iDQN. The modifications added to DQN are marked in green.

```
1: Inputs: number of epochs $N$, number of training steps per epoch $n$, sampling head policy $\mu$,
   online and target parameters $\theta = \bar{\theta}$, replay buffer $\mathcal{D}$, gradient step frequency $G$, rolling step
   frequency $R$, target update frequency $T$.
2:
3: $i \leftarrow 0$                                              ▷ number of overall training steps
4: performance $\leftarrow$ empty list
5: for $N$ epochs do
6:     $j \leftarrow 0$                                          ▷ number of training steps within an epoch
7:     $s \leftarrow$ env.init()
8:     absorbing $\leftarrow$ false; sum_reward $\leftarrow 0$; n_episodes $\leftarrow 0$
9:     while $j < n$ and absorbing $=$ false do
10:        sample $k \sim \mu$                                  ▷ sample a neural network head
11:        sample $a \sim \epsilon$-greedy $Q_k(s, \cdot|\theta)$
12:        $(s', r,$ absorbing$) \leftarrow$ env.step($a$)
13:        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s, a, r, s')\}$
14:        $s \leftarrow s'$; sum_reward $+= r$
15:        if absorbing $=$ true then
16:            $s \leftarrow$ env.init()
17:            n_episodes $+= 1$
18:        end if
19:
20:        if $i = 0[G]$ then
21:            $d \sim \mathcal{U}(\mathcal{D})$
22:            $\theta \leftarrow$ Adam_update($\mathcal{L}, d, \theta, \bar{\theta}$)      ▷ $\mathcal{L}$ is defined in (2)
23:        end if
24:        if $i = 0[R]$ then
25:            $(\theta, \bar{\theta}) \leftarrow$ rolling_step($\theta, \bar{\theta}$)      ▷ explained in Section 4
26:        end if
27:        if $i = 0[T]$ then
28:            $\bar{\theta} \leftarrow \theta$
29:        end if
30:        $i += 1; j += 1$
31:    end while
32:    performance.append$\left(\frac{\text{sum\_reward}}{\text{n\_episodes}}\right)$
33: end for
34: return $\theta$
```

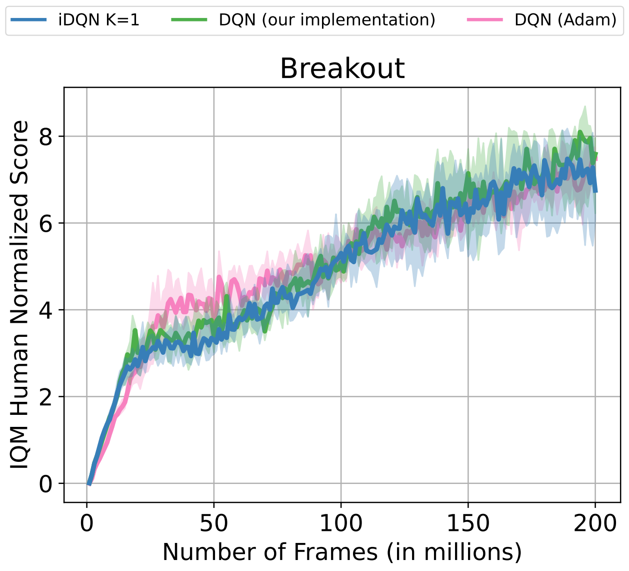


Figure 13: Our implementation of DQN yields similar performance as the implementation of Dopamine (DQN (Adam)). This certifies that we can compare the results released in Dopamine with our method. Both DQN implementations and iDQN with $K = 1$ have a similar behavior. This certifies the trustworthiness of our code base.

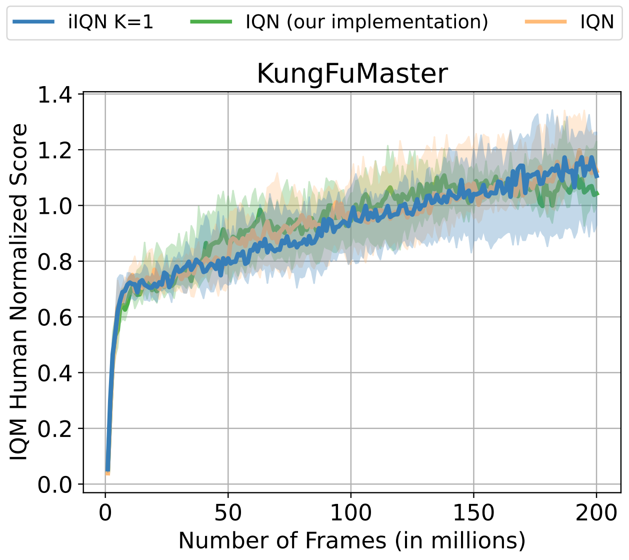


Figure 14: Our implementation of IQN yields similar performance as the implementation of Dopamine (IQN). This certifies that we can compare the results released in Dopamine with our method. Both IQN implementations and iIQN with $K = 1$ have a similar behavior. This certifies the trustworthiness of our code base.

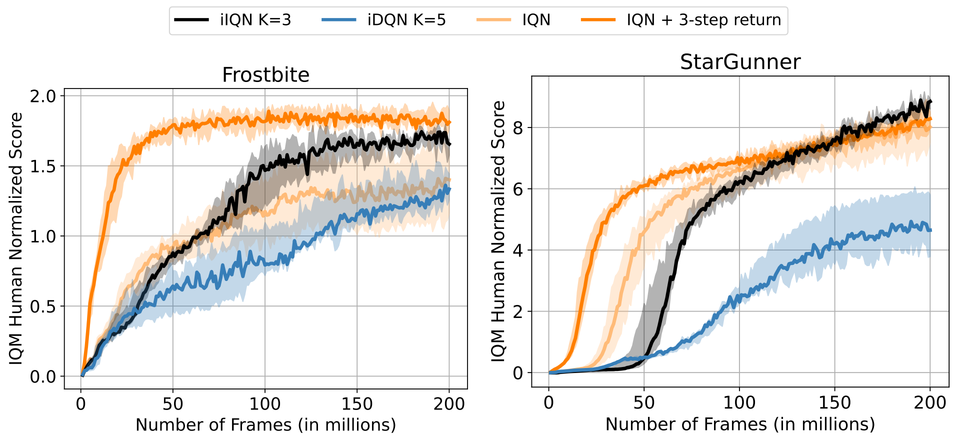


Figure 15: iIQN with $K = 3$ outperforms iDQN with $K = 5$ and IQN, showing that not only is it technically feasible to combine those two algorithms, but that it can yield better performance. Interestingly, iIQN even outperforms IQN + 3-step return on StarGunner.

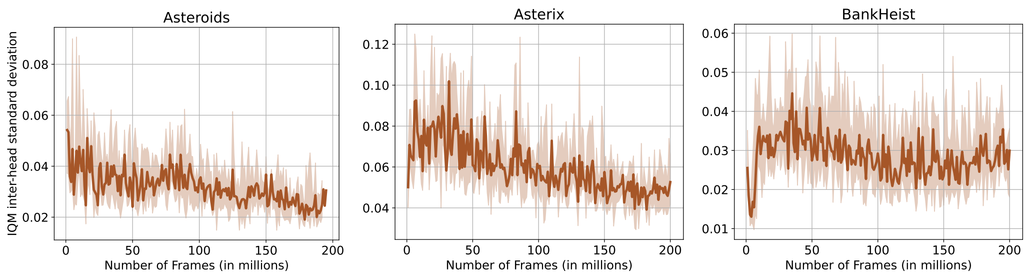


Figure 16: Standard deviation of the online networks' output averaged over 3200 samples at each iteration.

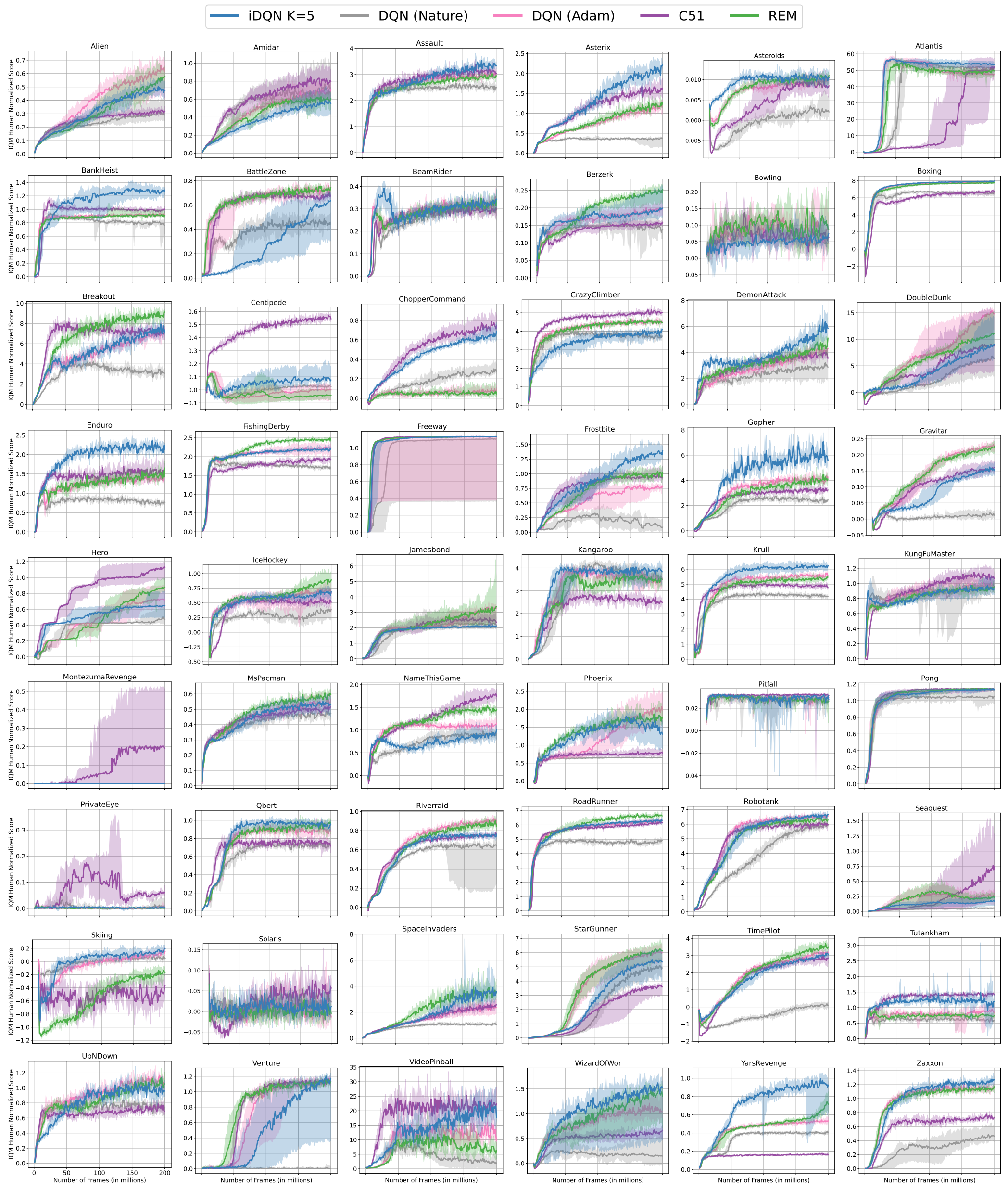


Figure 17: Performances of iDQN ($K = 5$) on the 54 Atari games along with the considered baselines.

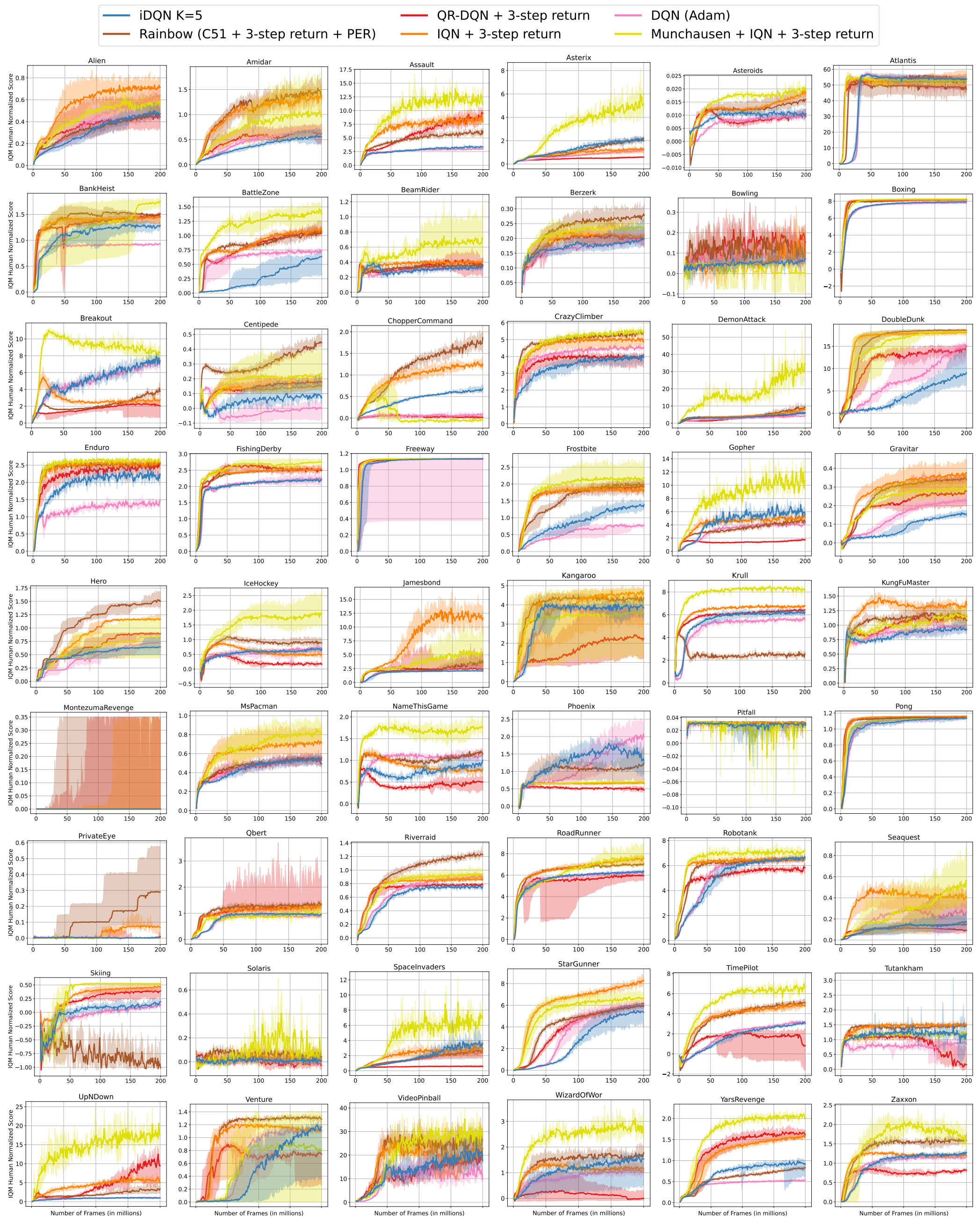


Figure 18: Performances of iDQN ($K = 5$) on the $54$ Atari games along with other improvements over DQN.