BEYOND TARGET NETWORKS: IMPROVING DEEP Q-LEARNING WITH FUNCTIONAL REGULARIZATION

Anonymous authors
Paper under double-blind review

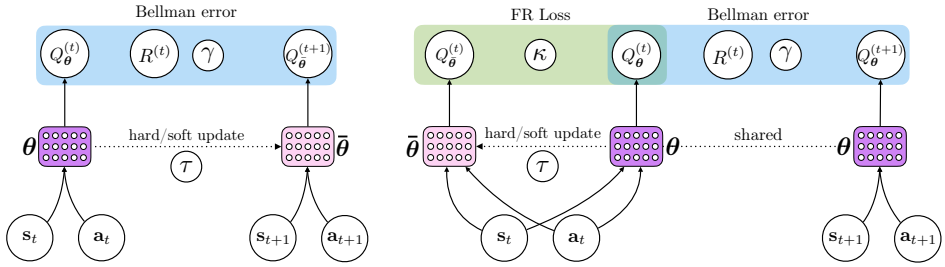
ABSTRACT

Target Networks are at the core of the recent successes of Deep Reinforcement Learning due to their ability to stabilize the training of Deep Neural Networks by using lagging parameters to estimate the target Q-values used to compute the Bellman error. Yet using Target Networks also periodically inhibits the propagation of newly-encountered rewards which could ultimately slow down training in term of performance and recovering the true value function. In this work, we propose to augment the Bellman error with a Functional Regularizer to a prior network which allows us to explicitly tune the regularization weight and achieve stable and fast learning. Unlike the traditional Bellman error, which requires Target Networks to be stable, the functional regularized Bellman error uses up-to-date parameters to estimate the target Q-values, thereby speeding up training while maintaining stability by using a Functional Regularizer to a prior network. We also demonstrate how target network implicitly operate a form of Functional Regularizer in the Bellman error albeit with fixed regularization weight. Using our objective, the Functionally Regularized Bellman error, we show that we can recover the true optimal value function on a simple task, while the Bellman error with Target Networks fails to do so. Additionally, we show empirical improvements in sample efficiency and performance across a range of Atari environments. In summary, the Functionally Regularized Bellman error can be used as a drop in replacement for the Bellman error and lead to more stable and faster learning.

1 INTRODUCTION

Deep Q-learning (DQL) methods demonstrate instability in practice due to being trained with off-policy data, function-approximations, and bootstrapping (Sutton & Barto, 2018; Van Hasselt et al., 2018). Part of the instability is due to the constantly changing target Q-values use in the Bellman error caused by the updating of the Deep Neural Networks (DNNs) parameters. In this context, Target Networks improve stability by updating the parameters generating the targets only periodically (Mnih et al., 2013). Because of this, target network based training is now a standard practice in DQL (Mnih et al., 2013; Lillicrap et al., 2015; Abdolmaleki et al., 2018b; Haarnoja et al., 2018a; Fujimoto et al., 2018; Hausknecht & Stone, 2015; Van Hasselt et al., 2016) and has achieved state-of-the-art performance on a variety of difficult tasks ranging from the Atari suite (Hessel et al., 2018) to robotics applications (Haarnoja et al., 2018b).

Despite the impressive performance of Target Networks, using separate sets of parameters to estimate the target Q-values is problematic. This is because by not updating the parameters of the target network, newly encountered rewards are not propagated through the state space, slowing down learning. A common alternative to periodically updating the target weights has been to parametrize the target network using a moving average of the parameters (Lillicrap et al., 2015), which is the default choice for MLP architectures, but has not been showed to be effective in practice for larger architectures such as CNN. The approach can be seen as a type of *weight-regularization* that slows down the parameter updates, which might be insufficient for DNNs where regularizing the parameters does not necessarily imply stability of the network outputs (Benjamin et al., 2018). Alternatives based on preconditioning (Knight & Lerner, 2018; Achiam et al., 2019) have been proposed to maintain stability without the use of Target Networks. Albeit more sophisticated, such ideas still fall within the realm of weight-regularization and are usually computationally costly and limited to simple network architectures.



(a) Bellman error with Target Networks.

(b) Functionally Regularized Bellman error with Prior Network.

Figure 1: (a) Target networks use a separate set of parameters, $\bar{\boldsymbol{\theta}}$ (pink), to estimate target values which are then used in the Bellman error. The target network stabilizes the Q-value estimates, but is not up-to-date. (b) In our approach, we simply use the up-to-date $\boldsymbol{\theta}$ to estimate the Q-value for the Bellman error (see the rightmost block), and add a κ weighted Functional Regularization (FR) loss to stabilize the current Q-value estimate (parameterized by $\boldsymbol{\theta}$) by comparing it to the old Q-values (parameterized by $\bar{\boldsymbol{\theta}}$).

In this paper, we show that, contrary to popular belief, Target Networks are not required to stabilize DQL training. In fact, by using the Functionally Regularized Bellman error, we can better approximate the true value function and improve performance. Specifically, we used up-to-date parameters to estimate the target Q-values and stabilize training via *functional regularization* (FR) to a periodically updated prior network (see Fig. 1). This decoupling enables us to explicitly tune the regularization level for a given task, which is not possible in the Bellman error. We validate our new objective on the Four Rooms environment (Sutton et al., 1999) with DNNs function approximation and show that we can recover the true value function and learning quickly. Finally, we demonstrate that DQL methods with the FR Bellman error outperform popular algorithms based on the Bellman error on a subset of the Atari suite (Bellemare et al., 2013) which is the most common bechmark for DQL methods.

2 Background

Preliminaries We consider the general case of a Markov Decision Process (MDP) defined by $\{\mathcal{S}, \mathcal{A}, P, R, \gamma, \mu\}$, where $\mathcal{S}$ and $\mathcal{A}$ respectively denote the finite state and action spaces, $P\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ are the environment transition dynamics, where $P(\cdot|\mathbf{s}, \mathbf{a})$ is the distribution of the next state taking action $\mathbf{a}$ in state $\mathbf{s}$. $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function, $\gamma \in [0, 1)$ is the discount factor, and μ is the initial state distribution. We use an MDP to represent the sequential interaction between an agent and the environment, where at each round t the agent observes its current state $\mathbf{s}_t$, selects an action $\mathbf{a}_t$ which results in a reward $R(\mathbf{s}_t, \mathbf{a}_t)$ and the next state $\mathbf{s}_{t+1} \sim P(\cdot|\mathbf{s}_t, \mathbf{a}_t)$. The reinforcement learning (RL) objective involves defining an agent that maximizes the expected discounted sum of rewards $\mathbb{E}_{P,\pi} \sum_t \gamma^t R(\mathbf{s}_t, \mathbf{a}_t)$, (with $\mathbf{s}_0 \sim \mu$), by means of a policy $\pi(\mathbf{a}|\mathbf{s})$, that given a state selects in expectation the best action. Bellman optimality equations are defined as:

$$Q^*(\mathbf{s}, \mathbf{a}) \equiv R(\mathbf{s}, \mathbf{a}) + \mathbb{E}_{P,\pi^*}\left[\sum_{t=1}^{\infty} \gamma^t R(\mathbf{s}_t, \mathbf{a}_t) \middle| \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}\right], \tag{1}$$

$$= R(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}', \mathbf{a}' \sim P, \pi^*}\left[Q^*(\mathbf{s}', \mathbf{a}')\right], \tag{2}$$

where the optimal policy is defined as $\pi^*(\mathbf{a}|\mathbf{s}) = \delta(\mathbf{a} - \arg\max_{\mathbf{a}} Q^*(\mathbf{s}, \mathbf{a}))$, $\delta(\cdot)$ is the Dirac delta function, and we use π^* to evaluate the expectations in Eqs. (1) and (2) (see (Watkins, 1989)). Note that the value function can be obtained as $V^*(\mathbf{s}_t) = \max_{\mathbf{a}} Q^*(\mathbf{s}_t, \mathbf{a})$.

Q-value Estimation We can estimate the Q-value for each state-action pair by formulating it as a regression problem, e.g., treating the right hand side of Eq. (1) as the target. However, we generally do not have access to the optimal policy or to samples from the optimal return. Fortunately, it is possible to learn from arbitrary policy by using the Q-learning algorithm (Watkins, 1989). Specif-

ically, we can minimize the one-step Mean Squared Bellman Error (MSBE) for a given transition $(\mathbf{s}_t, \mathbf{a}, r_t, \mathbf{s}_{t+1})$ (Sutton, 1988):

$$l(Q) = \frac{1}{2}\Bigg(\underbrace{R(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1}, \mathbf{a}_{t+1} \sim P, \pi}\left[Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})\right]}_{\text{target } Q\text{-value}} - Q(\mathbf{s}_t, \mathbf{a}_t)\Bigg)^2, \tag{3}$$

where $\pi(\mathbf{a}|\mathbf{s}) = \delta(\mathbf{a} - \arg\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}))$, and the loss is then averaged over multiple transitions. This approach of using future value estimates as regression targets is referred to as *bootstrapping*. Sometimes, the target is expressed using the Bellman operator $\mathcal{T}Q(\mathbf{s}_t, \mathbf{a}_t) = R(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1}, \mathbf{a}_{t+1} \sim P, \pi}\left[Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})\right]$.

Function Approximation The original Q-learning algorithm by Watkins, estimates each state-action pair separately, a prohibitive task in real-world applications. Instead, it is possible to approximate Q by means of a parametrized function $Q_{\boldsymbol{\theta}}$ (e.g., a deep network), which can be learned by minimizing Eq. (3) with an iterative algorithm such as gradient descent. Unfortunately, combining off-policy data, function approximation, and bootstrapping makes learning unstable and potentially diverge (Van Hasselt et al., 2018; Sutton & Barto, 2018; Achiam et al., 2019). The problem arises when the parameters of the Q-network are updated to better approximate the Q-value of a state-action pair at the cost of worsening the approximation of other Q-values, including the ones used as targets.

One approach to stabilize learning involves estimating the regression targets with a *target network* $Q_{\overline{\boldsymbol{\theta}}}$, a copy of $Q_{\boldsymbol{\theta}}$ where $\overline{\boldsymbol{\theta}}$ is periodically updated only after a certain number of iterations (Mnih et al., 2013). This causes regression targets to no longer directly depend on the most recent estimate of $\boldsymbol{\theta}$. With this, the Mean Squared Projected Bellman Error (MSPBE) loss for a given state transition is:

$$l^{\text{target}}(\boldsymbol{\theta}) = \frac{1}{2}\Bigg(R(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1}, \mathbf{a}_{t+1} \sim P, \pi}\left[Q_{\overline{\boldsymbol{\theta}}}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})\right] - Q_{\boldsymbol{\theta}}(\mathbf{s}_t, \mathbf{a}_t)\Bigg)^2. \tag{4}$$

A common strategy is to update the target parameters only after a fixed number of training iterations ("the hard update") (Mnih et al., 2013). In between these updates, the target is using the old parameters, and the newly encountered rewards are not immediately reflected in the estimated target Q-values. However, the delay in the reward propagation can slow down training. For instance, consider a Markov chain environment of N states with deterministic transitions, in which all states have reward of 0 except the terminal, right most state which has a reward of 1. Consider using tabular Q-values and the corresponding target Q-values, which are updated every H training steps. Even with access to all the state-action pairs, this scheme will require NH training steps to propagate the right-most reward through the chain, i.e., to the state N steps away.

An alternative to hard-updating is to use a moving average of the weights to parametrize the target Q-network as done with Polyak's averaging ("the soft update") (Lillicrap et al., 2015). The target network's weights are updated as $\overline{\boldsymbol{\theta}} \leftarrow (1-\tau)\overline{\boldsymbol{\theta}} + \tau\boldsymbol{\theta}$, with $\tau \in (0, 1)$. However, we are not using $\boldsymbol{\theta}$ to estimate the target Q-values, thus, our regression targets are no longer up-to-date and learning is slowed.

Weight Regularization in DQN The soft updates can be seen as a simple *weight regularization* where the update of $\overline{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$ is delayed by adding an L_2 regularizer $\tau\|\boldsymbol{\theta} - \overline{\boldsymbol{\theta}}\|^2$. Essentially, we do not want the value of $\overline{\boldsymbol{\theta}}$ to abruptly change. Such idea to improve stability is common in non-stationary time-series analysis (Brown, 1959; Holt et al., 1960; Gardner Jr, 1985), online learning (Cesa-Bianchi & Lugosi, 2006), and continual learning (Kirkpatrick et al., 2017; Nguyen et al., 2017) in general. Recent attempts to go beyond target networks employ preconditioning methods which can also be seen as weight-regularization methods (Knight & Lerner, 2018; Achiam et al., 2019). For example, natural-gradient descent used in Knight & Lerner (2018) employs the Kullback-Leibler (KL) for Gaussian distributions $\mathcal{N}(q|Q_{\boldsymbol{\theta}}(s, a), 1)$ over the Q-values q, giving rise to a quadratic weight-regularizer. Another approach of Achiam et al. (2019) uses a different preconditioner which can also be seen as a quadratic weight-regularization. The computation in these approaches is challenging and so far they have only been applied to small networks.

A more serious problem with weight regularization is its ineffectiveness for DNNs. For neural networks, the network outputs depend on the weights in a complex way and exact value of the

weights may not matter much. What matters ultimately is the network outputs (Benjamin et al., 2018), and it is better to directly regularize those. Recently such *functional regularization* has been applied to continual learning (Pan et al., 2020) to avoid catastrophic forgetting, where it has shown to perform much better than weight-regularization. We will resort to this approach in this paper to improve the stability of DQL without using target networks.

3 DEEP Q-LEARNING WITH FUNCTIONAL REGULARIZATION

Deep Q-learning methods are known to be unstable, not recover the true value function, and sometimes even (soft-)diverge in practice (Van Hasselt, 2010; Van Hasselt et al., 2016; 2018). A horde of improvements have been propose to stabilize training, but in this work we will mainly focus on the role played by Target Networks. Target networks stabilize the non-stationarity that arises when combining function approximation and bootstrapping. However, by not using the most up-to-date target Q-value estimates, this also has the effect of slowing down training. Intuitively, since the lagging network is used to estimate the target Q-value, it can be interpreted as a form of *prior*, preventing the Q-value estimates from changing too quickly. In this section, we use this insight to introduce FR Q-learning, a regularized loss to stabilize training while using up-to-date target values. In the process, we help formalize the notion of target networks as a form of functional prior.

3.1 FUNCTIONAL REGULARIZATION

We can see target networks as having a dual role: providing target Q-values used for learning, and stabilizing training. We can separate these two roles: use up-to-date parameters to estimate the target Q-values, and use a functional prior to regularize the Q-values. Following Benjamin et al. (2018), we propose to use an L_2 functional regularizer which penalizes the divergence between the current estimate of Q-value function $Q_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a})$ and an old version of it $Q_{\overline{\boldsymbol{\theta}}}(\mathbf{s}, \mathbf{a})$. This gives us the following loss function where up-to-date parameter $\boldsymbol{\theta}$ are used in the MSPBE loss along with a FR loss to stabilize $\boldsymbol{\theta}$ by regularizing the current state-action pair only,

$$l^{\text{FR}}(\boldsymbol{\theta}) = \frac{1}{2}\Big(R(\mathbf{s}_t, \mathbf{a}_t) + \texttt{stop_grad}\left(\gamma \mathbb{E}\left[Q_{\boldsymbol{\theta}}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})\right]\right) - Q_{\boldsymbol{\theta}}(\mathbf{s}_t, \mathbf{a}_t)\Big)^2 + \frac{\kappa}{2}\Big(Q_{\boldsymbol{\theta}}(\mathbf{s}_t, \mathbf{a}_t) - Q_{\overline{\boldsymbol{\theta}}}(\mathbf{s}_t, \mathbf{a}_t)\Big)^2. \tag{5}$$

The expectation above is taken over the transition and policy, and $\kappa > 0$ is the regularization parameter. By `stop_grad`, we indicate a function that prevents gradients from propagating into the target Q-value estimate.

Critically, unlike Eq. (4), the target Q-value estimates are supplied by the up-to-date Q-network, with the functional prior now separately serving to stabilize the estimate. As with a target network, we can update the functional prior periodically. Overall, l^{FR} is arguably similar in complexity to l^{target}, requiring only an additional function evaluation for $Q_{\boldsymbol{\theta}}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$ and an additional hyperparameter, κ, which we found was not difficult to tune in all our experiments (see Section 4). As will be discussed in Section 3.2, target networks implicitly make a fixed trade-off between speed and stability, whereas κ provides needed flexibility in adjusting this trade-off.

Functional regularization of the Q-network outputs has multiple advantages over regularization of the Q-network parameters (Shashua & Mannor, 2019; Knight & Lerner, 2018; Achiam et al., 2019). DNN parameter regularization is difficult to interpret and tune, as the effect of parameters may vary substantially within and across architectures. While various architectures can be used to parameterize Q-networks, the interpretation of the output Q-value is consistent in all settings. FR provides a straightforward way to decouple target value estimation and stability (via regularization). FR is also compatible with ensembles of Q-networks (Van Hasselt et al., 2016; Osband et al., 2016a;b; Fujimoto et al., 2018).

Similarly to Knight & Lerner (2018), this can also be seen as a KL divergence between two Gaussian process whose mean function is the Q-values at different state-action pairs and covariance function is the identity. It is indeed possible to inject correlations, and also regularize multiple past state-action pairs. Since this increase the computation cost, we did not pursue this in this work.

3.2 INTERPRETING TARGET NETWORKS THROUGH FUNCTIONAL REGULARIZATION

Given that FR also stabilizes estimation through the use of a lagging estimate, we might reasonably ask whether target networks can be interpreted through the lens of FR. To approach this question, we compare the gradients, $\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta})$, arising from each case (Eq. (4) & Eq. (5)) allowing us to see how these learning procedures differ. For a complete derivation, see Appendix A. To simplify notation, we define the Q-value estimate at time t as $Q_{\boldsymbol{\theta}}^{(t)} \equiv Q_{\boldsymbol{\theta}}(\mathbf{s}_t, \mathbf{a}_t)$, the corresponding expected change in the Q-value estimate as $\Delta Q_{\boldsymbol{\theta}}^{(t)} \equiv \mathbb{E}_{P,\pi}[Q_{\boldsymbol{\theta}}^{(t+1)} | \mathbf{s}_t, \mathbf{a}_t] - Q_{\boldsymbol{\theta}}^{(t)}$, and the reward as $R^{(t)} \equiv R(\mathbf{s}_t, \mathbf{a}_t)$. Then, for a given state-action pair, we can compare the two gradients:

$$\nabla_{\boldsymbol{\theta}} l^{\text{FR}}(\boldsymbol{\theta}) = -\big(R^{(t)} + \gamma(Q_{\boldsymbol{\theta}}^{(t)} + \underline{\Delta Q_{\boldsymbol{\theta}}^{(t)}}) - Q_{\boldsymbol{\theta}}^{(t)}\big)\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}^{(t)} + \underline{\kappa}\big(Q_{\overline{\boldsymbol{\theta}}}^{(t)} - Q_{\boldsymbol{\theta}}^{(t)}\big)\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}^{(t)}. \tag{6}$$

Replacing $\underline{\Delta Q_{\boldsymbol{\theta}}^{(t)}}$ with $\Delta Q_{\overline{\boldsymbol{\theta}}}^{(t)}$ and $\underline{\kappa}$ with γ we recover the gradient of $l^{\text{target}}(\boldsymbol{\theta})$:

$$\nabla_{\boldsymbol{\theta}} l^{\text{target}}(\boldsymbol{\theta}) = -\big(R^{(t)} + \gamma(Q_{\boldsymbol{\theta}}^{(t)} + \underline{\Delta Q_{\overline{\boldsymbol{\theta}}}^{(t)}}) - Q_{\boldsymbol{\theta}}^{(t)}\big)\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}^{(t)} + \underline{\gamma}\big(Q_{\overline{\boldsymbol{\theta}}}^{(t)} - Q_{\boldsymbol{\theta}}^{(t)}\big)\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}^{(t)}. \tag{7}$$

We see that, despite differences in their loss formulations, target networks and FR result in nearly identical gradients, with two key exceptions. The first distinction, underlined in orange, is the estimated change in the Q-value. With target networks, this term is supplied by the lagging network parameters, $\overline{\boldsymbol{\theta}}$, and therefore is not up-to-date, whereas with FR, it is supplied by the up-to-date parameters. Thus, FR can propagate updated value estimates faster. The second distinction, underlined in blue, is the weighting on the difference between the lagging and up-to-date Q-value estimates. With target networks, this is the discount factor, γ, whereas with FR, this is a separate hyper-parameter. Using the gradient in Eq. (7), and defining $\hat{Q}_{\boldsymbol{\theta},\overline{\boldsymbol{\theta}}}^{(t)} \equiv Q_{\boldsymbol{\theta}}^{(t)} + \Delta Q_{\overline{\boldsymbol{\theta}}}^{(t)}$, we can derive a loss function, $\widetilde{l}^{\text{target}}$, that yields equivalent gradients as l^{target}, but is written in the FR form:

$$\widetilde{l}^{\text{target}}(\boldsymbol{\theta}) = \frac{1}{2}\Bigg(R(\mathbf{s}_t, \mathbf{a}_t) + \texttt{stop_grad}\left(\gamma \mathbb{E}\left[\hat{Q}_{\boldsymbol{\theta},\overline{\boldsymbol{\theta}}}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})\right]\right) - Q_{\boldsymbol{\theta}}(\mathbf{s}_t, \mathbf{a}_t)\Bigg)^2 + \frac{\gamma}{2}\Bigg(Q_{\boldsymbol{\theta}}(\mathbf{s}_t, \mathbf{a}_t) - Q_{\overline{\boldsymbol{\theta}}}(\mathbf{s}_t, \mathbf{a}_t)\Bigg)^2.$$

From this perspective, target networks effectively perform a special, restricted form of Gaussian FR, with $p(Q_{\boldsymbol{\theta}}) = \mathcal{N}(Q_{\overline{\boldsymbol{\theta}}}, \gamma^{-1})$, i.e., a precision weight that is not under direct control. In contrast, l^{FR} allows us to separately adjust this weight to trade-off stability and learning speed. Overall, this analysis frames target networks as closely related to FR, but with lagging target value estimates and an inability to separately control the regularization weight.

3.3 COMPARISON WITH POLYAK AVERAGING

We mentioned Polyak's averaging as a common technique used to achieve similar results during training. Interestingly, if we assume the target network is an estimate in weight space of the latest Q-network, we can show that Polyak's updates implicitly operate parameter regularization on such estimate. Suppose that after each gradient step (indexed by i) we identify the target network as the solution to the problem:

$$\overline{\boldsymbol{\theta}}_{i+1} = \min_{\overline{\boldsymbol{\theta}}} \frac{1}{2}\|\overline{\boldsymbol{\theta}} - \boldsymbol{\theta}_i\|^2 + \frac{1-\tau}{2\tau}\|\overline{\boldsymbol{\theta}} - \overline{\boldsymbol{\theta}}_i\|^2 \tag{8}$$

where $\boldsymbol{\theta}_i$ is the latest set of weights of the Q-network and $\overline{\boldsymbol{\theta}}_i$ is the previous instance of the target network. The minimizing $\overline{\boldsymbol{\theta}}$ is obtained by computing the gradient of the problem and equating it 0, leading to $\overline{\boldsymbol{\theta}}_{i+1} = (1-\tau)\boldsymbol{\theta}_i + \tau\overline{\boldsymbol{\theta}}$, which is exactly Polyak's averaging. This shows that Polyak's update is indeed a form of weight regularization with respect to the most recent target network. However, weight regularization does not guarantee that the output of the regularized network matches the previous target network. In fact, while this technique has found success in control problems (Lillicrap et al., 2015; Haarnoja et al., 2018a; Fujimoto et al., 2018), hard updating of the parameters is usually preferred for complex DNN architectures (Mnih et al., 2013; Hausknecht & Stone, 2015; Hessel et al., 2018; Kapturowski et al., 2018; Parisotto et al., 2020).

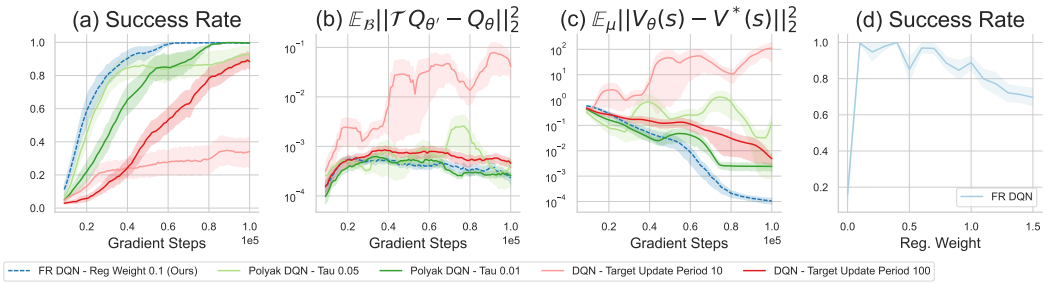


Figure 2: **Four Rooms Comparison**. We compare the performance of different Deep Q-learning algorithms on the Four Rooms environment. The reward position is kept fixed through training (upper-left room), but the agent's position is randomized at the beginning of every episodes to bypass exploration difficulties. In Figure (a), we can see that FR DQN can quickly complete the task and maintain stability, the other methods are either fast but unstable or slow and stable, e.g., Polyak DQN 0.05 and 0.01. Figure (b) highlights the stability of the loss during training for FR DQN and Polyak DQN. Figure (c) shows the ability of FR DQN to recover the true value function, while all the other methods (including Polyak DQN) fail to do so. Figure (d) highlights the stability of FR DQN for a wide range of regularization weights. Regularization from 0.1 to 1. achieves competitive performance.

4 Experiments

We compare performance on the Four Rooms environment and the same subset of Atari games (Bellemare et al., 2013) used by Mnih et al. (Mnih et al., 2013). In all cases, the experimental setup is identical across FR and the baseline target network approach, the only difference being the use of l^{FR} (Eq. (5)) instead of l^{target} (Eq. (4)). The experimental code is based on DQN Zoo (Quan & Ostrovski, 2020), implemented in Jax (Bradbury et al., 2018) and Haiku (Hennigan et al., 2020). Additional experiment details can be found in the Appendix.

4.1 Four Rooms

In order to better understand the effect of FR combined with Deep Q-learning algorithms, we compare a suite of algorithms and hyper-parameters on the Four Rooms environment (Sutton et al., 1999) with neural approximation. The advantage of using this environment (a grid world) over more complex ones is that because we know the underlying transition matrix, the true optimal value function can be computed using tabular methods (which will be referred to as V^*). We can then compare the agent's parametrized value estimate $V_{\boldsymbol{\theta}}(\mathbf{s}) = \max_{\mathbf{a}} Q_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a})$ with the optimal one $V^*(\mathbf{s})$. The goal is set in the upper-left room near the right door, and the initial position is drawn randomly from all other cells excluding the walls position. The position is given through a one hot encoding. The actions are up, down, left and right. The discount factor is 0.99, the reward is always 0 except at the goal state where every action leads to a reward of 1. Specifically all actions from the goal state receive a reward of 1. The episode terminate either once the agent reach the goal or after 500 steps. The environment is implemented using Easy MDP (Ahmed, 2020). The Q-value is parametrized by a MLP with 2 layers of 128 hidden units each. The shared hyper-parameters (learning rate and batch size) were tuned on 5 seeds for DQN to maximize success rate, and then use for all the other methods. The final performances are reported on 20 different seeds after 100k gradient steps (and as many transitions). Finally, the transitions are collected by the ϵ-greedy policy, with ϵ of 0.05. Finally a batch size of 512 transitions and a learning rate of 0.0001 are used. All hyper parameters are available in Table 1

As observed in Figure 2 (a), the target network update period greatly affects performance. On the one hand, frequently updating the target network lead to faster initial performance since the reward is propagated through the space. On the other, hand updating the network too frequently can lead to instability, e.g., larger Bellman error (Fig. 2 (b)). For example, Tau 0.05 initially outperforms Tau 0.01, but becomes unstable, and never reaches optimal performance. FR DQN bypass this issue, by

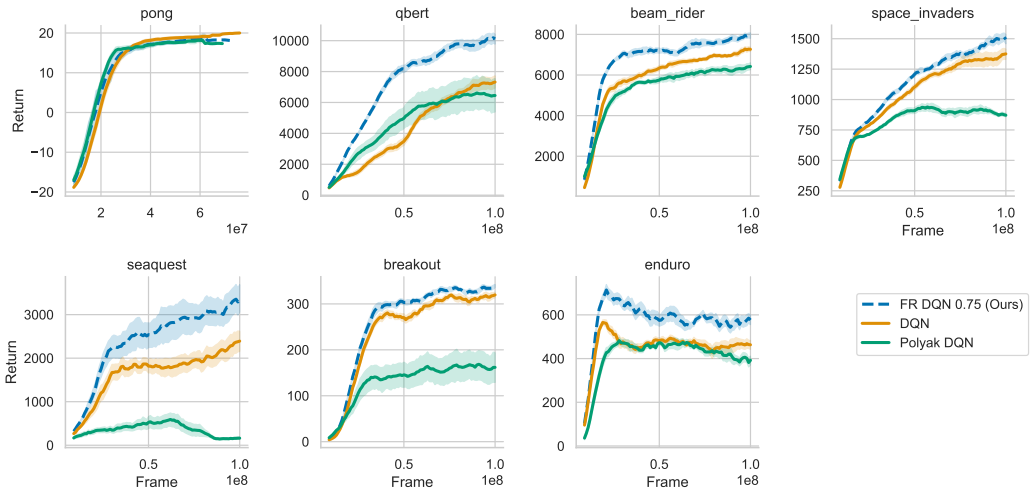


Figure 3: **DQN Atari Comparison**. Performance curves for the subset of Atari games from Mnih et al. (Mnih et al., 2013). The returns are averaged over 10 trials using the ϵ-greedy policy with $\epsilon = 0.05$.

using the previous Q-values estimates exclusively for regularization which allows it to use the most up-to-date estimate for learning.

Q-learning methods should converge to the optimal value function V^*, but in complex settings like Atari games, it is impossible to compute V^* and monitor our distance to the optimum. In our simple Four Rooms setting, however, measuring the accuracy of their approximation is tractable. Interestingly, even if the MSBPE loss is roughly the same for FR DQN and Polyak DQN (Fig. 2 (b)), FR DQN approximates the true value function with much greater accuracy than Polyak DQN (Fig. 2 (c)). This is of particular interest since the MSBPE is the metric that is usually monitored in practice and might not be an accurate measure of the accuracy of the Q-value.

4.2 ATARI

The Arcade Learning Environment (Bellemare et al., 2013) provides a set of benchmark Atari games for evaluating RL algorithms, with rich visual observations and diverse reward functions. Due to the complexity of these environments, complex DNN architectures are preferred and hard value network updates are typically used. To evaluate FR in this setting, we compare with a vanilla Deep Q-network (DQN) (Mnih et al., 2013) and Polyak DQN, allowing us to isolate the benefits of FR. We tuned the hyper-parameter for each method (except DQN as they have already been tuned by the authors) by performing experiments on 2 seeds (only 1 set of global HPs used for every tasks). Once the overall best set of hyper-parameters have been selected, we use 10 additional seeds (and exclude the 2 initial seeds) to compare the algorithms. All hyper parameters are available in Table 2. For experiments, we use the same suite of representative environments from Mnih et al. (Mnih et al., 2013).

DQN Atari performance results are presented in Fig. 3. Across all environments, FR matches or exceeds DQN in terms of sample efficiency and final performance at 100M steps. Each seed takes 48 hours to run on our resources for a total of 14400 hours of gpus usage. FR's ability to use up-to-date target Q-values also plays a significant role in improving performance over target networks.

Double DQN Since the introduction of Target Networks, there has been numerous techniques to stabilize the training and improve the performance of DQL methods (Hessel et al., 2018). While it is outside of the scope of this paper to combine FR with each and every DQN improvements, we show that it is fairly easy to combine FR with double DQN (DDQN) (Van Hasselt et al., 2016). DDQN decouples the action selection from the action evaluation to provide a target during training and counter the maximum bias. It has been shown an effective way to stabilise DQN and prevent

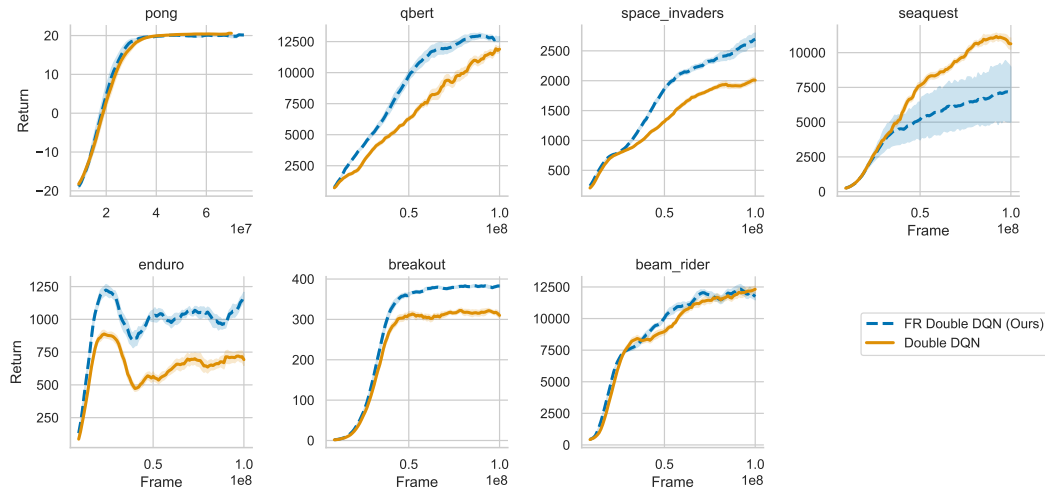


Figure 4: **Double DQN Atari Comparison**. Performance curves for the subset of Atari games from Mnih et al. (Mnih et al., 2013). The baseline (dotted blue line) and 4 different FR weights (solid lines) average returns over 10 trials using ϵ-greedy with $\epsilon = 0.05$.

(soft-)divergence. The results presented in Fig. 4 clearly demonstrates that FR can be combined with DDQN to achieves even higher performances on this subset of Atari games.

5 Related Work

Multiple previous works have investigated improving value estimation through various constraints and regularization. Farahmand et al. (massoud Farahmand et al., 2009) is perhaps the closest work to our own, functionally regularizing the L_2 norm of the Q-value estimate, i.e., penalizing $\kappa||Q_{\boldsymbol{\theta}}||^2$. This can be interpreted as using a fixed Gaussian prior, whereas we use a periodically updated set of parameters to provide a moving prior. Shao et al. (2020) propose adding an additional backward MSPBE loss to the next Q-value to stabilise training and remove the need for a target network. Other works have sought to regularize the Q-value estimator by constraining parameter updates, e.g., through regularization (Farebrother et al., 2018), conjugate gradient methods (Schulman et al., 2015b), pre-conditioning the gradient updates (Knight & Lerner, 2018; Achiam et al., 2019), or using Kalman filtering (Shashua & Mannor, 2019; 2020). As argued in Sections 2 and 3, functional regularization, as opposed to parameter regularization, is more appropriate in this setting, as parameter regularization does not necessarily imply that the Q-value estimates remain stable during training. Additional works have regularized the Q-value estimator using temporal regularization (Xu et al., 2017; Thodoroff et al., 2018; 2019), smoothing value estimates over trajectories. In contrast, FR Q-learning regularizes value estimates across training updates to maintain stability. Other works have proposed methods for regularizing policy optimization using trust regions (Peters et al., 2010; Schulman et al., 2015a), proximal methods (Schulman et al., 2017), entropy/KL penalties (Haarnoja et al., 2018a; Abdolmaleki et al., 2018b;a), and natural policy gradients (Kakade, 2001). As these methods are applied to a separately parameterized policy, they can readily be combined with FR Q-learning. Finally, while we apply FR to improve value estimation in RL, FR has also been used in a variety of other settings, such as supervised learning (Benjamin et al., 2018; Khan et al., 2019), unsupervised learning (Garg & Liang, 2020), and continual learning (Pan et al., 2020). Similarly, works outside of RL, e.g., semi-supervised learning (He et al., 2020), have also used lagging networks to stabilize training. More fully connecting ideas from these various approaches may prove fruitful in multiple areas of machine learning.

Other previous works have sought to address orthogonal issues with DQL algorithms, and, in principle, these could be combined with FR Q-learning. For instance, Kim et al. (2019) removed the need for target network on Atari games by using the mellow max operator as an alternative to the max operator used in bootstrapping. This new operator could easily be used in combination with

FR. As mentionned above, Double Deep Q-learning (**?**Van Hasselt et al., 2016) addresses positive bias in the value estimate (Thrun & Schwartz, 1993) by using two Q-networks, with each target network providing target values for the other. Fujimoto et al. (Fujimoto et al., 2018) extend and apply this technique to continuous control, instead using the minimum of the two target networks in the target value. FR can also be combined with other forms of TD-learning, such as Huber (Fujimoto et al., 2018) or logistic (Serrano et al., 2021) loss functions. Other works have proposed methods for regularizing policy optimization using trust regions (Peters et al., 2010; Schulman et al., 2015a), proximal methods (Schulman et al., 2017), entropy/KL penalties (Haarnoja et al., 2018a; Abdolmaleki et al., 2018b;a), and natural policy gradients (Kakade, 2001). As these methods are applied to a separately parameterized policy, they can readily be combined with FR Q-learning.

6 Discussion

Using functional regularization, we propose the FR Bellman error as an alternative to the Bellman error. The Bellman error requires Target Networks to stabilize the training of DQL methods due to the non-stationary nature of the regression problem. Like the Bellman error, the FR Bellman error uses a lagging set of parameters to stabilize estimation. However, critically, this is decoupled from target value estimation, instead taking the form of a functional prior. In this way, we can 1) use up-to-date target value estimates, thereby propagating reward information more quickly, and 2) separately control the degree of stabilization via the FR weight, κ. Indeed, by comparing the gradients resulting from MSPBE losses with target networks and FR, we noted that target networks implicitly perform a form of FR, but with lagging target values and a non-independent weight ($\kappa = \gamma$). This helps illuminate why target networks stabilize training in practice, while also highlighting their drawbacks.

We demonstrate the stability of of FR DQN on the Four Rooms environment by showing that it is the only method that can recover the true optimal value function, and show that we achieve higher performance than DQN and Polyak DQN. Then we showed that FR DQN consistently outperform DQN and Polyak DQN on a subset of the Atari suite. Finally we showed that FR DQN can be combined with DDQN to further improve performance. Thus, we conclude that FR provides a conceptually simple, easy to implement, and effective method for improving DQL methods.

While the FR Bellman error requires an additional weighting parameter, κ, we noted that the performance does not change dramatically for small change in κ. We see κ as providing needed flexibility in setting the degree of regularization, unlike Target Networks, which cannot separately control this aspect. In our experiments, we used a fixed κ throughout training, however, future work could investigate methods for automatically adjusting κ during training (Haarnoja et al., 2018b) or conditioning it on individual state-action pairs. Likewise, other forms of FR losses, corresponding to alternative functional priors, may yield further improvement. Nevertheless, the FR Q-learning approach that we have presented here performs well across a wide range of environments, providing a drop-in replacement for target networks.

References

Abbas Abdolmaleki, Jost Tobias Springenberg, Jonas Degrave, Steven Bohez, Yuval Tassa, Dan Belov, Nicolas Heess, and Martin Riedmiller. Relative entropy regularized policy iteration. *arXiv preprint arXiv:1812.02256*, 2018a.

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018b.

Joshua Achiam, Ethan Knight, and Pieter Abbeel. Towards characterizing divergence in deep q-learning. *arXiv preprint arXiv:1903.08894*, 2019.

Zafarali Ahmed. Easy MDPs implemented in a gym like interface with access to transition dynamics., 2020. URL `https://github.com/zafarali/emdp`.

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

Ari S Benjamin, David Rolnick, and Konrad Kording. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Robert Goodell Brown. *Statistical forecasting for inventory control*. McGraw/Hill, 1959.

Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1587–1596. PMLR, 2018.

Everette S Gardner Jr. Exponential smoothing: The state of the art. *Journal of forecasting*, 4(1): 1–28, 1985.

Siddhant Garg and Yingyu Liang. Functional regularization for representation learning: A unified theoretical perspective. *arXiv preprint arXiv:2008.02447*, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1861–1870. PMLR, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.

Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020. URL `http://github.com/deepmind/dm-haiku`.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Charles C. Holt, Franco Modigliani, John F. Muth, and Herbert A. Simon. *Planning Production, Inventories, and Work Force*. Englewood Cliffs, 1960.

Sham M Kakade. A natural policy gradient. *Advances in neural information processing systems*, 14, 2001.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

Mohammad Emtiyaz E Khan, Alexander Immer, Ehsan Abedi, and Maciej Korzepa. Approximate inference turns deep networks into gaussian processes. In *Advances in neural information processing systems*, pp. 3094–3104, 2019.

Seungchan Kim, Kavosh Asadi, Michael Littman, and George Konidaris. Deepmellow: removing the need for a target network in deep q-learning. In *Proceedings of the Twenty Eighth International Joint Conference on Artificial Intelligence*, 2019.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Ethan Knight and Osher Lerner. Natural gradient deep q-learning. *arXiv preprint arXiv:1803.07482*, 2018.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Amir massoud Farahmand, Mohammad Ghavamzadeh, Csaba Szepesvári, and Shie Mannor. Regularized fitted q-iteration for planning in continuous-space markovian decision problems. In *2009 American Control Conference*, pp. 725–730. IEEE, 2009.

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

Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pp. 4026–4034, 2016a.

Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In *International Conference on Machine Learning*, pp. 2377–2386, 2016b.

Pingbo Pan, Siddharth Swaroop, Alexander Immer, Runa Eschenhagen, Richard E Turner, and Mohammad Emtiyaz Khan. Continual deep learning by functional regularisation of memorable past. *arXiv preprint arXiv:2004.14070*, 2020.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, pp. 7487–7498. PMLR, 2020.

Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 2010.

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

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Joan Bas Serrano, Sebastian Curi, Andreas Krause, and Gergely Neu. Logistic q-learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 3610–3618. PMLR, 2021.

Lin Shao, Yifan You, Mengyuan Yan, Qingyun Sun, and Jeannette Bohg. Grac: Self-guided and self-regularized actor-critic. *arXiv preprint arXiv:2009.08973*, 2020.

Shirli Di-Castro Shashua and Shie Mannor. Trust region value optimization using kalman filtering. *arXiv preprint arXiv:1901.07860*, 2019.

Shirli Di-Castro Shashua and Shie Mannor. Kalman meets bellman: Improving policy evaluation through value tracking. *arXiv preprint arXiv:2002.07171*, 2020.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Pierre Thodoroff, Audrey Durand, Joelle Pineau, and Doina Precup. Temporal regularization in markov decision process. *arXiv preprint arXiv:1811.00429*, 2018.

Pierre Thodoroff, Nishanth Anand, Lucas Caccia, Doina Precup, and Joelle Pineau. Recurrent value functions. *arXiv preprint arXiv:1905.09562*, 2019.

Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

Hado Van Hasselt. Double q-learning. *Advances in neural information processing systems*, 23: 2613–2621, 2010.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

Zhongwen Xu, Joseph Modayil, Hado van Hasselt, Andre Barreto, David Silver, and Tom Schaul. Natural value approximators: Learning when to trust past estimates. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 2117–2125, 2017.

A Appendix

A.1 Q-learning with Target Network

Letting $R^{(t)} \equiv R(\mathbf{s}_t, \mathbf{a}_t)$, $Q^{(t)} \equiv Q(\mathbf{s}_t, \mathbf{a}_t)$ and $\Delta Q^{(t)}_{\bar{\boldsymbol{\theta}}} \equiv \mathbb{E}[Q^{(t+1)}_{\bar{\boldsymbol{\theta}}} - Q^{(t)}_{\bar{\boldsymbol{\theta}}} | \mathbf{s}_t, \mathbf{a}_t] = \mathbb{E}[Q^{(t+1)}_{\bar{\boldsymbol{\theta}}} | \mathbf{s}_t, \mathbf{a}_t] - Q^{(t)}_{\bar{\boldsymbol{\theta}}}$

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} l^{\text{target}}(\boldsymbol{\theta}) &= -\big(R^{(t)} + \gamma \mathbb{E}_{P,\pi}[Q^{(t+1)}_{\bar{\boldsymbol{\theta}}}] - Q^{(t)}_{\boldsymbol{\theta}}\big)\nabla_{\boldsymbol{\theta}} Q^{(t)}_{\boldsymbol{\theta}} \\
&= -\big(R^{(t)} + \gamma(Q^{(t)}_{\bar{\boldsymbol{\theta}}} + \Delta Q^{(t)}_{\bar{\boldsymbol{\theta}}}) - Q^{(t)}_{\boldsymbol{\theta}}\big)\nabla_{\boldsymbol{\theta}} Q^{(t)}_{\boldsymbol{\theta}} \\
&= -\big(R^{(t)} + \gamma(Q^{(t)}_{\bar{\boldsymbol{\theta}}} + \Delta Q^{(t)}_{\bar{\boldsymbol{\theta}}}) - (1-\gamma+\gamma)Q^{(t)}_{\boldsymbol{\theta}}\big)\nabla_{\boldsymbol{\theta}} Q^{(t)}_{\boldsymbol{\theta}} \\
&= -\big(R^{(t)} + \gamma Q^{(t)}_{\bar{\boldsymbol{\theta}}} + \gamma \Delta Q^{(t)}_{\bar{\boldsymbol{\theta}}} - (1-\gamma)Q^{(t)}_{\boldsymbol{\theta}} - \gamma Q^{(t)}_{\boldsymbol{\theta}}\big)\nabla_{\boldsymbol{\theta}} Q^{(t)}_{\boldsymbol{\theta}} \\
&= -\big(R^{(t)} + \gamma \Delta Q^{(t)}_{\bar{\boldsymbol{\theta}}} - (1-\gamma)Q^{(t)}_{\boldsymbol{\theta}}\big)\nabla_{\boldsymbol{\theta}} Q^{(t)}_{\boldsymbol{\theta}} + \gamma\big(Q^{(t)}_{\bar{\boldsymbol{\theta}}} - Q^{t}_{\boldsymbol{\theta}}\big)\nabla_{\boldsymbol{\theta}} Q^{(t)}_{\boldsymbol{\theta}} \\
&= -\big(R^{(t)} + \gamma \Delta Q^{(t)}_{\bar{\boldsymbol{\theta}}} - Q^{(t)}_{\boldsymbol{\theta}} + \gamma Q^{(t)}_{\boldsymbol{\theta}}\big)\nabla_{\boldsymbol{\theta}} Q^{(t)}_{\boldsymbol{\theta}} + \gamma\big(Q^{(t)}_{\bar{\boldsymbol{\theta}}} - Q^{(t)}_{\boldsymbol{\theta}}\big)\nabla_{\boldsymbol{\theta}} Q^{(t)}_{\boldsymbol{\theta}} \\
&= -\big(R^{(t)} + \gamma(Q^{(t)}_{\boldsymbol{\theta}} + \underline{\Delta Q^{(t)}_{\bar{\boldsymbol{\theta}}}}) - Q^{(t)}_{\boldsymbol{\theta}}\big)\nabla_{\boldsymbol{\theta}} Q^{(t)}_{\boldsymbol{\theta}} + \underline{\gamma}\big(Q^{(t)}_{\bar{\boldsymbol{\theta}}} - Q^{(t)}_{\boldsymbol{\theta}}\big)\nabla_{\boldsymbol{\theta}} Q^{(t)}_{\boldsymbol{\theta}}
\end{aligned}
$$

A.2 Q-LEARNING WITH FUNCTIONAL REGULARIZATION

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} l^{\text{FR}}(\boldsymbol{\theta}) &= -\big(R^{(t)} + \gamma \mathbb{E}_{P,\pi}[Q_{\boldsymbol{\theta}}^{(t+1)}] - Q_{\boldsymbol{\theta}}^{(t)}\big)\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}^{(t)} + \kappa\big(Q_{\overline{\boldsymbol{\theta}}}^{t} - Q_{\boldsymbol{\theta}}^{(t)}\big)\nabla Q_{\boldsymbol{\theta}}^{(t)} \\ &= -\big(R^{(t)} + \gamma(Q_{\theta}^{(t)} + \underline{\Delta Q_{\theta}^{(t)}}) - Q_{\boldsymbol{\theta}}^{(t)}\big)\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}^{(t)} + \kappa\big(Q_{\overline{\boldsymbol{\theta}}}^{(t)} - Q_{\boldsymbol{\theta}}^{t}\big)\nabla Q_{\boldsymbol{\theta}}^{(t)}\end{aligned}$$

B APPENDIX

B.1 FOUR ROOMS

Table 1: Four Rooms Hyperparameters

Hyperparameter	DQN	Polyak DQN	FR DQN
Learning rate	1e-3, **1e-4**	**1e-4**	**1e-4**
Batch size	32, **512**	512	512
Target Update Period	10, 25, 50, 100	1	50, 500, **5000**
τ	-	5e-1, 1e-1, 5e-2 1e-2	-
κ	-	-	[**0.1**, . . . , 0.9]
Deep Neural Network	[128, 128]	[128, 128]	[128, 128]
ϵ	0.05	0.05	0.05

B.2 ATARI

Most of the hyper-parameters have been kept from the defaut DQN and Double DQN respectively. Here is a list of the tuning that has been performed and the resulting best hyper-parameters.

Table 2: Atari Hyperparameters

Hyperparameter	Polyak DQN	FR DQN	FR Double DQN
τ	5e-5, 2.5e-5, **1e-5**, 5e-6	-	-
Target Update Period	100	-	-
κ	-	0.25, 0.5, **0.75**	**0.75**