Comparison of Deep Reinforcement Learning Algorithms in Atari 2600 Games

Hany Raza

Northeastern University, CS Department
raza.h@northeastern.edu

Abstract

The Deep Q-Network (DQN) algorithm has become very popular and several variants have been developed over it. However, it is also important to understand the intricate details in its variants, for further improvement and also to identify and exploit the most suited applications for them. In this paper, we compare DQN base algorithm along with its variants Double DQN and Dueling Double DQN algorithm. The experiments are performed on three Atari domains and extensively analyzed. We also compare a modified version of DQN with original DQN algorithm. With our experiments and analysis, we understand the differences between them and successfully make inference about the benefit of each algorithm in different situations and their suitability for different environments.

Introduction

Reinforcement Learning (RL) is a field focused on sequential decision-making, training agents to achieve optimal behavior in an environment to maximize rewards or reach defined goals. Deep Reinforcement Learning, a combination of Deep Learning and RL, has led to algorithms like Deep Q-Networks (DQN) (DQN; Mnih et al. 2013, 2015), integrating Q-learning (Watkins, 1989), convolutional neural networks, and experience replay. This approach, initially excelling in Atari games, has seen enhancements for speed and stability. Understanding variations in these algorithms is crucial for leveraging benefits, addressing drawbacks, and making improvements.

While DQN surpassed human expertise in some Atari 2600 games, it has drawbacks, notably overestimation bias. Double DQN (DDQN; van Hasselt, Guez, and Silver 2016) resolves this issue by decoupling action selection and evaluation, resulting in improved stability and performance in Atari 2600 environments. Dueling Double DQN (Wang et al. 2016) introduces a dueling network architecture, enhancing policy evaluation by separating state value and action advantage functions. In certain environments, this architecture enables RL agents to outperform previous benchmarks.

This paper compares DQN, Double DQN, and Dueling Double DQN across Pong, Krull, and Boxing Atari 2600 environments. Experiment results are analyzed on various parameters, including modifications to the DQN algorithm. The findings contribute to understanding the strengths and limitations of these algorithms in different gaming scenarios.

Background

Below are the details of these algorithms and some important sub-components.

Q-Learning and Double Q-Learning

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

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

where $\gamma \in [0,1]$ is a discount factor giving more value to immediate rewards than later rewards. The optimal value can be calculated as $Q_*(s,a) = \max_\pi Q_\pi(s,a)$. For this, we use Q-learning, a form of temporal difference learning. We learn a parameterized value function $Q(s, a; \theta_t)$. The Q-learning update formula 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

$$\theta_{t+1} = \theta_t + \alpha \left(Y_t^{\mathrm{Q}} - Q\left(S_t, A_t; \theta_t\right)\right) \nabla_{\theta_t} Q\left(S_t, A_t; \theta_t\right) \quad (1)$$

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

$$Y_t^{\mathrm{Q}} \equiv R_{t+1} + \gamma \max_a Q\left(S_{t+1}, a; \theta_t\right) \quad (2)$$

The above equation is often compared to stochastic gradient descent, updating the current value $Q(S_t, A_t; \theta_t)$ towards target value Y_t^{Q}.

In Q-learning, in (2), the max operator uses the same values both to select and to evaluate an action causing overestimation of action values. Double Q-learning decouples the selection from the evaluation preventing the error.

$$Y_t^{DQ} \equiv R_{t+1} + \gamma Q\left(S_{t+1}, argmax_a Q\left(S_{t+1}, a; \theta_t\right); \theta_t'\right) \quad (3)$$

Here, 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, θ and θ'. For each update, one set of weights is used to determine the greedy policy and the other to determine its value.

Experience Replay

In the process of reinforcement learning, Experience Replay involves the accumulation of a dataset comprising experiences from various episodes. Unlike standard temporal difference learning, where only the current experience is utilized, Experience Replay trains the Q-network by randomly sampling mini-batches of experiences from the dataset. This method enhances data efficiency by reusing experience samples in multiple updates, reducing variance through uniform sampling from the replay buffer. An additional benefit is that, when the algorithm forgets previously learned experiences, uniform data sampling from the buffer facilitates relearning. Experience Replay proves valuable for both data efficiency and mitigating variance in reinforcement learning algorithms.

Deep Q Networks (DQN)

A deep Q network (DQN) is a deep neural network with multiple layers. For a given state s as input, it outputs action values in vector form $Q(s, \cdot; \theta)$, where θ are the weights or parameters of the network. For an x-dimensional state space and an action space containing y actions, the neural network is a function from R^x to R^y. DQN algorithm (Mnih et al. (2015)) has a target network, online network and experience replay. The target network, with parameters θ^-, has the same architecture as the online or behavior network which is constantly updated. Its parameters are copied every τ steps from the online network, and kept fixed on all other steps to enhance stability of derived values. Target network is given by:

$$Y_t^{\mathrm{DQN}} \equiv R_{t+1} + \gamma \max_a Q\left(S_{t+1}, a; \theta_t^-\right).$$

Double Deep Q Networks (DDQN)

The only difference in DDQN from DQN is Double Q-Learning algorithm which reduces overestimation of values and results in better stability and performance of algorithm. The target network has significantly less updates and hence has more stable values, compared to online network.
The update equation for Double DQN is:

$$Y_t^{DDQN} \equiv R_{t+1} + \gamma Q\left(S_{t+1}, argmax_a Q\left(S_{t+1}, a; \theta_t\right), \theta_t^-\right)$$

Dueling Double Deep Q Networks (DDQN)

This algorithm has a new duel architecture which is combined with DDQN and hence named Dueling DDQN.

The first architecture in Figure 1 is used by DDQN and DQN. The second neural network architecture is dueling architecture which is designed for value based RL. It has the same initial convolutional encoder and then divides in two streams of computation, value and advantage streams. It then

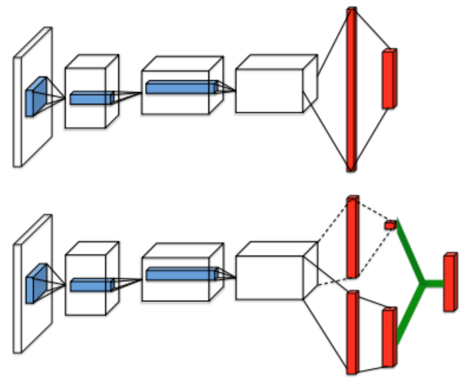

Figure 1: Duel Network Architecture (Wang et al. 2016)

merges by a special aggregator as shown in Figure 1 (Wang et al. 2016). This aggregation operation is represented by the following mathematical equation:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A\left(s, a'; \theta, \alpha\right)\right).$$

The estimated state-value function $V(s; \theta, \beta)$ forms the base of the Q-values and estimated Advantage function of action is added over it. Here, θ refers to weights of the convolutional layers, and α and β refer to parameters of fully-connected layers of the state-value and advantage streams respectively.
The dueling architecture can more efficiently approximate value of states, without considering the estimated impact of each action from the state. This can be beneficial in noisy and highly dynamic environments. The dueling architecture is also able to prioritize higher advantage value actions than multiple average or similar value advantage actions during policy evaluation.
The value stream V is updated with every update in Q-values which is in contrast to DQN and DDQN algorithm.

Related Work

There has been a lot of development in this series of DQN algorithms, with many variants including Multi-step Learning (Sutton 1988; Sutton and Barto 1998), Prioritized replay (Schaul et al. 2015), Distributional RL (Bellemare, Dabney, and Munos 2017), Noisy Nets (Fortunato et al. 2017) and also multiple combination of these which are called Rainbow Algorithm's variant (Matteo et al. 2017). Rainbow algorithm encompasses all the mentioned DQN variants here and DQN, DDQN and Dueling DDQN.
Each of these algorithms improve an aspect of DQN algorithm.

In Multi-step learning we perform n-step bootstrap in contrast to Q-learning which is one step boostrapping. This also reduces the deadly triad effect due to n-step bootstrapping which increases variance and reduces bias. Hence, this algorithm has the potential to exploit a good balance between variance and bias of the bootstrapped values and results in better performance and stability with fine-tuning.

Prioritized DQN uses a Prioritized Experience Replay buffer which samples the data based on TD-error as priority. This has multiple advantages as it trains the agent with the best states and transitions where there is more room for improvement.

Noisy Nets are improving the exploration aspect of DQN and similar algorithms. Here, learned perturbations of the network weights are used to drive exploration. The key insight is that a single change to the weight vector can induce a consistent, and potentially very complex, state-dependent change in policy over multiple time steps unlike dithering approaches where de-correlated noise (example: e-greedy) is added to the policy at every step. The perturbations are sampled from a noise distribution. The variance of the perturbation is a parameter that can be considered as the energy of the injected noise. These variance parameters are learned using gradients from the reinforcement learning loss function, along side the other parameters of the agent. This way of inducing stochasticity by adding noise to weight values enables better exploration and agent's policy results in greater results although with some increase in computation.

Distributional Reinforcement Learning, approximates distribution of the return instead of computing typical expected return. It uses a distributional perspective to design a new algorithm which applies Bellman's equation to the learning of approximate value distributions. This algorithm also achieves state of the art results.

The final variant Rainbow combines all these methods in a single algorithm and evaluates performance on all Atari 2600 games, where it demonstrated state of the art results for most games among DQN variants.

Project Description

In this project, focus is on the DQN, Double DQN and Dueling DDQN algorithms and extensively comparing them on 3 Atari game domains. A modified version of DQN has also been compared.

Image Processing and Environment Wrappers

Since the state space is in RGB image format, it can be resource intensive and inefficient to directly train on those state spaces. Hence, below are some modification methods used for image processing.
After extensive research and experiment, OpenAI gym wrappers have been selected for the pre-processing and also some environment modifications.

- NoopResetEnv(30): This class has three functions, init(), reset() and step(). It samples initial states by taking random number of no-ops on reset. Here, No-op is assumed to be action 0. It takes random number of no-op actions in reset() bounded by specified range which is this case is 30.
- FireResetEnv(): This class takes action on reset for environments that are fixed until firing.
- MaxAndSkipEnv(4): This class boosts performance speed of training by choosing maximum reward resulting observation of N observations (four in this case) and returns this as an observation for the step combining reward from all N observations. It takes the maximum of every pixel in the last two frames and sends it as an observation.
- ProcessFrame84(): This class converts the input state image from dimensions $(210, 160, 3)$ to $(84, 84, 1)$ using gray-scale conversion. It crops non-relevant parts of the image and then scales them down.
- BufferWrapper(4): This class stacks 4 images to the state, to increase the observation of game dynamics.
- ImageToPyTorch(): This class converts the input state from $(84, 84, 4)$ to $(4, 84, 84)$. Basically shifting the channel or in our case, number of stacked states.
- ScaledFloatFrame(): This class converts the grayscale pixel input values $[0, 255]$ to float value $[0.0, 1.0]$ and is applied at the end of processing.

DQN

For DQN the Algorithm 1 from (Mnih et al.,2015) has been implemented in PyTorch. The states in this algorithm are pre-processed using the wrappers discussed in previous sub section.
In all the three algorithms, the epsilon value is linear decayed till Final Exploration Frame mentioned in hyperparameter table in next section.

Neural Network Architecture:
Convolution layer 1: 32 channels, filter size 8*8, stride 4
Convolution layer 2: 64 channels, filter size 4*4, stride 2
Convolution layer 3: 64 channels, filter size 3*3, stride 1
Linear Layer 1: Hidden Dimension 512
Linear Layer 2: Output Dimension: Number of actions

Double DQN

For Double DQN, the same algorithm and same network architecture as DQN has been implemented with just the replacement of Target value being (3).

$$Y_t^{DQ} \equiv R_{t+1} + \gamma Q\left(S_{t+1}, argmax_a Q\left(S_{t+1}, a; \theta_t\right); \theta_t'\right)$$

Dueling DDQN

This algorithm has been implemented using Double DQN algorithm with the difference being in Network Architecture.
Additionally, the gradient norms in Dueling DDQN have been clipped to value of 10 for stability.

Convolution layer 1: 32 channels, filter size 8*8, stride 4
Convolution layer 2: 64 channels, filter size 4*4, stride 2
Convolution layer 3: 64 channels, filter size 3*3, stride 1

Value Network:

Algorithm 1: deep Q-learning with experience replay

Input:Initialize replay memory D to capacity N
Input:Initialize action-value function Q with random weights θ
Input:Initialize target action-value function Q with weights $\theta^{-} = \theta$
For episode $= 1, M$ do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1,$ T do
    With probability ε select a random action a_t
    otherwise select $a_t = argmax_{\mathrm{r}} Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and
    observe reward r_t and image x_{i+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess
    $\phi_{t+1} = \phi(s_{t+1})$
    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
    Set $y_j = \begin{cases} r_j & if A \\ r_j + \gamma \max_{\alpha'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & else \end{cases}$
    Perform a gradient descent step on
    $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\dot{Q} = Q$
  End For
End For

Where Condition A is $j + 1$ episode being terminal.

Linear Layer 1: Hidden Dimension 512
Linear Layer 2: Output Dimension: 1

Advantage (State-Action)
Linear Layer 1: Hidden Dimension 512
Linear Layer 2: Output Dimension: Number of actions

Further hyperparameters are in Experiments Section.

Experiments and Analysis

Atari 2600 game domains are popular environments to evaluate Deep Reinforcement Learning Algorithms. They range from simple to moderately complex environments. By training our Agents on them we can have a good understanding of the flaws and possible improvements in a resource efficient manner. These environments can be used as a substitution or a simulation of real world scenarios.

The domains chosen for current algorithm comparison are Pong, Krull and Boxing. These have good amount of variance in complexity of environment and also have the potential to bring out the uniqueness of our chosen algorithms, hence they can be properly evaluated and their positive effects and negative effects will be potentially observed.

Hyperparameters

The hyperparameters for Pong, Krull and Boxing environments have been provided in Figure 3 and Figure 4, respectively. These parameters have been tuned by multiple trial and error, utilizing Ray Tune library from PyTorch and referring to Algorithm specific original papers. It was observed that learning rate, Target network update step, and epsilon decay, total amount of training steps have dominant effect on learning of algorithm.

Training Platform

The Training platform for this project was Google Colab Pro on CUDA (GPU) and partially on local machine CPU. These have been resource intensive experiments.

Pong Experiment Analysis

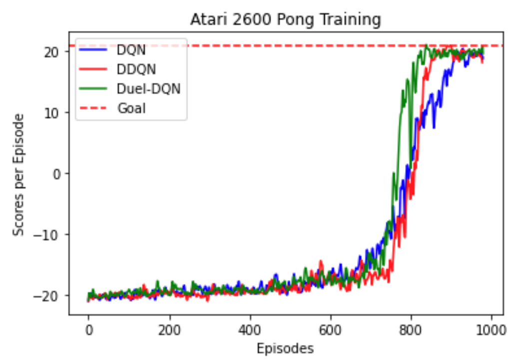

Figure 2: Atari 2600 Pong Results

Pong is relatively simpler domain of Atari 2600 yet complex enough for the algorithms to be tested over it for evaluation. It has action-space of size 6 and reward +1 and -1 over interaction in the environment, with the goal of reaching to a score of 21 before the opponent. It is a deterministic version environment.
Environment Reference: PongNoFrameskip-V4
The results in the graph (Figure 2) have been averaged by a sliding window of range 5. The algorithms took about 2 hours for training, but slightly more for Dueling DDQN, perhaps due to the dueling architecture and hence more weights to update.
Based on the graph results, it can be observed that all three algorithms converge almost to the optimal/goal value, with Dueling DDQN (duel) receiving 20.5 test score, Double DQN test score of 19.7 and DQN test score of 19.1. (Since it's deterministic environment, there's no need to average test scores) Further, it can be observed that Dueling DQN (duel) most of the time had higher score and rate of learning compared to DDQN and DQN. There can be seen an instance where for short duration overestimated state-action values in DQN had higher rate of score increment (slope) than Dueling DDQN, however the Dueling overcame it soon.
Double DQN (DDQN) initially performed similar to DQN however fell behing DQN, but later it recovered its learning

speed having learned optimal State-Action pairs compared to DQN and performed better. Dueling DDQN having the benefits of DDQN, also has the benefit of its dueling network structure which it seems gave it an advantage to distinguish between optimal states and the optimal state-action values at those states. It converged approximately 5 percent episodes before DDQN and 10 percent episodes before DQN based on the given results.
Another point to mention is without sliding window average, it could be observed that DQN had highest local variance followed by DDQN.

PONG

Mini-Batch Size	32
Replay Memory Size	50_000
Action Repeat	4
Target Network Update Frequency	10_000
Behavior Network Update Frequency	4
Discount Factor	0.99
Agent History Length	4
Learning Rate (DQN, DDQN)	2.25e-4
Learning Rate (Dueling DDQN)	1.25e-4
Initial Exploration (e)	1.0
Final Exploration (e)	0.01
Final Exploration Frame	1_600_000
Replay start size	5000
No-op max	30
Total Time steps	1_800_000
Maximum Episode Length	15000

Figure 3: Atari 2600 Pong Hyperparameters

Krull Experiment Analysis

Krull can be deemed moderate-high complexity domain of Atari 2600 with 18 action-spaces and many states. The agent receives multiple types of rewards while interacting with the environment (basically killing different types of monsters and even the same monsters gives different rewards if they have different speeds) The reward earning potential is spread over multiple states in this environment. The game ends when the player loses all lives. It is a deterministic version environment, however, as game proceeds difficulty level increases (Monsters have higher speeds and change their pattern).
Environment Reference: KrullNoFrameskip-V4
Due to high variance, the results in the graph have been averaged by a sliding window of range 11. The DDQN and DQN algorithms took about 6.5 hours for training, with around 7.5 hours for Dueling DDQN, due to the same reason as mentioned above.
Based on the graph (Figure 5) results, it can be observed that training by all three algorithms result in agent learning

Boxing and Krull

Mini-Batch Size	32
Replay Memory Size	50_000
Action Repeat	4
Target Network Update Frequency	10_000
Behavior Network Update Frequency	4
Discount Factor	0.99
Agent History Length	4
Learning Rate (DQN, DDQN)	2.25e-4
Learning Rate (Dueling DDQN)	1.25e-4
Initial Exploration (e)	1.0
Final Exploration (e)	0.01
Final Exploration Frame	4_250_000
Replay start size	5000
No-op max	30
Total Time steps	5_000_000
Maximum Episode Length	20000

Figure 4: Krull And Boxing Hyperparameters

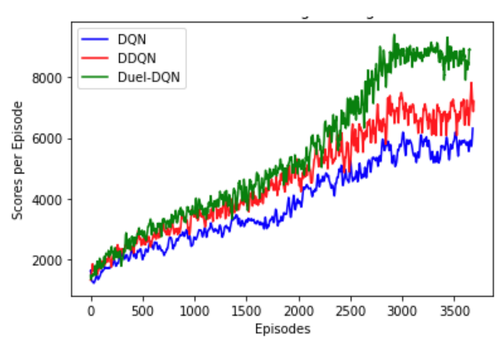

Figure 5: Atari 2600 Krull Results

how to maximize rewards in the environment, with Dueling DDQN (duel) converging at 9,014 test score, Double DQN 7481 test score and DQN 6272 test score.
Further, it can be observed that Dueling DQN (duel) most of the time had higher score and rate of learning compared to DDQN and DQN. It can be seen that due to the maximization bias DQN over-estimated certain state-actions and had a drop in scoring rate. Even though it did improve later it still converged to a lower score than DDQN and Dueling DQN.
It can be observed that DDQN score is lower than Duel DQN most of the time, however, it is still higher than DQN and has more stable learning compared DQN. However, once it reached its maximum score, it seems the environment difficulty increased (Monster patterns and speed changed) its score reduced for those episodes as it didn't have the optimal state-action for those scenarios.
This however, can only be seen slightly affecting the Dueling DQN value. The advantage of having a better approximated state value and perhaps a generalized Q-value can be seen here. Agent is able to adapt better to changing patterns. This doesn't seem to affect the DQN values perhaps due to the overestimated state-action values were in general applicable to the situation.
It can be seen that Dueling DDQN also had, a sudden increase in score rate increment in between training. Perhaps this was due to having learned considerable amount of optimal values of state and the optimal state-action values till those episodes that enabled distinguishing them and hence choosing better state and actions at those states. Finally, Dueling DDQN scored around 20 percent higher than DDQN and 42 percent higher compared to DQN.

Boxing Experiment Analysis

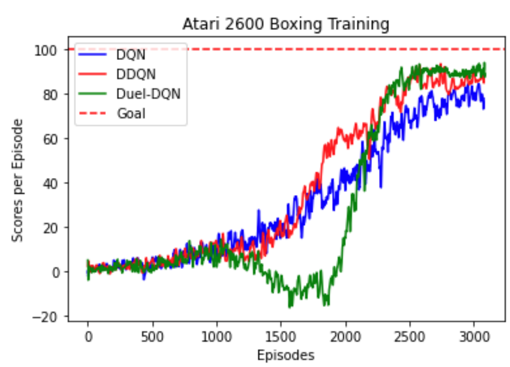

Figure 6: Atari 2600 Boxing Results

Boxing can be deemed moderate complexity domain of Atari 2600 with 18 action-spaces and multiple states. The agent receives multiple types of rewards, specifically two different rewards for two different types of punches +1 and +2 and -1 and -2 if get punched by opponent. The reward earning potential is spread over multiple states in this environment. The game ends if anyone of the player the scores 100, winner is the one who scores it first.
Environment Reference: BoxingNoFrameskip-V4
Due to high variance, the results in the graph have been averaged by a sliding window of range 11. The DDQN and DQN algorithms took about 6-7 hours for training, with around 7 hours for Dueling DDQN.
Based on the graph results, it can be observed that training by all three algorithms result in agent learning how to maximize rewards in the environment, with Dueling DDQN (duel) converging at 92.4 test score, Double DQN 85.1 test score and DQN 78.9 test score.
This graph (Figure 6) has multiple points. First considerable amount of score in Dueling during training was first in negative reward. For DDQN this observation is comparatively less and DQN even less, perhaps due to restricted exploration of ”seemingly” un-optimal state spaces. First, for Double DQN, the initial drop in learning was perhaps due to biased state-action values which was corrected and it overcame DQN quickly. In case of DQN, there was no apparent considerable drop and maximization bias didn't affect much initially however, after mid-way point in training it's slow score increment rate/slope was noticeable and it converged to a lower score value due to biased/overestimated state-action pair values.

Dueling DDQN is beneficial in scenarios where the action results are similar and state values hold significant importance. In this scenario, the right action at the right state, both are important. In this case, Dueling DDQN had similar starting score rate/slope as the other two, however, the negative rewards it received were more than the positive rewards per state. This occurrence happening multiple times lead to the state value being negative and hence also reducing State-action pair (Q-values) for all actions. At this stage perhaps the algorithm started to explore the values at other states which were non-negative/relatively higher. With more training it was able to finally distinguish the relative importance of state values and then started stabilizing and then choosing the right action at the right state. After being trained with this, the agent was able to make better choices and soon its rate of score increment overcame the other two algorithms and converged to a higher value compared to them.
Finally, Dueling DDQN scored around 8 percent higher than DDQN and 17 percent higher compared to DQN.

In Boxing, along with the three general algorithms, a simple new experiment was done with DQN.
In this DQN the learning rate was set to be 2.5e-4 and the gradients were clipped to norm of 10 as in Dueling DDQN. This modification can be called DQN-X. The result (Figure 7) showed DQN-X converging in close to 60 percent time steps as compared to standard DQN algorithm in Boxing environment.
The reason can be that clipping the gradients resulted in counteracting the overestimation bias of DQN. Further, increment in learning rate also gave it a boost and it con-

verged to an average test value of 82.5, higher than standard DQN (trained in 5M training steps), in just over 2.5 million training steps.
With this we can understand the importance of gradient clipping or stability in Q-values and also that learning rates for DQN and DDQN may be further tuned for better efficiency.
Hyperparameters for DQN-X algorithm are shown in Figure 8.

Conclusion

In this paper, we have learnt and observed the differences between three variant of DQN algorithms Double DQN (DDQN), Dueling DDQN and its base form algorithm on three Atari domains. A modified version of DQN was also compared with DQN, emphasizing importance of reducing variance in Q-values due to maximization bias in DQN algorithm. We conclude that DDQN performs better than DQN in moderate to high complexity environments, however, in some simple environments such as Pong the benefit may not be significant if the variance or the number of actions in environment are less or when the actions do not have significantly different impact in environment. We also clearly identified the dominance of Dueling DDQN in large state and action spaces and understand that it typically has better performance than DDQN and DQN. We also get an understanding that Dueling DDQN may lose its advantage in environments that have less states and actions and the impact of actions on the state value is relatively homogeneous.

Code Link has been provided below.

References

Bellemare, M. G.; Dabney, W.; and Munos, R. 2017. A distributional perspective on reinforcement learning. In ICML.

Fortunato, M.; Azar, M. G.; Piot, B.; Menick, J.; Osband, I.; Graves, A.; Mnih, V.; Munos, R.; Hassabis, D.; Pietquin, O.; Blundell, C.; and Legg, S. 2017. Noisy networks for exploration. CoRR abs/1706.10295.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing atari with deep reinforcement learning. CoRR abs/1312.5602.

van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double Q-learning. In Proc. of AAAI, 2094–2100.

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

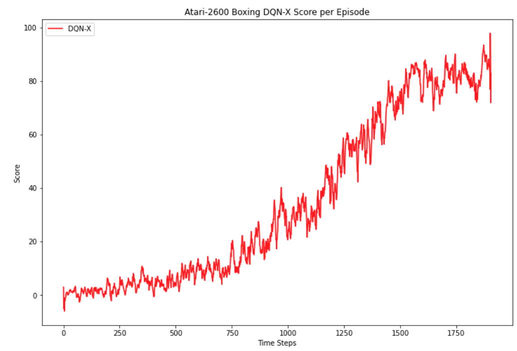

Figure 7: Atari 2600 Boxing Result DQN-X

DQN-X Boxing

Mini-Batch Size	32
Replay Memory Size	50_000
Action Repeat	4
Target Network Update Frequency	10_000
Behavior Network Update Frequency	4
Discount Factor	0.99
Agent History Length	4
Learning Rate (DQN-X)	2.5e-4
Initial Exploration (e)	1.0
Final Exploration (e)	0.01
Final Exploration Frame	2_200_000
Replay start size	5000
No-op max	30
Total Time steps	2_500_000
Maximum Episode Length	20000
Gradient Clip	10

Figure 8: DQN-X Hyperparameters

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

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

Wang, Z.; Schaul, T.; Hessel, M.; van Hasselt, H.; Lanctot, M.; and de Freitas, N. 2016. Dueling network architectures for deep reinforcement learning. In Proceedings of The 33rd International Conference on Machine Learning, 1995–2003.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized experience replay. In Proc. of ICLR.

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

Experiment Code Link

CODE LINK