Final Project

Nikolai Pagh Pedersen, Rasmus Burkarl Pedersen
npede22, rpede22

1 Design choices of setup

We made our main file as tweakable as possible, making it easy to change hyper-parameters or environment-specific settings. We did this to make it easy to test multiple different setups—this makes it easier to compare our algorithms and how they perform on each environment.

This means that all of our experimental variables like learning rate, discount facter, greedy schedule, episode length and ensemble size are defined in the beginning. Furthermore when running the file, it will ask which agent to use, and on which enviorment. If you want to run REDQ or Q-ENSEMBLE-MIN, you will be prompted with how big the ensemble size should be. When an experiment is done, it will create a PNG showing learning rate and bias.

1.1 PseudoCode

1.1.1 DDQN

Double-Q Learning (DDQN) maintains two independent Q-tables, Q_1 and Q_2, each updated according to the Bellman equation. At each transition, we draw $u \sim \mathcal{U}(0,1)$; if $u < 0.5$, we update Q_1 using

$$Q_1(s,a) \;\leftarrow\; Q_1(s,a) \;+\; \alpha\big[r + \gamma\, Q_2\big(s', \arg\max_{a'} Q_1(s',a')\big) - Q_1(s,a)\big],$$

otherwise we update Q_2 , using Q_1 to evaluate the selected action. This 50/50 split ensures both tables are trained equally eliminating the overestimation bias of the max-operator in vanilla Q-learning.

Algorithm 1 Double-DQN Q-table Update

if $\mathrm{random}() < 0.5$ **then**
  $a^* \leftarrow \arg\max_a Q_1(s',a)$
  $Q_1(s,a) \leftarrow Q_1(s,a) + \alpha\big[r + \gamma\, Q_2(s',a^*) - Q_1(s,a)\big]$
else
  $b^* \leftarrow \arg\max_a Q_2(s',a)$
  $Q_2(s,a) \leftarrow Q_2(s,a) + \alpha\big[r + \gamma\, Q_1(s',b^*) - Q_2(s,a)\big]$
end if

1.1.2 DDQN-MIN

Clipped double q-learning is a extension of DQN that helps reducing over-estimation bias by doing ”clipping”, this means taking the lower value in the q-table. In practice this means that if one table over-estimates, the other is lower - so taking the minimum makes your target lower, making is a more conservative estimate.

$$a^* \;=\; \arg\max_a \big(Q_1(s',a) + Q_2(s',a)\big), \qquad \text{clipped} \;=\; \min\big(Q_1(s',a^*),\, Q_2(s',a^*)\big).$$

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

Algorithm 2 DDQN-MIN Q-table Update

$a^* \leftarrow \arg\max_a \big(Q_1(s', a) + Q_2(s', a)\big)$
clipped $\leftarrow \min\big(Q_1(s', a^*),\, Q_2(s', a^*)\big)$
target $\leftarrow r + \gamma$ clipped
if random() < 0.5 **then**
    $Q_1(s, a) \leftarrow Q_1(s, a) \;+\; \alpha\big[\text{target} - Q_1(s, a)\big]$
else
    $Q_2(s, a) \leftarrow Q_2(s, a) \;+\; \alpha\big[\text{target} - Q_2(s, a)\big]$
end if

1.1.3 Q-ENSEMBLE-MIN

Q-ENSEMBLE-MIN is an extension of DDQN and DDQN-MIN, that uses an ensemble of M q-tables to further stabilize learning. This helps against over-estimation. First you find the best actions

$$a^* \;=\; \arg\max_{a \in \mathcal{A}} \sum_{i=1}^{M} Q_i(s_{t+1}, a)$$

Then you perform the clipping - now just using multiple q-tables. Then each time you get a sample, you choose M tables at random and do an update towards the clipped target.

Algorithm 3 Q-Ensemble-MIN Q-table Update

$a^* \leftarrow \arg\max_a \sum_{i=1}^{M} Q_i(s', a)$
clipped $\leftarrow \min_{i=1,\ldots,M} Q_i(s', a^*)$
target $\leftarrow r \;+\; \gamma \cdot$ clipped
$k \leftarrow$ UniformRandom$\{1, 2, \ldots, M\}$
$Q_k(s, a) \;\leftarrow\; Q_k(s, a) \;+\; \alpha\,\big[\text{target} - Q_k(s, a)\big]$

1.1.4 REDQ

REDQ works mostly the same as Q-Ensemble-Min, the difference is that REDQ randomly picks two different "critics" (one Q-value estimator) from the ensemble of M q-tables, these are used for our clipped target.

$$\text{clipped} \;=\; \min\big\{\, Q_i(s_{t+1}, a^*),\; Q_j(s_{t+1}, a^*)\big\}$$

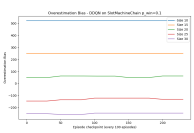
By then taking the minimum of two randomly selected critics, we should reduce over-estimation bias.

We then pick one critic at random to update toward the clipped target. Over many updates all critics should learn, but they should not all reinforce the same over-optimistic target.

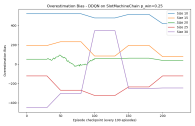
Algorithm 4 REDQ Q-Table Update

$i, j \;\sim$ UniformDistinct$\{1, \ldots, M\}$
$a^* \;\leftarrow\; \arg\max_a \sum_{n=1}^{M} Q_n(s', a)$
clipped $\;\leftarrow\; \min\big\{\, Q_i(s', a^*),\; Q_j(s', a^*)\big\}$
$y \;\leftarrow\; r \;+\; \gamma\,$clipped
$k \;\sim$ Uniform$\{1, \ldots, M\}$
$Q_k(s, a) \;\leftarrow\; Q_k(s, a) \;+\; \alpha\,\big[y - Q_k(s, a)\big]$

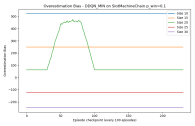
2 Overestimation and learning curve plots



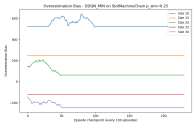
(a) DDQN, $p_{\text{win}} = 0.1$



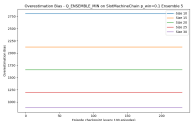
(b) DDQN, $p_{\text{win}} = 0.25$



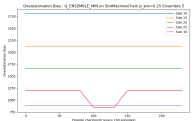
(c) DDQN-MIN, $p_{\text{win}} = 0.1$



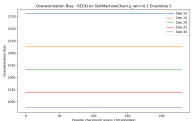
(d) DDQN-MIN, $p_{\text{win}} = 0.25$



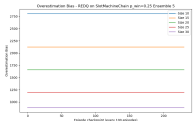
(e) Q-Ensemble-MIN, $p_{\text{win}} = 0.1$



(f) Q-Ensemble-MIN, $p_{\text{win}} = 0.25$

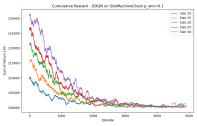


(g) REDQ, $p_{\text{win}} = 0.1$



(h) REDQ, $p_{\text{win}} = 0.25$

Figure 1: Over-estimation bias for all four algorithms on SlotMachineChain (ensemble size 5 where applicable).



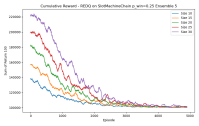
(a) DDQN, $p_{\text{win}} = 0.1$

(b) DDQN, $p_{\text{win}} = 0.25$

(c) DDQN-MIN, $p_{\text{win}} = 0.1$

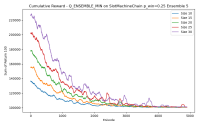
(d) DDQN-MIN, $p_{\text{win}} = 0.25$

(e) REDQ, $p_{\text{win}} = 0.1$



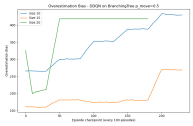
(f) REDQ, $p_{\text{win}} = 0.25$

(g) Q-Ensemble-MIN, $p_{\text{win}} = 0.1$

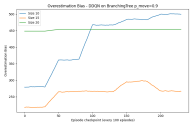


(h) Q-Ensemble-MIN, $p_{\text{win}} = 0.25$

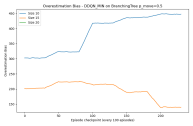
Figure 2: Sum returns of last 100 episodes for all four algorithms on SlotMachineChain (ensemble size 5 where applicable).



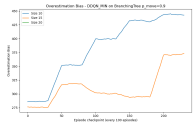
(a) DDQN, $p_{\text{move}} = 0.5$



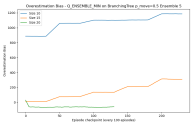
(b) DDQN, $p_{\text{move}} = 0.9$



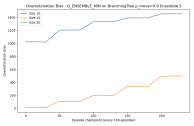
(c) DDQN-MIN, $p_{\text{move}} = 0.5$



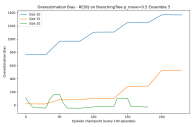
(d) DDQN-MIN, $p_{\text{move}} = 0.9$



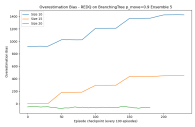
(e) Q-Ensemble-MIN, $p_{\text{move}} = 0.5$



(f) Q-Ensemble-MIN, $p_{\text{move}} = 0.9$

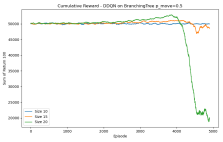


(g) REDQ, $p_{\text{move}} = 0.5$

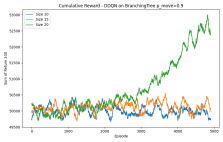


(h) REDQ, $p_{\text{move}} = 0.9$

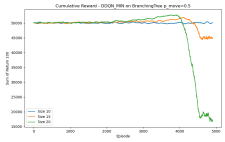
Figure 3: Over-estimation bias for all four algorithms on BranchingTree (ensemble size 5).



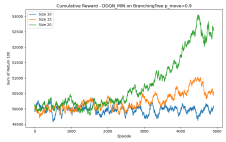
(a) DDQN, $p_{\text{move}} = 0.5$



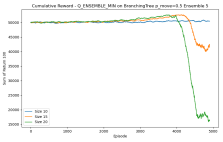
(b) DDQN, $p_{\text{move}} = 0.9$



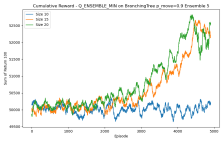
(c) DDQN-MIN, $p_{\text{move}} = 0.5$



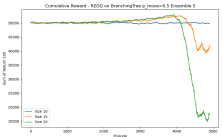
(d) DDQN-MIN, $p_{\text{move}} = 0.9$



(e) Q-Ensemble-MIN, $p_{\text{move}} = 0.5$



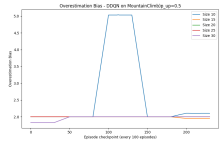
(f) Q-Ensemble-MIN, $p_{\text{move}} = 0.9$



(g) REDQ, $p_{\text{move}} = 0.5$

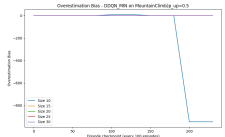
(h) REDQ, $p_{\text{move}} = 0.9$

Figure 4: Sum returns of last 100 episodes for all four algorithms on BranchingTree (ensemble size 5).

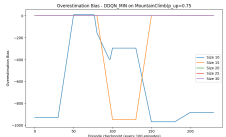


(a) DDQN, $p_{\text{up}} = 0.5$

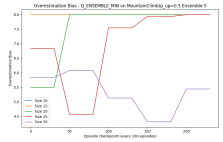
(b) DDQN, $p_{\text{up}} = 0.75$



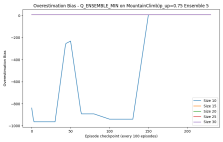
(c) DDQN-MIN, $p_{\text{up}} = 0.5$



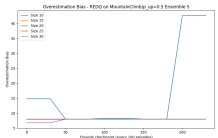
(d) DDQN-MIN, $p_{\text{up}} = 0.75$



(e) Q-Ensemble-MIN, $p_{\text{up}} = 0.5$



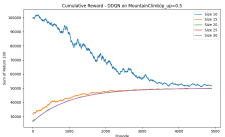
(f) Q-Ensemble-MIN, $p_{\text{up}} = 0.75$



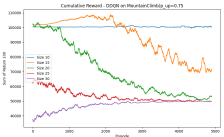
(g) REDQ, $p_{\text{up}} = 0.5$

(h) REDQ, $p_{\text{up}} = 0.75$

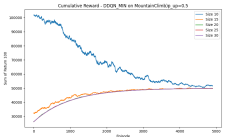
Figure 5: Over-estimation bias for all four algorithms on MountainClimb (ensemble size 5).



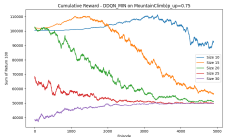
(a) DDQN, $p_{\text{up}} = 0.5$



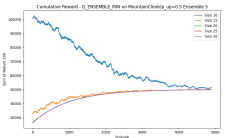
(b) DDQN, $p_{\text{up}} = 0.75$



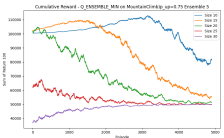
(c) DDQN-MIN, $p_{\text{up}} = 0.5$



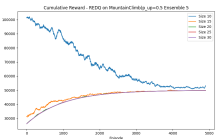
(d) DDQN-MIN, $p_{\text{up}} = 0.75$



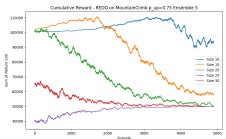
(e) Q-Ensemble-MIN, $p_{\text{up}} = 0.5$



(f) Q-Ensemble-MIN, $p_{\text{up}} = 0.75$



(g) REDQ, $p_{\text{up}} = 0.5$



(h) REDQ, $p_{\text{up}} = 0.75$

Figure 6: Sum returns of last 100 episodes for all four algorithms on MountainClimb (ensemble size 5).

3 Overestimation Bias: Technical Comparison

A central challenge in Q-learning is *over-estimation bias*: because the same value estimates are used both to select and to evaluate actions, the quantity

$$\max_a Q(s, a)$$

tends to exceed the true optimal return from state s.

Double DQN (DDQN) DDQN decouples action selection from action evaluation by maintaining two tables Q_1 and Q_2. On each update one table chooses the greedy action

$$a^* \;=\; \arg\max_a Q_1(s', a)$$

while the other evaluates it (and vice versa). This separation cuts the selection–evaluation feedback loop in half, roughly halving the over-estimation bias compared to standard Q-learning.

Clipped Double DQN (DDQN-MIN) DDQN-MIN further reduces bias by *min-clipping* its bootstrap target. First select

$$a^* \;=\; \arg\max_a \big(Q_1(s', a) + Q_2(s', a)\big),$$

then compute

$$\text{target} = r + \gamma\, \min\big\{Q_1(s', a^*),\, Q_2(s', a^*)\big\}.$$

Bootstrapping from the minimum of the two estimates yields a more conservative target and may even slightly under-estimate.

Q-Ensemble-MIN Q-Ensemble-MIN generalizes DDQN-MIN to an ensemble of M critics. Action selection uses the sum

$$a^* = \arg\max_a \sum_{i=1}^{M} Q_i(s', a),$$

and the target is

$$\text{target} = r + \gamma \min_{i=1,\ldots,M} Q_i(s', a^*).$$

Averaging over M tables makes the greedy choice robust to outliers clipping to the global minimum makes the bootstrap highly conservative.

Randomized Ensembled Double Q (REDQ) REDQ builds on Q-Ensemble-MIN by introducing *randomized clipping*. This extra randomization further decorrelates the bootstrap target from the critic being updated, yielding the smallest and fastest decaying over-estimation bias of all four methods.

4 Discussion of relationship between overestimation and learning curves

4.1 SlotmachineChain

DDQN has the highest and most persistent positive bias, especially when win = 0.25. This optimism, translates into slower and noisier learning curves. Early in training it looks like the agent is overestimating bad actions and continues to explore more suboptimal branches, which looks like delays the convergence of high rewards.

DDQN-MIN Bias is reduced compared to DDQN, it looks like learning curves rise more steeply and stabilize sooner than DDQN.

Q-Ensemble-Min Bias looks very small, almost close to zero from the beginning. The learning curves are also smooth and fast rising. Most likely because of reducing variance and bias, by doing very little wasted exploration.

REDQ Shows the lowest bias, the randomized critics produce very low bias. The return plots show a steep climb and then flattens at the ”highest” reward level. This indicates that randomized clipping prevents a lot of over and under shooting, and instead gives some more accurate targets to do more efficient improvement.

4.2 BranchingTree

DDQN DDQN shows substantial persistent and positive bias, where the q-values looks like they overshoot the true returns. The corresponding return curves rise slowly and below the optimal value, this indicates that the algorithm is over-optimistic and explores suboptimal branches after it should have found the best path.

DDQN-MIN Gives a bias near zero very quickly, the learning curves start flatter than REDQ and Q-Ensemble-Min. This could indicate that it is being too conservative early on - and catches up later.

Q-Ensemble-Min Bias is close to zero - and learning is steady and converges faster than DDQN and DDQN-min.

REDQ Very low bias, learning curve is the fastest and most smooth. Looks like it discovers high reward branches quickly and avoids noise.

4.3 MountainClimb

DDQN We observe large fluctuations in both bias and return, especially early in training. This is due to the high ϵ, which leads to extensive exploration. Sometimes the agent randomly reaches the goal state early, giving high returns before any meaningful learning has occurred. As ϵ decays, reaching the goal becomes harder without deliberate strategy, and the bias stabilizes. Since the environment only allows a 50% chance of success, returns level off around half the optimal value.

DDQN-MIN Compared to DDQN, this algorithm shows a marked reduction in bias. Learning progresses more slowly at first but is more stable throughout. The agent overestimates less often, and its conservative updates result in smoother learning curves. Over time, performance converges to the same average as DDQN, reflecting the probabilistic limit of reaching the top.

Q-Ensemble-MIN Bias is near zero throughout. Aggregating across multiple Q-tables leads to more robust action selection, and min-clipping ensures conservative estimates. Learning curves are smooth and rise quickly to a performance level close to the theoretical optimum. There is little noise and virtually no overestimation, even during the early exploration phase.

REDQ REDQ combines ensemble learning with additional randomization when selecting critics. This de correlates the value estimate from the update target, reducing bias further. Learning is fast and stable, and the agent quickly identifies high-reward strategies without being misled by noise. The randomized critic selection helps avoid reinforcing overly optimistic estimates.

5 Explanation of results

The results show a strong connection between overestimation bias and learning dynamics. Algorithms with high bias, like DDQN, can show inflated early returns, but these are often due to chance during exploration and not genuine improvement. This leads to unstable learning and delayed convergence.

As bias is reduced (e.g., via min-clipping in DDQN-MIN), updates become more stable and reliable. Although learning starts off slower, it results in more predictable performance over time.

Ensemble-based methods (Q-Ensemble-MIN and REDQ) reduce both bias and variance, yielding faster and smoother learning. REDQ, in particular, benefits from randomized target construction, making it the most robust approach across environments.

In all three environments, Q-Ensemble-MIN and REDQ consistently achieve lower bias and better performance. This supports the hypothesis that using multiple value estimates and conservative targets leads to more efficient learning, especially in complex or noisy settings.