An Improved Algorithm for Deep Q-Networks (Postprint)

Authors: Xia Zongtao, Qin Jin

Date: 2018-10-11T00:00:00+00:00

Abstract

Deep Q-Network suffers from a severe overestimation problem, which degrades the agent' s ability to find the optimal policy. To mitigate the overestimation problem in Deep Q-Network, a correction function is proposed to improve the value function in Deep Q-Network. When the selected action is the optimal action, the correction function is set to 1, leaving the current state-action value unmodified; when the selected action is not the optimal action, the correction function is less than 1, reducing the current state-action value. This thereby increases the difference between optimal state-action values and non-optimal state-action values, reducing the impact of the overestimation problem. Experimental results demonstrate that the improved algorithm achieves better performance on Playing Atari 2600 video games and OpenAI Gym, indicating that the improved algorithm finds a better policy than Deep Q-Network.

Full Text

Preamble

Title: An Improved Algorithm for Deep Q-Network

Authors: Xia Zongtao, Qin Jin
(College of Computer Science & Technology, Guizhou University, Guiyang 550025, China)

Abstract: Deep Q-Network (DQN) suffers from severe overestimation, which degrades the agent' s ability to find optimal policies. To mitigate this overestimation problem in DQN, this paper proposes a correction function to improve the evaluation function. The correction function equals 1 when the selected action is optimal, leaving the current state-action value unchanged. When the selected action is suboptimal, the correction function is less than 1, reducing the current state-action value. This increases the disparity between optimal

and non-optimal state-action values, thereby reducing the impact of overestimation. Experiments demonstrate that the improved algorithm achieves better performance in Playing Atari 2600 video games and OpenAI Gym, indicating that it finds superior policies compared to standard DQN.

Keywords: Deep Q-Network; overestimation problem; correction function; state-action value

0 Introduction

Reinforcement learning fundamentally involves mapping from states to actions through continuous interaction between an agent and its environment, thereby improving the agent' s adaptability to unknown environments and maximizing cumulative rewards to ultimately obtain optimal policies. Reinforcement learning has broad applications in artificial intelligence, including industrial manufacturing, path planning, and game playing. Traditional algorithms such as Q-learning and Sarsa store Q-values in a Q-table, limiting them to problems with small state or action spaces. For large-scale problems, constructing a sufficiently large Q-table becomes infeasible, making it impossible to fully represent the state and action spaces.

To address this limitation, DeepMind introduced deep learning concepts into reinforcement learning, proposing Deep Q-Network (DQN) that combines deep learning' s perception capabilities with reinforcement learning' s decision-making capacity. DQN uses convolutional neural networks to model the Q-table, enabling complete representation of state-action spaces and achieving remarkable success in Playing Atari 2600 video games, surpassing human professional players in 49 games. However, combining reinforcement learning with neural network function approximation introduces overestimation problems, causing the final model to select suboptimal policies.

Several approaches have been proposed to address this issue. Double DQN decouples action selection from evaluation, mitigating overestimation but introducing underestimation. Speedy Q-learning combines state-action values from steps k and k-1 to update the Q-table, achieving faster convergence and partially solving overestimation, but its neural network variant fails to outperform DQN. Averaged DQN utilizes not just the current network but the past n networks to obtain target values, averaging them before taking the maximum as the new target. While this yields higher average rewards, it suffers from high training overhead and long training times. Bias-Corrected Q-learning constructs a correction term to offset the bias introduced by the max operator, demonstrating superior performance in roulette games compared to Q-learning and Speedy Q-learning. However, it is better suited for large action spaces and less appropriate for the relatively small action spaces typical of Playing Atari 2600 games. Advantage learning can alleviate overestimation by shrinking non-optimal values to preserve the ordering of optimal and suboptimal values despite estimation

errors, but it applies uniform reduction to different non-optimal values, which is unreasonable.

This paper proposes a novel method that applies larger reduction to non-optimal values closer to the optimal value and smaller reduction to those farther away. Using neural network modeling, we apply this approach to DQN to create an improved algorithm called Corrected-DQN. Experiments demonstrate that Corrected-DQN learns superior policies and achieves higher returns.

---

1 Background

1.1 Reinforcement Learning

Reinforcement learning is a process of learning a mapping from states to actions. Through interaction with the environment, the agent collects real-world samples and learns from them via trial and error to discover optimal policies. A reinforcement learning system typically comprises an agent and an environment. The agent continuously interacts with the environment by executing actions, transitioning to new states, and receiving rewards until reaching an absorbing state where no further actions are taken and the state no longer changes. Repeating this process enables the agent to eventually learn an optimal policy. The decision-making process in reinforcement learning is based on Markov Decision Processes (MDPs), which typically assume the learning task satisfies the Markov property.

An MDP can be defined as a quadruple $\langle S, A, P, R \rangle$, where S represents the set of all possible states, A denotes the set of all possible actions, P is the state transition probability (the probability of reaching the next state after taking action a in state s), and R is the reward function (the immediate reward obtained when reaching state s' after taking action a in state s). Generally, P and R are unknown in a reinforcement learning task, requiring the agent to continuously explore through trial and error.

Reinforcement learning tasks commonly use value functions to represent the cumulative reward expected from executing a policy in a given state. Value functions fall into two categories: state value functions and state-action value functions. The state value function $V^{\pi}(s)$ represents the cumulative expected return starting from state s when following policy π. The state-action value function $Q^{\pi}(s, a)$ represents the cumulative expected return starting from state s and taking action a under policy π.

The optimal policy π^* maximizes cumulative rewards. A reinforcement learning task may have multiple optimal policies, denoted as π^*. The state value function under optimal policy π^* is called the optimal state value function, denoted as $V^*(s)$. The state-action value function under optimal policy π^* is called the optimal state-action value function, denoted as $Q^*(s, a)$.

Traditional algorithms like Q-learning and Sarsa store $Q^*(s,a)$ in a Q-table, which works for small-scale discrete spaces. However, when state and action spaces become too large, hardware limitations prevent constructing such massive tables, necessitating function approximation. The most popular approach uses deep learning to model the Q-table.

1.2 Deep Reinforcement Learning

Deep reinforcement learning employs neural networks as function approximation models. By repeatedly adjusting network parameters through reinforcement learning, the neural network eventually models the Q-table. This approach combines reinforcement learning' s decision-making capabilities with deep learning's perception abilities, using deep learning to transform large-scale raw inputs through simple yet nonlinear transformations into higher-level abstract representations that discover intrinsic data patterns. Through reinforcement learning, the network is continuously updated to achieve a good fit to the Q-table, enabling the agent to obtain a satisfactory policy via the neural network.

DeepMind first proposed DQN in 2013, successfully combining Convolutional Neural Networks (CNN) with Q-learning, achieving human-level performance in 49 Atari 2600 games. Subsequent variants emerged, such as Double DQN and Deep Recurrent Q-Network.

---

2 Improved Deep Reinforcement Learning Algorithm

2.1 Overestimation Problem

Combining neural networks with reinforcement learning introduces overestimation. Function approximation models like neural networks contain estimation errors, so their outputs cannot perfectly reflect true values—there is always deviation between estimated and actual values. Most reinforcement learning algorithms use the max operator to select the optimal action in the current state. When estimation errors are uniformly distributed, the action selected by the max operator may not be truly optimal. The model tends to favor actions with inflated state-action values. When differences between state-action values corresponding to the same state are small, the model may select a suboptimal action, preventing it from learning an optimal policy and degrading agent performance.

Traditional Q-learning evaluates and selects the optimal action in the current state by computing the optimal state value function $V^*(s)$. Suppose in state s, the agent can only choose between two actions a_1 and a_2, where $Q(s,a_1) > Q(s,a_2)$, making a_1 optimal and a_2 suboptimal. Due to uniform estimation error ε in the function approximation model, the following situation may occur:

$$\langle MATH_8 \rangle$$

In this case, the max operator selects the suboptimal action a_2 instead of the optimal action a_1, causing the agent to learn a suboptimal policy.

2.2 Difference Amplification

One improvement approach is to increase the gap between optimal and suboptimal values by reducing suboptimal values. When the error between true and estimated values is relatively small, the ordering of optimal and non-optimal values remains unaffected by estimation errors and the max operator. Advantage learning adopts this concept by shrinking non-optimal values to preserve the ordering. However, advantage learning applies uniform reduction to different non-optimal values, which is unreasonable—it shrinks values far from the optimal more aggressively than those close to it, yet the suboptimal value most likely to be selected by the max operator receives the smallest reduction, diminishing the effectiveness of difference amplification.

This paper proposes a novel method that introduces a correction function. While reducing all non-optimal values, it applies greater reduction to values closer to the optimal and smaller reduction to those farther away.

Define a correction function:

$$\langle MATH_4 \rangle$$

where u is the action taken in state s (from historical data in the replay memory), and a^* is the action corresponding to the maximum state-action value output by the online value network. Due to the properties of exponential functions, when base $b \in (0, 1)$, as the difference between $Q(s, u)$ and $\max_a Q(s, a)$ decreases, $B(s, u)$ also decreases. This means non-optimal values closer to the optimal receive smaller correction functions. When $Q(s, u) = \max_a Q(s, a)$, the correction function equals 1, leaving the optimal state-action value unchanged.

The modified evaluation function becomes:

$$\langle MATH_5 \rangle$$

This approach leaves the optimal state-action value function unchanged while scaling down non-optimal values, thereby amplifying the gap between non-optimal and optimal values. The overestimation problem caused by neural network estimation errors is consequently mitigated, enabling the improved optimal state value function to find better policies.

2.3 Improved Deep Q-Network

DeepMind first proposed DQN in 2013. Earlier, Neural Fitted Q-learning (NFQ) attempted to use a DQN-like architecture for simple control problems. However, NFQ used resilient backpropagation (RPROP) for parameter updates and

suffered from neural network instability, preventing it from handling complex control problems. DQN dramatically improved neural network stability through target networks and experience replay, while using stochastic gradient descent (SGD) to update the online value network. This enabled DQN to handle complex control problems without requiring additional data—raw data could be directly input into a convolutional neural network, making data processing more human-like. The agent then used Q-learning to interact with the environment, obtaining reward-labeled samples to update the online value network parameters. After a certain number of steps, the online value network parameters were copied to the target value network, which computed target values. The update process for online and target value networks is shown in [Figure 1: see original paper].

DQN uses a simplified state value function to obtain target values:

$$\langle MATH_6 \rangle$$

where a is the action taken in state s (not necessarily optimal). To apply the proposed correction function to deep reinforcement learning, we must model the correction function appropriately.

2.3.1 Modeling In DQN, the target value network models target values while the online value network models estimated values. Since our correction function modifies target values, we use the target value network to model it. By processing the current state' s corresponding image and inputting it into the target value network, at the output layer, if $a = a^*$, the correction function is set to 1; if $a \neq a^*$, we obtain:

$$\langle MATH_9 \rangle$$

Thus non-optimal values are scaled down. As the agent continuously interacts with the environment, the online value network model is continuously updated, making it possible that historical model selections differ from the latest model' s selections, which enables implementation of our correction function modeling.

2.3.2 Training Using the proposed correction function, we redefine DQN to obtain the improved Corrected-DQN algorithm:

Algorithm: Corrected-DQN

Initialize replay memory D with capacity N, target network update step C, discount factor γ, correction function base b, and episode count M.

Initialize online value network $Q(s, a; \theta)$ with random weights θ, and initialize target value network $Q(s, a; \theta^-)$ with weights $\theta^- = \theta$.

For each episode until M episodes complete: 1. Initialize state s_1 and preprocess raw image to obtain $\phi_1 = \phi(s_1)$. 2. For each step in the episode: - With probability ε, select a random action a_t; otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$. - Execute action a_t in the emulator, observe reward r_t and next state s_{t+1}, preprocess to obtain $\phi_{t+1} = \phi(s_{t+1})$. - Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D. - Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D. - Compute correction function: If $a_j = \arg\max_a Q(\phi_j, a; \theta)$, set $B = 1$; otherwise compute $B = b^{Q(\phi_j, a_j; \theta^-) - \max_a Q(\phi_j, a; \theta^-)}$. - Compute target value: If ϕ_{j+1} is terminal, set $y_j = r_j$; otherwise $y_j = r_j + \gamma \max_a Q(\phi_{j+1}, a'; \theta^-) \times B$. - Compute loss: $L = [y_j - Q(\phi_j, a_j; \theta)]^2$ and update θ via gradient descent. - Every C steps, set $\theta^- = \theta$.

---

3 Experimental Results and Analysis

3.1 Experimental Design

To validate Corrected-DQN' s performance, we conducted comparative experiments between NIPS-DQN and Corrected-DQN in two environments: Arcade Learning Environment (ALE) and OpenAI Gym.

In ALE, we used identical experimental settings to NIPS-DQN and employed TensorFlow-GPU 1.2 with an Nvidia Quadro P6000 GPU for computational efficiency. To reduce computational requirements, we directly used the NIPS-DQN algorithm from the original paper as our baseline. We evaluated both algorithms on five classic Atari 2600 control tasks: Breakout, Seaquest, Phoenix, Krull, and Amidar, using the original paper' s hyperparameters. The replay memory capacity was initialized to 1,000,000 transitions, the discount factor γ to 0.95, and the correction function base b was initialized to 0.95 and decayed to 0.5 with step size 0.05. The exploration probability ε in ε-greedy policy decayed from 1 to 0.1 with increasing steps. Raw Atari 2600 game images (210×210 pixels, 128 colors) were preprocessed into 84×84 grayscale images.

The network architecture consisted of two convolutional layers and two fully connected layers. The first convolutional layer had 16 filters of size 8×8 with stride 4; the second had 32 filters of size 4×4 with stride 2, transforming a 20×20×16 input space to 9×9×32 output space, using ReLU activation. The first fully connected layer had 256 neurons; the second had neurons equal to the action space size, outputting estimated Q-values for each possible action. The online value network produced current state estimates. We sampled 32×4 transitions randomly from replay memory to break correlations. Each input to the online value network consisted of 4 consecutive concatenated state images (84×84 each). The output layer' s Q-values were processed by a max operator to select the optimal action. The agent executed this action, received reward, transitioned to the next state, and stored the transition tuple in replay memory. When full, memory was overwritten from the beginning. The target value

network computed target values, and the online network was updated via SGD. Every 10,000 steps, online network parameters were copied to the target network, which modeled the correction function for integration into DQN.

In OpenAI Gym, we adjusted parameters due to differences from ALE. Replay memory size was reduced to 10,000, target network update period to 1,000 steps, and each epoch to 10,000 steps. For the three selected control tasks (Acrobot, CartPole, and MountainCar), the state representations were feature vectors (position, velocity, etc.) rather than images, significantly reducing problem scale. Consequently, we simplified the network architecture by removing convolutional layers while keeping other parameters unchanged. No image preprocessing was required—states were directly input to the network.

3.2 Experimental Results and Analysis

In ALE, we compared Corrected-DQN and NIPS-DQN across five control tasks, evaluating average reward, algorithm stability, and convergence speed. Higher average reward indicates better performance. Results show Corrected-DQN achieved higher average rewards than NIPS-DQN on Seaquest, Phoenix, Amidar, Breakout, and Krull, demonstrating superior policy learning and agent performance. As shown in Figure 2: see original paper-(e), Corrected-DQN outperformed NIPS-DQN in all five tasks. Notably, on Krull, Corrected-DQN converged faster, stabilizing at a high level after only 20 epochs (approximately 50,000 steps per epoch).

Regarding stability, neural networks as Q-function approximators are inherently unstable. While DQN' s experience replay and dual-network architecture improve stability, significant reward fluctuations persist. Our experiments reveal that mitigating overestimation increases the probability of selecting optimal actions. Consequently, Corrected-DQN exhibits improved neural network stability on Seaquest, Phoenix, and Krull, with smoother learning curves compared to NIPS-DQN. However, both algorithms occasionally produce zero average rewards on Phoenix, indicating room for further improvement.

In OpenAI Gym, we compared the algorithms on Acrobot, CartPole, and MountainCar. Performance on CartPole was measured by average reward, while Acrobot and MountainCar were evaluated by target-reaching frequency per epoch (10,000 steps). As shown in Figure 3: see original paper, Corrected-DQN demonstrated better stability than NIPS-DQN on Acrobot, with reduced oscillation amplitude. Averaging cumulative rewards across 250 epochs yielded 65 for Corrected-DQN versus 50 for NIPS-DQN, confirming superior policy learning. On MountainCar Figure 3: see original paper, Corrected-DQN reached the target more frequently, indicating better final policies. For CartPole Figure 3: see original paper, while not visually obvious, averaging cumulative rewards across 250 epochs gave 1,401 for Corrected-DQN versus 896 for NIPS-DQN, demonstrating superior strategy learning.

We observed that performance improvements were more pronounced in ALE

than OpenAI Gym. Preliminary analysis suggests this is because OpenAI Gym' s selected tasks have significantly smaller input scales, providing fewer data features and consequently limiting algorithmic performance.

---

4 Conclusion

This paper addresses DQN' s overestimation problem by proposing a deep reinforcement learning algorithm based on difference amplification. By designing a novel correction function that applies greater reduction to non-optimal values closer to the optimal value and smaller reduction to those farther away, the gap between optimal and non-optimal values gradually increases. This reduces the impact of nonlinear estimation model errors on action selection, mitigates overestimation, and enables the agent to learn superior policies.

Experiments on Playing Atari 2600 and OpenAI Gym demonstrate that Corrected-DQN achieves improvements in average reward, stability, and convergence compared to NIPS-DQN. We conclude that applying the proposed correction function to DQN effectively alleviates overestimation.

References

[1] Zhihua Zhou. *Machine Learning* [M]. Beijing: Tsinghua University Press, 2016: 371-373.

[2] Yang Gao, Ruyi Zhou, Hao Wang, Zhixin Cao. Research on average reward reinforcement learning algorithm [J]. *Chinese Journal of Computers*, 2007, 29(8): 2806-2810.

[3] Wenchen Yang, Lun Zhang. Multi-agent reinforcement learning based traffic signal control for integrated urban network: survey of state of art [J]. *Application Research of Computers*, 2018, 35(6): 1613-1618.

[4] Tan M. Multi-agent reinforcement learning: independent vs. cooperative agents [C]// Proc of the 10th International Conference on Machine Learning. San Francisco, CA: Morgan Kaufmann Publishing, 1993: 487-494.

[5] Peter D, et al. Q-learning [J]. *Machine Learning*, 1992, 8(3): 279-292.

[6] Rummery G, Niranjan M. On-line Q-learning using connectionist systems [C]// CUED/F-INFENG/TR 166. England, 1994.

[7] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[8] Thrun S, Schwartz A. Issues in using function approximation for reinforcement learning [C]// Proc of the 4th Connectionist Models. 1993.

[9] Van H V, Guez A, Silver D. Deep reinforcement learning with double Q-learning [C]// Proc of the AAAI Conference on Artificial Intelligence. 2016.

[10] Hasselt H V, et al. Double Q-learning [C]// Advances in Neural Information Processing Systems. 2010: 2613-2621.

[11] Azar M, Munos R, Ghavamzadeh M, Kappen H. Speedy Q-learning [C]// Advances in Neural Information Processing Systems, 2011.

[12] Oron A, Nir B, Nahum S. Averaged-DQN: variance reduction and stabilization for deep reinforcement learning [EB/OL]. arXiv:1611.01929.

[13] Lee D, Defourny B, Powell W. Bias-corrected Q-learning to control max-operator bias in Q-learning [C]// Proc of IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning. 2013: 93-99.

[14] Baird L C. Reinforcement learning through gradient descent [D]. Carnegie Mellon University, USA, 1999.

[15] Yi Qiu, Yanyan Dong. Linear programming method based on Markov process [J]. *Statistics & Decision*, 2017, 10: 88-90.

[16] Lecun Y, Botton L, Bengio Y, Haffer P. Gradient-based learning applied to document recognition [J]. *Proceedings of the IEEE*, 1998, 86(11): 2278-2324.

[17] Hausknecht M, Stone P. Deep recurrent Q-learning for partially observable MDPs [EB/OL]. arXiv:1507.06527, 2015.

[18] Martin R. Neural fitted q iteration-first experiences with a data efficient neural reinforcement learning method [C]// Machine Learning: ECML 2005.

Note: Figure translations are in progress. See original paper for figures.