Terminology

Mathematical notations and definitions

- $x : t : y = x + t(y - x)$
- $x \setminus y \setminus z = \frac{y-x}{z-x}$
- $x [y] z = \min(z, \max(x, y))$
- $\lambda = \min_a Q(s,a) \setminus \aleph \setminus \max_a Q(s,a)$
- $G_t = \sum_{t'=t}^{\infty} \gamma^{t'} r_{t'}$
- $Q(s,a) = r(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim (s,a)} V(s')$
- $V(s) = \mathop{\mathbb{E}}_{a' \sim \pi(s')} Q(s', a')$

Acronyms

AR: Aspiration Rescaling

DL: Deep Learning

DNN: Deep Neural Network

DRL: Deep Reinforcement Learning

DQN: Deep Q-Network

LRA: Local Relative Aspiration (represented by λ)

MAB: Multi-Armed Bandit

RL: Reinforcement Learning

SB3: Stable Baselines 3

Satisficing Reinforcement learning

Clément Dumas, under the supervision of Jobst Heitzig

August 2023

Abstract

Reinforcement Learning (RL) predominantly trains agents by maximizing the expected return of a reward function, which is an approximation of the agent's true utility. However, the relationship between the reward and utility is not always straightforward, leading to scenarios where reward maximization does not necessarily yield high utility. Such scenarios are termed "reward hacking". This report delves into the concept of Satisficing Reinforcement Learning as a potential solution to this issue.

Satisficing agents aim to reach a specific aspiration level, rather than solely maximizing the reward. This concept was explored through the implementation of variants of the Q-learning algorithm and its Deep Learning equivalent, with the primary goal of demonstrating the feasibility of transforming maximizing agents into satisficers.

Preliminary results show promise in multi-armed Bandit environments. However, the current implementation lacks the stability needed to be a viable alternative to its maximizing counterpart.

1 Preliminaries

1.1 Reinforcement Learning

Reinforcement Learning (RL) is a subfield of machine learning that focuses on training *agents* to make optimal decisions through interaction with their *environment*. The core idea of RL is that an agent learns from its experiences by performing actions within an environment and receiving feedback in the form of a scalar *reward* signal. The agent's goal is to learn a policy, a probability distribution over actions for each state, that maximizes the cumulative reward over time.

In the RL paradigm, an agent interacts with an environment over a sequence of discrete time steps. At each time step t, the agent observes the current state s_t of the environment, selects an action a_t based on its current policy π, and executes this action. The environment then transitions to a new state s_{t+1} and provides the agent with a reward signal r_{t+1}. The agent's objective is to learn a policy that maximizes the expected sum of these reward signals, also known

as the return:

$$G_t = \sum_{t'=t+1}^{\infty} \gamma^{t'} r_{t'} \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor that determines the present value of future rewards.

The RL problem is typically formalized as a Markov Decision Process (MDP) $\langle S, r, T, \gamma \rangle$, defined by a set of states S, a set of actions A, a transition function $T(s, a, s') = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$, and a reward function $r(s, a) = \mathbb{E}(r_{t+1} | s_t = s, a_t = a)$.

1.2 Q-learning

Q-learning is a widely used and studied RL algorithm introduced by Watkins and Dayan [1992]. It estimates the action-value function, denoted as $Q(s, a)$, for each state-action pair. The action-value function represents the expected return for choosing a particular action in a given state and following a specific policy thereafter. Likewise, we can also define the state-value function, denoted as $V(s)$, representing the expected return starting from state s:

$$V^{\pi}(s, a) = \mathbb{E}_{\pi}(G_t | s_t = s) \tag{2}$$

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}(G_t | s_t = s, a_t = a) \tag{3}$$

Both V^{π} and Q^{π} are governed by the Bellman equation as follows:

$$Q^{\pi}(s, a) = r(s, a) + \gamma \mathop{\mathbb{E}}_{s' \sim (s,a)} V^{\pi}(s') \tag{4}$$

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a \sim \pi(s)} Q^{\pi}(s, a) \tag{5}$$

where π is the policy, s is the state, a is the action, and γ is the discount factor.

In Q-learning, a Q-table representing the Q values of each (state, action) pair is randomly initialized. Then, the agent interacts with the environment, using an ε-greedy policy, which selects the action with the highest estimated Q-value with probability $1 - \varepsilon$ and a random action with probability ε. This policy encourages a balance between exploration of the environment and exploitation of the current knowledge. The agent continuously updates its Q-table based on the rewards it receives and the states it explores. The update rule for Q-learning when, at time t, the agent chooses action a_t in state s_t and ended up in state s_{t+1} is given by:

$$y_t = r_t + \gamma \max_{a} Q(s_{t+1}, a) \tag{6}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t [y_t - Q(s_t, a_t)] \tag{7}$$

where y is the target Q value, α_t is the learning rate, and r_t is the reward received after taking action a_t in state s_t. It is proven that over certain condition on α, the Q-table will converge to the optimal Q-function, $Q^*(s, a)$, which is the maximum expected return achievable by the optimal policy. The whole Q-learning algorithm is shown in algorithm 1 of appendix A.

1.3 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a fusion of traditional Reinforcement Learning and Deep Learning. In DRL, deep neural networks (DNNs) are employed as function approximators to estimate value functions or policies directly from raw inputs. This eliminates the need for manual feature engineering that is typically required in traditional RL, allowing DRL agents to autonomously learn representations from raw data.

The primary motivations for using DNNs in RL are:

- **Capability to Handle High Dimensionality:** DNNs can automatically extract relevant features from high-dimensional inputs, such as images, making them ideal for tasks like video game playing, robotics, and autonomous driving.
- **Generalization:** DNNs have the ability to generalize across similar states, thereby reducing the number of samples required to learn an effective policy.
- **Scalability:** DNNs can be scaled up to achieve better performance, which is particularly beneficial for challenging RL tasks.

A notable example of DRL is the Deep Q-Network (DQN) algorithm introduced by Mnih et al. [2013], which utilizes a neural network to approximate the Q-function. Unlike traditional Q-learning that maintains a table of Q-values, DQN trains a neural network to predict Q-values for all possible actions in a given state.

Given a state s, the neural network produces a vector of action values $Q(s, \cdot; \theta)$, where θ are the parameters of the network. The DQN learning rule is inspired by the Q-learning update:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{8}$$

$$L(\theta) = \mathbb{E}[(Q(s, a; \theta) - y)^2] \tag{9}$$

Where θ^- represents the parameters of a target network, which are periodically copied from the main network. This target network stabilizes learning by providing fixed Q-value targets. At each training step, gradient descent is applied with a learning rate α to find the parameters that minimize the loss.

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$$

To further stabilize DRL training, experience replay is often used. Instead of learning from the most recent experience, the agent stores its experiences in a replay buffer and samples mini-batches from this buffer to train the neural network. This approach breaks the temporal correlation of consecutive experiences and provides a more diverse set of training samples. Putting everything together, we get the algorithm 2 of appendix A.

1.4 Reward hacking

However, RL, including its DRL variant, is not without challenges. One significant issue is the alignment problem, where the reward function used to train the agent does not perfectly represent the true objective, leading to undesired behaviors. One kind of failure is called *reward hacking*, where the agent finds ways to maximize the reward without genuinely achieving the intended goal.

Designing rewards that closely align with the real utility remains a significant challenge in RL, as highlighted by Amodei et al. [2016]. Theoretical work by Zhuang and Hadfield-Menell [2021] demonstrates that maximizing the return when the reward function does not capture the real utility can result in arbitrarily low utility across certain hypotheses. Skalse et al. [2022] further illustrates that finding a proxy aligned with the true utility in finite Markov decision processes (MDP) is highly unlikely. Empirical instances of reward hacking in RL have also been documented, as compiled by Krakovna et al. [2020]. An example of such failure is shown in 1

Figure 1: The agent learned to circle to collect the green bonuses instead of racing.

As we train increasingly advanced autonomous AI systems, it is crucial to ensure that their objectives remain aligned with ours. This alignment becomes even more critical as optimal policies tend to seek power, as noted by Turner et al. [2023]. Skalse et al. suggest that one potential solution to this problem could be to explore approaches not based on optimizing reward functions.

Our work on satisficing attempts to address this underexplored approach to mitigating reward hacking. Several other strategies have been proposed to address this issue. For instance, Brown et al. [2020] proposes detecting reward hacking in the same way as detecting unseen strategies: as out-of-distribution. While this approach shows promise in detecting undesired or surprising behavior, it does not provide a solution for training agents to avoid over-optimizing their proxies while yielding poor utility. Another strategy involves incorporat-

ing human input, as seen in Frye and Feige [2019], which builds on the work of Christiano et al. [2017]. However, these methods are not perfect as the feedback given to the agent is not directly provided by a human due to efficiency concerns, but by a reward model trained on human preferences. These reward models are still proxies of real human beliefs and depend on the human's ability to oversee the agent's behavior. Both these problems have been empirically observed. Gao et al. [2022] found that over-optimizing a specific human reward model leads to worse performance, and Christiano et al. reported that some humans were deceived by a robot arm that pretended to grab a Lego by moving between the Lego and the camera. While there are ongoing efforts to enhance human oversight, such as Bowman et al. [2022], it remains unclear whether these issues can be fully resolved.

1.5 Satisficing

The term of *satisficing* was first introduced in economics by Simon [1956]. According to Simon's definition, a satisficing agent with an aspiration $\aleph$[1] will search through available alternatives, until it finds one that give it a return greater than $\aleph$. Satisficing objectives have been formalized in different ways by Reverdy et al. [2016] for Multi-Armed Bandit (MAB) environments, and they propose several algorithms with logarithmic or finite regret for each objective. In those environments, satisficing has been used by Tamatsukuri and Takahashi [2019] and Russo et al. [2017] to find a close to optimal policy faster.

Goodrich and Quigley [2004] use a satisficing version of Q learning to avoid taking risks during exploration. There is no aspiration in this algorithm, instead, it estimates the gain and the cost of an action and only consider it if the gain is greater than the cost. Thus, it's not addressing our maximizing reward concern.

Little research has been undertaken on implementing satisficing RL algorithms that generalize to environments beyond the MAB setting. Moreover, some work that has adopted the satisficing approach has actually used it to maximize rewards, which is precisely what we aim to avoid. While the definition by Simon might offer interesting insights within the RL and DRL frameworks, it fails to address our concerns about maximization. By his definition, a maximizer would be classified as a satisficer.

Therefore, we introduce a novel concept of a *satisficing* agent. In our study, an $\aleph$-satisficing agent or $\aleph$-satisficer does not seek gains exceeding $\aleph$. Instead, it aims to achieve an expected gain of $\aleph$:

$$\mathbb{E}\, G_0 = \aleph$$

2 Local Relative Aspiration

In the context of Q-learning, both the maximization and minimization policies (i.e., selecting $\operatorname{argmax}_a Q(s,a)$ or $\operatorname{argmin}_a Q(s,a)$) can be viewed as the

[1] read "aleph", the first letter of the Hebrew alphabet

extremities of a continuum of LRA^λ policies, where $\lambda \in [0, 1]$ denotes the Local Relative Aspiration (LRA). At time t, such a policy samples an action a from a probability distribution $\pi(s_t) \in \Delta(A)$, satisfying the relation:

$$\mathop{\mathbb{E}}_{a \sim \pi(s_t)} Q(s_t, a) = \min_a Q(s_t, a) : \lambda : \max_a Q(s_t, a) = \aleph_t \tag{10}$$

Here, $x : u : y$ represents the interpolation between x and y with a factor u, defined as:

$$x : u : y = x + u(y - x)$$

With this formulation, the agent satisfices $\aleph_t$ at each time t. Setting $\lambda = 0$ corresponds to minimization, while $\lambda = 1$ corresponds to maximization.

The most direct method to determine π is to deterministically select a such that $Q(s, a) = \aleph_t$. If no such a exists, we can define π to select a_t^+ with probability p and a_t^- with probability $1 - p$, where:

$$a_t^+ = \mathop{\mathrm{argmin}}_{a: Q(s_t, a) > \aleph_t} Q(s_t, a) \tag{11}$$

$$a_t^- = \mathop{\mathrm{argmax}}_{a: Q(s_t, a) < \aleph_t} Q(s_t, a) \tag{12}$$

$$p = Q(s_t, a_t^-) \setminus \aleph_t \setminus Q(s_t, a_t^+,) \tag{13}$$

where $x \setminus y \setminus z$ denotes the interpolation factor of y relative to the interval between x and z. It's defined as:

$$x \setminus y \setminus z = \frac{y - x}{z - x}$$

Here, p is selected to satisfy (10).

What elevates the significance of this method is its similarity to Q-Learning/DQN, especially since the update target resembles those in (7) and (8):

$$y = r_t + \gamma \aleph_{t+1} \tag{14}$$

By using this update target and replacing $a \leftarrow \mathrm{argmax}_a Q(s, a)$ by $a \leftarrow a \sim \pi$ with π defined above, we create two variations of Q learning and DQN called *LRA Q-learning* and *LRA-DQN*. Furthermore, LRA Q-learning retains certain properties of Q-learning. Another intern proved that for all λ, Q converges to a function Q^λ, with $Q^1 = Q^*$.

Nevertheless, a direct relationship between the value of λ and the agent performance across different environments remains elusive. If the actions in the environment only affect the reward i.e. $\forall s, s', a, a'$

$$\mathbb{P}(s'|s, a) = \mathbb{P}(s'|s, a')$$

for example in an iterated MAB setup, $\mathbb{E}_\lambda G_0$ is linear in respect to λ:

$$\mathbb{E}_{\pi^\lambda} G_0 = \mathbb{E}_{\pi^0} G_0 : \lambda : \mathbb{E}_{\pi^1} G_0$$

However as soon as the distribution of the next state is influenced by a, which is the case in most environments, we can loose this property as shown in figure 2.

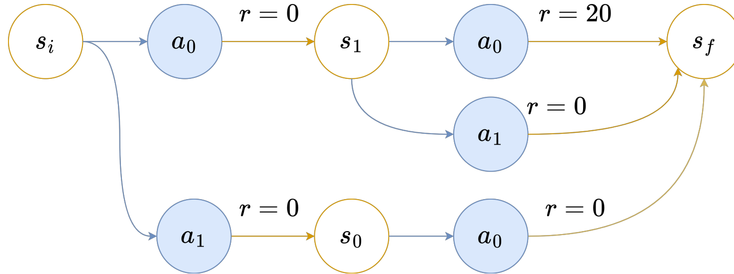


Figure 2: In this MDP, where s_i is the initial state and s_f the terminal state, $\mathbb{E}_{\pi^\lambda} G_0 = 20\lambda^2$

3 Aspiration Propagation

The inability to robustly predict agent performance for a specific value of λ show that we can not build an $\aleph$-satisficer with LRA alone. The only certainty we have is that if $\lambda < 1$, the agent will not maximize. However, it might be so close to maximizing that it attempts to exploit the reward system. This uncertainty motivates the transition to a global aspiration satisficing algorithm. Instead of specifying the LRA, we aim to directly specify the agent's aspiration, $\aleph_0$, representing the return we expect the agent to achieve. The challenge then becomes how to propagate this aspiration from one timeframe to the next. It is crucial that aspirations remain *consistent* as per (15), ensuring recursive fulfillment of $\aleph_0$.

$$\aleph_t = \mathop{\mathbb{E}}_{a_t} \mathop{\mathbb{E}}_{(r_t, s_{t+1})} (r_t + \gamma \aleph_{t+1}) \tag{15}$$

A direct approach to ensure consistent aspiration propagation would be to employ a *hard* update:

$$\aleph_{t+1} = (\aleph_t - r_t)/\gamma \tag{16}$$

However, this method of updating aspirations does not guarantee that the aspiration remains *feasible* as defined in (17).

$$\min_a Q(s_t, a) \leqslant \aleph_t \leqslant \max_a Q(s_t, a) \tag{17}$$

Ensuring feasibility is paramount because, at each step, the agent must be able to select a policy that meets the aspiration. If the aspiration is consistently feasible, applying (15) to $t = 0$ guarantees that $\mathbb{E}\, G_0 = \aleph_0$.

To elucidate the importance of feasibility and demonstrate why hard updates might be inadequate (since they do not ensure feasibility), consider the Markov Decision Process (MDP) illustrated in figure 3. Assume the agent is parameterized by $\gamma = 1$ and $\aleph_0 = 10$, and possesses a comprehensive understanding of the reward distribution.

Upon interacting with the environment and reaching s_0 after its initial action, the agent's return is 15, leading to a new aspiration of $\aleph = -5$. This aspiration is no longer feasible, culminating in an episode end with $G_0 = 15$.

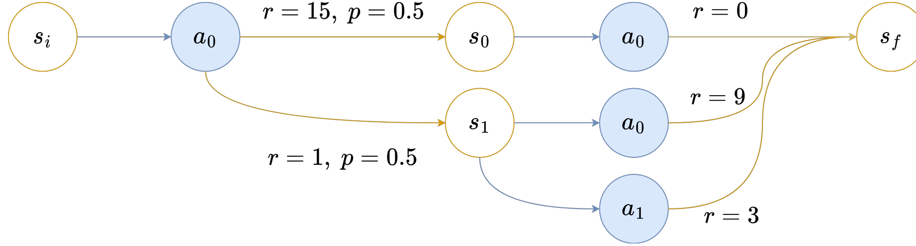


Figure 3: s_i is the initial state and s_f is the terminal state.

If the agent reaches s_1, then $\aleph_1 = 9$. Consequently, the agent selects a_0 and receives $r = 9$, ending the episode with $G = 10$. As a result, $\mathbb{E}\, G_0 = 12.5 \neq \aleph_0$.

3.1 Aspiration Rescaling

To address the aforementioned challenges, we introduce *Aspiration Rescaling* (AR). This approach ensures that the aspiration remains both *feasible* (as per (17)) and *consistent* (as per (15)) during propagation. To achieve this, we introduce two additional values, $\overline{Q}$ and $\underline{Q}$:

$$\overline{Q}(s,a) = \mathop{\mathbb{E}}_{(r,s')\sim(s,a)} r + \overline{V}(s')$$

$$\underline{Q}(s,a) = \mathop{\mathbb{E}}_{(r,s')\sim(s,a)} r + \underline{V}(s')$$

where

$$\overline{V}(s) = \max_a Q(s,a)$$

$$\underline{V}(s) = \min_a Q(s,a)$$

These values provide insight into the potential bounds of subsequent states. The AR strategy computes λ_{t+1}, the LRA for the next step, at time t, rather than directly determining $\aleph_{t+1}$. By calculating an LRA, we ensure the aspiration will be feasible in the next state. Furthermore, by selecting it such that

$$\mathop{\mathbb{E}}_{(r_t,s_{t+1})\sim(s_t,a)} \min_a Q(s_{t+1},a) : \lambda_{t+1} : \max_a Q(s_{t+1},a) - \frac{\aleph_t - r_t}{\gamma} = 0 \tag{18}$$

we ensure consistency. More precisely, at each step, the algorithm propagates its aspiration using the AR formula:

$$\lambda_{t+1} = \underline{Q}(s_t,a_t) \setminus Q(s_t,a_t) \setminus \overline{Q}(s_t,a_t) \tag{19}$$

Subsequently, we can compute the aspiration using the new LRA:

$$\aleph_{t+1} = \min_a Q(s_{t+1},a) : \lambda_{t+1} : \max_a Q(s_{t+1},a) \tag{20}$$

which fulfills (18), as depicted in figure 4. The formal proof of the algorithm's consistency is provided in appendix C.

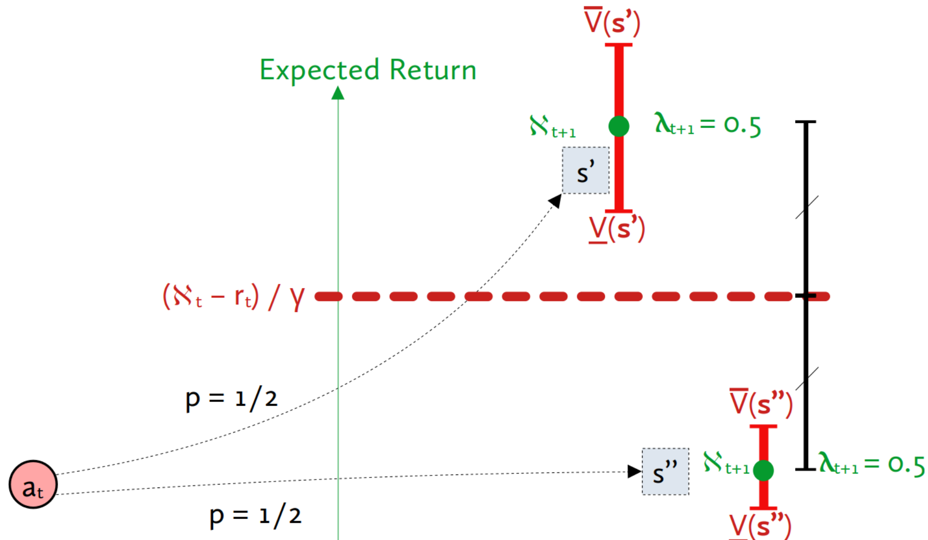


Figure 4: By choosing $\lambda_{t+1} = 0.5$, the two aspirations have a mean of $\frac{\aleph_t - r_t}{\gamma}$, ensuring consistency.

Implementing this algorithm introduces new challenges as $\overline{Q}$ and $\underline{Q}$ cannot be derived from Q. Indeed, in most settings, we lack access to the distribution of $s_{t+1} \sim (s_t, a_t)$.

We also introduce a smoothing factor, μ, to make the sequence $(\lambda_t)_t$ more continuous. When learning from the transition at time t, λ' defined as

$$\lambda' = \lambda_{t+1} : \mu : \lambda_t \tag{21}$$

is used instead of λ_{t+1} for the update target. Setting $\mu = 0$ corresponds to no smoothing, while $\mu = 1$ represents the highest level of smoothing, completely overriding λ_{t+1}. Smoothing can help prevent the agent from oscillating between extremely low and high LRA values.

Combining these elements, we derive AR-Q learning and AR-DQN (refer to algorithms 3 and 4 in appendix B). However, this presents several challenges. The original DQN algorithm does not require Q values to be close to reality to choose the optimal policy. It only requires that, for each state s, when sorting the action set $\mathcal{A}$ based on the key function $Q(s, \cdot)$, the resulting order matches sorting $\mathcal{A}$ based on the key function $Q^*(s, \cdot)$. But to fulfill $\aleph_0$, aspiration rescaling demands **exact** Q values. Another complication arises as the three Q estimators are interdependent, potentially leading to unstable learning.

3.2 Generalization of Aspiration Rescaling

At the internship's conclusion, we tried a novel approach that we did not have the opportunity to thoroughly explore. This method leverages the fact that in the proof of AR's consistency in appendix C, V^+ and V^- do not necessarily need to be $\max_a Q(\cdot, a)$ and $\min_a Q(\cdot, a)$. Thus, we can use the Q functions derived from any bounds $V^{+/-}$ in the aspiration rescaling. Those Vs can serve as "safety bounds" we want the Q values of our action to be between. We can then actually derive Q from Q^+, Q^- and $\aleph$:

$$Q(s_t, \aleph_t, a_t) = Q^-(s_t, a_t)\, [\, \aleph_t \,]\, Q^+(s_t, a_t) \tag{22}$$

where

$$x\, [\, y\,]\, z = \min(z, \max(x, y))$$

The rationale is that if the aspiration is included within the safety bounds, our algorithm will, on average, achieve it, hence $Q = \aleph_t$. Otherwise, we will approach the aspiration as closely as our bounds permit. This method offers several advantages over our previous AR algorithms:

Adaptability: $\aleph_0$ can be adjusted without necessitating retraining.

Stability: Q^+ and Q^- can be trained independently, offering greater stability compared to training Q alongside both of them simultaneously.

Flexibility: Q^+ and Q^- can be trained using any algorithm as soon as the associated V^+ and V^- respect $V^-(s) \leqslant Q(s, a) \leqslant V^+(s)$.

Modularity: There are minimal constraints on the choice of the action lottery, potentially allowing the combination of satisficing with safety criteria for possible actions.

For instance, we can use LRA to learn Q^{λ^+} and Q^{λ^-} for $\lambda^- < \lambda^+$ and use them along with $V^+(s) = \min_a Q^+(s, a) : \lambda^+ : \max_a Q^+(s, a)$ and V^- defined analogously. This corresponds to algorithm 5 in appendix B.

4 Experiments

For all figures, unless otherwise stated, the Y-axis denotes $\mathbb{E}\ G_0$, averaged over 10 learning runs. Each run undergoes 100 evaluations to estimate $\mathbb{E}\ G_0$. The standard deviation, averaged across these runs, is shown in a lighter shade. Algorithms were implemented using the stable baselines 3 (SB3) framework developed by Raffin et al. [2021]. The presented results utilize the DRL version of the previously discussed algorithms, enabling performance comparisons in more complex environments. The DNN architecture employed is the default SB3 "MlpPolicy". All environment rewards have been standardized such that the optimal policy's return is 1. The three environments used in the experiment are presented in the next section.

4.1 Environments

4.1.1 Iterated MAB

In this environment, the agent can choose between different arms for N_{round} times. Each arm gives a certain reward plus Gaussian noise. The observation is $k \leqslant N_{\text{round}}$ the number of rounds played. The optimal policy is to choose the arm with the highest average reward.

4.1.2 Boat Racing

Originating from Krakovna et al. [2020], this gridworld environment rewards the agent with $r = 3$ every time it lands on an arrow tile that aligns with the arrow's direction. Any other move incurs a reward of -1. The environment's layout is depicted in Figure 5. One optimal strategy would be to circle around the environment to collect reward.

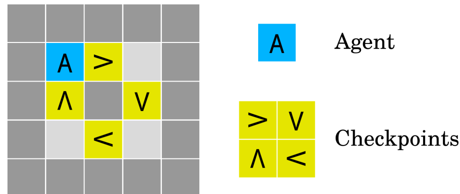


Figure 5: Boat racing gridworld (figure 4 in Krakovna et al. [2020])

4.1.3 Empty Grid

The Empty grid environment represents a vacant room, where the agent's primary task is to navigate to the green goal square in the bottom right of the room, as shown in figure 6. Achieving this yields a sparse reward, with a minor penalty based on the steps taken. The agent can rotate left, right, or move forward. The observation space encodes each tile as a an integer. Success grants a reward calculated as

$$R = 1 - 0.9 * (\text{step count}/\text{max steps})$$

while failure results in no reward. The episode ends when the agent reaches the goal or upon a timeout. The optimal policy is to move to the bottom, turn left and reach the goal. The fact that the reward is only given at the end of the episode make it more complex to learn.

4.2 LRA-DQN

We conducted experiments to explore the relationships between G_0 and λ. In the MAB setup, as expected, the relationship appears linear, as seen in Figure 7a. In boat racing, the relationship seems quadratic, as shown in Figure 7b.

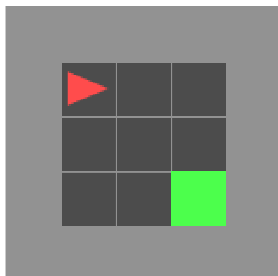
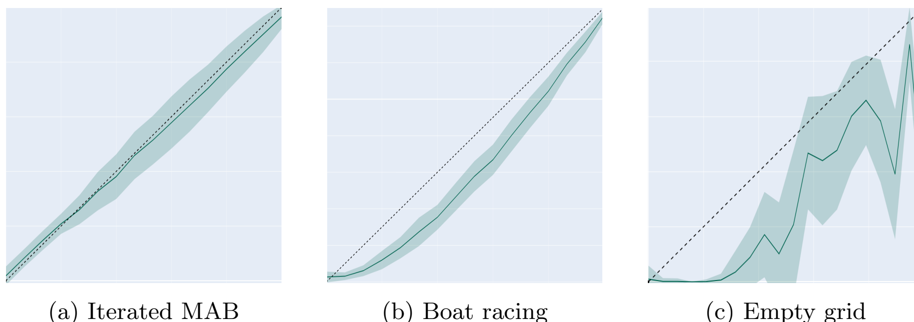


Figure 6: The minigrid environment

Figure 7c also suggests a quadratic relationship, but with noticeable noise and a drop at $\lambda = 1$. Experiments with DQN showed that DQN was unstable in this environment, as indicated by this decline. Unfortunately, we did not have time to optimize the DQN hyperparameters for this environment. Overall we can see that even in



(a) Iterated MAB (b) Boat racing (c) Empty grid

Figure 7: $\mathbb{E}\, G_0$ as a function of λ. The dashed line represents y=x. Bounds are [0,1]

4.3 AR-DQN

Our experiment show that using a hard update, as described in (16), yielded more stable results. The AR update is primarily unstable due to the inaccuracy of aspiration rescaling in the initial stages, where unscaled Q-values lead to suboptimal strategies. As the exploration rate converges to 0, the learning algorithm gets stuck in a local optimum, failing to meet the target on expectation. In the MAB environment, the problem was that the algorithm was too pessimistic about what is feasible because of too low Q values. the algorithm's excessive pessimism about feasibility, stemming from undervalued Q-values, was rectified by subtracting $Q(s, a)$ using (24). However, this modification causes the algorithm to maximize rewards in the initial steps. Since the Q-value is small when training starts, it incentivizes the agent to select the maximizing action

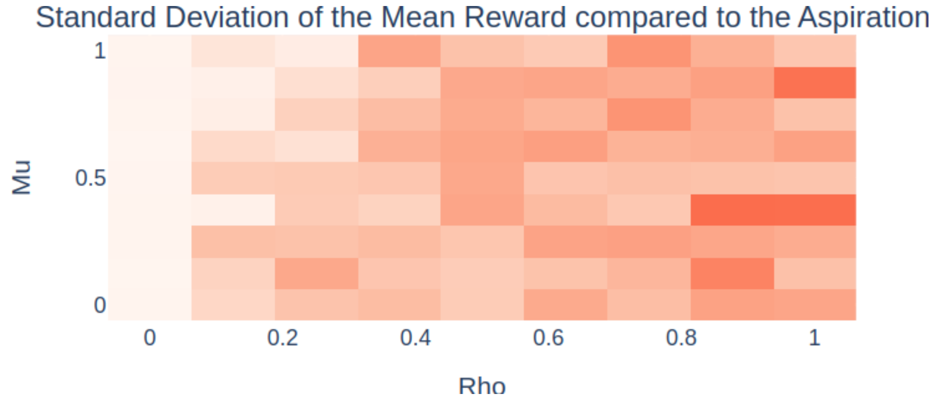


Figure 8: The scale ranges from 0 to 1, with 1 representing the maximum achievable gain. Each (rho, mu) pair is evaluated using 10 aspirations

during these early stages. To study the performance of hard and AR updates, we introduced a new hyperparameter, ρ, to interpolate between hard updates and aspiration rescaling, leading to an updated aspiration rescaling function:

$$\delta_{\text{hard}} = -r_t/\gamma \tag{23}$$

$$\delta_{\text{AR}} = -Q(s,a)/\gamma + \aleph_{t+1} \tag{24}$$

$$\aleph \leftarrow \aleph_t/\gamma + \delta_{\text{hard}} : \rho : \delta_{\text{AR}} \tag{25}$$

Here, $\rho = 0$ corresponds to a hard update, and on expectation, $\rho = 1$ is equivalent to AR as per (20).

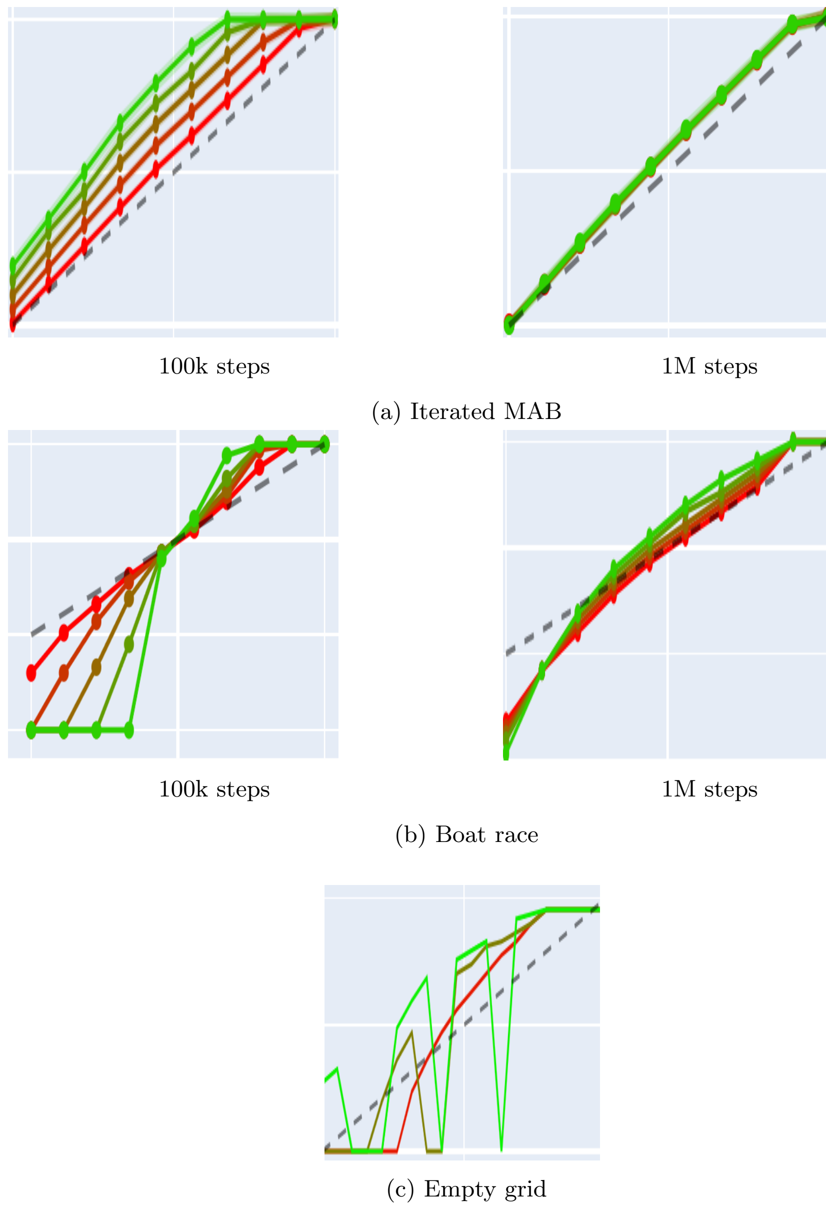
Figure 8 study the influence of ρ and μ on the performance of the algorithm. The algorithm is evaluated using a set of target aspirations $(\aleph_0^i)_{1 \leqslant i \leqslant n}$. For each aspiration, we train the algorithm and evaluate it using:

$$\text{Err} = \sqrt{\frac{\sum_i^n (\mathbb{E}\ G^i - \aleph_0^i)^2}{n}} \tag{26}$$

This would be minimized by a perfect satisficing algorithm. As observed, having a small ρ is crucial for good performance, while μ has a less predictable effect. This suggests that aspiration rescaling needs further refinement to be effective.

4.4 LRAR-DQN

Results on LRAR-DQN confirm our hypothesis that precise Q values are essential for aspiration rescaling. After 100k steps, in both boat racing and iterated MAB, the two LRA-DQN agents, Q^+ and Q^-, have already converged to their final policies. However, the Q-estimator still underestimates the Q values. As illustrated in figure 9, waiting for 1M steps does not alter the outcome with



(a) Iterated MAB

(b) Boat race

(c) Empty grid

Figure 9: On each graph, the X-axis is $\aleph_0$. The colorscale represents ρ, from red=0 to green=1

hard updates ($\rho = 0$), which depend less on the exact Q values. Nevertheless, they enable AR ($\rho = 1$) to match its performance.

In our experiments, the LRAR-DQN algorithm exhibited suboptimal performance on the empty grid task. A potential explanation, which remains to be empirically validated, is the divergence in state encounters between the Q^+ and Q^- during training. Specifically, Q^- appears to predominantly learn behaviors that lead to prolonged stagnation in the top-left corner, while Q^+ seems to be oriented towards reaching the exit within a reasonable timeframe. As a future direction, we propose extending the training of both Q^+ and Q^- under the guidance of the LRAR-DQN policy to ascertain if this approach rectifies the observed challenges.

5 Conclusion

Throughout the duration of this internship, we successfully laid the groundwork for satisficing DQN algorithms. These were implemented using Stable Baseline 3 (Raffin et al. [2021]), a distinguished open-source framework that offers state-of-the-art RL algorithms. By adopting Stable Baseline 3, we ensure that, once fully functional, satisficing algorithms can be readily assessed across a wide range of environments, notably Atari games. Future work will focus on refining the DQN algorithms, exploring the possibility of deriving satisficing algorithms from other RL methodologies such as the Soft Actor-Critic as presented by Haarnoja et al. [2018], and investigating the behavior of satisficers in multi-agent environments both with and without maximizing agents.

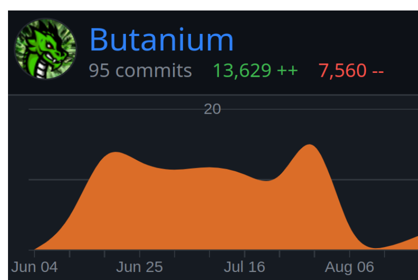
6 Meta information

Throughout my internship, I dedicated a significant amount of time to coding and experimenting with various strategies to enhance the stability of the algorithms. While many of these strategies are not detailed in this report, it's worth noting that they didn't always lead to notable improvements in algorithm performance. To gain insights into the intricacies of the algorithms and understand potential failure cases, I utilized Tensorboard, broadcasting metrics such as loss and average gain during training, as illustrated in figure 10b. My experience also provided me with the opportunity to familiarize myself with Slurm, enabling me to run experiments on the cluster concurrently. I also acquired proficiency in using Ray, a multiprocessing framework, which allowed me to train multiple models simultaneously on my computer. My commit activity throughout the internship, which offers a glimpse into my consistent engagement with the project, can be viewed in figure 10a.

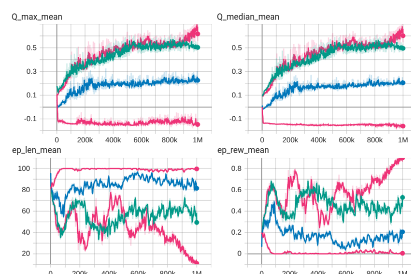
Every week, I actively participated in the lab's "jour fixe," where I learned about the work and challenges faced by other researchers. Notably, during one of these sessions, I had the opportunity to present my own work and received constructive feedback and suggestions. Additionally, I attended several confer-

ences on diverse subjects studied in different departments of the lab and those presented by invited researchers. Of particular note, I attended a talk by Guillaume Falmagne, an ENS Cachan alumnus, on human collective behavior in r/place, after which we had a discussion about ecosystem simulation.

Finally, I assisted my supervisor in the preparation and delivery of an online conference talk, showcasing our collective research findings, which sparked interesting insight and conversations.



(a) My GitHub contribution to our repository



(b) An example of Tensorboard curves from the Empty grid LRA experiment

References

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety, 2016.

Samuel R. Bowman, Jeeyoon Hyun, Ethan Perez, Edwin Chen, Craig Pettit, Scott Heiner, Kamilė Lukošiūtė, Amanda Askell, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Christopher Olah, Daniela Amodei, Dario Amodei, Dawn Drain, Dustin Li, Eli Tran-Johnson, Jackson Kernion, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Liane Lovitt, Nelson Elhage, Nicholas Schiefer, Nicholas Joseph, Noemí Mercado, Nova DasSarma, Robin Larson, Sam McCandlish, Sandipan Kundu, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Timothy Telleen-Lawton, Tom Brown, Tom Henighan, Tristan Hume, Yuntao Bai, Zac Hatfield-Dodds, Ben Mann, and Jared Kaplan. Measuring progress on scalable oversight for large language models, 2022.

Daniel Brown, Russell Coleman, Ravi Srinivasan, and Scott Niekum. Safe imitation learning via fast Bayesian reward inference from preferences. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1165–1177. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/brown20a.html`.

Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2017.

Christopher Frye and Ilya Feige. Parenting: Safe reinforcement learning from human input, 2019.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization, 2022.

Michael A Goodrich and Morgan Quigley. Satisficing q-learning: Efficient learning in problems with dichotomous attributes. In *2004 International Conference on Machine Learning and Applications, 2004. Proceedings.*, pages 65–72. IEEE, 2004.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, and Shane Legg. Specification gaming: the flip side of ai ingenuity. *DeepMind Blog*, 3, 2020.

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

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

Paul Reverdy, Vaibhav Srivastava, and Naomi Ehrich Leonard. Satisficing in multi-armed bandit problems, 2016.

Daniel Russo, David Tse, and Benjamin Van Roy. Time-sensitive bandit learning and satisficing thompson sampling, 2017.

Herbert A. Simon. Rational choice and the structure of the environment. *Psychological review*, 63 2:129–38, 1956. URL `https://api.semanticscholar.org/CorpusID:8503301`.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking, 2022.

Akihiro Tamatsukuri and Tatsuji Takahashi. Guaranteed satisficing and finite regret: Analysis of a cognitive satisficing value function. *Biosystems*, 180:46–53, 2019. ISSN 0303-2647. doi: https://doi.org/10.1016/j.biosystems.2019.02.009. URL `https://www.sciencedirect.com/science/article/pii/S0303264718304453`.

Alexander Matt Turner, Logan Smith, Rohin Shah, Andrew Critch, and Prasad Tadepalli. Optimal policies tend to seek power, 2023.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.

Simon Zhuang and Dylan Hadfield-Menell. Consequences of misaligned AI. *CoRR*, abs/2102.03896, 2021. URL https://arxiv.org/abs/2102.03896.

A Q learning and DQN algorithms

Algorithm 1 Q-learning

Require: discount factor $\gamma \in (0,1]$, sequence of learning rates $\alpha_t \in (0,1]$, sequence of exploration rate $\epsilon_t \in [0,1]$

1: Initialize $Q(s,a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$
2: **for** $t = 0, \ldots, T-1$ **do**
3: ▷ *Act:* ◁
4: with probability ϵ_t, $a_t \leftarrow \text{Explore}(s_t)$, else $a_t \leftarrow \arg\max_a Q(s_t, a)$
5: $(r_t, s_{t+1}) \leftarrow \text{Env}(a_t)$
6: ▷ *Learn:* ◁
7: $y \leftarrow r_t + \gamma \max_a Q(s_{t+1}, a)$ ▷ *update target*
8: $Q(s_t, a_t) \mathrel{+}= \alpha_t\big(y - Q(s_t, a_t)\big)$ ▷ *update the Q-table*
9: **procedure** EXPLORE(s)
10: **return** a sampled uniformly from the action space

Algorithm 2 DQN

Require: Discount factor $\gamma \in (0,1]$, sequence of exploration rate $\epsilon_t \in [0,1]$

1: Initialize empty replay memory D
2: Randomly initialize action-value Neural network Q
3: Initialize corresponding target Neural network $Q' \leftarrow Q$
4: Reset the environment and get the initial observation $s_0 \leftarrow \text{resetEnv}$
5: **for** $t = 0, \ldots, T-1$ **do**
6: ▷ *Act:* ◁
7: with probability ϵ_t, $a_t \leftarrow \text{Explore}(s_t)$, else $a_t \leftarrow \arg\max_a Q(s_t, a)$
8: $r_t, s_{t+1}, \text{done} \leftarrow \text{Env}(a_t)$ ▷ *Perform action a_t*
9: **if** not done **then**
10: store transition (s_t, a_t, r_t, s_{t+1}) in D
11: ▷ *Learn:* ◁
12: sample some minibatch B from D
13: **for** $(s_j, a_j, r_j, s_{j+1}) \in B$ **do**
14: **if** done **then**
15: $y_j \leftarrow r_j$
16: **else**
17: $y_j \leftarrow r_j + \gamma \max_a Q'(s_{j+1}, a)$ ▷ *update target*
18: gradient descent on Q with loss $(y_j - Q(s_j, a_j))^2$
19: every C steps: $Q' \leftarrow Q$
20: **procedure** EXPLORE(s)
21: **return** a sampled uniformly from the action space

B AR algorithms

Algorithm 3 AR-Q learning, Difference with Q-learning are highlighted in blue

---

Require: discount factor $\gamma \in (0, 1]$, sequence of learning rates $\alpha_t \in (0, 1]$, sequence of exploration rate $\epsilon_t > 0$, initial aspiration $\aleph_0 \in \mathbb{R}$, parameter $\mu \in [0, 1)$

1: Initialize $Q(s, a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$
2: Initialize empty replay memory D
3: Initialize two further, tables $\overline{Q}$, $\underline{Q}$
4: $s_0 \leftarrow$ reset Env
5: $\lambda_0 \leftarrow \min_a Q(s_0, a) \setminus \aleph_0 \setminus \max_a Q(s_0, a)$
6: **for** $t = 0, \ldots, T - 1$ **do**
7:     ▷ *Act:* ◁
8:     with probability ϵ_t, $a_t \leftarrow \text{Explore}(s_t)$, else $a_t \leftarrow \text{Exploit}(s_t, \aleph_t)$
9:     $(r_t, s_{t+1}, \text{done}) \leftarrow \text{Env}(a_t)$
10:     **if** done **then**
11:         $\aleph_{t+1} \leftarrow \aleph_0$
12:     **else**
13:         $\aleph_{t+1} \leftarrow \text{RescaleAspiration}(s_t, a_t, s_{t+1})$ ▷ *prepare next move*
14:     $\lambda_{t+1} \leftarrow \min_a Q(s_{t+1}, a) \setminus \aleph_{t+1} \setminus \max_a Q(s_{t+1}, a)$
15:     ▷ *Learn:* ◁
16:     $\lambda' \leftarrow \lambda_{t+1} : \mu : \lambda_t$ ▷ *smoothen relative aspiration levels*
17:     $\underline{v} \leftarrow \min_a Q(s_{t+1}, a)$, $\overline{v} \leftarrow \max_a Q(s_{t+1}, a)$
18:     $v \leftarrow \underline{v} : \lambda' : \overline{v}$
19:     $y_j \leftarrow r_j + \gamma v'$, $\underline{y}_j \leftarrow r_j + \gamma \underline{v}'$, $\overline{y}_j \leftarrow r_j + \gamma \overline{v}'$ ▷ *update targets*
20:     $Q(s_t, a) \mathrel{+}= \alpha_t \big(y - Q(s_t, a)\big)$
21:     $\overline{Q}(s_t, a) \mathrel{+}= \alpha_t \big(\overline{y} - \overline{Q}(s_t, a)\big)$
22:     $\underline{Q}(s_t, a) \mathrel{+}= \alpha_t \big(\underline{y} - \underline{Q}(s_t, a)\big)$
23: **procedure** EXPLORE(s)
24:     **return** a sampled uniformly from the action space
25: **procedure** EXPLOIT($s, \aleph$) ▷ *draw a so that $\underset{a}{\mathbb{E}} Q(s, a) = \aleph$ if possible*
26:     **if** exists a with $Q(s, a) = \aleph$ **then**
27:         **return** any such a
28:     **else if** exists a with $Q(s, a) > \aleph$ and a with $Q(s, a) < \aleph$ **then**
29:         $a_- \leftarrow \arg\max_{a \text{ with } Q(s,a) < \aleph} Q(s, a)$
30:         $a_+ \leftarrow \arg\min_{a \text{ with } Q(s,a) > \aleph} Q(s, a)$
31:         $p \leftarrow Q(s, a_-) \setminus \aleph \setminus Q(s, a_+)$
32:         with probability p, **return** a_+, else **return** a_-
33:     **else if** exists a with $Q(s, a) > \aleph$ **then**
34:         **return** $\arg\min_a Q(s, a)$
35:     **else**
36:         **return** $\arg\max_a Q(s, a)$
37: **procedure** RESCALEASPIRATION(s_t, a_t, s_{t+1})
38:     $\lambda_{t+1} \leftarrow \underline{Q}(s_t, a_t) \setminus Q(s_t, a_t) \setminus \overline{Q}(s_t, a_t)$
39:     **return** $\min_a Q(s_{t+1}, a) : \lambda_{t+1} : \max_a Q(s_{t+1}, a)$

Algorithm 4 AR-DQN, differences with DQN are highlighted in blue.

Require: initial observation o_0, discount factor $\gamma \in (0, 1]$, initial aspiration $\aleph_0 \in \mathbb{R}$, parameter $\mu \in [0, 1)$

1: Initialize empty replay memory D
2: Randomly initialize action-value neural network Q
3: Initialize corresponding target DNN $Q' \leftarrow Q$
4: Randomly initialize two further, DNNs $\overline{Q}$, $\underline{Q}$
5: $s_0 \leftarrow$ reset Env
6: $\lambda_0 \leftarrow \min_a Q(s_0, a) \setminus \aleph_0 \setminus \max_a Q(s_0, a)$
7: **for** $t = 0, \dots, T-1$ **do**
8: ▷ *Act:* ◁
9: with probability ϵ, $a_t \leftarrow \text{Explore}(s_t)$, else $a_t \leftarrow \text{Exploit}(s_t, \aleph_t)$
10: $(r_t, s_{t+1}, \text{done}) \leftarrow \text{Env}(a_t)$
11: **if** done **then**
12: $\aleph_{t+1} \leftarrow \aleph_0$
13: **else**
14: $\aleph_{t+1} \leftarrow \text{RescaleAspiration}(s_t, a_t, s_{t+1})$ ▷ *prepare next move*
15: $\lambda_{t+1} \leftarrow \min_a Q(s_{t+1}, a) \setminus \aleph_{t+1} \setminus \max_a Q(s_{t+1}, a)$
16: **if** not done **then**
17: store transition $(s_t, \lambda_t, a_t, r_t, s_{t+1}, \lambda_{t+1})$ in D
18: ▷ *Learn:* ◁
19: sample some minibatch B from D
20: **for** $(s_j, \lambda_j, a_j, r_j, s_{j+1}, \lambda_{j+1}) \in B$ **do**
21: $\lambda' \leftarrow \lambda_{j+1} : \mu : \lambda_j$ ▷ *smoothen relative aspiration levels*
22: $\underline{v}' \leftarrow \min_a Q'(s_{j+1}, a)$, $\overline{v}' \leftarrow \max_a Q'(s_{j+1}, a)$
23: $v' \leftarrow \underline{v}' : \lambda' : \overline{v}'$
24: $y_j \leftarrow r_j + \gamma v'$, $\underline{y}_j \leftarrow r_j + \gamma \underline{v}'$, $\overline{y}_j \leftarrow r_j + \gamma \overline{v}'$ ▷ *update targets*
25: gradient descent on Q with loss $\left(y_j - Q(s_j, a_j)\right)^2$
26: gradient descent on $\overline{Q}$ with loss $\left(\overline{y}_j - \overline{Q}(s_j, a_j)\right)^2$
27: gradient descent on $\underline{Q}$ with loss $\left(\underline{y}_j - \underline{Q}(s_j, a_j)\right)^2$
28: every C steps: $Q' \leftarrow Q$
29: **procedure** Explore(s)
30: **return** a sampled uniformly from the action space
31: **procedure** Exploit($s, \aleph$) ▷ *draw a so that $\mathbb{E}_a Q(s, a) = \aleph$ if possible*
32: **if** exists a with $Q(s, a) = \aleph$ **then**
33: **return** any such a
34: **else if** exists a with $Q(s, a) > \aleph$ and a with $Q(s, a) < \aleph$ **then**
35: $a_- \leftarrow \arg\max_{a \text{ with } Q(s,a) < \aleph} Q(s, a)$
36: $a_+ \leftarrow \arg\min_{a \text{ with } Q(s,a) > \aleph} Q(s, a)$
37: $p \leftarrow Q(s, a_-) \setminus \aleph \setminus Q(s, a_+)$
38: with probability p, **return** a_+, else **return** a_-
39: **else if** exists a with $Q(s, a) > \aleph$ **then**
40: **return** $\arg\min_a Q(s, a)$
41: **else**
42: **return** $\arg\max_a Q(s, a)$
43: **procedure** RescaleAspiration(s_t, a_t, s_{t+1})
44: $\lambda_{t+1} \leftarrow \underline{Q}(s_t, a_t)) \setminus Q(s_t, a_t) \setminus \overline{Q}(s_t, a_t)$
45: **return** $\min_a Q(s_{t+1}, a) : \lambda_{t+1} : \max_a Q(s_{t+1}, a)$

Algorithm 5 LRA based Rescaling Deep Q-Networks (LRAR-DQN)

Require: initial state s_0, discount factor $\gamma \in (0,1]$, exploration schedule $\epsilon(t)$, local relative target bounds $\underline{\lambda} < \overline{\lambda} \in [0,1]$, rescaling rate $\rho \in [0,1]$

1: initialize function approximators $Q^-(s,a)$ and $Q^+(s,a)$
2: let $V^-(s) \equiv \min_a Q^-(s,a) : \lambda^- : \max_a Q^-(s,a)$ and $V^+(s) \equiv \min_a Q^+(s,a) : \lambda^+ : \max_a Q^+(s,a)$
3: ▷ *Learn functions Q^-, Q^+ based on λ^-, λ^+ via LRA-DQN* ◁
4: learn Q^- and Q^+ with targets $Q^-(s,a) = \mathop{\mathbb{E}}_{(r,s')\sim(s,a)} \big(r + \gamma V^-(s')\big)$ and $Q^+(s,a) = \mathop{\mathbb{E}}_{(r,s')\sim(s,a)} \big(r + \gamma V^+(s')\big)$
5: ▷ *Derive from them the aspiration-parameterized state-value function Q:* ◁
6: put $Q(s,\aleph,a) \equiv Q^-(s,a)\,[\,\aleph\,]\,Q^+(s,a)$
7: ▷ *Deployment:* ◁
8: deploy the agent defined by Deploy(), using the functions Q^-, Q^+, Q
9: **procedure** DEPLOY($\aleph_0$)
10: $s \leftarrow$ EnvReset(); $\aleph \leftarrow \aleph_0$
11: **while** True **do**
12: $\ell \leftarrow$ ActionLottery$(s,\aleph)$ ▷ *action lottery and action*
13: Sample action $a \sim \ell$
14: $(r, s', \text{done}) \leftarrow$ EnvStep(a) ▷ *act, get reward, observe*
15: **if** done **then**
16: Break
17: **else**
18: $\aleph \leftarrow$ PropagateAspiration$(s,\aleph,a,r,s')$ ▷ *prepare next step*
19: **procedure** ACTIONLOTTERY$(s,\aleph)$
20: $\tilde{\aleph} \leftarrow V^-(s)\,[\,\aleph\,]\,V^+(s)$ ▷ *Ensure $\aleph$ is feasible*
21: among those lotteries $\ell \in \Delta(A)$ with $Q(s,\aleph,\ell) = \tilde{\aleph}$, ▷ *meet target in expectation*
choose one based on additional safety criteria and **return** it
22: **procedure** PROPAGATEASPIRATION$(s,\aleph,a,r,s')$
23: **return** $V^-(s') : \big(Q^-(s,a) \setminus q \setminus Q^+(s,a)\big) : V^+(s')$

C Consistency of Aspiration Rescaling

Let V^- and V^+ be functions defining upper and lower bounds for Q, verifying $\forall s, a$:

$$V^-(s) \leqslant Q(s,a) \leqslant V^+(s) \tag{27}$$

We can then define

$$Q^-(s,a) = \mathop{\mathbb{E}}_{(r,s')\sim(s,a)} r + V^-(s') \tag{28}$$

$$Q^+(s,a) = \mathop{\mathbb{E}}_{(r,s')\sim(s,a)} r + V^+(s') \tag{29}$$

Which therefore verify

$$Q^-(s,a) \leqslant Q(s,a) \leqslant Q^+(s,a) \tag{30}$$

For example, in AR $V^+(s) = \max_a Q(s,a)$, $V^-(s) = \min_a Q(s,a)$, $Q^+ = \overline{Q}$ and $Q^- = \underline{Q}$.

In the following proof all $\mathbb{E}$ are meant $\mathbb{E}_{(r_t,s_{t+1})\sim(s_t,a_t)}$ if not specified. We want to show that when the aspiration is propagated like this:

$$\begin{aligned}\lambda_{t+1} &= Q^-(s_t,a_t) \setminus Q(s_t,a_t) \setminus Q^+(s_t,a_t)\\ \aleph_{t+1} &= V^-(s_{t+1}) : \lambda_{t+1} : V^+(s_{t+1})\end{aligned}$$

And that the policy π is chosen at time t fulfills

$$\mathop{\mathbb{E}}_{a\sim\pi} Q(s,a) = \aleph_t \tag{31}$$

the aspiration remains consistent:

$$\mathop{\mathbb{E}}_{a_t\sim\pi} \mathbb{E}\; r_t + \aleph_{t+1} = \aleph_t \tag{32}$$

By combining (31) with (32) we just need to prove that

$$\mathbb{E}\; r_t + \aleph_{t+1} = Q(s_t,a_t)$$

First we rewrite λ_{t+1} :

$$\begin{aligned}\lambda_{t+1} &= Q^-(s_t,a_t) \setminus Q(s_t,a_t) \setminus Q^+(s_t,a_t)\\ &= \frac{Q(s_t,a_t) - Q^-(s_t,a_t)}{Q^+(s_t,a_t) - Q^-(s_t,a_t)}\end{aligned}$$

using (28) and (29) we get

$$= \frac{\mathbb{E}\; V(s_{t+1}) - V^-(s_{t+1})}{\mathbb{E}\; V^+(s_{t+1}) - V^-(s_{t+1})}$$

Using this formula for λ we obtain:

$$\begin{aligned}&\mathbb{E}\; r_t + V^-(s_{t+1}) + \lambda_{t+1}(V^+(s_{t+1}) - V^-(s_{t+1}))\\ &= \mathbb{E}\; \big(r_t + V^-(s_{t+1})\big) + \lambda_{t+1}\mathbb{E}(V^+(s_{t+1}) - V^-(s_{t+1}))\\ &= \mathbb{E}\; r_t + V^-(s_{t+1}) + V(s_{t+1}) - V^-(s_{t+1})\\ &= \mathbb{E}\; r_t + V(s_{t+1})\\ &= Q(s_t,a_t) \quad \square\end{aligned}$$