Deep Reinforcement Learning with Double Q-learning

Hado van Hasselt and **Arthur Guez** and **David Silver**
Google DeepMind

arXiv:1509.06461v3 [cs.LG] 8 Dec 2015

Abstract

The popular Q-learning algorithm is known to overestimate action values under certain conditions. It was not previously known whether, in practice, such overestimations are common, whether they harm performance, and whether they can generally be prevented. In this paper, we answer all these questions affirmatively. In particular, we first show that the recent DQN algorithm, which combines Q-learning with a deep neural network, suffers from substantial overestimations in some games in the Atari 2600 domain. We then show that the idea behind the Double Q-learning algorithm, which was introduced in a tabular setting, can be generalized to work with large-scale function approximation. We propose a specific adaptation to the DQN algorithm and show that the resulting algorithm not only reduces the observed overestimations, as hypothesized, but that this also leads to much better performance on several games.

The goal of reinforcement learning (Sutton and Barto, 1998) is to learn good policies for sequential decision problems, by optimizing a cumulative future reward signal. Q-learning (Watkins, 1989) is one of the most popular reinforcement learning algorithms, but it is known to sometimes learn unrealistically high action values because it includes a maximization step over estimated action values, which tends to prefer overestimated to underestimated values.

In previous work, overestimations have been attributed to insufficiently flexible function approximation (Thrun and Schwartz, 1993) and noise (van Hasselt, 2010, 2011). In this paper, we unify these views and show overestimations can occur when the action values are inaccurate, irrespective of the source of approximation error. Of course, imprecise value estimates are the norm during learning, which indicates that overestimations may be much more common than previously appreciated.

It is an open question whether, if the overestimations do occur, this negatively affects performance in practice. Overoptimistic value estimates are not necessarily a problem in and of themselves. If all values would be uniformly higher then the relative action preferences are preserved and we would not expect the resulting policy to be any worse. Furthermore, it is known that sometimes it is good to be optimistic: optimism in the face of uncertainty is a well-known exploration technique (Kaelbling et al., 1996). If, however, the overestimations are not uniform and not concentrated at states about which we wish to learn more, then they might negatively affect the quality of the resulting policy. Thrun and Schwartz (1993) give specific examples in which this leads to suboptimal policies, even asymptotically.

To test whether overestimations occur in practice and at scale, we investigate the performance of the recent DQN algorithm (Mnih et al., 2015). DQN combines Q-learning with a flexible deep neural network and was tested on a varied and large set of deterministic Atari 2600 games, reaching human-level performance on many games. In some ways, this setting is a best-case scenario for Q-learning, because the deep neural network provides flexible function approximation with the potential for a low asymptotic approximation error, and the determinism of the environments prevents the harmful effects of noise. Perhaps surprisingly, we show that even in this comparatively favorable setting DQN sometimes substantially overestimates the values of the actions.

We show that the idea behind the Double Q-learning algorithm (van Hasselt, 2010), which was first proposed in a tabular setting, can be generalized to work with arbitrary function approximation, including deep neural networks. We use this to construct a new algorithm we call Double DQN. We then show that this algorithm not only yields more accurate value estimates, but leads to much higher scores on several games. This demonstrates that the overestimations of DQN were indeed leading to poorer policies and that it is beneficial to reduce them. In addition, by improving upon DQN we obtain state-of-the-art results on the Atari domain.

Background

To solve sequential decision problems we can learn estimates for the optimal value of each action, defined as the expected sum of future rewards when taking that action and following the optimal policy thereafter. Under a given policy π, the true value of an action a in a state s is

$$Q_\pi(s, a) \equiv \mathbb{E}\left[R_1 + \gamma R_2 + \dots \mid S_0 = s, A_0 = a, \pi\right] ,$$

where $\gamma \in [0, 1]$ is a discount factor that trades off the importance of immediate and later rewards. The optimal value is then $Q_*(s, a) = \max_\pi Q_\pi(s, a)$. An optimal policy is easily derived from the optimal values by selecting the highest-valued action in each state.

Copyright © 2016, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

crackbitswilp.in

Estimates for the optimal action values can be learned using Q-learning (Watkins, 1989), a form of temporal difference learning (Sutton, 1988). Most interesting problems are too large to learn all action values in all states separately. Instead, we can learn a parameterized value function $Q(s, a; \boldsymbol{\theta}_t)$. The standard Q-learning update for the parameters after taking action A_t in state S_t and observing the immediate reward R_{t+1} and resulting state S_{t+1} is then

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(Y_t^{\mathrm{Q}} - Q(S_t, A_t; \boldsymbol{\theta}_t)) \nabla_{\boldsymbol{\theta}_t} Q(S_t, A_t; \boldsymbol{\theta}_t) \,. \quad (1)$$

where α is a scalar step size and the target Y_t^{Q} is defined as

$$Y_t^{\mathrm{Q}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t) \,. \quad (2)$$

This update resembles stochastic gradient descent, updating the current value $Q(S_t, A_t; \boldsymbol{\theta}_t)$ towards a target value Y_t^{Q}.

Deep Q Networks

A deep Q network (DQN) is a multi-layered neural network that for a given state s outputs a vector of action values $Q(s, \cdot\,; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the parameters of the network. For an n-dimensional state space and an action space containing m actions, the neural network is a function from $\mathbb{R}^n$ to $\mathbb{R}^m$. Two important ingredients of the DQN algorithm as proposed by Mnih et al. (2015) are the use of a target network, and the use of experience replay. The target network, with parameters $\boldsymbol{\theta}^-$, is the same as the online network except that its parameters are copied every τ steps from the online network, so that then $\boldsymbol{\theta}_t^- = \boldsymbol{\theta}_t$, and kept fixed on all other steps. The target used by DQN is then

$$Y_t^{\mathrm{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t^-) \,. \quad (3)$$

For the experience replay (Lin, 1992), observed transitions are stored for some time and sampled uniformly from this memory bank to update the network. Both the target network and the experience replay dramatically improve the performance of the algorithm (Mnih et al., 2015).

Double Q-learning

The max operator in standard Q-learning and DQN, in (2) and (3), uses the same values both to select and to evaluate an action. This makes it more likely to select overestimated values, resulting in overoptimistic value estimates. To prevent this, we can decouple the selection from the evaluation. This is the idea behind Double Q-learning (van Hasselt, 2010).

In the original Double Q-learning algorithm, two value functions are learned by assigning each experience randomly to update one of the two value functions, such that there are two sets of weights, $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$. For each update, one set of weights is used to determine the greedy policy and the other to determine its value. For a clear comparison, we can first untangle the selection and evaluation in Q-learning and rewrite its target (2) as

$$Y_t^{\mathrm{Q}} = R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}_t) \,.$$

The Double Q-learning error can then be written as

$$Y_t^{\mathrm{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t) \,. \quad (4)$$

Notice that the selection of the action, in the argmax, is still due to the online weights $\boldsymbol{\theta}_t$. This means that, as in Q-learning, we are still estimating the value of the greedy policy according to the current values, as defined by $\boldsymbol{\theta}_t$. However, we use the second set of weights $\boldsymbol{\theta}'_t$ to fairly evaluate the value of this policy. This second set of weights can be updated symmetrically by switching the roles of $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$.

Overoptimism due to estimation errors

Q-learning's overestimations were first investigated by Thrun and Schwartz (1993), who showed that if the action values contain random errors uniformly distributed in an interval $[-\epsilon, \epsilon]$ then each target is overestimated up to $\gamma\epsilon\frac{m-1}{m+1}$, where m is the number of actions. In addition, Thrun and Schwartz give a concrete example in which these overestimations even asymptotically lead to sub-optimal policies, and show the overestimations manifest themselves in a small toy problem when using function approximation. Later van Hasselt (2010) argued that noise in the environment can lead to overestimations even when using tabular representation, and proposed Double Q-learning as a solution.

In this section we demonstrate more generally that estimation errors of any kind can induce an upward bias, regardless of whether these errors are due to environmental noise, function approximation, non-stationarity, or any other source. This is important, because in practice any method will incur some inaccuracies during learning, simply due to the fact that the true values are initially unknown.

The result by Thrun and Schwartz (1993) cited above gives an upper bound to the overestimation for a specific setup, but it is also possible, and potentially more interesting, to derive a lower bound.

Theorem 1. *Consider a state s in which all the true optimal action values are equal at $Q_*(s, a) = V_*(s)$ for some $V_*(s)$. Let Q_t be arbitrary value estimates that are on the whole unbiased in the sense that $\sum_a (Q_t(s, a) - V_*(s)) = 0$, but that are not all correct, such that $\frac{1}{m}\sum_a (Q_t(s, a) - V_*(s))^2 = C$ for some $C > 0$, where $m \geq 2$ is the number of actions in s. Under these conditions, $\max_a Q_t(s, a) \geq V_*(s) + \sqrt{\frac{C}{m-1}}$. This lower bound is tight. Under the same conditions, the lower bound on the absolute error of the Double Q-learning estimate is zero. (Proof in appendix.)*

Note that we did not need to assume that estimation errors for different actions are independent. This theorem shows that even if the value estimates are on average correct, estimation errors of any source can drive the estimates up and away from the true optimal values.

The lower bound in Theorem 1 decreases with the number of actions. This is an artifact of considering the lower bound, which requires very specific values to be attained. More typically, the overoptimism increases with the number of actions as shown in Figure 1. Q-learning's overestimations there indeed increase with the number of actions,

crackbitswilp.in

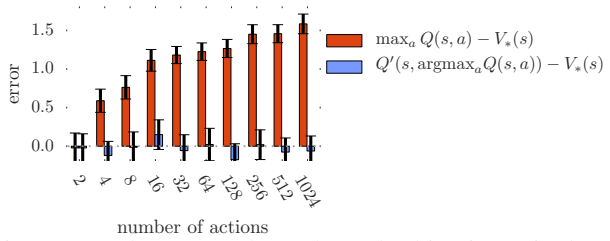


Figure 1: The orange bars show the bias in a single Q-learning update when the action values are $Q(s,a) = V_*(s) + \epsilon_a$ and the errors $\{\epsilon_a\}_{a=1}^m$ are independent standard normal random variables. The second set of action values Q', used for the blue bars, was generated identically and independently. All bars are the average of 100 repetitions.

while Double Q-learning is unbiased. As another example, if for all actions $Q_*(s,a) = V_*(s)$ and the estimation errors $Q_t(s,a) - V_*(s)$ are uniformly random in $[-1,1]$, then the overoptimism is $\frac{m-1}{m+1}$. (Proof in appendix.)

We now turn to function approximation and consider a real-valued continuous state space with 10 discrete actions in each state. For simplicity, the true optimal action values in this example depend only on state so that in each state all actions have the same true value. These true values are shown in the left column of plots in Figure 2 (purple lines) and are defined as either $Q_*(s,a) = \sin(s)$ (top row) or $Q_*(s,a) = 2\exp(-s^2)$ (middle and bottom rows). The left plots also show an approximation for a single action (green lines) as a function of state as well as the samples the estimate is based on (green dots). The estimate is a d-degree polynomial that is fit to the true values at sampled states, where $d = 6$ (top and middle rows) or $d = 9$ (bottom row). The samples match the true function exactly: there is no noise and we assume we have ground truth for the action value on these sampled states. The approximation is inexact even on the sampled states for the top two rows because the function approximation is insufficiently flexible. In the bottom row, the function is flexible enough to fit the green dots, but this reduces the accuracy in unsampled states. Notice that the sampled states are spaced further apart near the left side of the left plots, resulting in larger estimation errors. In many ways this is a typical learning setting, where at each point in time we only have limited data.

The middle column of plots in Figure 2 shows estimated action value functions for all 10 actions (green lines), as functions of state, along with the maximum action value in each state (black dashed line). Although the true value function is the same for all actions, the approximations differ because we have supplied different sets of sampled states.[1] The maximum is often higher than the ground truth shown in purple on the left. This is confirmed in the right plots, which shows the difference between the black and purple curves in orange. The orange line is almost always positive, indicating an upward bias. The right plots also show the estimates from Double Q-learning in blue[2], which are on average much closer to zero. This demonstrates that Double Q-learning indeed can successfully reduce the overoptimism of Q-learning.

The different rows in Figure 2 show variations of the same experiment. The difference between the top and middle rows is the true value function, demonstrating that overestimations are not an artifact of a specific true value function. The difference between the middle and bottom rows is the flexibility of the function approximation. In the left-middle plot, the estimates are even incorrect for some of the sampled states because the function is insufficiently flexible. The function in the bottom-left plot is more flexible but this causes higher estimation errors for unseen states, resulting in higher overestimations. This is important because flexible parametric function approximators are often employed in reinforcement learning (see, e.g., Tesauro 1995; Sallans and Hinton 2004; Riedmiller 2005; Mnih et al. 2015).

In contrast to van Hasselt (2010) we did not use a statistical argument to find overestimations, the process to obtain Figure 2 is fully deterministic. In contrast to Thrun and Schwartz (1993), we did not rely on inflexible function approximation with irreducible asymptotic errors; the bottom row shows that a function that is flexible enough to cover all samples leads to high overestimations. This indicates that the overestimations can occur quite generally.

In the examples above, overestimations occur even when assuming we have samples of the *true* action value at certain states. The value estimates can further deteriorate if we bootstrap off of action values that are already overoptimistic, since this causes overestimations to propagate throughout our estimates. Although *uniformly* overestimating values might not hurt the resulting policy, in practice overestimation errors will differ for different states and actions. Overestimation combined with bootstrapping then has the pernicious effect of propagating the wrong relative information about which states are more valuable than others, directly affecting the quality of the learned policies.

The overestimations should not be confused with optimism in the face of uncertainty (Sutton, 1990; Agrawal, 1995; Kaelbling et al., 1996; Auer et al., 2002; Brafman and Tennenholtz, 2003; Szita and Lőrincz, 2008; Strehl et al., 2009), where an exploration bonus is given to states or actions with uncertain values. Conversely, the overestimations discussed here occur only after updating, resulting in overoptimism in the face of apparent certainty. This was already observed by Thrun and Schwartz (1993), who noted that, in contrast to optimism in the face of uncertainty, these overestimations actually can impede learning an optimal policy. We will see this negative effect on policy quality confirmed later in the experiments as well: when we reduce the overestimations using Double Q-learning, the policies improve.

[1] Each action-value function is fit with a different subset of integer states. States -6 and 6 are always included to avoid extrapolations, and for each action two adjacent integers are missing: for action a_1 states -5 and -4 are not sampled, for a_2 states -4 and -3 are not sampled, and so on. This causes the estimated values to differ.

[2] We arbitrarily used the samples of action a_{i+5} (for $i \leq 5$) or a_{i-5} (for $i > 5$) as the second set of samples for the double estimator of action a_i.

crackbitswilp.in

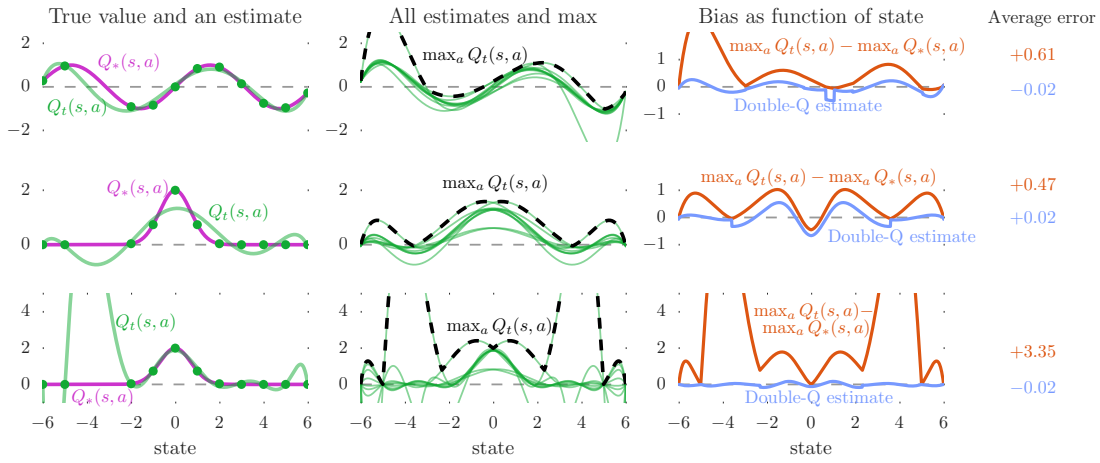


Figure 2: Illustration of overestimations during learning. In each state (x-axis), there are 10 actions. The **left column** shows the true values $V_*(s)$ (purple line). All true action values are defined by $Q_*(s,a) = V_*(s)$. The green line shows estimated values $Q(s,a)$ for one action as a function of state, fitted to the true value at several sampled states (green dots). The **middle column** plots show all the estimated values (green), and the maximum of these values (dashed black). The maximum is higher than the true value (purple, left plot) almost everywhere. The **right column** plots shows the difference in orange. The blue line in the right plots is the estimate used by Double Q-learning with a second set of samples for each state. The blue line is much closer to zero, indicating less bias. The three **rows** correspond to different true functions (left, purple) or capacities of the fitted function (left, green). (Details in the text)

Double DQN

The idea of Double Q-learning is to reduce overestimations by decomposing the max operation in the target into action selection and action evaluation. Although not fully decoupled, the target network in the DQN architecture provides a natural candidate for the second value function, without having to introduce additional networks. We therefore propose to evaluate the greedy policy according to the online network, but using the target network to estimate its value. In reference to both Double Q-learning and DQN, we refer to the resulting algorithm as Double DQN. Its update is the same as for DQN, but replacing the target Y_t^{DQN} with

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^-) .$$

In comparison to Double Q-learning (4), the weights of the second network $\boldsymbol{\theta}'_t$ are replaced with the weights of the target network $\boldsymbol{\theta}_t^-$ for the evaluation of the current greedy policy. The update to the target network stays unchanged from DQN, and remains a periodic copy of the online network.

This version of Double DQN is perhaps the minimal possible change to DQN towards Double Q-learning. The goal is to get most of the benefit of Double Q-learning, while keeping the rest of the DQN algorithm intact for a fair comparison, and with minimal computational overhead.

Empirical results

In this section, we analyze the overestimations of DQN and show that Double DQN improves over DQN both in terms of value accuracy and in terms of policy quality. To further test the robustness of the approach we additionally evaluate the algorithms with random starts generated from expert human trajectories, as proposed by Nair et al. (2015).

Our testbed consists of Atari 2600 games, using the Arcade Learning Environment (Bellemare et al., 2013). The goal is for a single algorithm, with a fixed set of hyperparameters, to learn to play each of the games separately from interaction given only the screen pixels as input. This is a demanding testbed: not only are the inputs high-dimensional, the game visuals and game mechanics vary substantially between games. Good solutions must therefore rely heavily on the learning algorithm — it is not practically feasible to overfit the domain by relying only on tuning.

We closely follow the experimental setting and network architecture outlined by Mnih et al. (2015). Briefly, the network architecture is a convolutional neural network (Fukushima, 1988; LeCun et al., 1998) with 3 convolution layers and a fully-connected hidden layer (approximately 1.5M parameters in total). The network takes the last four frames as input and outputs the action value of each action. On each game, the network is trained on a single GPU for 200M frames, or approximately 1 week.

Results on overoptimism

Figure 3 shows examples of DQN's overestimations in six Atari games. DQN and Double DQN were both trained under the exact conditions described by Mnih et al. (2015). DQN is consistently and sometimes vastly overoptimistic about the value of the current greedy policy, as can be seen by comparing the orange learning curves in the top row of plots to the straight orange lines, which represent the actual discounted value of the best learned policy. More precisely, the (averaged) value estimates are computed regularly during training with full evaluation phases of length $T = 125,000$ steps as

$$\frac{1}{T} \sum_{t=1}^{T} \operatorname*{argmax}_a Q(S_t, a; \boldsymbol{\theta}) .$$

crackbitswilp.in

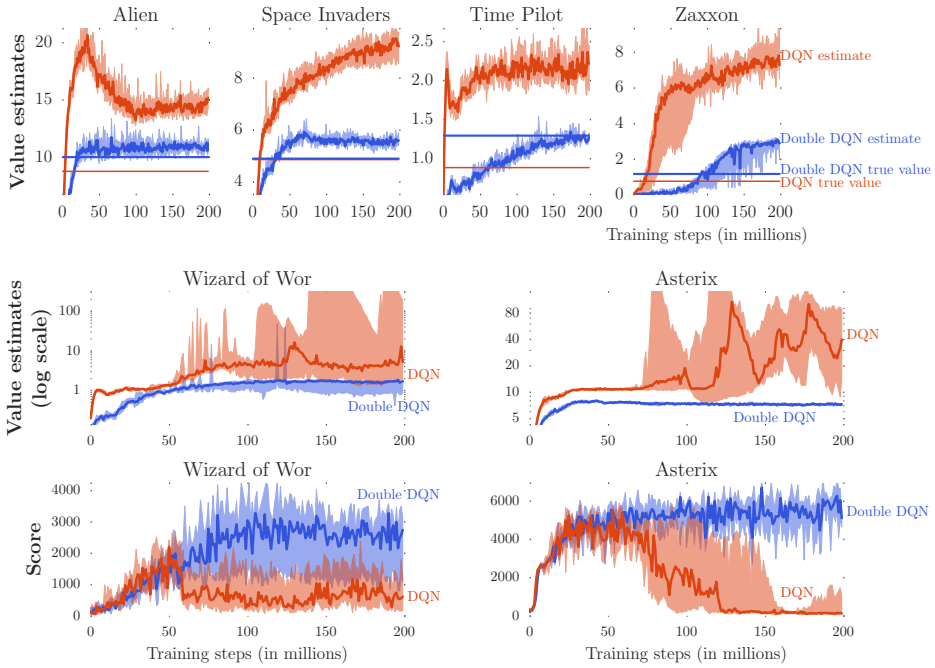


Figure 3: The **top** and **middle** rows show value estimates by DQN (orange) and Double DQN (blue) on six Atari games. The results are obtained by running DQN and Double DQN with 6 different random seeds with the hyper-parameters employed by Mnih et al. (2015). The darker line shows the median over seeds and we average the two extreme values to obtain the shaded area (i.e., 10% and 90% quantiles with linear interpolation). The straight horizontal orange (for DQN) and blue (for Double DQN) lines in the top row are computed by running the corresponding agents after learning concluded, and averaging the actual discounted return obtained from each visited state. These straight lines would match the learning curves at the right side of the plots if there is no bias. The **middle** row shows the value estimates (in log scale) for two games in which DQN's overoptimism is quite extreme. The **bottom** row shows the detrimental effect of this on the score achieved by the agent as it is evaluated during training: the scores drop when the overestimations begin. Learning with Double DQN is much more stable.

The ground truth averaged values are obtained by running the best learned policies for several episodes and computing the actual cumulative rewards. Without overestimations we would expect these quantities to match up (i.e., the curve to match the straight line at the right of each plot). Instead, the learning curves of DQN consistently end up much higher than the true values. The learning curves for Double DQN, shown in blue, are much closer to the blue straight line representing the true value of the final policy. Note that the blue straight line is often higher than the orange straight line. This indicates that Double DQN does not just produce more accurate value estimates but also better policies.

More extreme overestimations are shown in the middle two plots, where DQN is highly unstable on the games Asterix and Wizard of Wor. Notice the log scale for the values on the y-axis. The bottom two plots shows the corresponding scores for these two games. Notice that the increases in value estimates for DQN in the middle plots coincide with decreasing scores in bottom plots. Again, this indicates that the overestimations are harming the quality of the resulting policies. If seen in isolation, one might perhaps be tempted to think the observed instability is related to inherent instability problems of off-policy learning with function approximation (Baird, 1995; Tsitsiklis and Van Roy, 1997; Sutton et al., 2008; Maei, 2011; Sutton et al., 2015). However, we see that learning is much more stable with Double DQN,

	DQN	Double DQN
Median	93.5%	114.7%
Mean	241.1%	330.3%

Table 1: Summary of normalized performance up to 5 minutes of play on 49 games. Results for DQN are from Mnih et al. (2015)

suggesting that the cause for these instabilities is in fact Q-learning's overoptimism. Figure 3 only shows a few examples, but overestimations were observed for DQN in all 49 tested Atari games, albeit in varying amounts.

Quality of the learned policies

Overoptimism does not always adversely affect the quality of the learned policy. For example, DQN achieves optimal behavior in Pong despite slightly overestimating the policy value. Nevertheless, reducing overestimations can significantly benefit the stability of learning; we see clear examples of this in Figure 3. We now assess more generally how much Double DQN helps in terms of policy quality by evaluating on all 49 games that DQN was tested on.

As described by Mnih et al. (2015) each evaluation episode starts by executing a special no-op action that does not affect the environment up to 30 times, to provide different starting points for the agent. Some exploration during evaluation provides additional randomization. For Double DQN we used the exact same hyper-parameters as for DQN,

crackbitswilp.in

	DQN	Double DQN	Double DQN (tuned)
Median	47.5%	88.4%	116.7%
Mean	122.0%	273.1%	475.2%

Table 2: Summary of normalized performance up to 30 minutes of play on 49 games with human starts. Results for DQN are from Nair et al. (2015).

to allow for a controlled experiment focused just on reducing overestimations. The learned policies are evaluated for 5 mins of emulator time (18,000 frames) with an ϵ-greedy policy where $\epsilon = 0.05$. The scores are averaged over 100 episodes. The only difference between Double DQN and DQN is the target, using $Y_t^{\text{DoubleDQN}}$ rather than Y^{DQN}. This evaluation is somewhat adversarial, as the used hyperparameters were tuned for DQN but not for Double DQN.

To obtain summary statistics across games, we normalize the score for each game as follows:

$$\text{score}_{\text{normalized}} = \frac{\text{score}_{\text{agent}} - \text{score}_{\text{random}}}{\text{score}_{\text{human}} - \text{score}_{\text{random}}}. \quad (5)$$

The 'random' and 'human' scores are the same as used by Mnih et al. (2015), and are given in the appendix.

Table 1, under **no ops**, shows that on the whole Double DQN clearly improves over DQN. A detailed comparison (in appendix) shows that there are several games in which Double DQN greatly improves upon DQN. Noteworthy examples include Road Runner (from 233% to 617%), Asterix (from 70% to 180%), Zaxxon (from 54% to 111%), and Double Dunk (from 17% to 397%).

The Gorila algorithm (Nair et al., 2015), which is a massively distributed version of DQN, is not included in the table because the architecture and infrastructure is sufficiently different to make a direct comparison unclear. For completeness, we note that Gorila obtained median and mean normalized scores of 96% and 495%, respectively.

Robustness to Human starts

One concern with the previous evaluation is that in deterministic games with a unique starting point the learner could potentially learn to remember sequences of actions without much need to generalize. While successful, the solution would not be particularly robust. By testing the agents from various starting points, we can test whether the found solutions generalize well, and as such provide a challenging testbed for the learned polices (Nair et al., 2015).

We obtained 100 starting points sampled for each game from a human expert's trajectory, as proposed by Nair et al. (2015). We start an evaluation episode from each of these starting points and run the emulator for up to 108,000 frames (30 mins at 60Hz including the trajectory before the starting point). Each agent is only evaluated on the rewards accumulated after the starting point.

For this evaluation we include a tuned version of Double DQN. Some tuning is appropriate because the hyperparameters were tuned for DQN, which is a different algorithm. For the tuned version of Double DQN, we increased the number of frames between each two copies of the target network from 10,000 to 30,000, to reduce overestimations further because immediately after each switch DQN and Double DQN

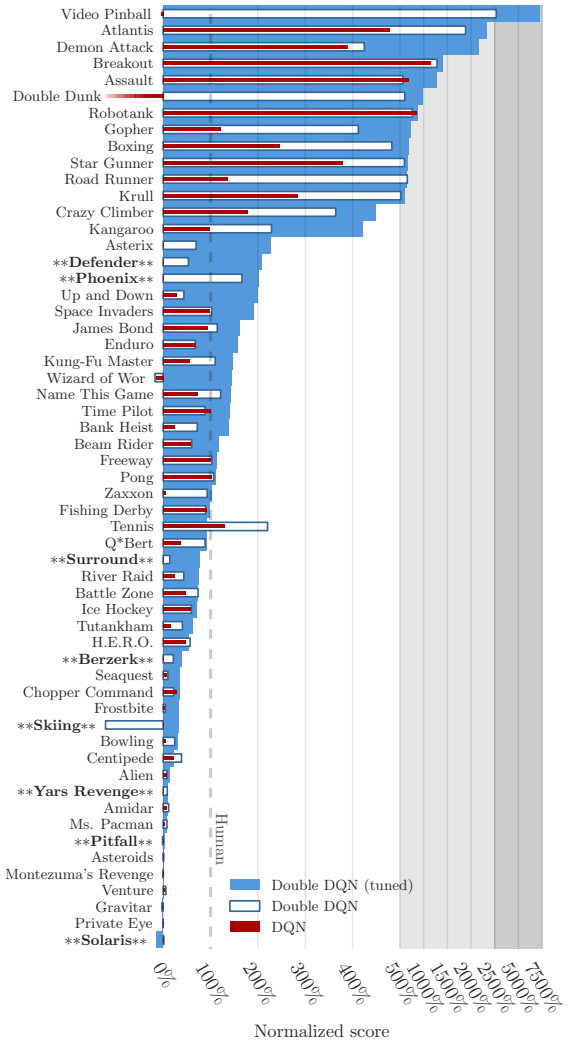


Figure 4: Normalized scores on 57 Atari games, tested for 100 episodes per game with human starts. Compared to Mnih et al. (2015), eight games additional games were tested. These are indicated with stars and a bold font.

both revert to Q-learning. In addition, we reduced the exploration during learning from $\epsilon = 0.1$ to $\epsilon = 0.01$, and then used $\epsilon = 0.001$ during evaluation. Finally, the tuned version uses a single shared bias for all action values in the top layer of the network. Each of these changes improved performance and together they result in clearly better results.[3]

Table 2 reports summary statistics for this evaluation on the 49 games from Mnih et al. (2015). Double DQN obtains clearly higher median and mean scores. Again Gorila DQN (Nair et al., 2015) is not included in the table, but for completeness note it obtained a median of 78% and a mean of 259%. Detailed results, plus results for an additional 8 games, are available in Figure 4 and in the appendix. On several games the improvements from DQN to Double DQN are striking, in some cases bringing scores much closer to

[3]Except for Tennis, where the lower ϵ during training seemed to hurt rather than help.

crackbitswilp.in

human, or even surpassing these.

Double DQN appears more robust to this more challenging evaluation, suggesting that appropriate generalizations occur and that the found solutions do not exploit the determinism of the environments. This is appealing, as it indicates progress towards finding general solutions rather than a deterministic sequence of steps that would be less robust.

Discussion

This paper has five contributions. First, we have shown why Q-learning can be overoptimistic in large-scale problems, even if these are deterministic, due to the inherent estimation errors of learning. Second, by analyzing the value estimates on Atari games we have shown that these overestimations are more common and severe in practice than previously acknowledged. Third, we have shown that Double Q-learning can be used at scale to successfully reduce this overoptimism, resulting in more stable and reliable learning. Fourth, we have proposed a specific implementation called Double DQN, that uses the existing architecture and deep neural network of the DQN algorithm without requiring additional networks or parameters. Finally, we have shown that Double DQN finds better policies, obtaining new state-of-the-art results on the Atari 2600 domain.

Acknowledgments

We would like to thank Tom Schaul, Volodymyr Mnih, Marc Bellemare, Thomas Degris, Georg Ostrovski, and Richard Sutton for helpful comments, and everyone at Google DeepMind for a constructive research environment.

References

R. Agrawal. Sample mean based index policies with O(log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, pages 1054–1078, 1995.

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning: Proceedings of the Twelfth International Conference*, pages 30–37, 1995.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res. (JAIR)*, 47:253–279, 2013.

R. I. Brafman and M. Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

K. Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2):119–130, 1988.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321, 1992.

H. R. Maei. *Gradient temporal-difference learning algorithms*. PhD thesis, University of Alberta, 2011.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.

A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver. Massively parallel methods for deep reinforcement learning. In *Deep Learning Workshop, ICML*, 2015.

M. Riedmiller. Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method. In J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo, editors, *Proceedings of the 16th European Conference on Machine Learning (ECML'05)*, pages 317–328. Springer, 2005.

B. Sallans and G. E. Hinton. Reinforcement learning with factored states and actions. *The Journal of Machine Learning Research*, 5:1063–1088, 2004.

A. L. Strehl, L. Li, and M. L. Littman. Reinforcement learning in finite MDPs: PAC analysis. *The Journal of Machine Learning Research*, 10:2413–2444, 2009.

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

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pages 216–224, 1990.

R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

R. S. Sutton, C. Szepesvári, and H. R. Maei. A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation. *Advances in Neural Information Processing Systems 21 (NIPS-08)*, 21:1609–1616, 2008.

R. S. Sutton, A. R. Mahmood, and M. White. An emphatic approach to the problem of off-policy temporal-difference learning. *arXiv preprint arXiv:1503.04269*, 2015.

I. Szita and A. Lőrincz. The many faces of optimism: a unifying approach. In *Proceedings of the 25th international conference on Machine learning*, pages 1048–1055. ACM, 2008.

G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School*, Hillsdale, NJ, 1993. Lawrence Erlbaum.

J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

H. van Hasselt. Double Q-learning. *Advances in Neural Information Processing Systems*, 23:2613–2621, 2010.

H. van Hasselt. *Insights in Reinforcement Learning*. PhD thesis, Utrecht University, 2011.

C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

crackbitswilp.in

Appendix

Theorem 1. *Consider a state s in which all the true optimal action values are equal at $Q_*(s,a) = V_*(s)$ for some $V_*(s)$. Let Q_t be arbitrary value estimates that are on the whole unbiased in the sense that $\sum_a (Q_t(s,a) - V_*(s)) = 0$, but that are not all zero, such that $\frac{1}{m}\sum_a (Q_t(s,a) - V_*(s))^2 = C$ for some $C > 0$, where $m \geq 2$ is the number of actions in s. Under these conditions, $\max_a Q_t(s,a) \geq V_*(s) + \sqrt{\frac{C}{m-1}}$. This lower bound is tight. Under the same conditions, the lower bound on the absolute error of the Double Q-learning estimate is zero.*

Proof of Theorem 1. Define the errors for each action a as $\epsilon_a = Q_t(s,a) - V_*(s)$. Suppose that there exists a setting of $\{\epsilon_a\}$ such that $\max_a \epsilon_a < \sqrt{\frac{C}{m-1}}$. Let $\{\epsilon_i^+\}$ be the set of positive ϵ of size n, and $\{\epsilon_j^-\}$ the set of strictly negative ϵ of size $m-n$ (such that $\{\epsilon\} = \{\epsilon_i^+\} \cup \{\epsilon_j^-\}$). If $n = m$, then $\sum_a \epsilon_a = 0 \implies \epsilon_a = 0 \ \forall a$, which contradicts $\sum_a \epsilon_a^2 = mC$. Hence, it must be that $n \leq m-1$. Then, $\sum_{i=1}^n \epsilon_i^+ \leq n \max_i \epsilon_i^+ < n\sqrt{\frac{C}{m-1}}$, and therefore (using the constraint $\sum_a \epsilon_a = 0$) we also have that $\sum_{j=1}^{m-n} |\epsilon_j^-| < n\sqrt{\frac{C}{m-1}}$. This implies $\max_j |\epsilon_j^-| < n\sqrt{\frac{C}{m-1}}$. By Hölder's inequality, then

$$\begin{aligned}\sum_{j=1}^{m-n} (\epsilon_j^-)^2 &\leq \sum_{j=1}^{m-n} |\epsilon_j^-| \cdot \max_j |\epsilon_j^-| \\ &< n\sqrt{\frac{C}{m-1}} n\sqrt{\frac{C}{m-1}}\,.\end{aligned}$$

We can now combine these relations to compute an upper-bound on the sum of squares for all ϵ_a:

$$\begin{aligned}\sum_{a=1}^{m} (\epsilon_a)^2 &= \sum_{i=1}^{n} (\epsilon_i^+)^2 + \sum_{j=1}^{m-n} (\epsilon_j^-)^2 \\ &< n\frac{C}{m-1} + n\sqrt{\frac{C}{m-1}} n\sqrt{\frac{C}{m-1}} \\ &= C\frac{n(n+1)}{m-1} \\ &\leq mC.\end{aligned}$$

This contradicts the assumption that $\sum_{a=1}^m \epsilon_a^2 < mC$, and therefore $\max_a \epsilon_a \geq \sqrt{\frac{C}{m-1}}$ for all settings of ϵ that satisfy the constraints. We can check that the lower-bound is tight by setting $\epsilon_a = \sqrt{\frac{C}{m-1}}$ for $a = 1, \ldots, m-1$ and $\epsilon_m = -\sqrt{(m-1)C}$. This verifies $\sum_a \epsilon_a^2 = mC$ and $\sum_a \epsilon_a = 0$.

The only tight lower bound on the absolute error for Double Q-learning $|Q'_t(s, \operatorname{argmax}_a Q_t(s,a)) - V_*(s)|$ is zero. This can be seen by because we can have

$$Q_t(s,a_1) = V_*(s) + \sqrt{C\frac{m-1}{m}}\,,$$

and

$$Q_t(s,a_i) = V_*(s) - \sqrt{C\frac{1}{m(m-1)}}\,, \text{for } i > 1.$$

Then the conditions of the theorem hold. If then, furthermore, we have $Q'_t(s,a_1) = V_*(s)$ then the error is zero. The remaining action values $Q'_t(s,a_i)$, for $i > 1$, are arbitrary. □

Theorem 2. *Consider a state s in which all the true optimal action values are equal at $Q_*(s,a) = V_*(s)$. Suppose that the estimation errors $Q_t(s,a) - Q_*(s,a)$ are independently distributed uniformly randomly in $[-1,1]$. Then,*

$$\mathbb{E}\left[\max_a Q_t(s,a) - V_*(s)\right] = \frac{m-1}{m+1}$$

Proof. Define $\epsilon_a = Q_t(s,a) - Q_*(s,a)$; this is a uniform random variable in $[-1,1]$. The probability that $\max_a Q_t(s,a) \leq x$ for some x is equal to the probability that $\epsilon_a \leq x$ for all a simultaneously. Because the estimation errors are independent, we can derive

$$\begin{aligned}P(\max_a \epsilon_a \leq x) &= P(X_1 \leq x \wedge X_2 \leq x \wedge \ldots \wedge X_m \leq x) \\ &= \prod_{a=1}^{m} P(\epsilon_a \leq x)\,.\end{aligned}$$

The function $P(\epsilon_a \leq x)$ is the cumulative distribution function (CDF) of ϵ_a, which here is simply defined as

$$P(\epsilon_a \leq x) = \begin{cases} 0 & \text{if } x \leq -1 \\ \frac{1+x}{2} & \text{if } x \in (-1,1) \\ 1 & \text{if } x \geq 1 \end{cases}$$

This implies that

$$\begin{aligned}P(\max_a \epsilon_a \leq x) &= \prod_{a=1}^{m} P(\epsilon_a \leq x) \\ &= \begin{cases} 0 & \text{if } x \leq -1 \\ \left(\frac{1+x}{2}\right)^m & \text{if } x \in (-1,1) \\ 1 & \text{if } x \geq 1 \end{cases}\end{aligned}$$

This gives us the CDF of the random variable $\max_a \epsilon_a$. Its expectation can be written as an integral

$$\mathbb{E}\left[\max_a \epsilon_a\right] = \int_{-1}^{1} x f_{\max}(x)\,\mathrm{d}x\,,$$

where $f_{\max}$ is the probability density function of this variable, defined as the derivative of the CDF: $f_{\max}(x) = \frac{\mathrm{d}}{\mathrm{d}x} P(\max_a \epsilon_a \leq x)$, so that for $x \in [-1,1]$ we have $f_{\max}(x) = \frac{m}{2}\left(\frac{1+x}{2}\right)^{m-1}$. Evaluating the integral yields

$$\begin{aligned}\mathbb{E}\left[\max_a \epsilon_a\right] &= \int_{-1}^{1} x f_{\max}(x)\,\mathrm{d}x \\ &= \left[\left(\frac{x+1}{2}\right)^m \frac{mx-1}{m+1}\right]_{-1}^{1} \\ &= \frac{m-1}{m+1}\,.\end{aligned}$$

□

Experimental Details for the Atari 2600 Domain

We selected the 49 games to match the list used by Mnih et al. (2015), see Tables below for the full list. Each agent step is composed of four frames (the last selected action is repeated during these frames) and reward values (obtained from the Arcade Learning Environment (Bellemare et al., 2013)) are clipped between -1 and 1.

crackbitswilp.in

Network Architecture

The convolution network used in the experiment is exactly the one proposed by proposed by Mnih et al. (2015), we only provide details here for completeness. The input to the network is a 84x84x4 tensor containing a rescaled, and gray-scale, version of the last four frames. The first convolution layer convolves the input with 32 filters of size 8 (stride 4), the second layer has 64 layers of size 4 (stride 2), the final convolution layer has 64 filters of size 3 (stride 1). This is followed by a fully-connected hidden layer of 512 units. All these layers are separated by Rectifier Linear Units (ReLu). Finally, a fully-connected linear layer projects to the output of the network, i.e., the Q-values. The optimization employed to train the network is RMSProp (with momentum parameter 0.95).

Hyper-parameters

In all experiments, the discount was set to $\gamma = 0.99$, and the learning rate to $\alpha = 0.00025$. The number of steps between target network updates was $\tau = 10,000$. Training is done over 50M steps (i.e., 200M frames). The agent is evaluated every 1M steps, and the best policy across these evaluations is kept as the output of the learning process. The size of the experience replay memory is 1M tuples. The memory gets sampled to update the network every 4 steps with minibatches of size 32. The simple exploration policy used is an ϵ-greedy policy with the ϵ decreasing linearly from 1 to 0.1 over 1M steps.

Supplementary Results in the Atari 2600 Domain

The Tables below provide further detailed results for our experiments in the Atari domain.

crackbitswilp.in

Game	Random	Human	DQN	Double DQN
Alien	227.80	6875.40	3069.33	2907.30
Amidar	5.80	1675.80	739.50	702.10
Assault	222.40	1496.40	3358.63	5022.90
Asterix	210.00	8503.30	6011.67	15150.00
Asteroids	719.10	13156.70	1629.33	930.60
Atlantis	12850.00	29028.10	85950.00	64758.00
Bank Heist	14.20	734.40	429.67	728.30
Battle Zone	2360.00	37800.00	26300.00	25730.00
Beam Rider	363.90	5774.70	6845.93	7654.00
Bowling	23.10	154.80	42.40	70.50
Boxing	0.10	4.30	71.83	81.70
Breakout	1.70	31.80	401.20	375.00
Centipede	2090.90	11963.20	8309.40	4139.40
Chopper Command	811.00	9881.80	6686.67	4653.00
Crazy Climber	10780.50	35410.50	114103.33	101874.00
Demon Attack	152.10	3401.30	9711.17	9711.90
Double Dunk	-18.60	-15.50	-18.07	-6.30
Enduro	0.00	309.60	301.77	319.50
Fishing Derby	-91.70	5.50	-0.80	20.30
Freeway	0.00	29.60	30.30	31.80
Frostbite	65.20	4334.70	328.33	241.50
Gopher	257.60	2321.00	8520.00	8215.40
Gravitar	173.00	2672.00	306.67	170.50
H.E.R.O.	1027.00	25762.50	19950.33	20357.00
Ice Hockey	-11.20	0.90	-1.60	-2.40
James Bond	29.00	406.70	576.67	438.00
Kangaroo	52.00	3035.00	6740.00	13651.00
Krull	1598.00	2394.60	3804.67	4396.70
Kung-Fu Master	258.50	22736.20	23270.00	29486.00
Montezuma's Revenge	0.00	4366.70	0.00	0.00
Ms. Pacman	307.30	15693.40	2311.00	3210.00
Name This Game	2292.30	4076.20	7256.67	6997.10
Pong	-20.70	9.30	18.90	21.00
Private Eye	24.90	69571.30	1787.57	670.10
Q*Bert	163.90	13455.00	10595.83	14875.00
River Raid	1338.50	13513.30	8315.67	12015.30
Road Runner	11.50	7845.00	18256.67	48377.00
Robotank	2.20	11.90	51.57	46.70
Seaquest	68.40	20181.80	5286.00	7995.00
Space Invaders	148.00	1652.30	1975.50	3154.60
Star Gunner	664.00	10250.00	57996.67	65188.00
Tennis	-23.80	-8.90	-2.47	1.70
Time Pilot	3568.00	5925.00	5946.67	7964.00
Tutankham	11.40	167.60	186.70	190.60
Up and Down	533.40	9082.00	8456.33	16769.90
Venture	0.00	1187.50	380.00	93.00
Video Pinball	16256.90	17297.60	42684.07	70009.00
Wizard of Wor	563.50	4756.50	3393.33	5204.00
Zaxxon	32.50	9173.30	4976.67	10182.00

Table 3: Raw scores for the no-op evaluation condition (5 minutes emulator time). DQN as given by Mnih et al. (2015).

crackbitswilp.in

Game	DQN	Double DQN
Alien	42.75 %	40.31 %
Amidar	43.93 %	41.69 %
Assault	246.17 %	376.81 %
Asterix	69.96 %	180.15 %
Asteroids	7.32 %	1.70 %
Atlantis	451.85 %	320.85 %
Bank Heist	57.69 %	99.15 %
Battle Zone	67.55 %	65.94 %
Beam Rider	119.80 %	134.73 %
Bowling	14.65 %	35.99 %
Boxing	1707.86 %	1942.86 %
Breakout	1327.24 %	1240.20 %
Centipede	62.99 %	20.75 %
Chopper Command	64.78 %	42.36 %
Crazy Climber	419.50 %	369.85 %
Demon Attack	294.20 %	294.22 %
Double Dunk	17.10 %	396.77 %
Enduro	97.47 %	103.20 %
Fishing Derby	93.52 %	115.23 %
Freeway	102.36 %	107.43 %
Frostbite	6.16 %	4.13 %
Gopher	400.43 %	385.66 %
Gravitar	5.35 %	-0.10 %
H.E.R.O.	76.50 %	78.15 %
Ice Hockey	79.34 %	72.73 %
James Bond	145.00 %	108.29 %
Kangaroo	224.20 %	455.88 %
Krull	277.01 %	351.33 %
Kung-Fu Master	102.37 %	130.03 %
Montezuma's Revenge	0.00 %	0.00 %
Ms. Pacman	13.02 %	18.87 %
Name This Game	278.29 %	263.74 %
Pong	132.00 %	139.00 %
Private Eye	2.53 %	0.93 %
Q*Bert	78.49 %	110.68 %
River Raid	57.31 %	87.70 %
Road Runner	232.91 %	617.42 %
Robotank	508.97 %	458.76 %
Seaquest	25.94 %	39.41 %
Space Invaders	121.49 %	199.87 %
Star Gunner	598.09 %	673.11 %
Tennis	143.15 %	171.14 %
Time Pilot	100.92 %	186.51 %
Tutankham	112.23 %	114.72 %
Up and Down	92.68 %	189.93 %
Venture	32.00 %	7.83 %
Video Pinball	2539.36 %	5164.99 %
Wizard of Wor	67.49 %	110.67 %
Zaxxon	54.09 %	111.04 %

Table 4: Normalized results for no-op evaluation condition (5 minutes emulator time).

crackbitswilp.in

Game	Random	Human	DQN	Double DQN	Double DQN (tuned)
Alien	128.30	6371.30	570.2	621.6	1033.4
Amidar	11.80	1540.40	133.4	188.2	169.1
Assault	166.90	628.90	3332.3	2774.3	6060.8
Asterix	164.50	7536.00	124.5	5285.0	16837.0
Asteroids	871.30	36517.30	697.1	1219.0	1193.2
Atlantis	13463.00	26575.00	76108.0	260556.0	319688.0
Bank Heist	21.70	644.50	176.3	469.8	886.0
Battle Zone	3560.00	33030.00	17560.0	25240.0	24740.0
Beam Rider	254.60	14961.00	8672.4	9107.9	17417.2
Berzerk	196.10	2237.50		635.8	1011.1
Bowling	35.20	146.50	41.2	62.3	69.6
Boxing	-1.50	9.60	25.8	52.1	73.5
Breakout	1.60	27.90	303.9	338.7	368.9
Centipede	1925.50	10321.90	3773.1	5166.6	3853.5
Chopper Command	644.00	8930.00	3046.0	2483.0	3495.0
Crazy Climber	9337.00	32667.00	50992.0	94315.0	113782.0
Defender	1965.50	14296.00		8531.0	27510.0
Demon Attack	208.30	3442.80	12835.2	13943.5	69803.4
Double Dunk	-16.00	-14.40	-21.6	-6.4	-0.3
Enduro	-81.80	740.20	475.6	475.9	1216.6
Fishing Derby	-77.10	5.10	-2.3	-3.4	3.2
Freeway	0.10	25.60	25.8	26.3	28.8
Frostbite	66.40	4202.80	157.4	258.3	1448.1
Gopher	250.00	2311.00	2731.8	8742.8	15253.0
Gravitar	245.50	3116.00	216.5	170.0	200.5
H.E.R.O.	1580.30	25839.40	12952.5	15341.4	14892.5
Ice Hockey	-9.70	0.50	-3.8	-3.6	-2.5
James Bond	33.50	368.50	348.5	416.0	573.0
Kangaroo	100.00	2739.00	2696.0	6138.0	11204.0
Krull	1151.90	2109.10	3864.0	6130.4	6796.1
Kung-Fu Master	304.00	20786.80	11875.0	22771.0	30207.0
Montezuma's Revenge	25.00	4182.00	50.0	30.0	42.0
Ms. Pacman	197.80	15375.00	763.5	1401.8	1241.3
Name This Game	1747.80	6796.00	5439.9	7871.5	8960.3
Phoenix	1134.40	6686.20		10364.0	12366.5
Pit Fall	-348.80	5998.90		-432.9	-186.7
Pong	-18.00	15.50	16.2	17.7	19.1
Private Eye	662.80	64169.10	298.2	346.3	-575.5
Q*Bert	183.00	12085.00	4589.8	10713.3	11020.8
River Raid	588.30	14382.20	4065.3	6579.0	10838.4
Road Runner	200.00	6878.00	9264.0	43884.0	43156.0
Robotank	2.40	8.90	58.5	52.0	59.1
Seaquest	215.50	40425.80	2793.9	4199.4	14498.0
Skiing	-15287.40	-3686.60		-29404.3	-11490.4
Solaris	2047.20	11032.60		2166.8	810.0
Space Invaders	182.60	1464.90	1449.7	1495.7	2628.7
Star Gunner	697.00	9528.00	34081.0	53052.0	58365.0
Surround	-9.70	5.40		-7.6	1.9
Tennis	-21.40	-6.70	-2.3	11.0	-7.8
Time Pilot	3273.00	5650.00	5640.0	5375.0	6608.0
Tutankham	12.70	138.30	32.4	63.6	92.2
Up and Down	707.20	9896.10	3311.3	4721.1	19086.9
Venture	18.00	1039.00	54.0	75.0	21.0
Video Pinball	20452.0	15641.10	20228.1	148883.6	367823.7
Wizard of Wor	804.00	4556.00	246.0	155.0	6201.0
Yars Revenge	1476.90	47135.20		5439.5	6270.6
Zaxxon	475.00	8443.00	831.0	7874.0	8593.0

Table 5: Raw scores for the human start condition (30 minutes emulator time). DQN as given by Nair et al. (2015).

crackbitswilp.in

Game	DQN	Double DQN	Double DQN (tuned)
Alien	7.08%	7.90%	14.50%
Amidar	7.95%	11.54%	10.29%
Assault	685.15%	564.37%	1275.74%
Asterix	-0.54%	69.46%	226.18%
Asteroids	-0.49%	0.98%	0.90%
Atlantis	477.77%	1884.48%	2335.46%
Bank Heist	24.82%	71.95%	138.78%
Battle Zone	47.51%	73.57%	71.87%
Beam Rider	57.24%	60.20%	116.70%
Berzerk		21.54%	39.92%
Bowling	5.39%	24.35%	30.91%
Boxing	245.95%	482.88%	675.68%
Breakout	1149.43%	1281.75%	1396.58%
Centipede	22.00%	38.60%	22.96%
Chopper Command	28.99%	22.19%	34.41%
Crazy Climber	178.55%	364.24%	447.69%
Defender		53.25%	207.17%
Demon Attack	390.38%	424.65%	2151.65%
Double Dunk	-350.00%	600.00%	981.25%
Enduro	67.81%	67.85%	157.96%
Fishing Derby	91.00%	89.66%	97.69%
Freeway	100.78%	102.75%	112.55%
Frostbite	2.20%	4.64%	33.40%
Gopher	120.42%	412.07%	727.95%
Gravitar	-1.01%	-2.63%	-1.57%
H.E.R.O.	46.88%	56.73%	54.88%
Ice Hockey	57.84%	59.80%	70.59%
James Bond	94.03%	114.18%	161.04%
Kangaroo	98.37%	228.80%	420.77%
Krull	283.34%	520.11%	589.66%
Kung-Fu Master	56.49%	109.69%	145.99%
Montezuma's Revenge	0.60%	0.12%	0.41%
Ms. Pacman	3.73%	7.93%	6.88%
Name This Game	73.14%	121.30%	142.87%
Phoenix		166.25%	202.31%
Pit Fall		-1.32%	2.55%
Pong	102.09%	106.57%	110.75%
Private Eye	-0.57%	-0.50%	-1.95%
Q*Bert	37.03%	88.48%	91.06%
River Raid	25.21%	43.43%	74.31%
Road Runner	135.73%	654.15%	643.25%
Robotank	863.08%	763.08%	872.31%
Seaquest	6.41%	9.91%	35.52%
Skiing		-121.69%	32.73%
Solaris		1.33%	-13.77%
Space Invaders	98.81%	102.40%	190.76%
Star Gunner	378.03%	592.85%	653.02%
Surround		13.91%	76.82%
Tennis	129.93%	220.41%	92.52%
Time Pilot	99.58%	88.43%	140.30%
Tutankham	15.68%	40.53%	63.30%
Up and Down	28.34%	43.68%	200.02%
Venture	3.53%	5.58%	0.29%
Video Pinball	-4.65%	2669.60%	7220.51%
Wizard of Wor	-14.87%	-17.30%	143.84%
Yars Revenge		8.68%	10.50%
Zaxxon	4.47%	92.86%	101.88%

Table 6: Normalized scores for the human start condition (30 minutes emulator time).

crackbitswilp.in