From DQN to Double DQN: A Two-Paper Technical Analysis of Value Learning from Atari Pixels

Jared Young

***Abstract*—This report analyzes two closely related papers: *Playing Atari with Deep Reinforcement Learning* by Mnih *et al.* and *Deep Reinforcement Learning with Double Q-learning* by van Hasselt *et al.* The first paper is the breakthrough result. It shows that a convolutional Q-network trained from raw Atari frames can learn control policies without hand-crafted state features. The second paper is a follow-up that focuses on one specific weakness in that approach. It argues that DQN-style targets can systematically overestimate action values because the maximization step uses the same noisy estimates for both action selection and action evaluation, and it proposes Double DQN as a simple fix. My main argument is that these papers capture two stages of progress in the same line of research. DQN shows that end-to-end value learning from pixels can actually work. Double DQN then makes that same setup more reliable by improving the target used during training. Read together, the papers show how early deep reinforcement learning moved from proving the idea could work to being more careful about bias, stability, and policy quality.**

***Index Terms*—deep reinforcement learning, DQN, Double DQN, Q-learning, Atari**

I. Introduction

Deep reinforcement learning became especially important once researchers started asking whether an agent could learn control directly from raw sensory input instead of from a hand-engineered state representation. Atari 2600 is a natural testbed for that question because the agent only receives pixels, rewards, and terminal signals, while the true emulator state remains hidden from the learner [1]. In that setting, the problem is hard in two ways at once: the agent has to learn a useful visual representation, and it has to learn a control policy on top of that representation.

The two papers I selected form a clean technical progression. Mnih *et al.* show that a convolutional network trained with a Q-learning objective and experience replay can learn directly from Atari frames [1]. Van Hasselt *et al.* then ask a more specific question: once deep Q-learning works, are the learned action values actually trustworthy? Their answer is no. They show that DQN can systematically overestimate action values and that this overestimation can degrade the learned policy, not just distort the reported value function [2].

My argument throughout this report is that DQN and Double DQN are best read as two steps in the same value-based line of work. DQN is the paper that makes deep value learning from pixels look practical. Double DQN is the paper that comes back and fixes a specific problem in that setup.

II. Technical Baseline: Q-Learning and the Atari Problem

Both papers build on the standard reinforcement learning setting in which an agent interacts with an environment through states, actions, and rewards [1], [2]. The goal is to maximize discounted future return. In the DQN paper, the future discounted return is defined as

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}, \tag{1}$$

where $\gamma \in [0, 1)$ controls how much future rewards matter relative to immediate ones [1]. The optimal action-value function is then written as

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[R_t \mid s_t = s, a_t = a, \pi], \tag{2}$$

which asks a simple question: if the agent is in state s, takes action a, and behaves optimally afterward, how much long-term reward should it expect? [1]

The Bellman equation is the central identity behind both papers:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]. \tag{3}$$

This says that the value of an action now is equal to the immediate reward plus the discounted value of the best future action. Q-learning turns that identity into a bootstrapped update. In the Double DQN paper, the parameterized Q-learning update is written as

$$\theta_{t+1} = \theta_t + \alpha\big(Y_t^Q - Q(S_t, A_t; \theta_t)\big)\nabla_{\theta_t} Q(S_t, A_t; \theta_t), \tag{4}$$

with target

$$Y_t^Q \equiv R_{t+1} + \gamma \max_{a} Q(S_{t+1}, a; \theta_t). \tag{5}$$

So the method updates the current estimate toward a one-step Bellman target built from reward plus the estimated value of the best next action [2]. This is why Q-learning is powerful and risky at the same time: it can propagate useful long-range information through bootstrapping, but any systematic error in the target will also be propagated.

Atari makes these issues much harder. The DQN paper explains that the agent only observes images of the current screen, so the task is partially observed and many true emulator states are perceptually aliased [1]. A single frame may show where an object is, but not necessarily where it is moving.

Rewards can also be delayed for many time steps, which means the agent has to infer long-range consequences from sparse feedback [1]. On top of that, consecutive frames are strongly correlated, which is exactly the kind of data distribution that makes straightforward stochastic gradient updates unreliable [1]. This is the setting in which DQN has to work.

III. DQN: End-to-End Deep Q-Learning from Pixels

A. Problem Formulation and Architecture

The core question in [1] is whether a single deep reinforcement learning system can learn control directly from raw Atari frames without game-specific visual features. The network input is an $84 \times 84 \times 4$ tensor obtained by preprocessing and stacking the last four grayscale frames [1]. This matters because one frame usually does not reveal velocity or direction; the frame stack provides short-term temporal context.

In the uploaded DQN preprint, the Q-network uses two convolutional layers followed by a fully connected hidden layer and a linear output layer [1]. More specifically, the first hidden layer uses 16 filters of size 8×8 with stride 4, the second uses 32 filters of size 4×4 with stride 2, and the fully connected layer has 256 rectified linear units. The output layer contains one linear unit per legal action. That last point matters. The network does not take an action as input and score one pair at a time; instead, it maps the state representation directly to a full vector of Q-values, so all legal actions can be evaluated in a single forward pass [1].

The training objective is a squared temporal-difference loss. The DQN paper defines

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q(s,a;\theta_i)\right)^2\right], \tag{6}$$

where the target is

$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \mid s,a\right]. \tag{7}$$

In other words, DQN keeps the logic of Q-learning but replaces the tabular value function with a convolutional approximator trained by stochastic gradient descent [1]. In the preprint, the target is written using parameters from the previous iteration rather than the explicit target-network notation used in the later DQN family. That detail matters because it shows that the original paper is solving feasibility first, before the full later stabilization recipe appears in the second paper's background discussion.

B. Why the Training Architecture Matters

The most important contribution of DQN is not simply that it uses a convolutional network. The real contribution is the training recipe around that network. First, the method stores past transitions in replay memory and samples minibatches uniformly at random during learning [1]. This improves data reuse and breaks the short-range correlations that would otherwise make consecutive frames a poor training signal for gradient descent. Second, the method uses an ϵ-greedy behavior policy, which keeps exploration simple and generic across games [1]. Third, the paper normalizes the optimization problem by clipping rewards to $\{-1, 0, 1\}$ and by using a common architecture and hyperparameter setup across all seven reported Atari games [1].

The algorithm itself is straightforward. The agent preprocesses the current frame history, chooses an action using the current Q-network and ϵ-greedy exploration, executes that action in the emulator, stores the resulting transition, samples a random minibatch from replay memory, and performs a gradient step on the squared temporal-difference loss [1]. That procedure is important because it turns an otherwise unstable online stream of correlated video frames into something closer to a trainable supervised-learning loop.

These choices make the system trainable, but they also reveal its compromises. Replay memory stabilizes optimization, yet it makes learning highly dependent on the distribution already present in the buffer. Reward clipping improves gradient scale, but it also discards differences in reward magnitude that may matter strategically. The paper itself notes that uniform replay does not distinguish especially informative transitions from routine ones [1]. So even in the foundational paper, the method is engineered primarily for tractable optimization rather than for perfect value calibration or sample efficiency.

C. Strengths and Weaknesses

The main strength of DQN is that it proves end-to-end deep value learning from pixels is feasible. Using one shared architecture and one broadly shared training recipe, it outperforms prior Atari reinforcement learning methods on all seven reported games and exceeds human performance on Breakout, Enduro, and Pong [1]. That result matters because it shows that representation learning and control can be trained jointly rather than solved in separate stages.

At the same time, the limitations matter for the comparison that follows. The uploaded preprint evaluation is still small relative to later Atari benchmark practice. More importantly, strong representation learning does not guarantee that the value targets themselves are well behaved. Several games that require longer-horizon strategy, such as Q*bert, Seaquest, and Space Invaders, remain far from human performance [1]. The paper also shows stable learning curves for its setting, but it does not directly analyze whether any remaining instability comes from representation error, bootstrapping bias, or both. That gap is exactly where the Double DQN paper becomes valuable.

IV. Double DQN: Correcting Overestimation Bias

A. Where the DQN Target Goes Wrong

The key claim in [2] is that the maximization step in Q-learning can create an upward bias whenever action-value estimates are noisy or approximate. If the same estimator is used both to choose the greedy action and to evaluate it, then positive errors are more likely to survive the maximization step than negative ones. The paper formalizes that intuition rather than leaving it at the level of folklore. Under one of its theorem settings, if all true action values are equal in a state

and the average squared estimation error is $C > 0$, then the maximized estimate satisfies the lower bound

$$\max_a Q_t(s,a) \geq V^*(s) + \sqrt{\frac{C}{m-1}}, \tag{8}$$

where m is the number of actions [2]. The paper also gives a more concrete probabilistic example: if action-value errors are independently and uniformly distributed on $[-1,1]$, then the expected overoptimism becomes $(m-1)/(m+1)$ [2]. The exact constant is not the most important part. What matters is the basic idea: even if the errors at the action level are not biased overall, the max operator can still turn them into an overoptimistic estimate.

This matters because deep Q-learning is bootstrapped. If the target is systematically too high, that error does not stay local. It gets propagated through later updates. The paper is careful to distinguish this phenomenon from deliberate optimism used for exploration. Here, the optimism is accidental. It arises from the mechanics of the target and can directly hurt policy quality [2].

B. From DQN to Double DQN

The Double DQN paper presents the stabilized DQN target in the form

$$Y_t^{\text{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t^-), \tag{9}$$

where θ_t^- denotes target-network parameters copied periodically from the online network [2]. The proposed correction is

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q\big(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t^-\big). \tag{10}$$

The online network chooses the greedy next action, but the target network evaluates that chosen action [2]. That decoupling is the whole idea. Double DQN does not replace replay memory, the convolutional architecture, or off-policy learning. It changes only the construction of the target.

That small change is one of the best parts of the paper. Because the modification is so minimal, the authors can make a much cleaner argument about what is actually improving. If the values become less inflated and the policy becomes more stable, it is reasonable to connect that change back to the target itself instead of to some totally different architecture. Put more simply, DQN uses one estimator to both pick and score the next action. Double DQN splits those jobs apart.

C. Architecture, Evidence, and Limitations

The Double DQN paper follows the later DQN-style Atari setting and explicitly summarizes the network architecture it uses: an $84 \times 84 \times 4$ input, three convolutional layers, a fully connected hidden layer of 512 units, and approximately 1.5 million parameters overall [2]. The target network is copied every $\tau = 10{,}000$ steps, replay memory stores one million transitions, minibatches have size 32, and training runs for 50 million agent steps, or 200 million frames [2]. This is important for precision. The paper is not merely criticizing the 2013 preprint in its original form; it is diagnosing overestimation in the later stabilized DQN family.

The empirical results are pretty convincing. Under the standard no-op evaluation, the median normalized score rises from 93.5% for DQN to 114.7% for Double DQN, while the mean rises from 241.1% to 330.3% [2]. Under the harder human-start evaluation, the median rises from 47.5% to 88.4%, and a tuned Double DQN reaches 116.7% median and 475.2% mean [2]. The paper also highlights large improvements on individual games such as Road Runner, Asterix, Zaxxon, and Double Dunk [2]. More importantly, the paper shows cases where DQN's value estimates rise sharply while its actual score falls, which is exactly the signature one would expect if overestimation were harming policy quality rather than merely inflating reported Q-values [2]. In that sense, the empirical story is stronger than saying "Double DQN gives smaller numbers." The stronger claim is that it gives better calibrated numbers and often better control.

The limitations are narrower but still real. Double DQN improves the estimator inside the DQN framework, but it does not solve the bigger deep RL problems of poor sample efficiency, replay dependence, or expensive Atari-scale training. It makes the DQN setup better; it does not replace the whole value-based approach.

V. Comparative Analysis and Discussion

The easiest way to compare these papers is to ask what each one is mainly trying to solve. In DQN, the hard part is representation learning under high-dimensional visual input. The paper shows that a deep convolutional network, combined with replay-based Q-learning, can pull enough structure out of Atari frames to support useful control [1]. In Double DQN, representation is not really the main issue anymore. That paper starts from the fact that DQN already works well enough to learn strong policies and then asks whether the values driving those policies are biased [2].

That is why the two papers fit together so well. The second paper depends on the success of the first one, but it also shows that success by itself is not the whole story. A model can learn a useful representation and still train on a distorted target. To me, DQN is the more foundational paper because it changes what looks possible. Double DQN is the more focused paper because it comes back and asks what exactly is going wrong in the update.

There is also an important historical detail here. The uploaded DQN paper is the 2013 preprint, while the Double DQN paper evaluates the later DQN family that already includes a target network and a broader Atari benchmark [1], [2]. I do not think that weakens the comparison. If anything, it makes the progression clearer. The original DQN result shows feasibility. The later Double DQN result shows that even after the approach became stronger and more stable, the target still had a bias problem.

It is also worth noticing how much stays the same between the two methods. Both are still value-based, off-policy, replay-driven systems operating on stacked Atari frames. Both rely

on bootstrapping. Both try to use one network that generalizes across many states instead of memorizing a table of values. Because of that, the most important difference is not the whole deep learning pipeline. It is the target definition. That is what makes Double DQN such a clean follow-up. It keeps almost everything else fixed and changes the part most directly tied to overestimation.

The bigger lesson, at least in my view, is that progress in deep reinforcement learning is not just about using a stronger neural network. It is also about being careful with what target the network is being trained on and how estimation error gets pushed forward through bootstrapped updates. DQN shows that deep value learning from pixels can work. Double DQN shows that making it work is not the same thing as making it reliable.

VI. Conclusion

Taken together, these two papers show a pretty clear development path in early deep reinforcement learning. DQN demonstrates that value-based control from raw pixels is feasible with a convolutional Q-network, replay memory, and Atari-scale training [1]. Double DQN then shows that this success still comes with a built-in weakness: if the target uses one estimator to both select and evaluate actions, the result can be persistent overestimation [2]. By separating those two roles, Double DQN improves value accuracy, makes learning more stable, and often improves policy quality too.

The larger takeaway is that deep reinforcement learning was improving in two directions at the same time. It was getting better at learning representations from difficult visual input, and it was also getting better at handling the behavior of the targets used during training. That is why these papers work so well as a pair. The first paper explains why deep Q-learning became viable. The second explains why viable is not the same thing as well-calibrated, and why that difference matters.

For this assignment, I think the pairing works well because it supports both summary and critique. The first paper gives the architecture, the domain, and the baseline learning logic. The second paper gives the clearer error analysis and shows how a small change in the target can noticeably affect stability and performance. Reading them together makes the progression easy to see: first the method works, and then researchers figure out one important reason it can still go wrong and how to fix it.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," arXiv:1312.5602, 2013.

[2] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," arXiv:1509.06461, 2015.