Accelerating Q-learning through Efficient Value-sharing across Actions

Prabhat Nagarajan
University of Alberta
Edmonton, AB, Canada
nagarajan@ualberta.ca

Brett Daley
University of Alberta
Edmonton, AB, Canada
brett.daley@ualberta.ca

Martha White
University of Alberta, Canada CIFAR AI Chair
Edmonton, AB, Canada
whitem@ualberta.ca

Marlos C. Machado
University of Alberta, Canada CIFAR AI Chair
Edmonton, AB, Canada
machado@ualberta.ca

ABSTRACT

Action-values are foundational to many control algorithms such as Q-learning, therefore learning them efficiently is central to reinforcement learning (RL). However, action-value learning 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 shared value structure across 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. Empirically, we show that it improves DQN and IQN's performance in aggregate across 57 Atari games while increasing action gaps and dramatically reducing value overestimation.

KEYWORDS

DQN, Dueling, Value-sharing

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 their learned Q-function and selecting high-value actions. Consequently, learning action-value functions efficiently is a central focus in RL [21]. We draw attention to two observations.

The first observation is that the true action-values being learned are often of high norm (assumed to be Euclidean norm in this paper), due to the 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. To add to this, the update rules of action-value learning algorithms like Q-learning [32] make small, incremental changes to the predicted values. Consequently, when the true action-value functions have high norm, many updates are needed to change the small initial values to the large values over the course of training [11]. 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 are also likely to as well, 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 to an easier one of learning smaller residuals.

These two ideals — value sharing through a state-dependent baseline and learning lower norm solutions — are very related. Learning a single scalar baseline for a state that is shared by actions requires the baseline 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 [11, 27, 31]. 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 which we call the *mean-expansion layer* that does not require an explicit baseline nor extra model parameters. This layer allows us to to share values across actions in a state by constructing an *implicit baseline* which is shared by all actions to construct 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.

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 DQN and IQN in 57 Atari 2600 games [6]. Lastly, we find that it

Proc. of the Adaptive and Learning Agents Workshop (ALA 2026), Aydeniz, Delgrange, Mohammedalamen, Yang (eds.), May 25 – 26, 2026, Paphos, Cyprus, https://alaworkshop2026.github.io/. 2026.

substantially reduces overestimation and increases the action gap in DQN — both desirable properties in deep RL.

2 PRELIMINARIES

In RL, we formulate the problem as finite Markov decision processes (MDPs). 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 $\mu(s)$ denotes 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 state 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

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

To learn to maximize the expected discounted return, algorithms like Q-learning [32] learn an estimated action-value function Q that approximates the action-value function of the optimal policy $q^* = \text{argmax}_\pi q_\pi(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}$.

Deep Q-networks (DQN) [21] adapt Q-learning [32] to leverage neural networks. The algorithm learns a Q-network, which is a Q-function parametrized with a set of neural network parameters $\boldsymbol{\theta}$. Given a state s, the Q-network outputs a vector of action-values: $\mathbf{q}(s; \boldsymbol{\theta}) = Q(s, \cdot; \boldsymbol{\theta}) = [Q(s, a_1; \boldsymbol{\theta}), ..., Q(s, a_n; \boldsymbol{\theta})]^\top$. The agent's most recent M experience transitions (s, a, r, s') are stored in a replay buffer $\mathcal{D}$ [19]. This buffer serves as a training dataset for supervised regression of target values, where the targets $y(a, r, s')$ are constructed from the Q-learning update, $y(a, r, s') = r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}^-)$. 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 $\mathbb{E}_{(s,a,r,s')\sim\mathcal{U}(\mathcal{D})}[(y(a, r, s') - Q(s, a; \boldsymbol{\theta}))^2]$.

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 transformation*, which, when implemented as a layer in a neural network, is called the *mean-expansion layer* (ME layer). 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 and to further reduce norm, we seek a representation of $\boldsymbol{q}$ in which we do not have to explicitly carry 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 $\boldsymbol{z} = [z_1, ..., z_n]^\top$ is a *residual vector*.

Definition 3.1 (*Baseline-residual vector*)**.** We define the vector $\mathbf{u}(\boldsymbol{q}, b) = [z_1, ..., z_n, b]^\top$, to be 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 the shared component, b, changes all entries q_i, $i \in \{1, ..., n\}$. By contrast, changing z_i affects only q_i. The vector $\boldsymbol{u}(\boldsymbol{q}, b)$ can be used to construct $\boldsymbol{q}$:

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

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

3.1 Norm-reducing baselines

Representing $\boldsymbol{q}$ through a baseline-residual vector $\mathbf{u}(\boldsymbol{q}, b)$ has the advantage of being efficient, in that its L2-norm is often less than that of $\boldsymbol{q}$: $\|\mathbf{u}(\boldsymbol{q}, b)\|_2^2 < \|\boldsymbol{q}\|_2^2$. In fact, if the entries of $\boldsymbol{q}$ have non-zero mean, then it can be represented with a lower-norm baseline residual decomposition. In fact, 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 $b^* = \sum_{i=1}^n q_i/(n+1)$. That is,*

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

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 $\|\mathbf{u}(\boldsymbol{q}, b)\|^2 < \|\boldsymbol{q}\|^2$ 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 nonzero, then any baseline non-inclusively between 0 and $2b^*$ is a strictly norm-reducing baseline.

In this paper, we only consider baselines in the range $b \in [0, \mu_{\boldsymbol{q}})$, which, other than $b = 0$, are all strictly norm-reducing baselines. 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 Transformation

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 $\boldsymbol{z} = \boldsymbol{q} - b\mathbf{1}$ be the residual vector of $\boldsymbol{q}$ with

baseline b. The absolute value of the mean of this residual vector $g_{\boldsymbol{z}}(b) = |\frac{1}{n}\sum_i (q_i - b)| = |\mu_{\boldsymbol{q}} - b| = |\mu_{\boldsymbol{z}}|$ 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 mean $\mu_{\boldsymbol{z}}$ gets smaller from $b = 0$ to $b = \mu_{\boldsymbol{q}}$. Moreover, the L2-norm of the residual vector $\boldsymbol{z}$ is also decreasing from $b = 0$ to $b = \mu_{\boldsymbol{q}}$ (Proposition 6, Appendix A). Consequently, to encourage a lower norm residual vector, we minimize

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

where $k \geq 0$ is a penalty on the vector's mean. 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 $b \in [0, \mu_{\boldsymbol{q}})$, 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.

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, i.e., 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 transformation*, where k is a 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 and Regularization Properties

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

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

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

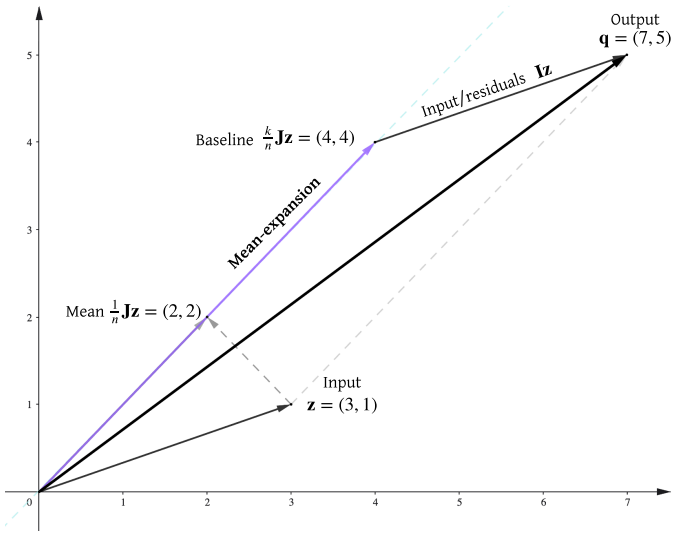


Figure 1: The mean-expansion layer/transformation. The input vector $z = (3, 1)$, is projected onto the all-ones vector to produce the mean component $(2, 2)$. This mean component is scaled by k, where $k = 2$, to produce the implicit baseline vector $\frac{k}{n}Jz = (4, 4)$. The input vector, which also serves as the residual vector, z, is added to this implicit baseline to produce the output $q = (I + \frac{k}{n}J)z = (7, 5)$. Applying this transformation allows us to store the low norm vector z instead of directly storing q or explicitly storing a baseline.

The mean component of $\boldsymbol{z}$ is scaled by $k + 1$ and added to its orthogonal component of $\boldsymbol{z}$ to produce $\boldsymbol{q}$, hence the name mean-expansion transformation. Said alternatively, the residual vector $\boldsymbol{z}$ is $\boldsymbol{q}$ with its mean component scaled down. Figure 1 provides a visual depiction of the mean-expansion transformation.

3.4 Selecting k

We have established how the mean-expansion transformation permits us to avoid the explicit modeling of a baseline parameter b. All that remains then, is to understand how to select k, based on the algorithm designer's desired implicit baseline b. The relationship between b and k can be formalized as follows.

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. □

Applying Equation 7, we see that as $k \to 0$, the baseline $b \to 0$. As $k \to \infty$, $b \to \mu_q$. In this paper, we use $k = n$ unless stated otherwise, which corresponds to the norm-minimizing baseline b^*.

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 analogous 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 in the range $(0, \mu_{\boldsymbol{q}})$,

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 transformation, k cannot be too large. While no mathematical issues arise with an arbitrarily large k, computational 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.

In this work, we set $k = n$, corresponding to the norm-minimizing baseline b^*. 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 transformation as 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 of residuals and produces an output that adds those residuals to a baseline which is constructed by scaling the mean of those very residuals. The layer increases the magnitude of the mean value of the input vector entries while preserving their relative differences. PyTorch code for the ME layer is provided in Appendix D.

The mean-expansion layer is added immediately before the output layer of a vector-valued 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 the system of equations without prior knowledge of the output vector $\boldsymbol{q}$. Since $\boldsymbol{q}$ is being learned, we instead implicitly change the problem to one of learning $\mathbf{z}$. The network's penultimate layer first outputs $\mathbf{z}$ 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 $\mathbf{z}$.

Taking as input the vector $\boldsymbol{z}$, the layer is implemented similar to Equation 6 to produce output $\boldsymbol{q}$:

$$\mu_z \leftarrow \frac{1}{n}\sum_{i=1}^{n} z_i, \tag{8}$$

$$\boldsymbol{q} \leftarrow \boldsymbol{z} - \mu_z + (k+1)\mu_z. \tag{9}$$

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 an 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 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, rather than run extensive hyperparameter sweeps.

4.2 Mean-expansion for Tabular Q-learning

The mean-expansion layer can also be applied to tabular Q-learning, though we posit its main use as being for neural networks. In lieu of a Q-function, we maintain residuals $Z(s,a)$ and construct the Q-values using the mean-expansion transformation M_k: $Q(s,\cdot) = M_k Z(s,\cdot)$. 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 update rule for all $a \in \mathcal{A}$ (see Appendix B for gradient computation):

$$Z(s_t,a_t) \leftarrow Z(s_t,a) + \alpha_t \delta_t \left(1 + \frac{k}{n}\right). \tag{10}$$

(11)

$$Z(s_t,a) \leftarrow Z(s_t,a) + \alpha_t \delta_t \left(\frac{k}{n}\right), \quad a \neq a_t. \tag{12}$$

We call this *Implicit-Baseline Q-learning* (IBQ), as we combine the mean-expansion transformation, 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 [32] as a special case when $k = 0$. More generally, this derivation applies to any TD-learning based action-value learning algorithm like Sarsa [25] and Expected Sarsa [18].

The update in Equations 10 and 12 highlight how the ME layer induces the 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 related class of approaches are dueling network approaches [11, 27, 31], 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 modification to 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 $b(s)$, which serves as the mean action-value.

It also outputs action-specific residuals $Z(s,a)$ to construct action-values as $Q(s,a) = b(s) + Z(s,a)$. 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) [11]. RDQ leverages the same architecture as Dueling DQN. However, $b(s)$, unlike in Dueling DQN where its value is the mean action-value prediction, does not serve as a predefined baseline in RDQ. Instead, RDQ augments the DQN loss with an additional L2-penalty, $\beta(\frac{1}{2}b(s)^2 + \frac{1}{2}\sum_a Z(s,a)^2)$, where β is a regularization coefficient.

Both of these methods learn an extra functional parameter to represent a baseline and both require an additional stream of learnable parameters. Dueling DQN, like our method, is a mere architectural modification to DQN with a predefined baseline. RDQ, by contrast, neither uses a predefined baseline, nor is it a mere architectural change, as it augments the DQN loss function.

Advantage updating [15] serves as a conceptual predecessor for the methods mentioned here. It is a 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 [4] avoids learning an explicit state-value function. These methods are emphasize stable learning through gradient descent with function approximation are less focused on accelerating learning through value-sharing.

Baseline-residual decompositions are broadly related to centering and normalization, which have also been explored in RL. Pop-Art [30] is a method which adaptively normalizes targets to be robust to different value scales. Sun et al. [26] 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 [9].

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 its sample efficiency, performance, overestimation, and action gaps.

6.1 Gridworld Experiment

To highlight the sample efficiency of IBQ (Equations 10 and 12), we examine it in a pedagogical and controlled 5x5 stochastic gridworld task. Episodes begin in the bottom left corner and terminate in 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 values of k from 0 to 7 in increments of 0.25. For each k, we report the best results across a sweep over 61 step sizes chosen from a logarithmic search over $(0, 1]$. All agents use an ϵ-greedy policy with $\epsilon = 0.1$. They are trained for 5,000 timesteps in each experiment, with 75 runs per combination of step size and k. Figure 2 shows, for different values of k, the percentage increase in the number of completed episodes when using IBQ over Q-learning, across 3 different sample complexity regimes. The shaded region corresponds to a 95% confidence interval.

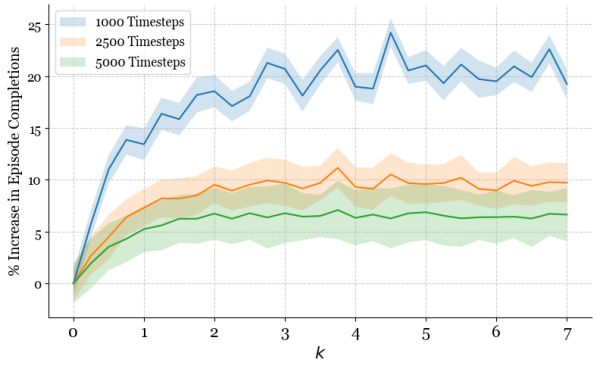


Figure 2: Gridworld results. We compare IBQ with different values of k, including $k = 0$, which is Q-learning. We report the percentage increase in episode completions across 3 sample complexity regimes. Shaded regions corresponds to a 95% confidence interval. For most k, IBQ(k) can complete over 20% more episodes than Q-learning within 1k timesteps. Both algorithms quickly master the task and their gap decreases with more timesteps.

We see that IBQ consistently learns faster in lower sample complexity regimes, with the gap between IBQ and Q-learning closing as both algorithms are given more experiences. We also observed that the optimal step size varied depending on k. Smaller step sizes worked better for large values of k, given the larger scaling, and larger step sizes worked better for smaller values of k.

6.2 Deep RL Empirical Methodology

Setting. We evaluate all of our agents in the Arcade Learning Environment (ALE) [6] on the 57 standard Atari environments used in the literature [29]. We follow the evaluation practices proposed by Machado et al. [20]. 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. All of our code[1] and baselines are implemented based on the PFRL library [14] and its reproduction of DQN. Our DQN implementation follows modern best practices, including the use of the Adam optimizer and the squared error loss, using the optimizer settings of Hessel et al. [16], as has become standard practice [8]. IB-DQN(k) refers to DQN with the mean-expansion layer applied to it. We do not modify the DQN algorithm or its hyperparameters for IB-DQN(n). We compare against the most related value-sharing method, Dueling DQN. We also scale up and evaluate RDQ, which was originally evaluated on MinAtar [33], to the full ALE. For RDQ, we set $\beta = 0.001$, matching the original

[1]This footnote will contain the link to the code in the camera-ready version of the paper.

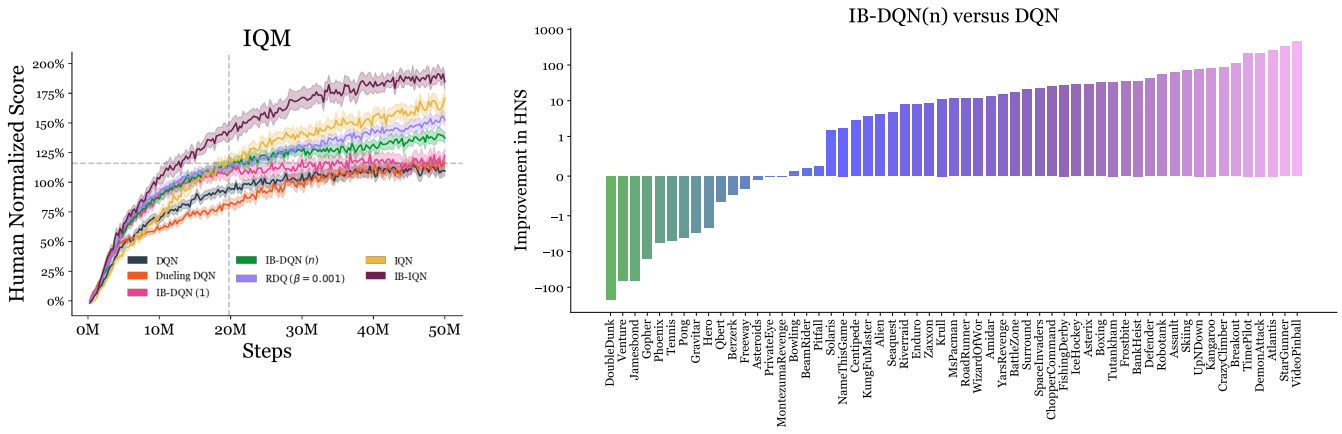


Figure 3: (*left*) Different algorithms and the interquartile mean of their human-normalized score across 57 games. All algorithms were run for 5 seeds, with the exception of IB-DQN(1) and IB-IQN(n), which were run for 3 seeds and 2 seeds, respectively. The shaded region depicts the 95% stratified bootstrap confidence interval. (*right*) The change in human-normalized score, measured as the average area-under-the-curve, when switching from DQN to IB-DQN($k = n$). Observe that the y-axis is log-scale.

paper after testing several values on a subset of games. For Implicit Quantile Networks (IQN) [10], 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.

6.3 Atari-57 Performance

Figure 3 (*left*) depicts the performance in terms of the interquartile mean (IQM) [2] of human-normalized score [21] of DQN, Dueling DQN, IB-DQN, RDQ, IQN and IB-IQN. All algorithms were run for 5 seeds, other than IB-DQN(1) and IB-IQN. Figure 3 (*right*) shows the change in human-normalized score when switching from DQN to IB-DQN. Figure 6 in Appendix F shows individual learning curves. Additionally, Table 2 in Appendix F reports the mean score over the last 3 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 vertical dotted line shows the first timestep at which the IQM of IB-DQN(n) exceeds that of the highest score of DQN (horizontal dotted line) across its curve. IB-DQN(n) is able to achieve DQN's best performance on the curve at just under 20M timesteps, or 40% of training time, exhibiting improved sample efficiency. When $k = 1$, which represents the conservative layer with a low condition number, we see visibly faster initial learning and a modest improvement through training. These results demonstrate that the mean-expansion layer can be an effective addition to a standard deep Q-network. Observe that IQN's performance also benefits substantially from adding the mean-expansion layer at the end of the network.

Scaling up RDQ demonstrates it to be an efficacious algorithm. IB-DQN(n) is also competitive with, though slightly worse than, RDQ. This is to be expected, as RDQ has an extra stream of parameters and an augmented loss. Unlike Dueling DQN, RDQ is able to effectively leverage these additional parameters through its regularization loss, with an additional hyperparameter in the form of a regularization coefficient. The appeal of IB-DQN is its simplicity as a simple parameter-free modification to DQN that can boost performance through an implicit baseline.

Observe that Dueling DQN does not outperform DQN, contrary to the original results of Wang et al. [31]. This is perhaps unsurprising, as it has been shown that modern implementations of DQN, utilizing Adam and the MSE loss, perform on par with distributional variants [1], which have been known to outperform Dueling Double DQN with prioritized replay [5].

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. In particular, we decided to look at two such metrics, overestimation and action gaps. Overestimation refers to the action-value prediction exceeding the true expected return [22, Sec. 2.2]. Reducing overestimation has historically been correlated with improved stability and performance in deep RL [13, 29].

Figure 4 (*left*) shows the percentage reduction in value overestimation when using IB-DQN(n) over DQN, measured as 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 7 in Appendix F), a reduction exceeding 100 indicates underestimation, while a reduction between 0% and 100% represents a reduction in 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 the penultimate layer's outputted features. In IB-DQN, the updates are explicit such that positive TD error increases the

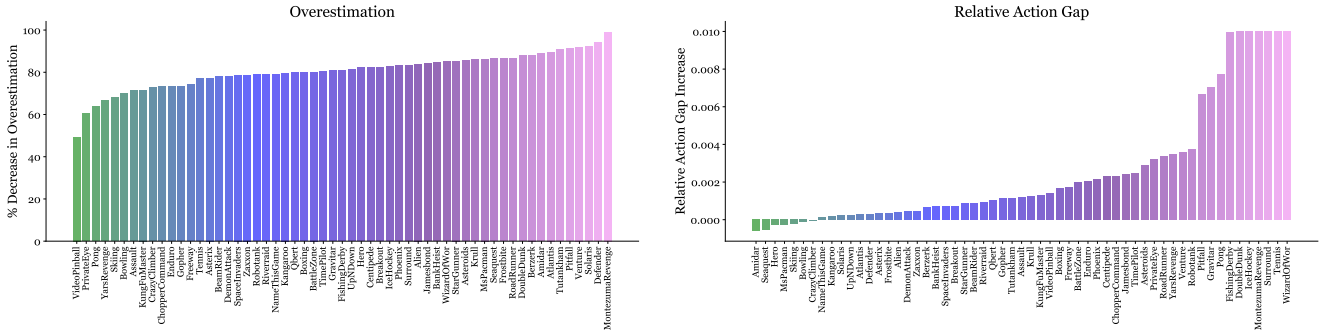


Figure 4: (*left*) The percentage *reduction* in overestimation when using IB-DQN over DQN, as a percentage of DQN's average overestimation area under the curve. In all games, IB-DQN reduces overestimation over DQN. (*right*) The *increase* in relative action gap from using IB-DQN instead of DQN. In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

values of all actions and a negative TD error decreases the value of all actions. A TD error on 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 7 in Appendix F.

Note that IB-DQN should be complementary to other methods for overestimation reduction, like Double DQN [29]. Double DQN uses both the Q-network and target network to produce bootstrap targets. This de-correlation in the bootstrap target reduces the degree of maximization bias. This de-correlation between the Q-network and target network is still present when ME layer is added to Double DQN.

The second metric we measured was the *action gap*. The action gap [12] 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. [31] report that the average action-value across states of a Double DQN [29] agent trained on the Atari game of 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 quickly change the greedy actions. For these reasons, increasing the action gap is known to be beneficial in deep RL [7].

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, as suggested by our overestimation results. We see that 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 model, not having to devote the majority of its capacity to fit the large baseline value common to all actions, can focus on the relative structure of the action-values.

6.5 Sensitivity Analysis of k

To examine the sensitivity of IB-DQN to its mean-scaling coefficient k, we ran it for several values of k on Gymnasium's [28] LunarLander-v3 environment for 120 seeds each. The results are shown in Figure 5. Figure 5 (left) shows that values of k approximately less than or equal to 64 can be beneficial for learning, producing a larger area under the curve than larger values of k. Performance drops dramatically as k grows too large, likely as the learning problem becomes ill-conditioned due to increasing condition number induced by the ME layer. Figure 5 (right) actually depicts the corresponding learning curves for several values of k, where we can visibly see the degradation as k grows large.

7 CONCLUSION AND DISCUSSION

In this paper, we presented the mean-expansion transformation, a simple way to represent and share values across actions. Utilizing the mean-expansion transformation allows us to avoid learning an explicit baseline parameter and merely learn a vector of residuals. The mean-expansion transformation can be applied to the DQN architecture, introducing no extra parameters and requiring no changes to the algorithm. We showed that applying the transformation provides a reliable boost to DQN's performance in aggregate across 57 Atari games, with large reductions in overestimation and an increased action gap.

The mean-expansion layer applied to RL is not without limitations. The first is that in our method, the condition number is $k + 1$, for $k \geq 0$. Selecting $k = n$ may not scale to extremely large action spaces. Fortunately, selecting $k = 1$ is well-conditioned and implies a baseline of $\mu_q/2$, which can be better than the implicit baseline of 0 in standard Q-learning, as supported by our empirical results. The second limitation is that our method has a discrepancy between the theory and practice. While our method is used to represent vector-valued outputs of action-values, our feedback is in the form of sample-based updates to individual state-action pairs. Dueling DQN also has this discrepancy as its $b(s)$, which represents the mean action-value, updates to minimize TD errors of the behavior policy, which may sample some actions in different proportion to

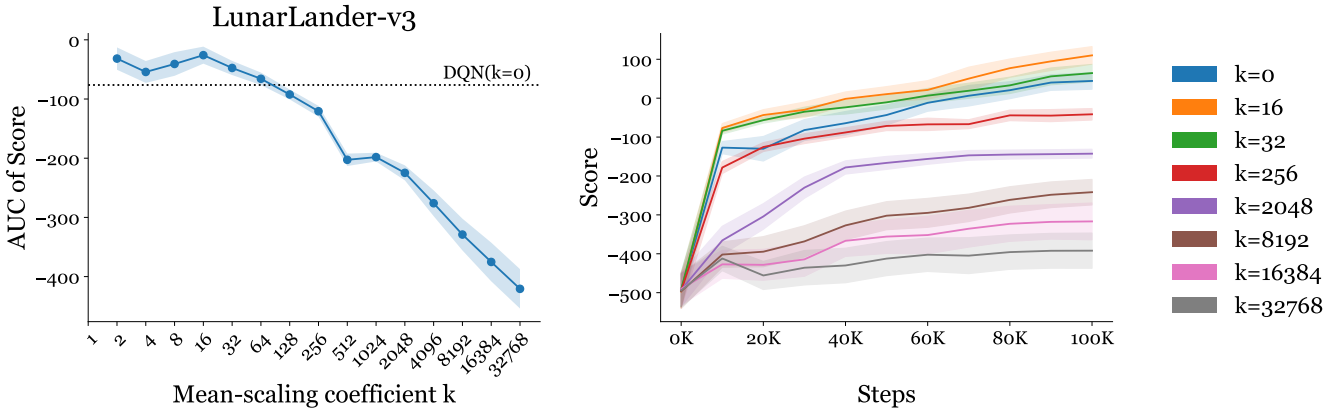


Figure 5: Sensitivity Analysis of k on LunarLander-v3. (*left*) The average area under the curve for several values of k (log scale). (*right*) The corresponding learning curves for a subset of values of k. In both plots, the shaded region depicts a 95% confidence interval for the mean across 120 seeds.

others. Examining the impact of this discrepancy is left for future work.

Our work also casts a novel perspective on value-sharing across actions in deep RL, especially as compared to Dueling DQN. The original dueling network architecture introduces extra parameters, a modified architecture, and a learned state-specific baseline. Any of one these could play an important role in improving performance. Our work suggests that the key ingredient may be the state-specific baseline, as we improve performance without extra parameters or large architectural modifications. We show that value-sharing does not necessarily require explicit separation of shared and independent components. Instead, we can have each action's output carry both individual and shared information effectively.

There is much to understand about the ME layer. A mechanistic explanation of the ME layer's impact on overestimation and action gaps remains unclear. How the ME layer impacts convergence and how it can be generalized to continuous actions remain active open questions.

Moreover, the source of the benefits of the ME layer can be better understood. In this paper had two motivations: value-sharing and learning a lower norm representation of the action-values. These two are not mutually exclusive, as the ME layer allows learning a lower norm residual representation through value-sharing and assigning We saw that the ME layer can yield sample efficiency benefits even in tabular settings, without the neural network, supporting the notion that value-sharing Yet, the gains from the ME layer seen in the deep RL setting possibly, perhaps even likely, also has to do with its impact on the optimization and learning dynamics of the Q-network induced by the ME layer.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful feedback. We thank Alex Lewandowski, Diego Gomez Noriega, and Esraa Elelimy for providing helpful feedback that improved the presentation of the paper. We thank Abhishek Naik for helpful discussions.

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.

REFERENCES

[1] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. 2020. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning (ICML)*.

[2] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. 2021. Deep reinforcement Learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems (NeurIPS)* (2021).

[3] Matthew Aitchison, Penny Sweetser, and Marcus Hutter. 2023. Atari-5: Distilling the Arcade Learning Environment down to Five Games. In *International Conference on Machine Learning (ICML)*.

[4] Leemon C Baird et al. 1999. *Reinforcement learning through gradient descent*. Technical Report. Carnegie Mellon University Pittsburgh, PA.

[5] Marc G Bellemare, Will Dabney, and Rémi Munos. 2017. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning (ICML)*.

[6] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research (JAIR)* (2013).

[7] Marc G Bellemare, Georg Ostrovski, Arthur Guez, Philip Thomas, and Rémi Munos. 2016. Increasing the action gap: New operators for reinforcement learning. In *AAAI conference on Artificial Intelligence (AAAI)*.

[8] Johan Samir Obando Ceron and Pablo Samuel Castro. 2021. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *International Conference on Machine Learning (ICML)*.

[9] Wesley Chung, Valentin Thomas, Marlos C Machado, and Nicolas Le Roux. 2021. Beyond variance reduction: Understanding the true impact of baselines on policy optimization. In *International Conference on Machine Learning (ICML)*.

[10] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. 2018. Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning (ICML)*.

[11] Brett Daley, Prabhat Nagarajan, Martha White, and Marlos C. Machado. 2025. An Analysis of Action-Value Temporal-Difference Methods That Learn State Values. *Reinforcement Learning Journal (RLJ)* (2025).

[12] Amir-Massoud Farahmand. 2011. Action-gap phenomenon in reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)* (2011).

[13] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*.

[14] Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka, and Takahiro Ishikawa. 2021. ChainerRL: A Deep Reinforcement Learning Library. *Journal of Machine Learning Research (JMLR)* (2021).

[15] Mance E Harmon, Leemon Baird, and A Harry Klopf. 1994. Advantage updating applied to a differential game. *Advances in Neural Information Processing Systems (NIPS)* (1994).

[16] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial intelligence (AAAI)*.

[17] Alston S. Householder. 1941. A theory of steady-state activity in nerve-fiber networks: I. Definitions and preliminary lemmas. *The Bulletin of Mathematical Biophysics* (1941).

[18] George H John. 1994. When the best move isn't optimal: Q-learning with exploration. In *AAAI*.

[19] Long-Ji Lin. 1992. *Reinforcement learning for robots using neural networks*. Carnegie Mellon University.

[20] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. 2018. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research (JAIR)* (2018).

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

[22] Prabhat Nagarajan, Martha White, and Marlos C. Machado. 2025. Double Q-learning for value-based deep reinforcement learning, revisited. *arXiv preprint arXiv:2507.00275* (2025).

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems (NeurIPS)* (2019).

[24] John Quan and Georg Ostrovski. 2020. DQN Zoo: Reference implementations of DQN-based agents. http://github.com/deepmind/dqn_zoo

[25] Gavin A Rummery and Mahesan Niranjan. 1994. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK.

[26] Hao Sun, Lei Han, Rui Yang, Xiaoteng Ma, Jian Guo, and Bolei Zhou. 2022. Exploit reward shifting in value-based deep-RL: Optimistic curiosity-based exploration and conservative exploitation via linear reward shaping. *Advances in Neural Information Processing Systems (NeurIPS)* (2022).

[27] Yunhao Tang, Rémi Munos, Mark Rowland, and Michal Valko. 2023. VA-learning as a more efficient alternative to Q-learning. In *International Conference on Machine Learning (ICML)*. PMLR.

[28] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. 2024. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032* (2024).

[29] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

[30] Hado P van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. 2016. Learning values across many orders of magnitude. *Advances in Neural Information Processing Systems (NeurIPS)* (2016).

[31] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*.

[32] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* (1992).

[33] Kenny Young and Tian Tian. 2019. 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 $b^* = \sum_{i=1}^n q_i/(n+1)$. That is,*

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

Proof. Let $f(b)$ be the squared L2-norm of $\boldsymbol{u}(\boldsymbol{q}, b)$.

$$f(b) = \|\boldsymbol{u}(\boldsymbol{q}, b)\|_2^2 = \sum_{i=1}^n (q_i - b)^2 + b^2. \tag{13}$$

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

$$\frac{df}{db} = 2b - \sum_{i=1}^n (q_i - b). \tag{14}$$

Setting the derivative to zero, we obtain

$$0 = 2b - 2\sum_{i=1}^n (q_i - b) \tag{15}$$

$$0 = b - \sum_{i=1}^n q_i + nb \tag{16}$$

$$\sum_{i=1}^n q_i = (n+1)b \tag{17}$$

$$b = \frac{\sum_{i=1}^n q_i}{n+1}. \tag{18}$$

□

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

Proof. Let $f(b)$ be the squared L2 norm of $\boldsymbol{u}(\boldsymbol{q}, b)$.

$$f(b) = \|\boldsymbol{u}(\boldsymbol{q}, b)\|_2^2 = \sum_{i=1}^n (q_i - b)^2 + b^2. \tag{19}$$

Expanding,

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

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

$$\begin{aligned} -2bn\mu_q + (n+1)b^2 &< 0 \\ (n+1)b^2 &< 2bn\mu_q \\ b^2 &< 2bb^*. \\ b(b - 2b^*) &< 0. \end{aligned}$$

This inequality holds if either $b < 0$ or $(b - 2b^*) < 0$, but not both.

If $\mu_q < 0$, then $b^* < 0$. If $\mu_q > 0$, then $b^* > 0$. 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 $\|\boldsymbol{u}(\boldsymbol{q}, \mu_{\boldsymbol{q}})\|_2^2$ 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 mean of the residual vector, $g_{\boldsymbol{q}}(b) = |\frac{1}{n}\sum_{i=1}^n (q_i - b)| = |\mu_{\boldsymbol{q}} - b| = \mu_z$, decreases from $b = 0$ to $b = \mu_{\boldsymbol{q}}$, regardless of the sign of $\mu_{\boldsymbol{q}}$.*

Proof.

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

So, if $\mu_{\boldsymbol{q}} > b$, $\frac{\mathrm{d}g_{\boldsymbol{q}}}{\mathrm{d}b} = -1$, 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, it 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 L2-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 $f_q(b) = \|\boldsymbol{q} - b\mathbf{1}\|_2^2 = \sum_{i=1}^n (q_i - b)^2$. Then

$$\begin{aligned} f_q(b) &= \sum_{i=1}^n \left(q_i^2 - 2bq_i + b^2\right) \\ &= \|\boldsymbol{q}\|_2^2 + nb^2 - 2b\sum_{i=1}^n q_i. \end{aligned}$$

To understand whether $f_q(b)$ is increasing or decreasing as a function of b, we examine its derivative:

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

Thus, $f_q(b)$ 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 $b = \mu_q$. Examining two cases:

- If $\mu_{\boldsymbol{q}} < 0$, then for $b \in (\mu_{\boldsymbol{q}}, 0]$ we have $b > \mu_{\boldsymbol{q}}$, so $f_q(b)$ is increasing on this interval and hence decreases as b moves from 0 toward $\mu_{\boldsymbol{q}}$.
- If $\mu_{\boldsymbol{q}} > 0$, then for $b \in [0, \mu_{\boldsymbol{q}})$ we have $b < \mu_{\boldsymbol{q}}$, so $f_q(b)$ 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 Transformation

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

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

Proof. Let $S_q = \sum_{i=1}^n q_i$ and let $S_z = \sum_{i=1}^n z_i$. By construction, for $k \geq 0$, we have that

$$\begin{aligned}
S_q &= S_z + \frac{nk}{n} S_z \\
\frac{k}{n} S_q &= \frac{k}{n} S_z + \frac{k^2}{n} S_z \\
\frac{k}{n} S_q &= \frac{k}{n} S_z (k+1) \\
\frac{\frac{k}{n} S_q}{k+1} &= \frac{k}{n} S_z \\
\frac{S_q}{\frac{n}{k}(k+1)} &= \frac{k}{n} S_z \\
k\mu_z &= \frac{S_q}{n + n/k}.
\end{aligned}$$

□

A.3 Properties of the Mean-expansion Transformation

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

$$\begin{aligned}
\boldsymbol{q} &= \left(I + \frac{k}{n} J\right)\mathbf{x} && (20)\\
&= \left(\frac{1}{n}J + \frac{k}{n}J + I - \frac{1}{n}J\right)\mathbf{x} && (21)\\
&= \left(\frac{1}{n}J\mathbf{x}\right) + \left(\frac{k}{n}J\mathbf{x}\right) + \left(I - \frac{1}{n}J\right)\mathbf{x} && (22)\\
&= (k+1)\left(\frac{1}{n}J\mathbf{x}\right) + \left(I - \frac{1}{n}J\right)\mathbf{x} && (23)\\
&= (k+1)\left(\frac{1}{n}J\mathbf{x}\right) + \left(\mathbf{x} - \frac{1}{n}J\mathbf{x}\right). && (24)
\end{aligned}$$

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

$$\begin{aligned}
f(\mathbf{z}) &= \frac{1}{n}\|\mathbf{z} - \boldsymbol{q}\|_2^2 + k\left(\frac{1}{n}\sum_{i=1}^n z_i\right)^2 && (25)\\
&= \frac{1}{n}(\mathbf{z} - \boldsymbol{q})^T(\mathbf{z} - \boldsymbol{q}) + k\left(\frac{1}{n}\sum_{i=1}^n z_i\right)^2 && (26)\\
&= \frac{1}{n}(\mathbf{z} - \boldsymbol{q})^T(\mathbf{z} - \boldsymbol{q}) + k\left(\frac{1}{n}\mathbf{1}^T\mathbf{z}\right)^2 && (27)\\
&= \frac{1}{n}(\mathbf{z} - \boldsymbol{q})^T(\mathbf{z} - \boldsymbol{q}) + \frac{k}{n^2}\mathbf{z}\mathbf{1}\mathbf{1}^T\mathbf{z} && (28)\\
&= \frac{1}{n}(\mathbf{z} - \boldsymbol{q})^T(\mathbf{z} - \boldsymbol{q}) + \frac{k}{n^2}\mathbf{z}^T J\mathbf{z} && (29)
\end{aligned}$$

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

$$\nabla_{\mathbf{z}} f(\mathbf{z}) = \frac{2}{n}(\mathbf{z} - \boldsymbol{q}) + \frac{2k}{n^2}J\mathbf{z}. \tag{30}$$

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

$$\begin{aligned}
0 &= \frac{1}{n}(\mathbf{z} - \boldsymbol{q}) + \frac{k}{n^2}J\mathbf{z} && (31)\\
0 &= (\mathbf{z} - \boldsymbol{q}) + \frac{k}{n}J\mathbf{z} && (32)\\
\boldsymbol{q} &= \mathbf{z} + \frac{k}{n}J\mathbf{z} && (33)\\
\boldsymbol{q} &= \left(I + \frac{k}{n}J\right)\mathbf{z}. && (34)
\end{aligned}$$

□

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 any action a in state s_t is

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial Z(s_t, a)} &= \frac{\partial \mathcal{L}}{\partial Q(s_t, a)}\frac{\partial Q(s_t, a)}{\partial Z(s_t, a)} && (35)\\
&= (-\delta_t)\frac{\partial Q}{\partial Z(s_t, a)} && (36)\\
&= (-\delta_t)\left(\mathbb{1}_{\{a_t = a\}} + \frac{k}{n}\right) && (37)
\end{aligned}$$

C SEMANTICS OF THE DUELING NETWORK ARCHITECTURE

In this Appendix, we discuss dueling methods more closely.

C.1 Dueling Deep Q-networks

Dueling DQN [31], under the terms "value" and "advantage", in fact learns a mean-residual decomposition of action-values. Its modification to the DQN architecture is to learn $b(s; \boldsymbol{\theta})$ in a baseline stream and $Z(s, \cdot)$ in a residual stream. The baseline and residual streams are aggregated to construct the Q-values

$$Q(s, a; \boldsymbol{\theta}) = b(s; \boldsymbol{\theta}) + Z(s, a; \boldsymbol{\theta}), \tag{38}$$

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 $Z(s, \cdot; \boldsymbol{\theta})$:

$$Z(s, a; \boldsymbol{\theta}) = x(s, a; \boldsymbol{\theta}) - \frac{1}{n} \sum_{a \in \mathcal{A}} x(s, a; \boldsymbol{\theta}). \tag{39}$$

Wang et al. [31] 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.

$$Q(s, a; \boldsymbol{\theta}) = b(s; \boldsymbol{\theta}) + x(s, a; \boldsymbol{\theta}) - \frac{1}{n} \sum_{a'} x(s, a'; \boldsymbol{\theta}). \tag{40}$$

Summing over actions:

$$\sum_a Q(s, a; \boldsymbol{\theta}) = \sum_a \left(b(s; \boldsymbol{\theta}) + x(s, a; \boldsymbol{\theta}) - \frac{1}{n} \sum_{a'} x(s, a'; \boldsymbol{\theta}) \right) \tag{41}$$

$$\sum_a Q(s, a; \boldsymbol{\theta}) = nb(s; \boldsymbol{\theta}) + \sum_a x(s, a; \boldsymbol{\theta}) - \sum_a x(s, a; \boldsymbol{\theta}) \tag{42}$$

$$\sum_a Q(s, a; \boldsymbol{\theta}) = nb(s; \boldsymbol{\theta}). \tag{43}$$

Dividing by n:

$$\frac{1}{n} \sum_a Q(s, a; \boldsymbol{\theta}) = b(s; \boldsymbol{\theta}). \tag{44}$$

Thus, Dueling DQN enforces

$$\boxed{b(s; \boldsymbol{\theta}) = \frac{1}{n} \sum_a Q(s, a; \boldsymbol{\theta})}, \tag{45}$$

i.e., $b(s; \boldsymbol{\theta})$ is the average action-value. As $Q(s, a; \boldsymbol{\theta}) = b(s) + Z(s, a; \boldsymbol{\theta})$, each $Z(s, a)$ 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) [11] uses the same Dueling DQN architecture and aggregates values according to Equation 38, but without the centering of the raw network outputs in Equation 39. In Dueling DQN, Equation 39 plays an important role in providing identifiability to the system in Equation 38 to ensure that the $n + 1$ variables, i.e., 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 $a \in \mathcal{A}$, RDQ adds the following penalty term to the DQN loss:

$$\beta \left(b(s; \boldsymbol{\theta})^2 + \sum_a Z(s, a; \boldsymbol{\theta})^2 \right),$$

where β is a regularization coefficient. In effect, this method encourages the baseline-residual vector of 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 be specific scalar multiples of the average action-value. RDQ simply uses $b(s; \boldsymbol{\theta})$ generally as a baseline, and penalizes it to ensure a low-norm representation.

Table 1: DQN training hyperparameters. * indicates 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.
*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.
step size	6.25e-5	The step size used by Adam.
*Adam ϵ	1.5e-4	The ϵ used by Adam.
*Adam β_1	0.9	β_1 hyperparameter value in Adam.
*Adam β_2	0.999	β_2 hyperparameter value in Adam.

D MEAN-EXPANSION LAYER PYTORCH CODE

Below we write the PyTorch [23] code for the ME layer, which can be implemented in fewer than a dozen lines.

```
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
```

E EXPERIMENTAL SETUP

In this section of the 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 [6], using the environment protocol proposed by Machado et al. [20]. 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, corresponding to the end of an episode, is upon the end of the game.

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. [21]. 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 Huber loss. Table 1 contains more implementation details and hyperparameters.

E.3 Metrics

Human-normalized scores (HNS) provide a mechanism for evaluating an agent's score in a manner that is comparable across games. The human-normalized score [21] of an agent can be computed as

$$\text{score}_{\text{hns}} = \frac{\text{score}_{\text{agent}} - \text{score}_{\text{random}}}{\text{score}_{\text{human}} - \text{score}_{\text{random}}}. \tag{46}$$

The human and random scores are taken from Quan and Ostrovski [24].

Measuring Overestimation. In this paper, we follow Nagarajan et al. [22, Appendix A.2]'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. On truncations due to 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 also ensure that this clipping is also applied when computing returns. This clipping, combined with discounting, often causes discounted returns to be much smaller than the raw, undiscounted, unclipped returns.

Measuring the Action Gap. To measure the action gap, we sample a minibatch from the buffer after each target network update and measure the action gap, i.e., 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 1e-8 in the denominator. These relative action gaps are averaged across all training phases and seeds of a game. We report the difference in relative action gap between the two algorithms, which are clipped to 0.01 for presentation.

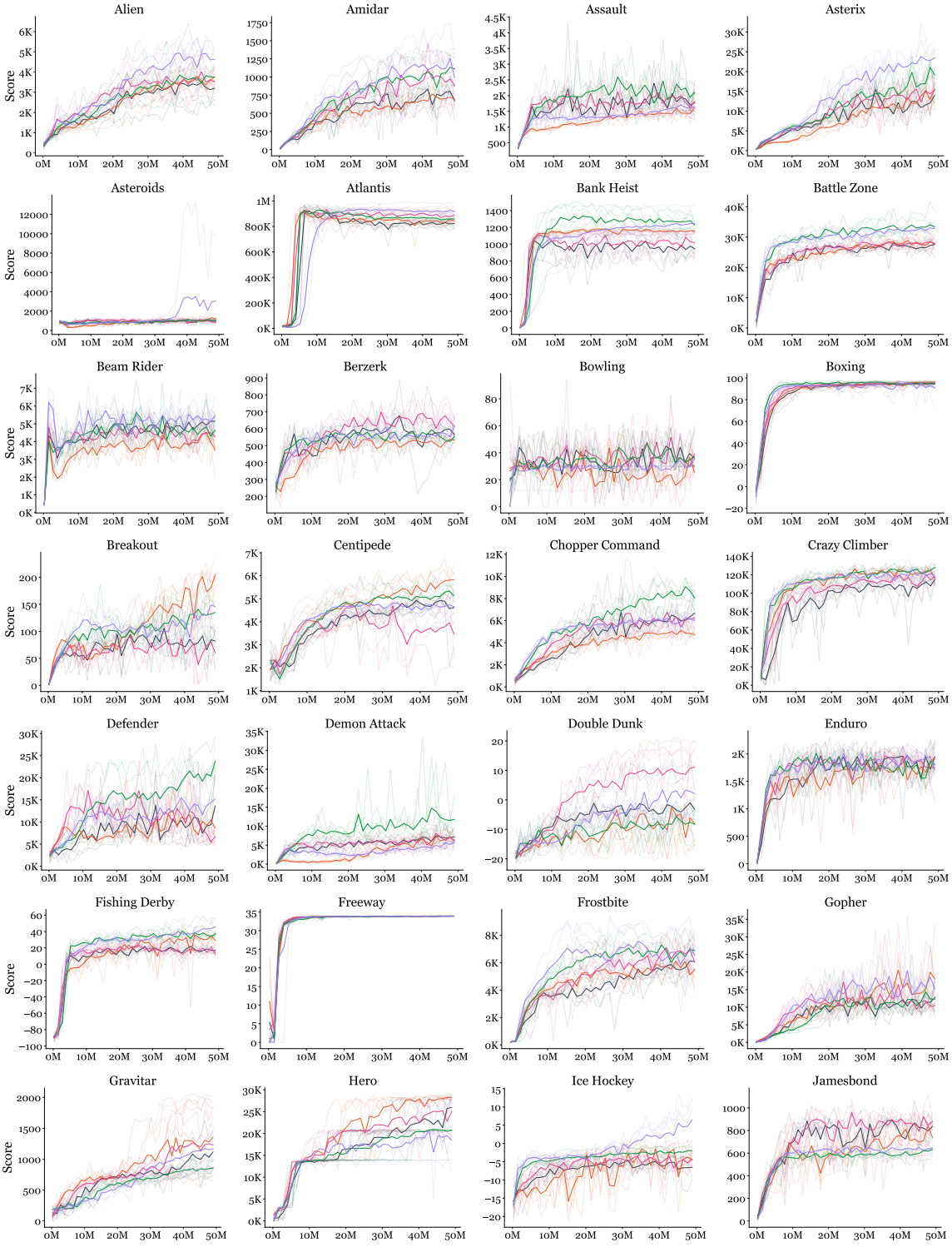
E.4 Baselines

For our Dueling DQN implementation, we used the exact same settings as DQN, but replace the network with the dueling network architecture. The final convolution layer outputs 64 filters of size 7×7 feature maps which are flattened into 3136 outputs. Following these flattened outputs, are two independent streams of 2-layer fully connected networks. Each stream has a hidden of layer of 512 units and ReLU activations [17]. 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 $\mathbf{z}(s;\boldsymbol{\theta}) = \mathbf{x}(s;\boldsymbol{\theta}) - \frac{1}{n}J\mathbf{x}(s;\boldsymbol{\theta})$.

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 [3], 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 for changing β from 0.001, as was set by Daley et al. [11], and thus kept this value.

F FULL RESULTS

In this Appendix, we include the full results corresponding to the experiments in Section 6. We include the full learning curves for each game. We also include curves that show the overestimation of DQN and IB-DQN throughout training. Lastly, we provide a table which reports the mean scores for each agent in every game, reporting the mean score of the final 5 evaluation phases.


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

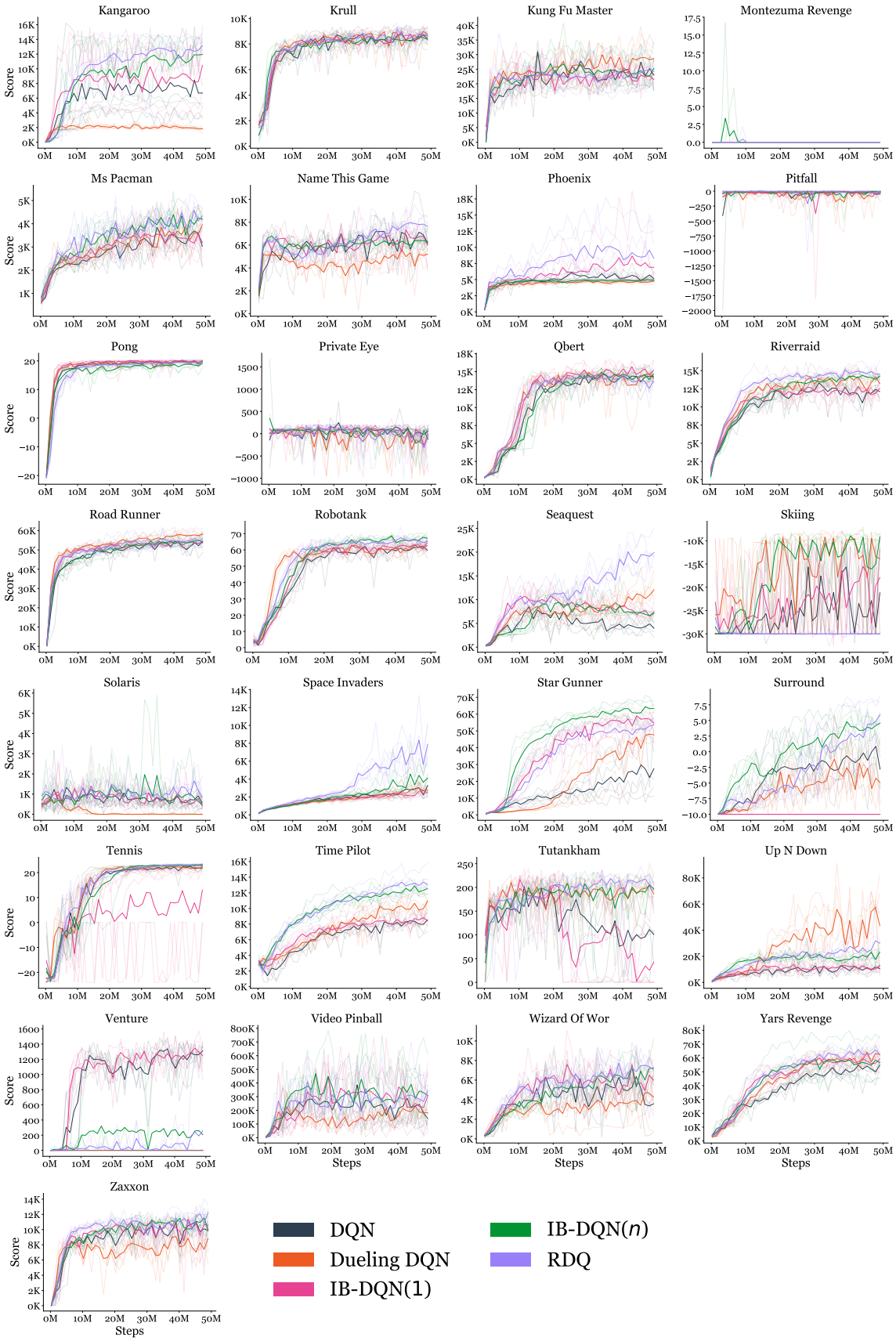
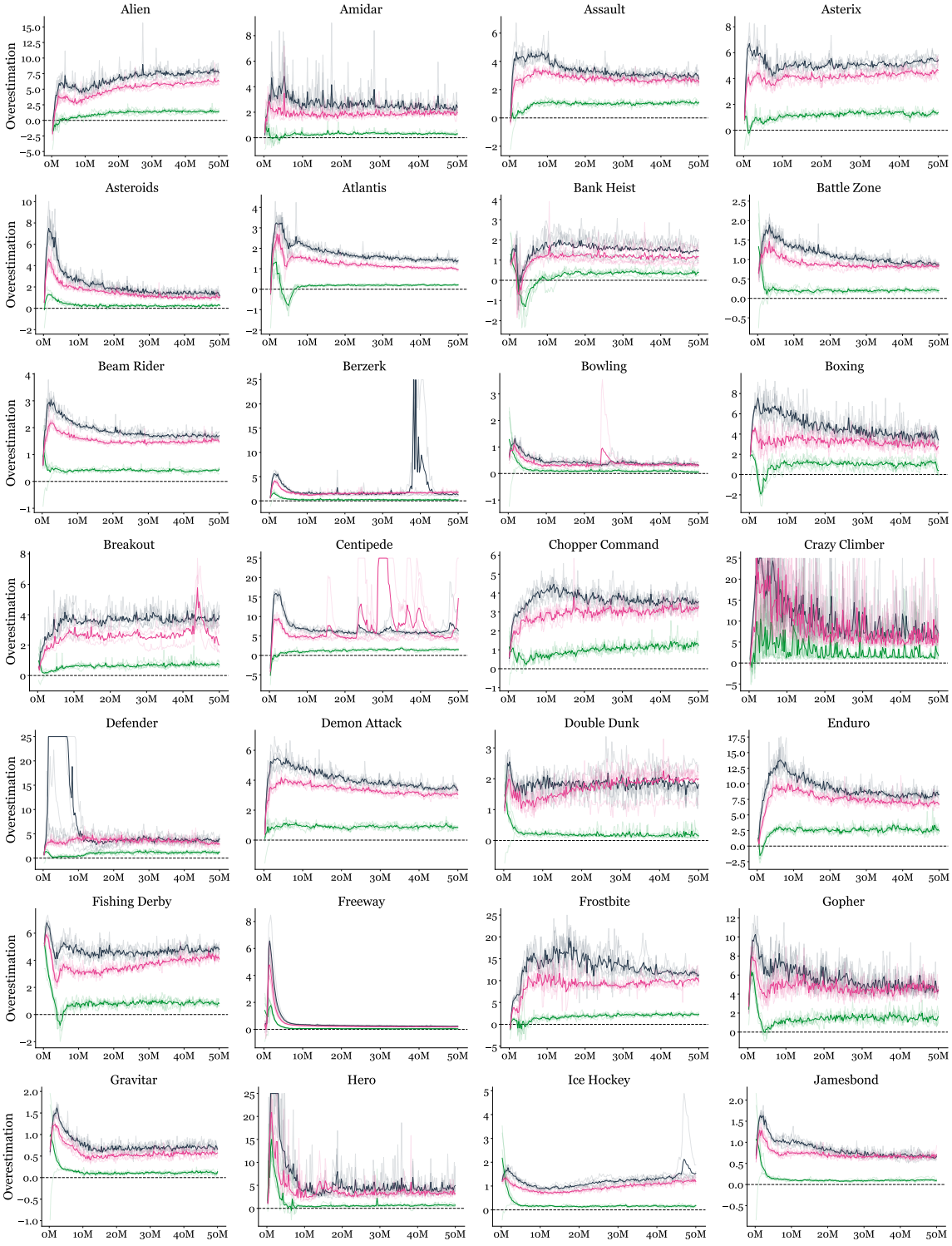


Figure 6: Mean score across 50M timesteps. Translucent curves are the individual seeds.


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

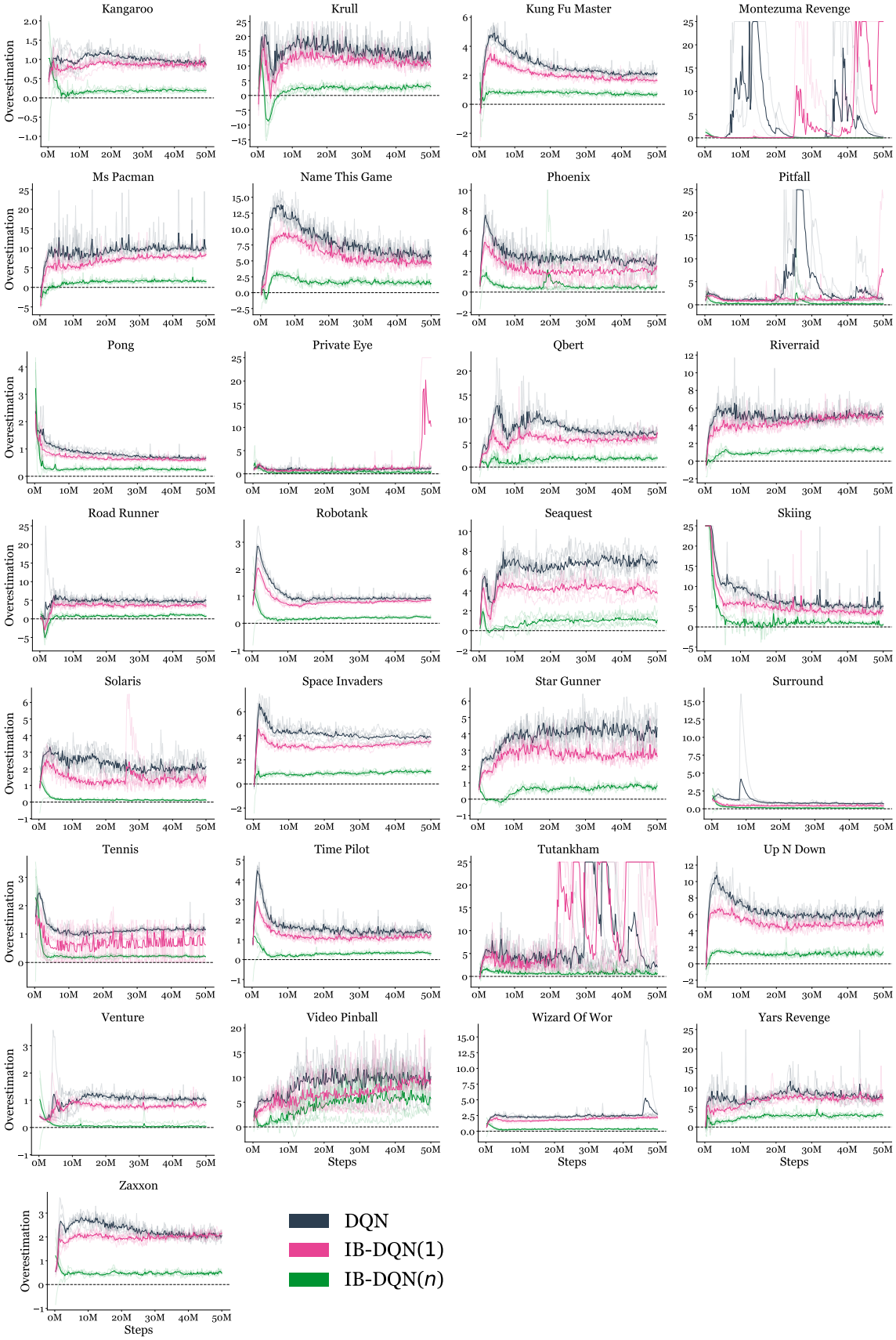


Figure 7: Mean overestimation across 50M timesteps. Overestimation capped at 25 for visibility. Translucent curves are the 5 individual seeds.

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	3146.4
Amidar	704.1	721.5	1076.5	1104.7	1262.4	952.9	849.2
Assault	1864.5	1481.4	1815.6	2039.6	1698.4	3479.8	5134.8
Asterix	11725.6	13975.8	13689.3	17452.8	24383.4	30104.1	41461.9
Asteroids	1071.2	1167.6	1012.9	1096.4	2856.2	1144.7	1184.4
Atlantis	821470.0	841251.7	886831.7	856981.7	916935.4	941850.0	875580.6
BankHeist	966.4	1168.1	1028.1	1270.8	1245.3	1203.8	1423.4
BattleZone	28035.5	28205.0	27649.0	33482.2	32937.7	27673.7	40798.3
BeamRider	5028.0	3838.2	4315.6	4982.4	5669.1	7803.1	8557.6
Berzerk	603.3	493.9	691.2	576.8	597.2	718.2	560.6
Bowling	28.6	25.4	30.5	33.5	30.4	39.4	35.9
Boxing	94.5	96.2	95.1	95.2	94.5	98.0	98.5
Breakout	91.3	172.7	75.1	133.8	133.3	309.3	274.1
Centipede	4714.9	5696.2	3308.1	5156.6	4606.5	5862.1	6172.4
ChopperCommand	5987.9	5173.5	6508.0	8174.8	6142.7	7857.3	9922.5
CrazyClimber	106037.2	121540.2	110589.1	124345.3	121425.0	119368.5	120500.6
Defender	9260.5	9195.2	7623.7	22271.0	13766.2	19813.2	26293.5
DemonAttack	7450.4	6052.2	6607.9	16491.9	6205.8	47948.2	63014.5
DoubleDunk	-2.3	-5.3	9.6	-7.0	3.1	-1.8	4.6
Enduro	1773.8	1695.6	1839.1	1748.2	1897.2	2204.7	2061.4
FishingDerby	14.6	30.8	15.5	39.0	44.0	42.5	46.2
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	8109.3
Gopher	15184.2	13862.0	13380.0	13337.7	17110.1	18495.5	15719.5
Gravitar	1058.7	1334.3	1261.3	846.3	1196.5	876.2	1007.7
Hero	26142.7	27788.5	24745.0	20885.0	19005.1	22822.0	25124.7
IceHockey	-7.1	-5.1	-4.5	-2.5	4.9	9.8	5.9
Jamesbond	848.4	770.2	844.1	648.9	624.2	679.4	549.5
Kangaroo	7690.1	1930.7	8704.5	11540.8	12261.9	12918.2	11038.6
Krull	8112.6	8666.3	8560.9	8283.4	8758.2	8887.2	9042.8
KungFuMaster	23924.3	26705.8	24500.2	24845.0	25119.4	25362.0	34233.3
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	4841.8
NameThisGame	6810.0	5134.5	6421.4	6203.6	8021.4	14222.0	18187.4
Phoenix	5128.0	4656.4	7005.4	4930.3	8814.6	6782.4	7252.7
Pitfall	-62.2	-59.7	-157.4	-8.9	-16.1	-12.3	-21.0
Pong	19.8	19.8	19.8	18.9	19.6	18.7	19.3
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	15966.3
Riverraid	11169.2	13658.7	12110.3	14068.4	14854.6	13305.8	13093.3
RoadRunner	53174.3	59137.8	53886.1	54619.0	54881.5	58900.4	61143.6
Robotank	62.0	62.7	60.8	67.4	67.0	70.5	70.9
Seaquest	4620.3	9953.0	7760.2	7155.8	18852.1	8418.8	31423.9
Skiing	-23539.1	-10948.9	-24026.5	-11028.6	-30000.0	-22549.1	-10037.6
Solaris	770.5	4.6	913.6	1189.4	1159.3	1490.9	1165.8
SpaceInvaders	2922.8	2843.5	3091.7	3906.0	7166.5	11839.3	8366.0
StarGunner	28529.9	46965.0	56808.9	62851.0	51615.8	82905.3	73060.2
Surround	-0.5	-2.6	-10.0	4.2	5.3	6.7	8.8
Tennis	22.3	22.1	6.7	22.9	23.3	17.1	22.7
TimePilot	8569.5	10599.5	8405.2	12774.8	13706.2	9316.3	17305.6
Tutankham	102.1	171.5	36.6	195.5	214.1	225.4	228.6
UpNDown	10570.1	46005.7	11025.4	20121.6	29311.0	27168.5	23779.1
Venture	1208.6	0.0	1295.9	238.7	208.4	476.6	875.2
VideoPinball	239194.8	219113.3	283539.1	315799.9	270117.5	315433.3	498268.9
WizardOfWor	4031.7	4185.5	6285.3	7660.8	7826.4	11630.1	8568.3
YarsRevenge	53891.5	61301.1	58932.9	56972.5	66570.2	51722.7	83602.1
Zaxxon	10034.8	6837.9	10370.4	10653.4	12278.4	12468.9	14008.6

Table 2: The mean evaluation score across the last 3 evaluations during training for different algorithms. All algorithms were run for 5 seeds except for IB-IQN which had 2 seeds and IB-DQN(1) which had 3 seeds.