Accelerating Q-learning through Efficient Value-Sharing across Actions

Prabhat Nagarajan [1 2] **Brett Daley** [3] **Martha White** [1 2 4] **Marlos C. Machado** [1 2 4]

Abstract

Action values are foundational to many control algorithms such as Q-learning. Therefore, efficient action-value learning is central to reinforcement learning (RL). However, learning them can be slow, requiring many updates to move values from their initialization, typically near zero, to their true values, which may be far from zero. Moreover, action-value learning algorithms typically update each state-action pair independently, without learning a value that is common to all actions within a state. In this paper, we address these inefficiencies by introducing the *mean-expansion layer*, which accelerates action-value learning by sharing values across actions within a state and by changing the problem from directly learning potentially large action-values to learning a lower-norm representation of them. In deep RL, this layer can be applied as a parameter-free addition to Q-network architectures without altering the underlying algorithm. Applied to deep Q-networks and implicit quantile networks, it improves aggregate performance across 57 Atari 2600 games while increasing action gaps and dramatically reducing value overestimation.

1. Introduction

Many reinforcement learning (RL) algorithms learn action-value functions, or Q-functions. Q-functions represent the expected discounted return for state-action pairs. Agents can make decisions by referencing a learned Q-function and selecting high-value actions. Consequently, learning action-value functions efficiently is a central focus in RL (Mnih et al., 2015). We draw attention to two observations.

[1]Department of Computing Science, University of Alberta, Edmonton, AB, Canada. [2]Alberta Machine Intelligence Institute. [3]Meta (independent work, not on Meta's behalf). [4]Canada CIFAR AI Chair. Correspondence to: Prabhat Nagarajan <nagarajan@ualberta.ca>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

The first observation is that the true action-values being learned are often of high (Euclidean) norm, due to each action-value representing an entire return. Large norms can be problematic when we consider that almost universally, action-value estimates are initialized to be close to zero. Furthermore, the update rules of action-value learning algorithms like Q-learning (Watkins, 1989; Watkins & Dayan, 1992) make small, incremental changes to the predicted values. Consequently, when the true action-values are large, several updates are required to change the small initial action-value estimates to the large values over the course of training (Daley et al., 2025). We can reason, then, that lower-norm outputs can be found faster.

The second motivating observation is that in many practical settings, action-values in a state are not mean-zero, as values are often correlated within a state. If one action in a state has a positive or negative action-value, other actions likely do too, and may even share similar values. The majority of action-value learning methods, however, update each action-value in a state individually and independently of other actions. Action-value learning could be accelerated, for example, by learning a common state-dependent baseline value that is shared across all actions in a state. Doing so would transform the challenging problem of independently learning potentially large individual action-values into an easier one of learning smaller residuals.

These two ideals—value sharing through a state-dependent baseline and learning lower-norm solutions—are closely related. Learning a single scalar baseline for each state that is shared across actions requires the baseline to be stored only once, rather than stored repeatedly in each action-value. These ideals motivate the goal of our approach, which is **to represent a vector of action-values with a lower-norm vector by efficiently sharing value across actions**.

In deep RL, there are existing methods that share values to accelerate learning, namely dueling network approaches (Wang et al., 2016; Tang et al., 2023; Daley et al., 2025). At each state, dueling network approaches produce a state-dependent baseline value along with action-specific residuals to construct Q-values. In practice, however, learning this baseline requires adding extra parameters to a Q-network through a separate multi-layer output stream.

In this paper, we derive and analyze a value-sharing method,

called the *mean-expansion layer* (ME layer), that does not require an explicit baseline or extra model parameters. This layer allows us to share values across actions in a state by constructing an *implicit baseline* that is shared by all actions to compute action-values. This layer is implementable as a parameter-free modification to a Q-network. It introduces no changes to the underlying algorithm and can be easily applied to both tabular and deep Q-learning. Simple PyTorch (Paszke et al., 2019) and JAX (Bradbury et al., 2018) implementations of the ME layer are in Appendix D.

Our empirical findings demonstrate several benefits of the mean-expansion layer. We first show that it can accelerate learning in a controlled gridworld setting. In deep RL, we find that it accelerates learning and improves aggregate performance when applied to deep Q-networks (DQN; Mnih et al., 2015) and implicit quantile networks (Dabney et al., 2018) in 57 Atari 2600 games (Bellemare et al., 2013). Lastly, we find that it substantially reduces overestimation and increases the action gap in DQN—both desirable properties in deep RL.

2. Preliminaries

We formulate the RL problem as a finite Markov decision process (MDP). A finite MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, \mu, R, \gamma)$, where $\mathcal{S}$ is a finite set of environment states and $\mathcal{A}$ is the finite set of actions available to the agent. In this paper, we use n to denote $|\mathcal{A}|$, the cardinality of the action space. The state transition probability $P(s'|s,a)$ is the probability that the agent transitions to state s' after taking action a in state s. The distribution $\mu \in \Delta(\mathcal{S})$ is a start-state distribution, where $\mu(s)$ represents the probability of an episode beginning in state s. The reward function R maps state transitions (s,a,s') to a bounded scalar reward $r = R(s,a,s')$, which the agent receives at the next timestep as it transitions to s'.

The agent's objective is to maximize its expected discounted return $\mathbb{E}_{P,\pi,\mu}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}\right]$. The quantity $\gamma \in [0,1)$ is a discount rate that exponentially discounts future rewards. A policy $\pi(\cdot|s) \in \Delta(\mathcal{A})$ is a decision rule defined for all actions that specifies the probability $\pi(a|s)$ of taking action a in state s. The expected discounted return for following policy π after taking action a in state s is known as the action-value function for policy π, formalized as

$$q_\pi(s,a) \coloneqq \mathbb{E}_{P,\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s, A_0 = a\right].$$

To learn to maximize the expected discounted return, algorithms like Q-learning (Watkins & Dayan, 1992) learn an estimated action-value function Q that approximates the action-value function of the optimal policy $q^*(s,a) = \max_\pi q_\pi(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$.

Deep Q-networks (DQNs) (Mnih et al., 2015) adapt Q-learning (Watkins & Dayan, 1992) to leverage neural networks. The algorithm trains a Q-network, which is a Q-function parameterized with a set of neural network parameters $\boldsymbol{\theta}$. Given a state s, the Q-network outputs a vector of action-values: $\mathbf{q}(s;\boldsymbol{\theta}) = Q(s,\cdot;\boldsymbol{\theta}) = [Q(s,a_1;\boldsymbol{\theta}), \ldots, Q(s,a_n;\boldsymbol{\theta})]^\top$. The agent's most recent M experience transitions (s,a,r,s') are stored in a replay buffer $\mathcal{D}$ (Lin, 1992). This buffer serves as a training dataset for supervised regression of target values, where the targets $y(r,s')$ are constructed from the Q-learning update, $y(r,s') = r + \gamma \max_{a'} Q(s',a';\boldsymbol{\theta}^-)$. The parameters $\boldsymbol{\theta}^-$ are the parameters of the *target network*, a time-delayed copy of the Q-network, used to produce stable targets for a fixed interval. The target network parameters are periodically copied from the Q-network parameters: $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$. As the agent interacts with the environment, the algorithm samples transitions from $\mathcal{D}$ uniformly at random and minimizes the loss $\mathbb{E}_{(s,a,r,s')\sim\mathcal{U}(\mathcal{D})}[(y(r,s') - Q(s,a;\boldsymbol{\theta}))^2]$.

3. The Mean-Expansion Layer

In this section, we aim to represent a vector $\boldsymbol{q}$ of action-values with a lower-norm vector in a manner that shares values efficiently across actions. To do so, we derive the *mean-expansion layer* (ME layer), a simple, invertible, linear transformation. We are motivated by the *baseline-residual decomposition*, which represents a vector with a scalar baseline and a vector of residuals. We begin this section by analyzing the norm-reducing properties of baseline-residual decompositions. To strive for simplicity in neural network implementation, we seek a representation of $\boldsymbol{q}$ that does not require explicit maintenance of an extra baseline parameter.

Let $\boldsymbol{q} \in \mathbb{R}^n$. We can represent each entry of $\boldsymbol{q}$ as $q_i = (q_i - b) + b$, where $b \in \mathbb{R}$ is a scalar *baseline* and each $z_i = q_i - b$ is a *residual*. The vector $\boldsymbol{z} = [z_1, \ldots, z_n]^\top$ is a *residual vector*.

Definition 3.1 (*Baseline-residual vector*)**.** The vector $\boldsymbol{u}(\boldsymbol{q}, b) = [z_1, \ldots, z_n, b]^\top$ is the *baseline-residual vector* of $\boldsymbol{q}$ with baseline b.

This decomposition of $\boldsymbol{q}$ explicitly separates the shared component b from $\boldsymbol{z}$. That is, changing b changes all entries $q_i, i \in \{1, \ldots, n\}$. By contrast, changing z_i affects only q_i. The vector $\boldsymbol{u}(\boldsymbol{q}, b)$ can be used to construct $\boldsymbol{q}$:

$$\boldsymbol{q} = (\boldsymbol{q} - b\mathbf{1}) + b\mathbf{1} = \boldsymbol{z} + b\mathbf{1}, \tag{1}$$

where $\mathbf{1}$ denotes the all-ones vector.

3.1. Norm-reducing baselines

Representing $\boldsymbol{q}$ through a baseline-residual vector $\boldsymbol{u}(\boldsymbol{q}, b)$ has the advantage of being efficient, in that its Euclidean norm is often less than that of $\boldsymbol{q}$: $\|\boldsymbol{u}(\boldsymbol{q}, b)\|_2^2 < \|\boldsymbol{q}\|_2^2$. In fact, if the entries of $\boldsymbol{q}$ have non-zero mean, then it can be

represented with a lower-norm baseline-residual decomposition. Moreover, we can compute the baseline that results in the minimum norm baseline-residual vector.

Proposition 1. *(Norm-minimizing baseline). Given a vector $\boldsymbol{q} \in \mathbb{R}^n$, the norm-minimizing baseline is $b^\star = \sum_{i=1}^n q_i/(n+1)$. That is,*

$$b^\star = \operatorname*{argmin}_b \|\mathbf{u}(\boldsymbol{q}, b)\|_2^2 = \frac{\sum_{i=1}^n q_i}{n+1}. \tag{2}$$

Proof. See Appendix A. □

More generally, we consider any baseline-residual decomposition efficient if its vector has lower norm than $\boldsymbol{q}$.

Definition 3.2 (*Norm-reducing baseline*)**.** Any $b \in \mathbb{R}$ such that $\|\boldsymbol{u}(\boldsymbol{q}, b)\|_2^2 < \|\boldsymbol{q}\|_2^2$ is called a *strictly norm-reducing baseline*.

In Proposition 4 in Appendix A, we show that in general, if the mean value of the entries of $\boldsymbol{q}$,

$$\mu_{\boldsymbol{q}} = \frac{1}{n}\sum_{i=1}^n q_i, \tag{3}$$

is non-zero, then any baseline strictly between 0 and $2b^\star$ is a strictly norm-reducing baseline.

In this paper, we only consider baselines between 0 and $\mu_{\boldsymbol{q}}$, which are all strictly norm-reducing baselines, other than $b = 0$. We go one step further in this section, eliminating the baseline from our representation, resulting in even further norm reduction than a baseline-residual vector.

3.2. Deriving the Mean-Expansion Layer

To avoid learning a baseline explicitly, our approach is to store only a residual vector and reconstruct a desired baseline implicitly. To decide our objective, consider how the mean of the residual vector relates to the baseline. Let $\boldsymbol{z}(b) = \boldsymbol{q} - b\mathbf{1}$ be the residual vector of $\boldsymbol{q}$ with baseline b. The absolute value of the mean of this residual vector $g_{\boldsymbol{z}}(b) = |\frac{1}{n}\sum_{i=1}^n (q_i - b)| = |\mu_{\boldsymbol{q}} - b| = |\mu_{\boldsymbol{z}(b)}|$ is a decreasing function from $b = 0$ to $b = \mu_{\boldsymbol{q}}$, regardless of the sign of $\mu_{\boldsymbol{q}}$ (Proposition 5, Appendix A). That is, the magnitude of the mean $\mu_{\boldsymbol{z}(b)}$ decreases from $b = 0$ to $b = \mu_{\boldsymbol{q}}$. Moreover, the Euclidean norm of the residual vector $\boldsymbol{z}(b)$ is also decreasing from $b = 0$ to $b = \mu_{\boldsymbol{q}}$ (Proposition 6, Appendix A). Consequently, to encourage a lower norm residual vector, we optimize the following objective:

$$\min_{\boldsymbol{z}} \left(\frac{1}{n}\|\boldsymbol{z} - \boldsymbol{q}\|_2^2 + k\left(\frac{1}{n}\sum_{i=1}^n z_i\right)^2 \right), \tag{4}$$

where $k \geq 0$ is a penalty coefficient on the mean value of the vector's entries. The first term encourages $\boldsymbol{z}$ to fit $\boldsymbol{q}$ exactly, or equivalently, $\boldsymbol{z} = \boldsymbol{q} - b\mathbf{1}$, with $b = 0$. The second term penalizes the mean of the entries of $\boldsymbol{z}$ for having large magnitude, encouraging $\boldsymbol{z} = \boldsymbol{q} - b\mathbf{1}$, for some b between 0 and $\mu_{\boldsymbol{q}}$, as we will soon show.

Solutions that minimize this objective trade off directly representing $\boldsymbol{q} - 0$ and representing some residual vector $\boldsymbol{q} - b\mathbf{1}$. The choice of k dictates the degree of this tradeoff, or how much the mean should be penalized for being large. The fraction $1/n$ on the first term reflects the fact that $\|\boldsymbol{z} - \boldsymbol{q}\|_2^2$ is a sum over n squared deviations, so its loss grows with n. To account for this sum, we normalize the sum of squared deviations by n. By contrast, regardless of the value of n, the second term squares a single scalar value (i.e., the mean), and does not require normalization as n grows.

We can characterize the solution to this objective as a solution to a linear system of equations. Let $\boldsymbol{J}$ be the all-ones matrix, that is, the $n \times n$ matrix of ones.

Proposition 2. *Given vector $\boldsymbol{q} \in \mathbb{R}^n$, the unique minimizer of Equation 4 is the solution to the system of equations:*

$$\boldsymbol{q} = \left(\boldsymbol{I} + \frac{k}{n}\boldsymbol{J}\right)\boldsymbol{z}. \tag{5}$$

That is, to solve the objective in Equation 4, we can solve the system in Equation 5. For $k \geq 0$, we call the invertible matrix $\boldsymbol{M}_k = \boldsymbol{I} + \frac{k}{n}\boldsymbol{J}$ the *mean-expansion layer*, where k is the *mean-scaling coefficient*.

Returning to the language of baselines and residuals, the vector $\boldsymbol{z}$ is still a residual vector, but the baseline vector $b\mathbf{1} = \frac{k}{n}\boldsymbol{J}\boldsymbol{z}$ is inferred from $\boldsymbol{z}$. In other words, given some target vector of values $\boldsymbol{q}$, for some desired baseline $b \in [0, \mu_{\boldsymbol{q}})$, $\boldsymbol{q}$ can be entirely represented by the residuals $\boldsymbol{z}$ that represent the solution to the system described in Equation 5.

3.3. Geometric Properties

As per its name, the mean-expansion layer scales the mean component of a vector. To understand this, first note that the matrix $\frac{1}{n}\boldsymbol{J}$ is the projection matrix onto the all-ones direction $\mathbf{1}$. Applied to a vector, it produces a constant vector of means $\frac{1}{n}\boldsymbol{J}\boldsymbol{q} = \mu_{\boldsymbol{q}}\mathbf{1} = [\mu_{\boldsymbol{q}}, \ldots, \mu_{\boldsymbol{q}}]^\top$. We call $\mu_{\boldsymbol{q}}\mathbf{1}$ the *mean component* of $\boldsymbol{q}$, that is, the projection of $\boldsymbol{q}$ onto $\mathbf{1}$. Geometrically, the mean-expansion layer scales the mean component by a factor of $k + 1$.

Equation 5 can be rewritten as (see Appendix A.3):

$$\boldsymbol{q} = \underbrace{\left((k+1)\frac{1}{n}\boldsymbol{J}\boldsymbol{z}\right)}_{\text{mean scaling}} + \underbrace{\left(\boldsymbol{z} - \frac{1}{n}\boldsymbol{J}\boldsymbol{z}\right)}_{\text{mean orthogonal}}. \tag{6}$$

The mean component of $\boldsymbol{z}$ is scaled by $k+1$ and added to the component of $\boldsymbol{z}$ orthogonal to $\mathbf{1}$ to produce $\boldsymbol{q}$. Equivalently,

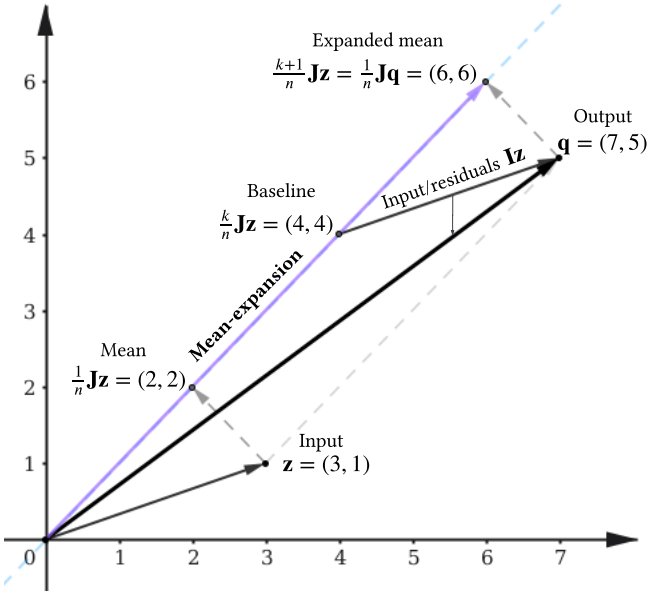


Figure 1. **The mean-expansion layer.** The input vector, $\boldsymbol{z} = (3,1)$, is projected onto the all-ones vector to produce the mean component $(2,2)$. This mean component is scaled by k, where $k = 2$, to produce the implicit baseline vector $\frac{k}{n}\boldsymbol{J}\boldsymbol{z} = (4,4)$. The input vector, $\boldsymbol{z}$, which also serves as the residual vector, is added to this implicit baseline to produce the output $\boldsymbol{q} = (\boldsymbol{I} + \frac{k}{n}\boldsymbol{J})\boldsymbol{z} = (7,5)$. The mean component of $\boldsymbol{q}$ is $(6,6)$, scaling the mean component of $\boldsymbol{z}$ by $k+1$. The ME layer allows us to store the low-norm vector $\boldsymbol{z}$ instead of $\boldsymbol{q}$, without explicitly storing a baseline.

the residual vector $\boldsymbol{z}$ is $\boldsymbol{q}$ with its mean component scaled down. Small changes to $\boldsymbol{z}$ along the all-ones dimension result in large changes to $\boldsymbol{q}$ along the all-ones dimension. Figure 1 provides a visual depiction of the transformation.

3.4. Selecting k

We have established how the mean-expansion layer avoids the explicit maintenance of a baseline parameter b. We have not yet, however, described the relationship between the implicit baseline b and the mean-scaling coefficient k. Fortunately, we can formally characterize their relationship, and do so in the following proposition.

Proposition 3. *Let $\mu_{\boldsymbol{z}} = \frac{1}{n}\sum_{i=1}^{n} z_i$. Let $b = k\mu_{\boldsymbol{z}}$, where $b\mathbf{1} = \frac{k}{n}\boldsymbol{J}\boldsymbol{z}$. Then if $k > 0$,*

$$b = k\mu_{\boldsymbol{z}} = \frac{\sum_{i=1}^{n} q_i}{n + \frac{n}{k}}. \tag{7}$$

Proof. See Appendix A. □

An algorithm designer can leverage this formal relationship to select k to induce some specific implicit baseline. For example, by applying Equation 7, we see that as $k \to 0^+$, that is, as k approaches 0 from the right, the baseline $b \to 0$. As $k \to \infty$, $b \to \mu_{\boldsymbol{q}}$. In this paper, we use $k = n$ unless stated otherwise, which corresponds to the norm-minimizing baseline $b^\star$.

It is easy to see that for $k > 0$, when $\boldsymbol{q}$ has non-zero mean, $\boldsymbol{z}$ has a lower norm than both $\boldsymbol{q}$ *and* its corresponding baseline-residual vector. That is, $\|\boldsymbol{z}\|_2^2 < \|\boldsymbol{u}(\boldsymbol{q}, k\mu_{\boldsymbol{z}})\|_2^2 < \|\boldsymbol{q}\|_2^2$. This follows from the fact that when $k > 0$, the implied baseline is between 0 and $\mu_{\boldsymbol{q}}$, which are all strictly norm-reducing baselines. By not explicitly storing the baseline, $\boldsymbol{z}$ has even lower norm than its baseline-residual counterpart.

3.5. Optimization and Numerical Stability

When using the mean-expansion layer, the coefficient k cannot be too large. While no mathematical issues arise with an arbitrarily large k, computational or numerical issues do. Computing $\boldsymbol{z}$ requires solving the system in Equation 5. For $k \geq 0$, the condition number of $\boldsymbol{M}_k$ is $k+1$, arising from scaling the all-ones direction by $k+1$. Hence, solving this system becomes numerically unstable for large values of k. For extremely large n, $k = 1$ is a safe choice, keeping the condition number at 2. The value $k = 1$ corresponds to the baseline $\mu_{\boldsymbol{q}}/2$, which is a strictly norm-reducing baseline when the mean is non-zero.

4. The Mean-Expansion Layer for Q-networks

Implementing the mean-expansion layer in a neural network is straightforward. Just as a softmax layer transforms a vector of logits into a probability vector, the ME layer takes as input an n-dimensional vector and outputs the input shifted by a baseline constructed by scaling the mean of its entries. The layer increases the magnitude of the mean value of the input vector entries while preserving their relative differences. PyTorch (Paszke et al., 2019) and JAX (Bradbury et al., 2018) code for the ME layer are provided in Appendix D.

The mean-expansion layer is added immediately before the vector-valued output of a neural network. In the previous section, we described how solving the system in Equation 5 allows us to produce the lower-magnitude residual-only representation of some vector $\boldsymbol{q}$. When producing a vector-valued output $\boldsymbol{q}$ for a state, as is done in Q-learning, we cannot solve Equation 5 without prior knowledge of $\boldsymbol{q}$. Since $\boldsymbol{q}$ is being learned, we instead implicitly change the problem to one of learning $\boldsymbol{z}$. The network's penultimate layer first outputs $\boldsymbol{z}$ and then we apply the final layer, an ME layer, to construct $\boldsymbol{q} = \boldsymbol{M}_k\boldsymbol{z}$. Losses on $\boldsymbol{q}$ are then backpropagated through $\boldsymbol{M}_k$ to $\boldsymbol{z}$. We transition from gradient descent on $\boldsymbol{q}$ in a standard Euclidean space to gradient descent on $\boldsymbol{q}$ in a modified Euclidean space where distances along the all-ones direction are shortened.

Taking as input the vector $\boldsymbol{z}$, the layer is implemented in a

manner similar to Equation 6 to produce output $\boldsymbol{q}$:

$$\mu_{\boldsymbol{z}} \leftarrow \frac{1}{n}\sum_{i=1}^{n} z_i,$$
$$\boldsymbol{q} \leftarrow \boldsymbol{z} - \mu_{\boldsymbol{z}}\mathbf{1} + (k+1)\mu_{\boldsymbol{z}}\mathbf{1}.$$

4.1. Implicit-Baseline Deep Q-networks

Concretely, in DQN, the standard output layer becomes the penultimate layer and we add an ME layer as the output layer. Given a state s, a standard Q-network outputs a vector $\boldsymbol{q}(s;\boldsymbol{\theta})$ containing the n values corresponding to each action-value $Q(s,a;\boldsymbol{\theta})$. Adding a mean-expansion layer causes the network to first output a residual vector $\boldsymbol{z}$, whose entries are aggregated according to the mean-expansion layer to produce the action-value outputs. We call this method Implicit-Baseline DQN, or IB-DQN(k), where $k \geq 0$ is the hyperparameter to set the desired baseline.

IB-DQN has several benefits. First, it generalizes DQN as a special case when $k = 0$. Moreover, the invertibility of the transformation ensures that with a sufficiently deep network, the representation capacity does not change relative to a standard Q-network. As a mere layer addition, IB-DQN does not modify the DQN algorithm itself and introduces no additional learnable parameters to the model. A major benefit of IB-DQN is that k is not an opaque hyperparameter. We can select k to produce some specific implicit baseline as a function of $\boldsymbol{q}$. This interpretability may make the problem of hyperparameter selection easier.

4.2. Mean-Expansion for Tabular Q-learning

The mean-expansion layer can also be applied to tabular Q-learning. In lieu of a lookup table of Q-values, we maintain a lookup table of residuals $Z(s,a)$ and construct the Q-values using the mean-expansion layer $\boldsymbol{M}_k$: $Q(s,\cdot) = \boldsymbol{M}_k Z(s,\cdot)$. This is equivalent to $Q(s,a) = Z(s,a) + \frac{k}{n}\sum_{a'} Z(s,a')$.

A gradient descent step with step size α_t produces the semi-gradient (i.e., the target is treated as a constant) update rule for all $a \in \mathcal{A}$ (see Appendix B for gradient computation):

$$Z(s_t,a_t) \leftarrow Z(s_t,a_t) + \left(1 + \frac{k}{n}\right)\alpha_t\delta_t, \tag{8}$$
$$Z(s_t,a) \leftarrow Z(s_t,a) + \frac{k}{n}\alpha_t\delta_t, \quad a \neq a_t. \tag{9}$$

We call this *Implicit-Baseline Q-learning* (IBQ), as we combine the mean-expansion layer, which generates an implicit baseline, with Q-learning. Each action's residual $Z(s_t,a)$ is incremented by $\frac{k}{n}\alpha_t\delta_t$. The chosen action's residual is incremented by an additional $\alpha_t\delta_t$. The action-value of the chosen action is incremented in greater proportion to the Q-learning error than the other actions, but the entire update emphasizes updating the baseline substantially more when k is large. Observe that we recover Q-learning (Watkins & Dayan, 1992) as a special case when $k = 0$. More generally, this derivation applies to any TD-learning-based action-value learning algorithm like Sarsa (Rummery & Niranjan, 1994) and Expected Sarsa (John, 1994).

The update rule in Equations 8 and 9 highlights how the ME layer induces value-sharing in Q-learning. The construction of $Q(s,a)$ shows that all the $Z(s,a)$ in a state each carry some component of the shared baseline used to construct *any* action's value in that state. Consequently, the credit for the TD error is distributed across all actions in a state, as indicated by the update equations. This manner of credit distribution should accelerate learning of the shared baseline in a state relative to the residuals, potentially accelerating overall learning.

5. Related Work

The most closely related approaches to ours are dueling network approaches (Wang et al., 2016; Tang et al., 2023; Daley et al., 2025), with the closest being Dueling DQN. Dueling DQN, like our method, is also an architectural modification to a Q-network that is otherwise trained through standard DQN, without altering the loss function. Dueling DQN, despite the use of the terms "value" and "advantage", in fact implements a baseline-residual decomposition of the action-value function. Dueling DQN outputs a state-specific baseline $B(s;\boldsymbol{\theta})$, which serves as the mean action-value. It also outputs action-specific residuals $Z(s,a;\boldsymbol{\theta})$ to construct action-values as $Q(s,a;\boldsymbol{\theta}) = B(s;\boldsymbol{\theta}) + Z(s,a;\boldsymbol{\theta})$. To learn this baseline, however, Dueling DQN adds an additional two-layer output stream to a Q-network, introducing a substantial number of extra parameters. In Appendix C, we show how Dueling DQN is a baseline-residual decomposition, specifically a mean-residual decomposition.

Another related method is called Regularized Dueling Q-learning (RDQ) (Daley et al., 2025). RDQ leverages the same architecture as Dueling DQN. However, $B(s;\boldsymbol{\theta})$ does not serve as a predefined baseline in RDQ, unlike in Dueling DQN where its output value is specifically the mean action-value. Instead, RDQ augments the DQN loss with an additional ℓ_2-penalty, $\beta(\frac{1}{2}B(s;\boldsymbol{\theta})^2 + \frac{1}{2}\sum_a Z(s,a;\boldsymbol{\theta})^2)$, where β is a regularization coefficient.

Both RDQ and Dueling DQN learn an explicit baseline parameter and represent it with an additional stream of learnable parameters. Dueling DQN, like the ME layer, is a mere architectural modification to DQN with a predefined baseline. RDQ, by contrast, neither uses a predefined baseline nor constitutes a purely architectural change, as it augments the DQN loss function.

Advantage Updating (Baird, 1993) serves as a conceptual predecessor to the methods mentioned here. It is a

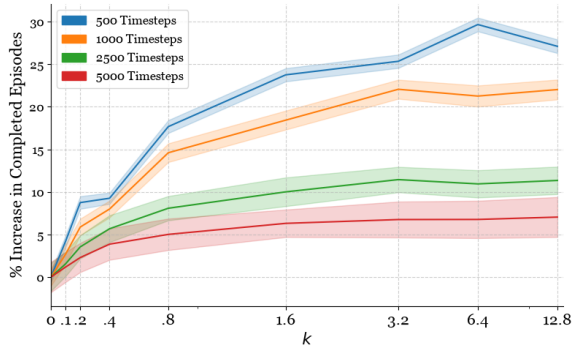


Figure 2. **Gridworld results.** We compare IBQ(k) under different k, including k=0, which is Q-learning. We report the percentage increase in episode completions across four sample complexity regimes. Shaded regions correspond to a 95% confidence interval. For most k, IBQ(k) can complete over 20% more episodes than Q-learning within 1k timesteps. Both algorithms quickly master the task and their gap decreases with more timesteps.

classical algorithm that decomposes a value function into state-values and action-advantages, where the state-value is shared by actions in a state. Its successor, Advantage Learning (Baird, 1999), avoids learning an explicit state-value function. These methods emphasize stable learning through gradient descent with function approximation rather than accelerating learning through value-sharing.

Baseline-residual decompositions are broadly related to centering and normalization, which have been explored in RL (Naik et al., 2024). Pop-Art (van Hasselt et al., 2016b) is a method that adaptively normalizes targets to be robust to different value scales. Sun et al. (2022) explore how shifting rewards can improve curiosity-driven exploration. More generally, baselines are commonly used in RL for variance reduction and stability, particularly in policy gradient methods (Williams, 1992; Chung et al., 2021).

6. Experiments

In this section, we evaluate our implicit-baseline Q-learning methods in a gridworld setting as well as in 57 Atari games. In particular, we examine their sample efficiency, performance, sensitivity to k, overestimation, and action gaps.

6.1. Gridworld Experiment

We first examine IBQ, the tabular algorithm which combines the mean-expansion layer with tabular Q-learning (Equations 8 and 9). To highlight its sample efficiency, we examine it in a pedagogical 5×5 stochastic gridworld task. Episodes begin in the bottom-left corner and terminate upon reaching the goal state in the top-right corner. The discount rate is 0.95. The agent receives a reward of 5 for reaching the goal and 0 otherwise. The agent's actions are the four cardinal directions. The agent transitions according to its chosen action with probability $3/4$ and according to a different randomly selected action with probability $1/4$.

We evaluate IBQ at different k. For each k, we report the best results across a sweep over 61 step sizes chosen from a logarithmic search over $(0, 1]$, running each combination for 128 seeds. All agents employ an ϵ-greedy policy with ϵ=0.1 for 5k timesteps. Figure 2 shows, for different values of k, the percentage increase in the number of completed episodes over Q-learning, across four different sample complexity regimes. The shaded regions correspond to a 95% confidence interval.

IBQ consistently learns faster than Q-learning. By distributing credit to all four actions rather than one, value propagates more quickly. The gap between Q-learning and IBQ decreases, however, with more experience, as both algorithms equally master the task. We also observed that when k is large, smaller step sizes are more suitable, given the larger scaling induced by k. Conversely, when k is small, larger step sizes are more suitable.

6.2. Deep RL Empirical Methodology

Setting. We evaluate all of our agents in the Arcade Learning Environment (ALE) (Bellemare et al., 2013) on the 57 standard Atari environments used in the literature (van Hasselt et al., 2016a). We follow the evaluation practices proposed by Machado et al. (2018). Specifically, we train and evaluate agents with sticky actions with probability 0.25, game-over termination signals, and the full action set.

Code and baselines. Our algorithms and baselines[1] are implemented based on the PFRL library (Fujita et al., 2021) and its reproduction of DQN. Our DQN implementation follows modern best practices, including the use of the Adam optimizer (Kingma & Ba, 2015) and the squared-error loss, using the optimizer settings of Hessel et al. (2018), as has become standard practice (Ceron & Castro, 2021). IB-DQN(k) refers to DQN which adds a final mean-expansion layer with mean-scaling coefficient k. For IB-DQN, we do not modify the underlying DQN algorithm or its hyperparameters. We compare against the most related value-sharing method, Dueling DQN. We also scale up RDQ, which was originally evaluated on MinAtar (Young & Tian, 2019), to the full ALE. For RDQ, we set $\beta = 0.001$, matching the original paper, after testing several values on a subset of games. For implicit quantile networks (IQN; Dabney et al., 2018), we largely match the training settings of the original paper. IB-IQN refers to IQN where an ME layer is added at the end of the network. Appendix E contains more training and evaluation details for reproducibility.

[1] github.com/prabhatnagarajan/me_layer.

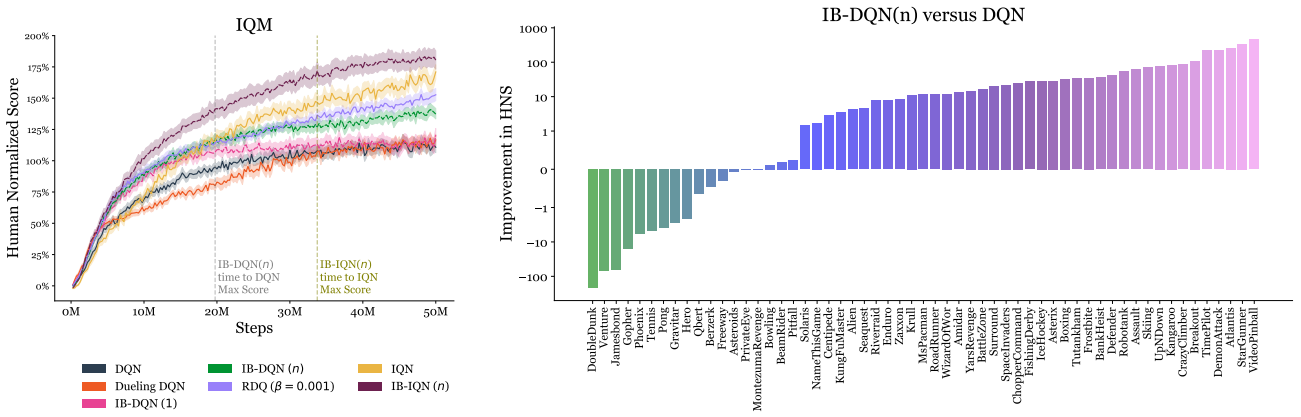


Figure 3. (*left*) Different algorithms and the interquartile mean of their human-normalized score across 57 games. All algorithms were run for five seeds per game. The shaded region depicts the 95% stratified bootstrap confidence interval (Agarwal et al., 2021). Dashed lines indicate the use of the mean-expansion layer. (*right*) The increase in human-normalized score, measured as the average area-under-the-curve, when switching from DQN to IB-DQN(k=n). Note that the y-axis is log-scale.

6.3. Atari-57 Performance

Figure 3 (*left*) depicts the performance in terms of the interquartile mean (IQM) (Agarwal et al., 2021) of the human-normalized score (Mnih et al., 2015) of DQN, Dueling DQN, IB-DQN, RDQ, IQN, and IB-IQN. All algorithms were run for five seeds per game. Figure 3 (*right*) shows the change in human-normalized score when switching from DQN to IB-DQN. Figure 7 in Appendix G shows the per-game learning curves. Additionally, Table 2 in Appendix G reports the mean score over the last three evaluations for each agent in each environment, across all seeds.

IB-DQN(n) clearly performs better than both DQN and Dueling DQN, with much better sample efficiency. The first vertical dotted line shows the first timestep at which the IQM of IB-DQN(n) exceeds DQN's highest score across its curve. IB-DQN(n) is able to achieve DQN's best performance at just under 20M timesteps, that is, within 40% of the total training time, exhibiting improved sample efficiency. IB-IQN(n), that is, IQN with the ME layer, also exhibits faster learning and performance over IQN. It achieves IQN's highest score across its curve at 33.75M timesteps, at just over $2/3$ of its total training time. To test the impact of the ME layer under a different optimizer, we evaluate the ME layer on DQN with RMSprop (Tieleman, 2012) and the Huber loss in Figure 6 of Appendix F. The ME layer accelerates learning and improves performance in this setting too, reinforcing our core findings.

When $k = 1$, which represents the conservative ME layer with a low condition number, we see faster initial learning and a modest improvement through training. By scaling the all-ones component of a vector, the gradients along the all-ones component are also scaled. This gradient scaling accelerates learning of a common baseline value for all entries of the vector. When $k = 1$, the mean action-value in a state is updated less aggressively than for larger values of k, yet this appears sufficient to accelerate initial learning. Overall, these results demonstrate that the ME layer can be an effective addition to a standard deep Q-network.

Scaling RDQ demonstrates it to be an effective algorithm. Unlike Dueling DQN, RDQ is able to effectively leverage its additional parameters through explicit regularization. RDQ outperforms IB-DQN(n), though the latter is competitive. This is to be expected; RDQ has an extra stream of parameters and an augmented loss. The primary appeal of IB-DQN(n) is its simplicity as a simple parameter-free layer that can be added to Q-networks which boosts performance through an implicit baseline.

Dueling DQN does not outperform DQN, contrary to the original results of Wang et al. (2016). It has been shown that modern implementations of DQN that use Adam and the squared-error loss perform on par with distributional variants (Agarwal et al., 2020), which have been known to outperform Dueling Double DQN with prioritized replay (Bellemare et al., 2017). Though it is difficult to draw definitive conclusions without extensive tuning, it may be the case that Dueling DQN's benefits are better realized in older implementations of DQN which did not use Adam or the squared-error loss.

6.4. Value Function Stability: Overestimation and Action Gaps

Aside from performance and sample efficiency, there are other metrics related to value-function dynamics that are known to be correlated with stability and performance in deep RL. We examined two such metrics, overestimation and action gaps. Overestimation refers to the action-value

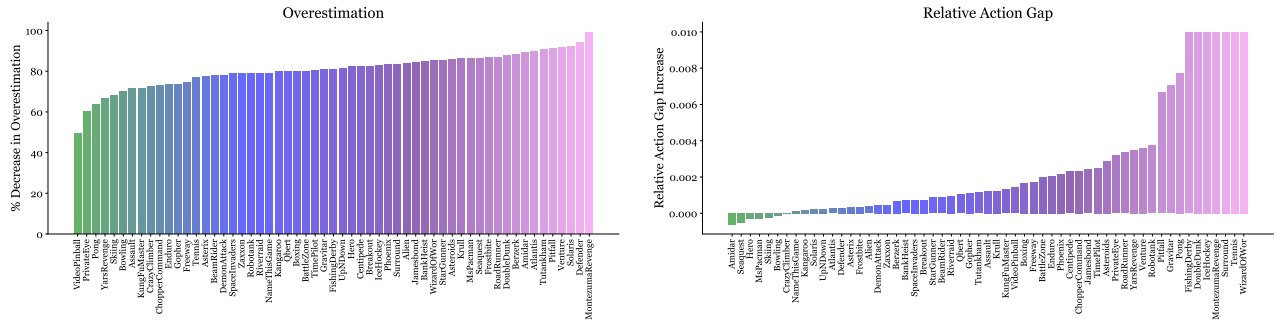


Figure 4. (*left*) The percentage *reduction* in overestimation when using IB-DQN over DQN, as a percentage of DQN's average overestimation area under the curve. In all games, IB-DQN reduces overestimation over DQN. (*right*) The *increase* in relative action gap from using IB-DQN instead of DQN. In the vast majority of games, IB-DQN increases the relative action gap compared to DQN. Values are clipped at 0.01 for visibility.

prediction exceeding the true expected return (Nagarajan et al., 2026). Reducing overestimation has historically been correlated with improved stability and performance in deep RL (van Hasselt et al., 2016a; Fujimoto et al., 2018).

Figure 4 (*left*) shows the percentage reduction in value overestimation when using IB-DQN(n) over DQN, measured as a percentage of DQN's overestimation. Overestimation is measured by comparing predicted returns to achieved returns during evaluation phases (see Appendix E). As DQN overestimates in all games (see Figure 8 in Appendix G), a reduction exceeding 100% indicates underestimation, while a reduction between 0% and 100% represents a reduction in, but not elimination of, overestimation. IB-DQN exhibits reduced overestimation in all games.

In standard DQN, a positive TD error increases the action-value of one action and indirectly modifies the other action-values via changes to hidden layers of the network. In IB-DQN, the updates are explicit such that a positive TD error increases the values of all actions and a negative TD error decreases the value of all actions. A TD error for one action is distributed across actions, perhaps tempering the outlier increases in the maximum action-value used for bootstrapping. This sensitivity to global changes in value and the implicit penalty on increasing the mean of the residuals (Equation 4) may be why we observe reduced overestimation. The full curves depicting overestimation throughout training are provided in Figure 8 in Appendix G. These curves also depict IB-DQN(1)'s overestimation, which is less than DQN's but more than IB-DQN(n)'s.

Note that IB-DQN should be complementary to other methods for overestimation reduction, such as Double DQN (van Hasselt et al., 2016a). Double DQN uses both the Q-network and target network to produce bootstrap targets. The use of separate networks in the bootstrap target mitigates the maximization bias. This use of separate networks is still present when the ME layer is added to Double DQN. Thus, the two methods should be complementary.

The second metric we measured was the action gap. The *action gap* (Farahmand, 2011) refers to the difference between the maximum action-value and the second-highest action-value. In deep RL, gaps between action-values in a state are often quite small. For example, Wang et al. (2016) report that the average action-value across states of a Double DQN agent trained on the game Seaquest was 15. However, the average action gap was a mere 0.04. A consequence of small action gaps is that minor changes to action-values can rapidly change the greedy actions. For these reasons, increasing the action gap is known to be generally beneficial in deep RL (Bellemare et al., 2016).

Figure 4 (*right*) shows the increase in the relative action gap when transitioning from DQN to IB-DQN(n). The relative action gap refers to the action gap divided by the absolute value of the average Q-value. We measure the relative action gap because IB-DQN produces much smaller action-values than DQN, in line with our overestimation results. In a large majority of environments, IB-DQN exhibits an increased relative action gap. This increase in relative action gap may be because our learned representations are themselves residuals. Consequently, the majority of the model's capacity need not be devoted to fitting a large baseline value common to all actions, and can instead be used to model the relative differences between action-values.

6.5. Sensitivity Analysis of Mean-Scaling Coefficient

To examine the sensitivity of IB-DQN to its mean-scaling coefficient k, we ran it for several values of k on three Atari 2600 games (Figure 5 (*left*)) for five seeds each and on Gymnasium's (Towers et al., 2024) LunarLander-v3 environment (Figure 5 (*center*)) for 120 seeds each. All environments share similar qualitative results. For some values of k, performance exceeds that of DQN, which is equivalent to $k = 0$. Then, as k grows too large, performance drops dramatically, likely due to gradients being scaled too much along the all-ones direction. Figure 5 (*right*) depicts the correspond-

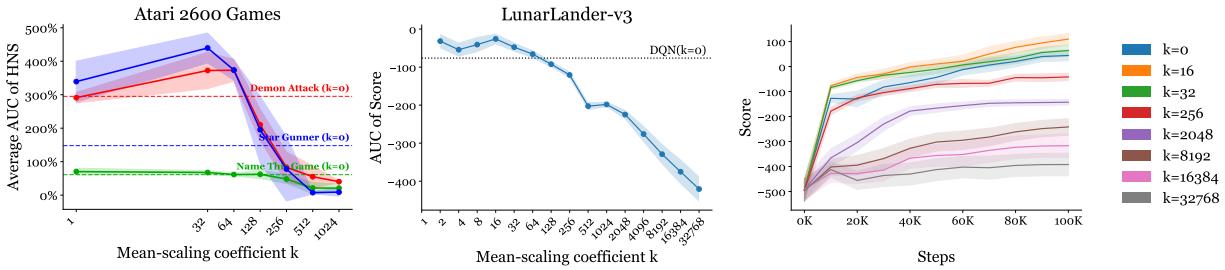


Figure 5. **Sensitivity Analysis of k.** (*left*) The average area under the curve (AUC) of the human-normalized score for several values of k (log scale) on Atari 2600 games, where the shaded region depicts the standard deviation across five seeds. (*center*) Similar to the *left* figure, but for LunarLander-v3, where the shaded region depicts a 95% confidence interval for the mean across 120 seeds. (*right*) The corresponding LunarLander learning curves for a subset of values of k, again with the shaded region depicting a 95% confidence interval.

ing LunarLander-v3 learning curves for several values of k, where we can visibly see the degradation as k grows large.

In this sensitivity analysis, we use the same step size for all k, and our results should be interpreted accordingly. In principle, we should tune the step size as we modify k, as done in the tabular experiments. Smaller step sizes should be more suitable for larger values of k. Nonetheless, the general result is that for any fixed step size, there is some large value of k beyond which performance degrades.

7. Conclusion and Discussion

In this paper, we presented the ***mean-expansion layer*** (ME layer), a simple way to represent and share values across actions. The ME layer allows us to avoid learning an explicit baseline parameter and instead learn a vector of residuals. This layer introduces no extra learnable parameters and does not change the underlying algorithm. Applying this layer at the end of a Q-network produces reliable improvements to DQN and IQN in terms of sample efficiency and performance in aggregate across 57 Atari 2600 games. It also provides large reductions in overestimation and increases the action gap, which are generally desirable in deep RL.

The ME layer applied to RL has its limitations. The first is that performance degrades as k grows large. Therefore, selecting $k = n$, where n is the number of actions, in large action spaces may lead to instabilities. This limitation is not specific to the ME layer; large action spaces generally create more difficult learning problems, including for Q-learning methods. Fortunately, selecting $k = 1$ is well-conditioned and implies a baseline of $\mu_q/2$, regardless of the number of actions. As such, setting $k = 1$ can be better than the implicit baseline of 0 in standard Q-learning, as supported by our empirical results.

A second limitation is that there is a discrepancy between the conceptual framing of the ME layer and its practical usage. While the ME layer is used to produce vector-valued outputs of action-values, feedback is in the form of sample-based updates to individual state-action pairs. This discrepancy is also present in Dueling DQN, where updates to individual state-action pairs update the baseline $B(s)$, which represents the mean action-value in the state. The updates occur according to the behavior policy, which may sample actions non-uniformly. Nonetheless, the mean is updated without accounting for variation in action-selection. Examining the impact of this discrepancy is left for future work.

Our study of the ME layer provides a novel interpretation of Dueling DQN. Dueling DQN explicitly learns separate baseline and action components. To do so, it maintains independent streams of parameters for each. Our work suggests that learning a state-specific baseline may be the key ingredient and that neither explicit separation nor additional parameters are strictly necessary to achieve this. Instead, each action's output can carry both individual and shared value effectively.

The ME layer can be viewed as a mechanism to generalize values across actions. In standard DQN, if different actions in a Q-network share active parameters, then some amount of generalization across actions is inevitable. The ME layer is one way to generalize across actions, in which inputs are collectively used to construct the output Q-values. Generalization can be beneficial but can also introduce approximation errors. The investigation of these tradeoffs as they pertain to the ME layer is left for future work.

Several questions regarding the ME layer remain open. We lack a comprehensive understanding of how the ME layer impacts the value function's learning path, how and when it improves performance, and how it affects convergence guarantees. Its generalization to continuous actions remains open, and we lack a mechanistic explanation for its impact on overestimation and action gaps. Despite these gaps in our understanding, we have established several fundamental aspects of the ME layer that can lay the foundation for answering these questions. It remains the case that the ME layer is an effective addition to Q-networks that can accelerate learning and improve performance.

Acknowledgements

We thank the anonymous reviewers, as well as Alex Lewandowski, Diego Gomez Noriega, Esraa Elelimy, and Roshan Shariff for their helpful feedback which improved the paper and its presentation.

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada CIFAR AI Chair Program, and the Alberta Machine Intelligence Institute (Amii). Prabhat Nagarajan is supported by the Alberta Innovates Graduate Student Scholarship. Computational resources were provided in part by the Digital Research Alliance of Canada.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Agarwal, R., Schuurmans, D., and Norouzi, M. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2020.

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

Aitchison, M., Sweetser, P., and Hutter, M. Atari-5: Distilling the arcade learning environment down to five games. In *International Conference on Machine Learning (ICML)*, 2023.

Baird, L. C. Advantage updating (technical report wl-tr-93-1146). *Wright Laboratory*, 1993.

Baird, L. C. *Reinforcement learning through gradient descent*. PhD thesis, Carnegie Mellon University, 1999.

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 (JAIR)*, 2013.

Bellemare, M. G., Ostrovski, G., Guez, A., Thomas, P., and Munos, R. Increasing the action gap: New operators for reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2017.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Katariya, Y., 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/jax-ml/jax.

Ceron, J. S. O. and Castro, P. S. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *International Conference on Machine Learning (ICML)*, 2021.

Chung, W., Thomas, V., Machado, M. C., and Le Roux, N. Beyond variance reduction: Understanding the true impact of baselines on policy optimization. In *International Conference on Machine Learning (ICML)*, 2021.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.

Daley, B., Nagarajan, P., White, M., and Machado, M. C. An analysis of action-value temporal-difference methods that learn state values. *Reinforcement Learning Journal (RLJ)*, 2025.

Farahmand, A.-M. Action-gap phenomenon in reinforcement learning. *Neural Information Processing Systems (NeurIPS)*, 2011.

Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, 2018.

Fujita, Y., Nagarajan, P., Kataoka, T., and Ishikawa, T. ChainerRL: A deep reinforcement learning library. *Journal of Machine Learning Research (JMLR)*, 2021.

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 *AAAI Conference on Artificial intelligence (AAAI)*, 2018.

Householder, A. S. A theory of steady-state activity in nerve-fiber networks: I. definitions and preliminary lemmas. *The Bulletin of Mathematical Biophysics*, 1941.

John, G. H. When the best move isn't optimal: Q-learning with exploration. In *AAAI Conference on Artificial Intelligence (AAAI)*, 1994.

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

Lin, L. *Reinforcement learning for robots using neural networks*. PhD thesis, Carnegie Mellon University, 1992.

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 (JAIR)*, 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*, 2015.

Nagarajan, P., White, M., and Machado, M. C. Deep Double Q-learning. *arXiv preprint arXiv:2507.00275*, 2026.

Naik, A., Wan, Y., Tomar, M., and Sutton, R. S. Reward centering. *Reinforcement Learning Journal (RLJ)*, 2024.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Quan, J. and Ostrovski, G. DQN Zoo: Reference implementations of DQN-based agents, 2020. URL http://github.com/deepmind/dqn_zoo.

Rummery, G. A. and Niranjan, M. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

Sun, H., Han, L., Yang, R., Ma, X., Guo, J., and Zhou, B. Exploit reward shifting in value-based deep-RL: Optimistic curiosity-based exploration and conservative exploitation via linear reward shaping. In *Neural Information Processing Systems (NeurIPS)*, 2022.

Tang, Y., Munos, R., Rowland, M., and Valko, M. VA-learning as a more efficient alternative to Q-learning. In *International Conference on Machine Learning (ICML)*. PMLR, 2023.

Tieleman, T. Lecture 6e-rmsprop: Divide the gradient by a running average of its recent magnitude. *Coursera: Neural Networks for Machine Learning*, pp. 26–30, 2012.

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double Q-learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016a.

van Hasselt, H. P., Guez, A., Hessel, M., Mnih, V., and Silver, D. Learning values across many orders of magnitude. *Neural Information Processing Systems (NeurIPS)*, 2016b.

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 (ICML)*, 2016.

Watkins, C. J. C. H. *Learning from delayed rewards*. PhD thesis, King's College, University of Cambridge, 1989.

Watkins, C. J. C. H. and Dayan, P. Q-learning. *Machine Learning*, 1992.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

Young, K. and Tian, T. Minatar: An Atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

A. Theoretical Results

A.1. Baseline Analysis

Proposition 1. *(Norm-minimizing baseline). Given a vector $\boldsymbol{q} \in \mathbb{R}^n$, the norm-minimizing baseline is $b^\star = \sum_{i=1}^n q_i/(n+1)$. That is,*

$$b^\star = \operatorname*{argmin}_b \|\mathbf{u}(\boldsymbol{q}, b)\|_2^2 = \frac{\sum_{i=1}^n q_i}{n+1}. \tag{2}$$

Proof. Let $f(b)$ be the squared Euclidean norm of $\boldsymbol{u}(\boldsymbol{q}, b)$.

$$f(b) = \|\boldsymbol{u}(\boldsymbol{q}, b)\|_2^2 = \sum_{i=1}^n (q_i - b)^2 + b^2.$$

This function is convex, so we can apply the first derivative test to find its minimum.

$$\frac{df}{db} = 2b - 2\sum_{i=1}^n (q_i - b).$$

Setting the derivative to zero, we obtain

$$\begin{aligned} 0 &= 2b - 2\sum_{i=1}^n (q_i - b) \\ 0 &= b - \sum_{i=1}^n q_i + nb \\ \sum_{i=1}^n q_i &= (n+1)b \\ b &= \frac{\sum_{i=1}^n q_i}{n+1}. \end{aligned}$$

□

Proposition 4. *(Norm-reducing baselines). Let $\mu_{\boldsymbol{q}} = \sum_{i=1}^n q_i/n$. If $\mu_{\boldsymbol{q}} < 0$, then the strictly norm-reducing baselines b are the ones that satisfy $2b^\star < b < 0$. If $\mu_{\boldsymbol{q}} > 0$, the strictly norm-reducing baselines b satisfy $0 < b < 2b^\star$.*

Proof. Let $f(b)$ be the squared Euclidean norm of $\boldsymbol{u}(\boldsymbol{q}, b)$.

$$f(b) = \|\boldsymbol{u}(\boldsymbol{q}, b)\|_2^2 = \sum_{i=1}^n (q_i - b)^2 + b^2.$$

Expanding,

$$\begin{aligned} f(b) &= \sum_{i=1}^n \left(q_i^2 + b^2 - 2q_i b\right) + b^2 \\ &= \sum_{i=1}^n q_i^2 - 2b\sum_{k=1}^n q_k + (n+1)b^2. \end{aligned}$$

Using $\sum_{i=1}^n q_i = n\mu_{\boldsymbol{q}}$, we obtain

$$f(b) = \|\boldsymbol{q}\|_2^2 - 2bn\mu_{\boldsymbol{q}} + (n+1)b^2.$$

Thus the norm of $\boldsymbol{u}(\boldsymbol{q}, b)$ is less than the norm of $\boldsymbol{q}$ when

$$\begin{aligned} -2bn\mu_{\boldsymbol{q}} + (n+1)b^2 &< 0 \\ (n+1)b^2 &< 2bn\mu_{\boldsymbol{q}} \\ b^2 &< 2bb^\star \\ b(b - 2b^\star) &< 0. \end{aligned}$$

This inequality holds if b and $(b - 2b^\star)$ have opposite signs.

If $\mu_{\boldsymbol{q}} < 0$, then $b^\star < 0$. Conversely, if $\mu_{\boldsymbol{q}} > 0$, then $b^\star > 0$. Thus, if $b^\star < 0$, then the norm is reduced for $2b^\star < b < 0$. If $b^\star > 0$, then the norm is reduced for $0 < b < 2b^\star$. This implies that if $\mu_{\boldsymbol{q}} \neq 0$, then $\|\boldsymbol{u}(\boldsymbol{q}, b)\|_2^2$ is less than $\|\boldsymbol{q}\|_2^2$. □

Proposition 5. *Let $\boldsymbol{q} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Assume $\mu_{\boldsymbol{q}} \neq 0$. The absolute value of the mean of the residual vector, $g_{\boldsymbol{q}}(b) = |\frac{1}{n}\sum_{i=1}^n (q_i - b)| = |\mu_{\boldsymbol{q}} - b| = |\mu_{\boldsymbol{z}}|$, decreases from $b = 0$ to $b = \mu_{\boldsymbol{q}}$, regardless of the sign of $\mu_{\boldsymbol{q}}$.*

Proof.

$$g_{\boldsymbol{q}}(b) = \begin{cases} \mu_{\boldsymbol{q}} - b & b \leq \mu_{\boldsymbol{q}}, \\ b - \mu_{\boldsymbol{q}} & b > \mu_{\boldsymbol{q}}. \end{cases}$$

So, if $b < \mu_{\boldsymbol{q}}$, $\frac{\mathrm{d}g_{\boldsymbol{q}}}{\mathrm{d}b} = -1$, and if $b > \mu_{\boldsymbol{q}}$, $\frac{\mathrm{d}g_{\boldsymbol{q}}}{\mathrm{d}b} = 1$. Thus, if $\mu_{\boldsymbol{q}} > 0$, then on the interval $b \in (0, \mu_{\boldsymbol{q}})$, $g_{\boldsymbol{q}}(b)$ is decreasing. If $\mu_{\boldsymbol{q}} < 0$ then on the interval $b \in (\mu_{\boldsymbol{q}}, 0)$ $g_{\boldsymbol{q}}(b)$ is increasing. Thus, $g_{\boldsymbol{q}}(b)$ is decreasing from $b = 0$ to $b = \mu_{\boldsymbol{q}}$. □

Proposition 6. *Let $\boldsymbol{q} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Assume $\mu_{\boldsymbol{q}} \neq 0$. The squared Euclidean norm of the residual vector $\|\boldsymbol{q} - b\mathbf{1}\|_2^2$ has decreasing magnitude from $b = 0$ to $b = \mu_{\boldsymbol{q}}$, regardless of the sign of $\mu_{\boldsymbol{q}}$.*

Proof. Define $f_{\boldsymbol{q}}(b) = \|\boldsymbol{q} - b\mathbf{1}\|_2^2 = \sum_{i=1}^n (q_i - b)^2$.

To understand whether $f_{\boldsymbol{q}}(b)$ is increasing or decreasing as a function of b, we examine its derivative:

$$\frac{df_{\boldsymbol{q}}}{db} = -2\sum_{i=1}^n (q_i - b) = 2\sum_{i=1}^n (b - q_i) = 2n(b - \mu_{\boldsymbol{q}}).$$

Thus, $f_{\boldsymbol{q}}(b)$ is decreasing when

$$2n(b - \mu_{\boldsymbol{q}}) < 0 \quad \Longleftrightarrow \quad b < \mu_{\boldsymbol{q}},$$

and increasing when $b > \mu_{\boldsymbol{q}}$. The norm of the residual vector is minimized at $b = \mu_{\boldsymbol{q}}$, a known fact. We can examine the two cases. In the first case, if $\mu_{\boldsymbol{q}} < 0$, then

for $b \in (\mu_{\boldsymbol{q}}, 0)$ we have $b > \mu_{\boldsymbol{q}}$, so $f_{\boldsymbol{q}}(b)$ is increasing on this interval and hence decreases from $b = 0$ to $\mu_{\boldsymbol{q}}$. In the second case, if $\mu_{\boldsymbol{q}} > 0$, then for $b \in [0, \mu_{\boldsymbol{q}})$ we have $b < \mu_{\boldsymbol{q}}$, so $f_{\boldsymbol{q}}(b)$ is decreasing on this interval.

Thus, from $b = 0$ to $b = \mu_{\boldsymbol{q}}$, the norm of the residual vector decreases regardless of the sign of $\mu_{\boldsymbol{q}}$. □

A.2. Analysis of the Mean-Expansion Layer

Proposition 3. *Let $\mu_{\boldsymbol{z}} = \frac{1}{n}\sum_{i=1}^{n} z_i$. Let $b = k\mu_{\boldsymbol{z}}$, where $b\mathbf{1} = \frac{k}{n}\boldsymbol{J}\boldsymbol{z}$. Then if $k > 0$,*

$$b = k\mu_{\boldsymbol{z}} = \frac{\sum_{i=1}^{n} q_i}{n + \frac{n}{k}}. \tag{7}$$

Proof. Let $S_{\boldsymbol{q}} = \sum_{i=1}^{n} q_i$ and let $S_{\boldsymbol{z}} = \sum_{i=1}^{n} z_i$. As each q_i is constructed with $q_i = z_i + \frac{k}{n}\sum_{j=1}^{n} z_j$, then for $k \geq 0$, we can sum all Q-values:

$$\begin{aligned}
\sum_{i=1}^{n} q_i &= \sum_{i=1}^{n}\left(z_i + \frac{k}{n}\sum_{j=1}^{n} z_j\right) \\
S_{\boldsymbol{q}} &= S_{\boldsymbol{z}} + kS_{\boldsymbol{z}} \\
S_{\boldsymbol{q}} &= (k+1)S_{\boldsymbol{z}} \\
\frac{S_{\boldsymbol{q}}}{k+1} &= S_{\boldsymbol{z}} \\
\frac{S_{\boldsymbol{q}}}{\frac{n}{k}(k+1)} &= \frac{k}{n}S_{\boldsymbol{z}} \\
k\mu_{\boldsymbol{z}} &= \frac{S_{\boldsymbol{q}}}{n + \frac{n}{k}}.
\end{aligned}$$

□

A.3. Properties of the Mean-Expansion Layer

In Equation 6, we described the geometric properties of the mean-expansion layer as stretching the mean component of the vector. This framing can be seen below.

$$\begin{aligned}
\boldsymbol{q} &= \left(\boldsymbol{I} + \frac{k}{n}\boldsymbol{J}\right)\boldsymbol{z} \\
&= \left(\boldsymbol{I} + \frac{k}{n}\boldsymbol{J} + \frac{1}{n}\boldsymbol{J} - \frac{1}{n}\boldsymbol{J}\right)\boldsymbol{z} \\
&= \left(\frac{k}{n}\boldsymbol{J}\boldsymbol{z} + \frac{1}{n}\boldsymbol{J}\boldsymbol{z}\right) + \left(\boldsymbol{I} - \frac{1}{n}\boldsymbol{J}\right)\boldsymbol{z} \\
&= (k+1)\left(\frac{1}{n}\boldsymbol{J}\boldsymbol{z}\right) + \left(\boldsymbol{z} - \frac{1}{n}\boldsymbol{J}\boldsymbol{z}\right).
\end{aligned}$$

The first term stretches the mean and the second term subtracts the mean component from the original vector.

Proposition 2. *Given vector $\boldsymbol{q} \in \mathbb{R}^n$, the unique minimizer of Equation 4 is the solution to the system of equations:*

$$\boldsymbol{q} = \left(\boldsymbol{I} + \frac{k}{n}\boldsymbol{J}\right)\boldsymbol{z}. \tag{5}$$

Proof. Let

$$\begin{aligned}
f(\boldsymbol{z}) &= \frac{1}{n}\|\boldsymbol{z} - \boldsymbol{q}\|_2^2 + k\left(\frac{1}{n}\sum_{i=1}^{n} z_i\right)^2 \\
&= \frac{1}{n}(\boldsymbol{z} - \boldsymbol{q})^\top(\boldsymbol{z} - \boldsymbol{q}) + k\left(\frac{1}{n}\sum_{i=1}^{n} z_i\right)^2 \\
&= \frac{1}{n}(\boldsymbol{z} - \boldsymbol{q})^\top(\boldsymbol{z} - \boldsymbol{q}) + k\left(\frac{1}{n}\mathbf{1}^\top\boldsymbol{z}\right)^2 \\
&= \frac{1}{n}(\boldsymbol{z} - \boldsymbol{q})^\top(\boldsymbol{z} - \boldsymbol{q}) + \frac{k}{n^2}\boldsymbol{z}^\top\mathbf{1}\mathbf{1}^\top\boldsymbol{z} \\
&= \frac{1}{n}(\boldsymbol{z} - \boldsymbol{q})^\top(\boldsymbol{z} - \boldsymbol{q}) + \frac{k}{n^2}\boldsymbol{z}^\top\boldsymbol{J}\boldsymbol{z}.
\end{aligned}$$

We can then take the gradient with respect to $\boldsymbol{z}$,

$$\nabla_{\boldsymbol{z}} f(\boldsymbol{z}) = \frac{2}{n}(\boldsymbol{z} - \boldsymbol{q}) + \frac{2k}{n^2}\boldsymbol{J}\boldsymbol{z}.$$

Setting the gradient to 0 and dividing by 2, we get:

$$\begin{aligned}
0 &= \frac{1}{n}(\boldsymbol{z} - \boldsymbol{q}) + \frac{k}{n^2}\boldsymbol{J}\boldsymbol{z} \\
0 &= (\boldsymbol{z} - \boldsymbol{q}) + \frac{k}{n}\boldsymbol{J}\boldsymbol{z} \\
\boldsymbol{q} &= \boldsymbol{z} + \frac{k}{n}\boldsymbol{J}\boldsymbol{z} \\
\boldsymbol{q} &= \left(\boldsymbol{I} + \frac{k}{n}\boldsymbol{J}\right)\boldsymbol{z}.
\end{aligned}$$

□

B. Tabular Update Rules

To derive the update rules for our residuals, we do Q-learning on the implied Q-values. Let the Q-learning error at time t be $\delta_t = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)$. We then apply gradient descent to the Q-learning error and minimize $\mathcal{L} = \frac{1}{2}\delta_t^2$. For transition $(s_t, a_t, r_{t+1}, s_{t+1})$, the derivative with respect to $Z(s_t, a)$, for action a in state s_t is

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial Z(s_t, a)} &= \frac{\partial \mathcal{L}}{\partial Q(s_t, a_t)}\frac{\partial Q(s_t, a_t)}{\partial Z(s_t, a)} && (10) \\
&= (-\delta_t)\frac{\partial Q(s_t, a_t)}{\partial Z(s_t, a)} && (11) \\
&= (-\delta_t)\left(\mathbb{1}_{\{a=a_t\}} + \frac{k}{n}\right). && (12)
\end{aligned}$$

The interpretation, as shown in Equations 8 and 9, is that the chosen action is updated in greater proportion to the TD error than the other actions. That is, the update adds $\alpha_t(1 + k/n)\delta_t$ to the chosen action's residual and $\alpha_t(k/n)\delta_t$ to the residuals of the unchosen actions.

C. Semantics of the Dueling Network Architecture

In this Appendix, we discuss dueling methods more closely.

C.1. Dueling Deep Q-networks

Dueling DQN (Wang et al., 2016) in fact learns a mean-residual decomposition of action-values, under the terms "value" and "advantage" for the mean and residuals, respectively. Its modification to the DQN architecture is to learn $B(s;\boldsymbol{\theta})$ in a baseline stream and $Z(s,\cdot;\boldsymbol{\theta})$ in a residual stream. The baseline and residual streams are aggregated to construct the Q-values

$$Q(s,a;\boldsymbol{\theta}) = B(s;\boldsymbol{\theta}) + Z(s,a;\boldsymbol{\theta}), \tag{13}$$

or $Q(s,\cdot;\boldsymbol{\theta}) = B(s;\boldsymbol{\theta})\mathbf{1}+Z(s,\cdot;\boldsymbol{\theta})$. Prior to outputting the residual vector $Z(s,\cdot;\boldsymbol{\theta})$, the residual stream first produces raw outputs $X(s,a;\boldsymbol{\theta})$, which are then zero-centered to produce the action residuals $Z(s,\cdot;\boldsymbol{\theta})$:

$$Z(s,a;\boldsymbol{\theta}) = X(s,a;\boldsymbol{\theta}) - \frac{1}{n}\sum_{a'\in\mathcal{A}} X(s,a';\boldsymbol{\theta}). \tag{14}$$

Wang et al. (2016) find this architecture to be effective, providing a substantial improvement over DQN.

We can analyze the semantics of the dueling network architecture through its construction of action-values. First, note that due to the zero-centering in Equation 14, the sum of residuals is zero:

$$\begin{aligned}\sum_{a\in\mathcal{A}} Z(s,a;\boldsymbol{\theta}) &= \sum_{a\in\mathcal{A}}\Big(X(s,a;\boldsymbol{\theta}) - \frac{1}{n}\sum_{a'\in\mathcal{A}} X(s,a';\boldsymbol{\theta})\Big)\\ &= \sum_{a\in\mathcal{A}} X(s,a;\boldsymbol{\theta}) - \frac{n}{n}\sum_{a'\in\mathcal{A}} X(s,a';\boldsymbol{\theta})\\ &= 0.\end{aligned}$$

Thus, summing the Q-values, we get

$$\begin{aligned}\sum_{a\in\mathcal{A}} Q(s,a;\boldsymbol{\theta}) &= \sum_{a\in\mathcal{A}}\Big(B(s;\boldsymbol{\theta}) + Z(s,a;\boldsymbol{\theta})\Big)\\ &= nB(s;\boldsymbol{\theta}) + \sum_{a\in\mathcal{A}} Z(s,a;\boldsymbol{\theta})\\ &= nB(s;\boldsymbol{\theta}).\end{aligned}$$

Dividing by n:

$$\frac{1}{n}\sum_{a\in\mathcal{A}} Q(s,a;\boldsymbol{\theta}) = B(s;\boldsymbol{\theta}).$$

Thus, Dueling DQN enforces

$$\boxed{B(s;\boldsymbol{\theta}) = \frac{1}{n}\sum_{a\in\mathcal{A}} Q(s,a;\boldsymbol{\theta})},$$

that is, the output of the baseline stream $B(s;\boldsymbol{\theta})$ is the mean action-value for the state. As $Q(s,a;\boldsymbol{\theta}) = B(s;\boldsymbol{\theta}) + Z(s,a;\boldsymbol{\theta})$, each $Z(s,a;\boldsymbol{\theta})$ is a residual with respect to the mean action-value. Thus, Dueling DQN implements a *mean-residual decomposition*.

C.2. Regularized Dueling Q-learning

The recently proposed Regularized Dueling Q-learning (RDQ) (Daley et al., 2025) uses the Dueling DQN architecture and aggregates values according to Equation 13, but without the centering of the raw network outputs in Equation 14. In Dueling DQN, Equation 14 plays an important role in providing identifiability to the system in Equation 13 to ensure that the $n+1$ variables, that is, a single baseline and n residuals, are not unconstrained.

RDQ adopts a different approach to constrain values through explicit regularization of the network outputs. In particular, if the network outputs $B(s;\boldsymbol{\theta})$ and $Z(s,a;\boldsymbol{\theta})$ for all $a\in\mathcal{A}$, then RDQ adds the following penalty term to the DQN loss:

$$\beta\left(B(s;\boldsymbol{\theta})^2 + \sum_{a\in\mathcal{A}} Z(s,a;\boldsymbol{\theta})^2\right),$$

where β is a regularization coefficient. In effect, this method encourages the agent to learn the baseline-residual vector corresponding to the optimal baseline in Equation 2. Unlike IB-DQN or Dueling DQN, RDQ does not use the residuals and baseline to represent any specific predefined baseline. Dueling DQN and IB-DQN both structurally enforce their baseline terms to have specific relationships to the action-values. RDQ simply uses $B(s;\boldsymbol{\theta})$ generally as a baseline, and penalizes it to ensure a low-norm representation.

D. Mean-expansion Layer PyTorch and JAX Code

The ME layer can be implemented in a dozen lines in PyTorch (Paszke et al., 2019).

```
import torch

class MeanExpansionLayer(torch.nn.Module):
    def __init__(self, mean_scaling_coefficient):
        super().__init__()
        self.register_buffer('scale', torch.tensor(1 + mean_scaling_coefficient))

    def forward(self, vec):
        mean = vec.mean(dim=-1, keepdim=True)
        residual = vec - mean
        output = self.scale * mean + residual
        return output
```

It can similarly be implemented efficiently in JAX (Bradbury et al., 2018).

```
import jax.numpy as jnp
from jax import jit

@jit
def mean_expansion_layer(vec, mean_scaling_coefficient):
    scale = 1 + mean_scaling_coefficient
    mean = jnp.mean(vec, axis=-1, keepdims=True)
    residual = vec - mean
    return scale * mean + residual
```

E. Experimental Setup

In this Appendix, we discuss the training and evaluation details for our agents.

E.1. Environment and training settings

We train agents in the Arcade Learning Environment (Bellemare et al., 2013), using the environment protocol proposed by Machado et al. (2018). We use sticky actions, where the agent's most recently executed simulator action is repeated in the subsequent frame with probability 0.25. We expose the agents to the full action set of 18 actions in all games. The only termination signal the agent receives is when the game is over, corresponding to the end of an episode.

The agents are trained for 50M timesteps. These 50M timesteps are divided into 200 training phases, each lasting 250k timesteps. After each training phase is an evaluation phase lasting 125k timesteps. During evaluation, the agents deploy an ϵ-greedy policy with $\epsilon = 0.001$. During both training and evaluation, episodes that reach the 30-minute time limit, which corresponds to 27,000 timesteps, are truncated.

E.2. Algorithm training details and hyperparameters

The training details of DQN largely follow Mnih et al. (2015). We use their architecture, pre-processing schema, and other details unless otherwise mentioned. One distinction worth noting is that we have our agents deploy an ϵ-greedy policy during training that is annealed from $\epsilon = 1$ to $\epsilon = 0.01$ over the first 1M timesteps. Another distinction is that we use the Adam optimizer and the squared error loss as opposed to RMSprop with the Huber loss. Table 1 contains more implementation details and hyperparameters. For IQN, our training details largely follow Dabney et al. (2018), including their architecture and hyperparameters. We use Adam with step size 5×10^{-5} and $\epsilon = 3.125 \times 10^{-4}$, and other hyperparameters follow the original paper.

Table 1. DQN training hyperparameters. We use $\star$ to indicate hyperparameters that differ from the original DQN paper.

Hyperparameter	Value	Description
minibatch size	32	Sample batch size of gradient updates.
replay memory capacity	1,000,000	Number of recent transitions stored in the replay buffer.
agent history length	4	Number of previous frames stacked in state representation.
target network update frequency	2,500	Frequency (in terms of gradient updates) of target network updates.
discount rate	0.99	Value of γ.
action repeat	4	In one timestep, actions are repeated for multiple simulator frames.
update frequency	4	Frequency (in timesteps) of parameter updates.
replay start size	50k	Minimum number of transitions in the replay buffer required before parameter updates begin.
initial exploration	1.0	Initial value of ϵ used for ϵ-greedy exploration.
$\star$ final exploration	0.01	Final value of ϵ used for ϵ-greedy exploration.
final exploration timestep	1,000,000	The timesteps over which ϵ is linearly annealed to its final ϵ.
maximum episode length	27,000	Timesteps after which an episode is truncated and the environment is reset.
$\star$ step size	6.25×10^{-5}	The step size used by Adam.
$\star$ Adam ϵ	1.5×10^{-4}	The ϵ used by Adam.
$\star$ Adam β_1	0.9	β_1 hyperparameter value in Adam.
$\star$ Adam β_2	0.999	β_2 hyperparameter value in Adam.

E.3. Metrics

The *human-normalized score* (HNS) provides a mechanism for evaluating an agent's score in a manner that is comparable across games. The human-normalized score (Mnih et al., 2015) of an agent can be computed as

$$\text{score}_{\text{hns}} = \frac{\text{score}_{\text{agent}} - \text{score}_{\text{random}}}{\text{score}_{\text{human}} - \text{score}_{\text{random}}}. \tag{15}$$

The human and random scores are taken from Quan & Ostrovski (2020).

Measuring Overestimation In this paper, we follow Nagarajan et al. (2026)'s protocol for measuring overestimation. During evaluations, we deploy a near-greedy evaluation policy. We only consider completed evaluation episodes, which are episodes that terminate or truncate due to reaching the maximum allowed episode length of 27,000 timesteps. Any episodes that are truncated prematurely due to the evaluation phase ending are discarded. Each state-action pair in a completed episode has a received empirical return. For episodes truncated due to the time limit, we bootstrap the value of the final state to compute the empirical return. Overestimation for a state-action pair is computed as the difference between the action-value prediction and the received empirical return. We average these over all state-action pairs in all evaluation episodes across all evaluation phases in a training run across all seeds to produce a single scalar measurement of overestimation. We then report the difference in overestimation between the two algorithms.

DQN agents (on Atari) employ reward clipping, where rewards are clipped to the range $[-1, 1]$. Since agents are trained to predict return estimates under this reward, we ensure that this clipping is also applied to compute returns. Combined with discounting, clipping often causes returns to be much smaller than the raw game scores, which are undiscounted and unclipped.

Measuring the Action Gap To measure the action gap, we sample a minibatch from the buffer after each target network update and measure the difference between the highest and second-highest action-value averaged across all (pre-transition) states in the minibatch. These quantities are averaged across all measurements in a training phase. We also maintain a running average of the last 1000 average minibatch action-values for each training phase. The relative action gap for a training phase is computed as the average action gap divided by the absolute value of the average action-value with a stability term of 10^{-8} added to the denominator. These relative action gaps are averaged across all training phases and seeds. Figure 4 (*right*) reports the difference in the relative action gap between the two algorithms, clipped at 0.01 for presentation.

E.4. Baselines

For our Dueling DQN implementation, we use the exact same settings as DQN, but replace the network with the dueling network architecture. The final convolution layer outputs 64 feature maps of size 7×7, which are flattened into 3,136 outputs. Following these flattened outputs are two independent streams of two-layer fully connected networks. Each stream has a hidden layer of 512 units and ReLU activations (Householder, 1941). The first stream outputs a single scalar $B(s;\boldsymbol{\theta})$, and the second stream outputs an n-dimensional vector $\mathbf{x}(s;\boldsymbol{\theta})$, which is centered to output the residuals $\mathbf{z}(s;\boldsymbol{\theta}) = \mathbf{x}(s;\boldsymbol{\theta}) - \frac{1}{n}\boldsymbol{J}\mathbf{x}(s;\boldsymbol{\theta})$.

RDQ shares the same dueling network architecture, but outputs the residuals $\mathbf{z}(s;\boldsymbol{\theta})$ directly, as opposed to first centering the raw outputs. It also includes a regularization coefficient β on the penalty $B(s;\boldsymbol{\theta})^2 + \sum_a Z(s,a;\boldsymbol{\theta})^2$. We tested 5 different values of β on Atari-5-Val (Aitchison et al., 2023), a subset of environments used for validation. As long as β was not too large, we found RDQ to be relatively robust. We did not observe an improvement from changing β from 0.001, as was set by Daley et al. (2025), and thus kept this value.

F. Mean-Expansion Layer with RMSprop

To examine how the ME layer impacts performance in deep RL outside of the Adam optimizer, we also tested it with the RMSprop (Tieleman, 2012) optimizer and the Huber loss, with the optimizer settings matching those of the original DQN (Mnih et al., 2015). In particular, we compare the interquartile mean in performance between DQN and IB-DQN(n) across the standard 57 games over three seeds, shown in Figure 6. Consistent with our prior results, we find that IB-DQN(n) substantially outperforms DQN in this setting, providing further evidence for the general efficacy of the ME layer.

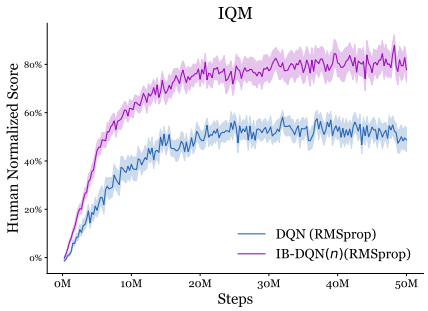
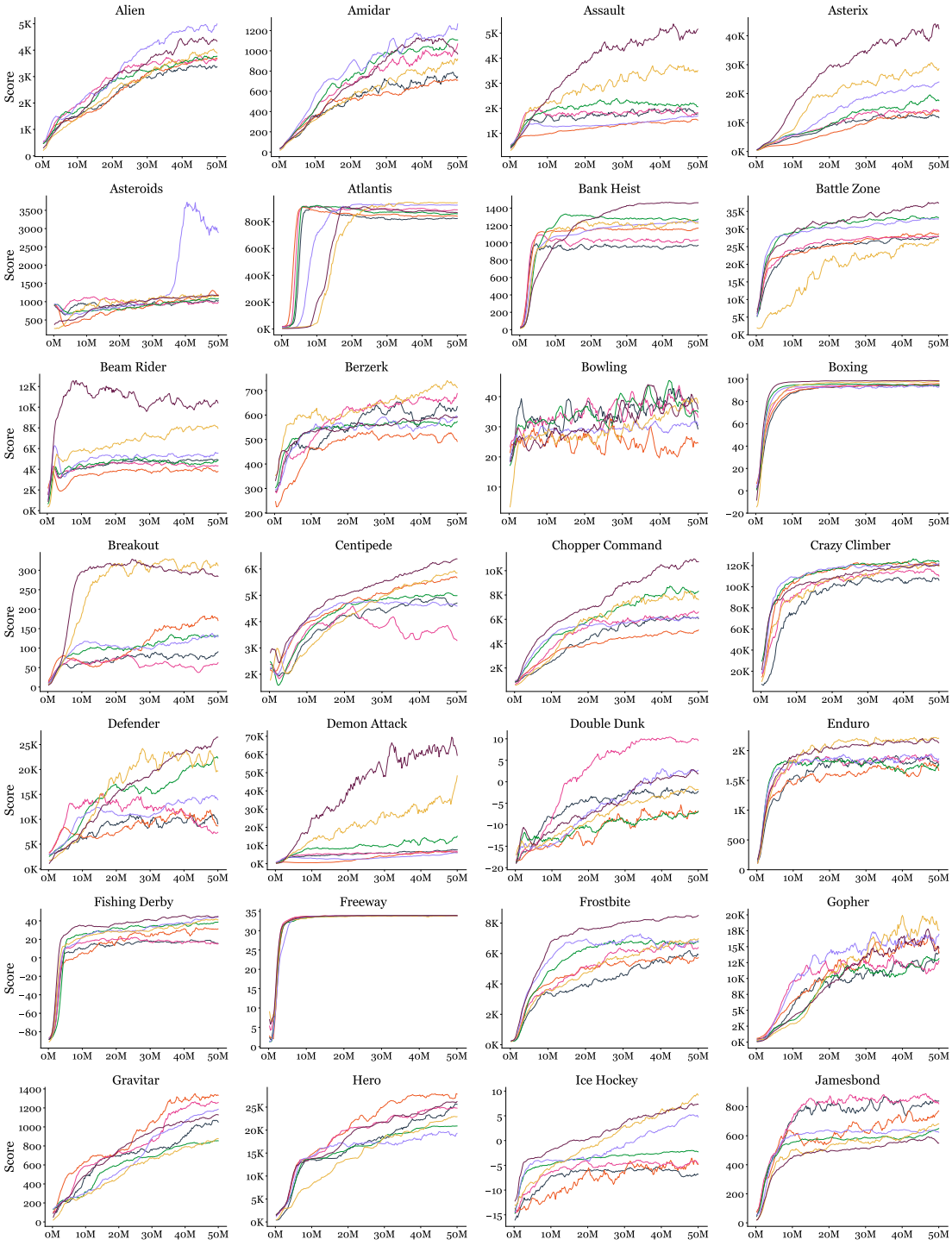


Figure 6. **The ME layer with RMSprop and the Huber loss.** The plot shows the interquartile mean of DQN with and without the ME layer across 57 games. All algorithms were run for three seeds per game. The shaded region depicts the 95% stratified bootstrap confidence interval (Agarwal et al., 2021).

G. Per-Game Results

In this Appendix, we include the per-game results corresponding to the Atari 2600 experiments in Section 6. We include the full learning curves for each game in Figure 7. We also include curves that show the overestimation of DQN and IB-DQN throughout training in Figure 8. Lastly, Table 2 reports the mean score across the final three evaluation phases for each algorithm and game.


Alien
Amidar
Assault
Asterix
Asteroids
Atlantis
Bank Heist
Battle Zone
Beam Rider
Berzerk
Bowling
Boxing
Breakout
Centipede
Chopper Command
Crazy Climber
Defender
Demon Attack
Double Dunk
Enduro
Fishing Derby
Freeway
Frostbite
Gopher
Gravitar
Hero
Ice Hockey
Jamesbond
Score
0M 10M 20M 30M 40M 50M

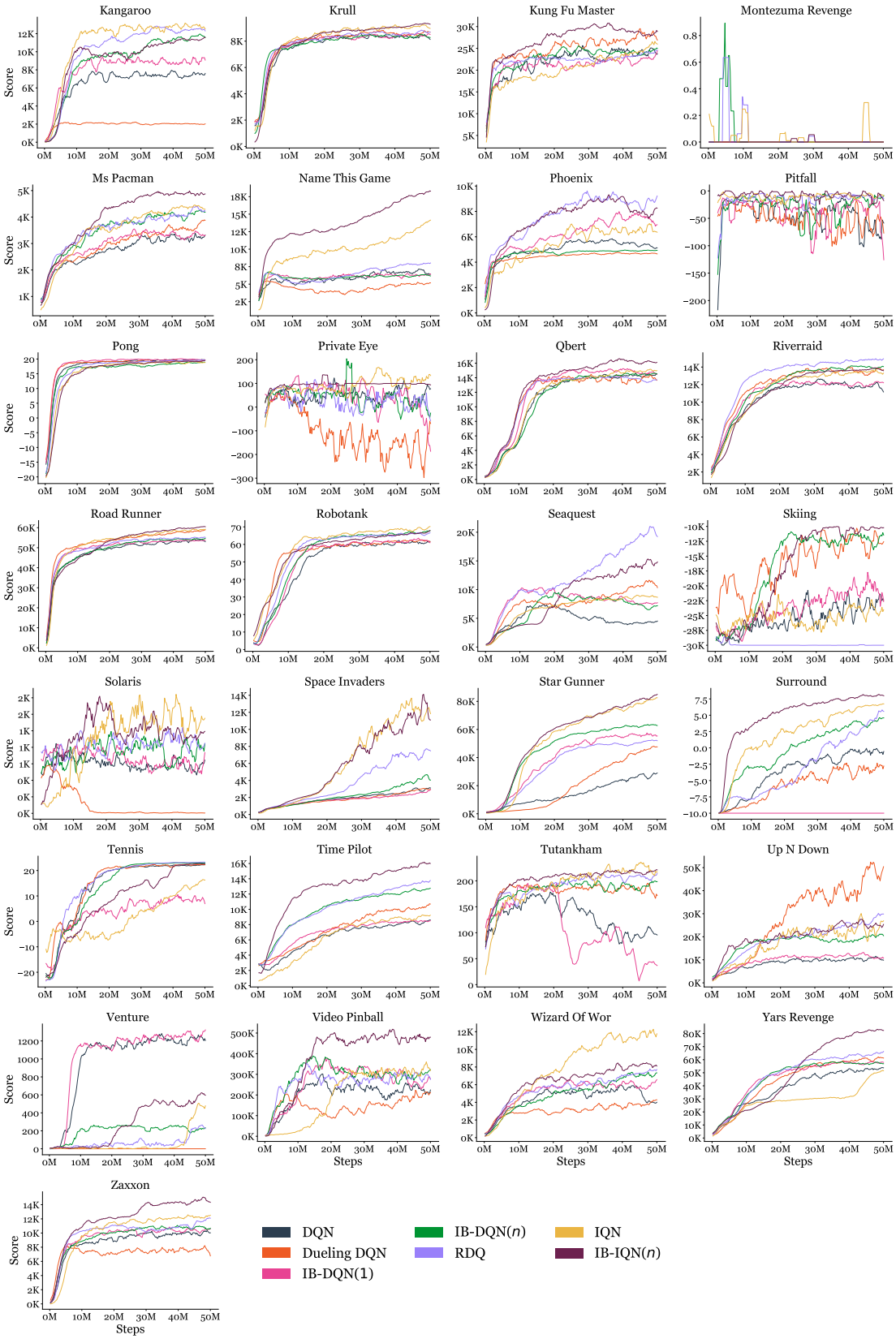
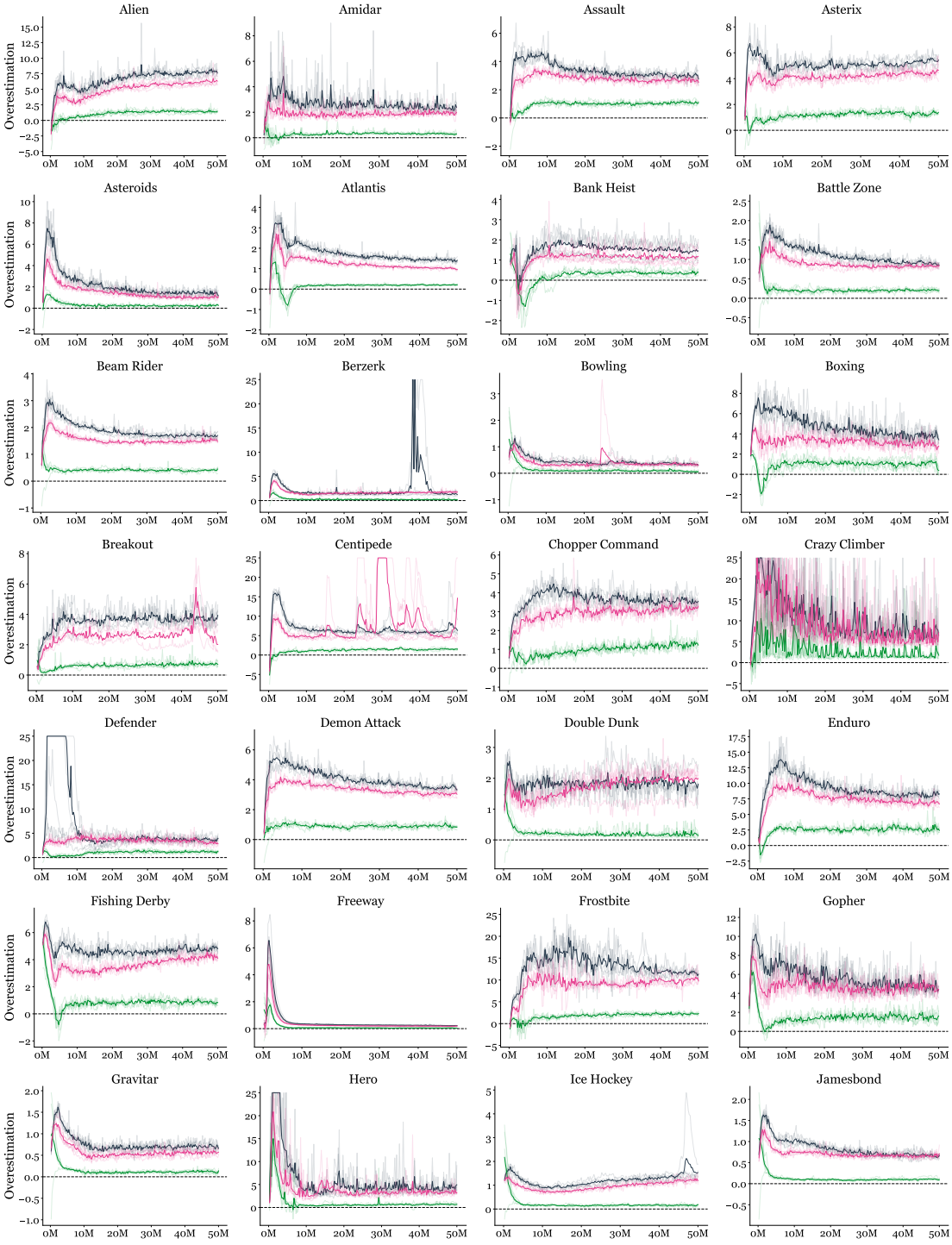


Figure 7. Mean score across 50M timesteps over five seeds per game. For readability, plots are smoothed with a moving average of seven.


Alien
Amidar
Assault
Asterix
Asteroids
Atlantis
Bank Heist
Battle Zone
Beam Rider
Berzerk
Bowling
Boxing
Breakout
Centipede
Chopper Command
Crazy Climber
Defender
Demon Attack
Double Dunk
Enduro
Fishing Derby
Freeway
Frostbite
Gopher
Gravitar
Hero
Ice Hockey
Jamesbond
Overestimation
0M 10M 20M 30M 40M 50M

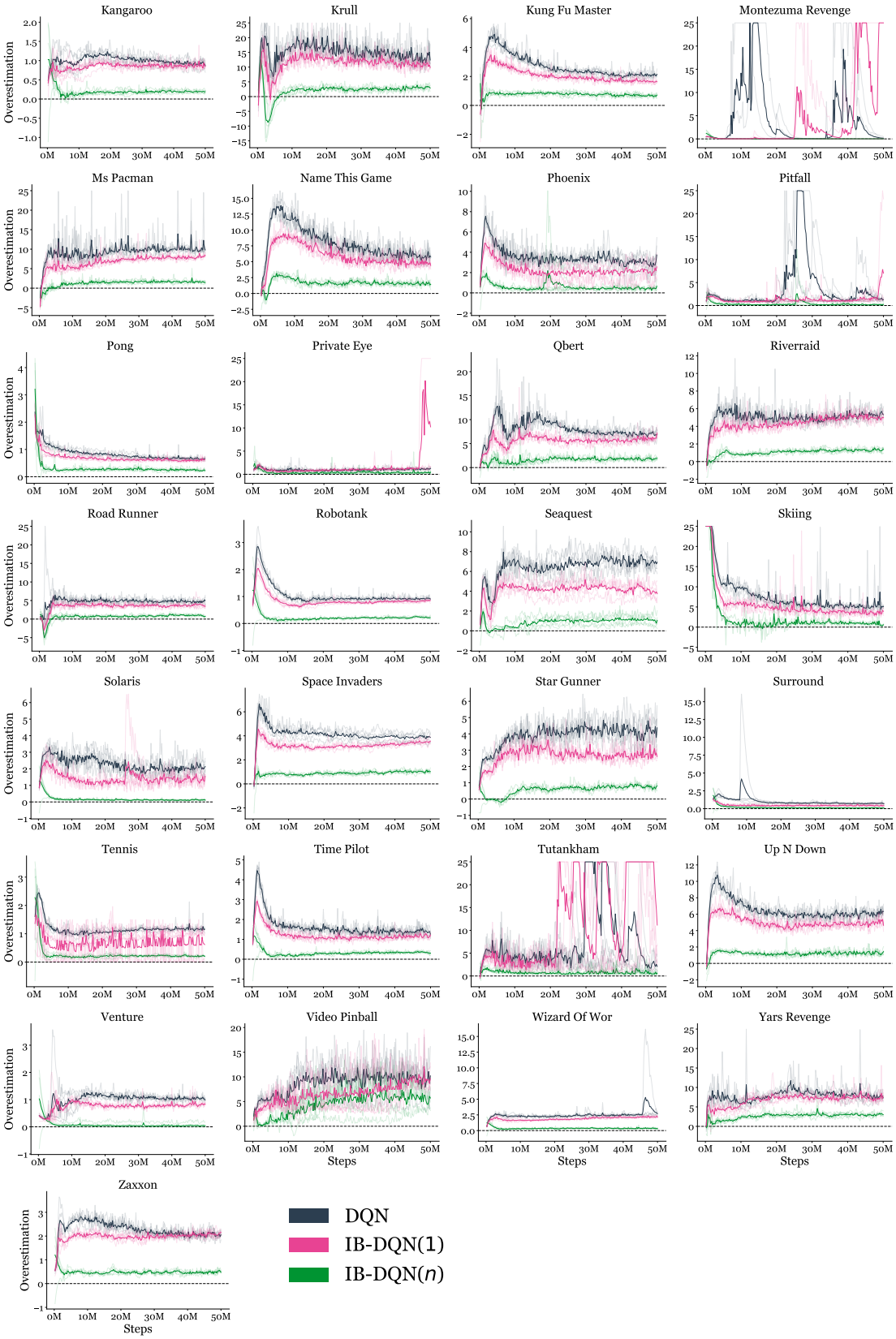


Figure 8. Mean overestimation across 50M timesteps across five seeds. Overestimation capped at 25 for visibility. Translucent curves are the individual seeds.

Environment	DQN	Dueling DQN	IB-DQN(1)	IB-DQN(n)	RDQ($\beta = 0.001$)	IQN	IB-IQN(n)
ALIEN	3446.7	3686.3	3602.8	3751.2	**5075.3**	3886.3	**4298.1**
AMIDAR	704.1	721.5	1076.5	1104.7	**1262.4**	952.9	**980.4**
ASSAULT	1864.5	1481.4	1815.6	**2039.6**	1698.4	3479.8	**5159.2**
ASTERIX	11725.6	13975.8	13689.3	17452.8	**24383.4**	30104.1	**39998.3**
ASTEROIDS	1071.2	1167.6	1012.9	1073.4	**2856.2**	1144.7	**1178.5**
ATLANTIS	821470.0	841251.7	886831.7	856981.7	**922795.0**	**941850.0**	865951.7
BANKHEIST	966.4	1168.1	1028.1	**1270.8**	1245.3	1203.8	**1463.0**
BATTLEZONE	28035.5	28205.0	27649.0	**33482.2**	32937.7	27673.7	**37347.3**
BEAMRIDER	5028.0	3838.2	4315.6	4982.4	**5669.1**	7803.1	**10219.5**
BERZERK	628.5	493.9	**691.2**	576.8	597.2	**718.2**	593.9
BOWLING	28.6	25.4	30.5	**33.5**	30.4	**39.4**	35.8
BOXING	94.5	**96.2**	95.1	95.2	94.5	98.0	**98.6**
BREAKOUT	91.3	**172.7**	61.0	133.8	133.3	**309.3**	288.0
CENTIPEDE	4714.9	**5696.2**	3308.1	5013.4	4606.5	5862.1	**6395.2**
CHOPPERCOMMAND	5987.9	5173.5	6508.0	**8174.8**	6142.7	7857.3	**10551.8**
CRAZYCLIMBER	106037.2	121540.2	110589.1	**124345.3**	121425.0	119368.5	**120757.1**
DEFENDER	9260.5	9195.2	7623.7	**22271.0**	13766.2	19813.2	**26517.9**
DEMONATTACK	7450.4	6052.2	6607.9	**16491.9**	6205.8	47948.2	**55029.9**
DOUBLEDUNK	-2.3	-5.3	**9.6**	-7.0	3.1	-1.8	**2.0**
ENDURO	1773.8	1695.6	1839.1	1748.2	**1897.2**	**2204.7**	2127.2
FISHINGDERBY	14.6	30.8	15.5	39.0	**44.0**	42.5	**44.9**
FREEWAY	33.8	33.9	33.8	**33.9**	33.8	33.9	**33.9**
FROSTBITE	6151.2	5714.5	6710.5	6697.5	**6775.6**	6913.1	**8556.4**
GOPHER	15184.2	13862.0	13380.0	13337.7	**17110.1**	**18495.5**	14897.3
GRAVITAR	1058.7	**1334.3**	1261.3	846.3	1196.5	876.2	**1125.9**
HERO	26142.7	**27788.5**	24745.0	20885.0	19005.1	22822.0	**26173.6**
ICEHOCKEY	-7.1	-5.1	-4.5	-2.5	**4.9**	**9.8**	7.9
JAMESBOND	**848.4**	770.2	844.1	648.9	624.2	**679.4**	539.6
KANGAROO	7690.1	1930.7	8704.5	11540.8	**12261.9**	**12918.2**	11563.1
KRULL	8112.6	8666.3	8560.9	8283.4	**8758.2**	8887.2	**9273.0**
KUNGFUMASTER	23924.3	**26705.8**	24500.2	24845.0	25119.4	25362.0	**28962.8**
MONTEZUMAREVENGE	**0.0**	0.0	0.0	0.0	0.0	**0.0**	0.0
MSPACMAN	3236.6	3882.0	3301.2	4222.4	**4318.7**	4261.9	**4857.0**
NAMETHISGAME	6810.0	5134.5	6421.4	6203.6	**8021.4**	14222.0	**18136.8**
PHOENIX	5128.0	4656.4	7005.4	4930.3	**8814.6**	6782.4	**8448.9**
PITFALL	-62.2	-59.7	-157.4	**-8.9**	-16.1	**-12.3**	-23.5
PONG	19.8	19.8	**19.8**	18.9	19.6	18.7	**19.6**
PRIVATEEYE	**68.5**	-105.4	-241.1	-47.9	-103.0	**125.2**	100.0
QBERT	14445.4	13913.9	14310.4	**14499.8**	13709.0	15153.6	**16185.5**
RIVERRAID	11169.2	13658.7	12110.3	14068.4	**14854.6**	13305.8	**13487.1**
ROADRUNNER	53174.3	**59137.8**	53886.1	54619.0	54881.5	58900.4	**60329.1**
ROBOTANK	62.0	62.7	60.8	**67.4**	67.0	**70.5**	68.4
SEAQUEST	4620.3	9953.0	7760.2	7155.8	**18852.1**	8418.8	**15313.7**
SKIING	-23539.1	**-10948.9**	-24026.5	-11007.6	-30000.0	-22549.1	**-10119.8**
SOLARIS	770.5	4.6	913.6	**1189.4**	1159.3	**1490.9**	1273.4
SPACEINVADERS	2922.8	2843.5	3091.7	3906.0	**7166.5**	**11839.3**	10930.3
STARGUNNER	28529.9	46965.0	56808.9	**62851.0**	51615.8	82905.3	**84907.1**
SURROUND	-0.5	-2.6	-10.0	4.2	**5.3**	6.7	**7.8**
TENNIS	22.3	22.1	6.7	22.9	**23.3**	17.1	**22.5**
TIMEPILOT	8569.5	10599.5	8405.2	12774.8	**13706.2**	9316.3	**16053.7**
TUTANKHAM	102.1	171.5	36.6	195.5	**214.1**	**225.4**	221.7
UPNDOWN	10570.1	**46005.7**	11025.4	20121.6	29311.0	**27168.5**	25773.2
VENTURE	1208.6	0.0	**1295.9**	238.7	208.4	476.6	**581.6**
VIDEOPINBALL	239194.8	219113.3	283539.1	**315799.9**	270117.5	315433.3	**498255.0**
WIZARDOFWOR	4031.7	4185.5	6285.3	7660.8	**7826.4**	**11630.1**	8146.0
YARSREVENGE	53891.5	61301.1	58932.9	56972.5	**66570.2**	51722.7	**83357.6**
ZAXXON	10034.8	6837.9	10370.4	10653.4	**12278.4**	12468.9	**14341.8**

Table 2. The mean evaluation score across the last 3 evaluations during training for each algorithm over five seeds. The highest scores for an environment are bolded, with separate comparisons for the IQN variants.