Multi Agent Seminar

Kenzo Clauw

Vrije Universiteit Brussel

kclauw@vub.ac.be.

Abstract

Dueling Network Architectures changes the architecture of the Deep Reinforcement Learning algorithm by splitting the estimation of the Q-values into a separate value and an advantage stream. This paper discusses the implementation of a new architecture in combination with existing improvements to the algorithm. The first improvement prioritizes the experience replay memory, while the latter prevents overestimation of the Q-values. We will perform our experiments in the Atari game environment and statistically determine whether the improvements outperform the baseline algorithm.

Introduction

During the last decade, researchers have been trying to push the boundaries of artificial intelligence by successfully outperforming humans in games. Recently a team at Google Deepmind introduced the deep Q-network algorithm that combines Reinforcement Learning with Convolutional Neural Networks. By running the same algorithm, Deepmind managed to reach human level performance in 47 Atari games with only screen pixels as input. Unlike previous approaches in AI, DQN is general enough to find the best solution to the problem without being programmed for a certain task. Since then multiple improvements have been made to DQN but despite these efforts, most of these implementations are still using standard neural networks which aren't entirely made for Reinforcement Learning. The Dueling Network Architecture(DNN) Wang et al. (2015) maximizes the benefits of RL by changing the underlying architecture of the network. This allows us to get a faster estimate of the state and value function without having to learn the effect of redundant actions, resulting in better performance. The key thing to notice in DNA is that the network is compatible with existing improvements such as double DQN van Hasselt et al. (2015) and Prioritized Experience Replay Schaul et al. (2015). Double DQN resolves the overestimation of the Q-values by separating the selection and evaluation in the target network. The idea behind the prioritized experience replay is to increase the replay probability of experience tuples with the highest TD-error. As in the original papers, the DNAWang et al. (2015) is implemented with the aforementioned improvements and evaluated using the Space Invaders and Atlantis games in the Atari environment. In the experiments, the results are then compared to the original paper to determine if our application is capable of achieving similar results.

Background

Reinforcement Learning describes the task in which the agent interacts with an environment through a sequence of observations, actions and rewards. In the case of Atari, the agent receives M video frames representing the current state of the game $S_t = (x_{t-M+1}, ..., x_t)$ at timestep t. At each timestep, the agent select an action from a set of possible actions and in return receives a reward signal produced by the emulator. The goal of the agent is to selects the actions that maximize the cumulative discounted reward until a terminal state T is reached $R_t = \sum_{r=t}^{T} \gamma^{r-t}$.
The state-action function $Q^\pi(s, a)$ represents the maximum discounted future reward when performing action a in state s behaving according to a stochastic policy π, and continue optimally from that point onwards. The value of the state is then determined by the expected value of the Q-function. Both functions are defined as :

$$Q^\pi(s, a) = \mathbb{E}[R_t \mid s_t = s, a_t = a, \pi] \tag{1}$$

$$V^\pi(s) = \mathbb{E}_{a \sim s_{(\pi)}}[Q^\pi(s, a)] \tag{2}$$

The Q-function can be computed recursively with dynamic programming as follows :

$$Q^\pi(s, a) = \mathbb{E}_{s'}[r + \gamma \mathbb{E}_{s'}[Q^\pi(s', a')] \mid s, a, \pi] \tag{3}$$

When the Q-function is following the optimal policy, it is defined as $Q^*_{s,a} = max_\pi Q^\pi(s, a)$, the optimal value function is then $V^* = max(Q^*)$. Based on these assumptions, the bellman equation is rewritten as :

$$Q^*(s, a) = \mathbb{E}_{s'}[r + max_\pi Q^*(s, a) \mid s, a] \tag{4}$$

The influence of an action relative to the value of a state with the advantage function is determined as :

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \tag{5}$$

Deep Q-network

The goal of Reinforcement Learning is to estimate the Q-function by performing iterative improvement with the Bellman equation. In the case of Atari this method is unfeasible, the number of states is equal to the amount of atoms in the universe, which makes it impossible to store the results in a lookup-table. As a solution, the Q-values are estimated with a function approximate such as a neural network, the Q-function is then parametrized with weights θ. To train the Deep Q-network a sequence of loss functions is minimized that change at each iteration i :

$$L_i(\theta) = \mathbb{E}_{s,a,r,s'}[(y_i^{DQN} - Q(s,a;\ \theta_i))^2] \tag{6}$$

with the target network as :

$$y_i^{DQN} = r + \gamma max_{a'} Q(s', a';\ \theta^i) \tag{7}$$

To simulate the effect of supervised learning, the parameters of the target network are held fixed and are only updated every t time steps. The loss function is then differentiated with respect to the weights retrieving the following gradient :

$$\nabla_\theta L_i(\theta) = \mathbb{E}_{s,a,r,s'}[(y_i^{DQN} - Q(s,a;\ \theta_i))\nabla_\theta Q(s,a;\ \theta_i)] \tag{8}$$

A neural network is a form of supervised learning that trains from a large set of independent and pre-labeled data with a fixed underlying distribution. Training a neural network with samples from an online-learning algorithm breaks these assumptions because the data is usually highly correlated and the distribution changes at runtime.

To resolve these issues,the experience tuples $e_t = (s, a, r, s)$ are stored in a replay memory. When training the network with gradient descent, mini batches of experiences are sampled uniformly at random from this distribution.

Methods

Double Q-learning

The Q-learning algorithm is known to overestimate it's values when determining the action in the q-function. This is because the same max operation in equation 7 is used to both select and evaluate an action. According to van Hasselt et al. (2015), this issue is resolved by decoupling the selection from the evaluation. To implement this upgrade, the target network is replaced with the following formula:

$$y_i^{DDQN} = r + \gamma Q(s', argmax Q(s', a');\ \theta_i);\ \theta^-) \tag{9}$$

Prioritized experience replay

In the original DQN algorithm, the network is trained with random samples of experience from the memory. The problem with this method is that experiences that give us good learning are ignored in the process of learning. As an alternative, we could prioritize which experiences get replayed, when sampling batches from the memory. As an alternative, a prioriterization criteria determines which experiences get replayed, when sampling mini batches from the memory. The importance of each experience tuple is determined by the temporal difference error between the transitions :

$$TD_error = \mid Q(s,a) - Target \mid \tag{10}$$

Intuitively, the TD-error indicates how far away the current value is from the next estimation, if the error is big then the experience yields the most learning in the long. To avoid expensive sweeps of the entire memory, the TD-errors of the transitions that are replayed are updated. As a consequence, the transitions with low error are not getting replayed and the memory is sensitive to noise. Moreover, the greedy priorization approach of determining the error only focuses on the experiences with high errors. These issues might lead to over fitting because of the lack of diversity in the model. As a compromise, a stochastic sampling method is used that interpolates between pure greedy priorization and uniform random sampling. The probability of sampling transition i is then defined as :

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{11}$$

With the α parameter controlling the amount priorization. The experiences in the memory are ranked according to the following formula s:

$$p_i = \frac{1}{rank(i)} \tag{12}$$

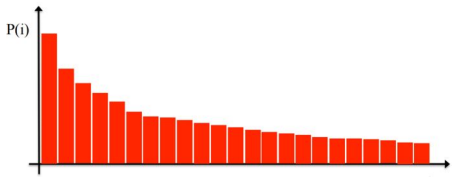

Figure 1:

As indicated in figure 1, the experiences are sorted according to the lowest td-error which results in a power distribution.

The problem with this sampling method is that the distribution is biased because of the ranked priorizations. To counter the bias in the experience distribution, the TD errors of the Q-learning update are multiplied with importance-sampling weights :

$$w(i) = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta \tag{13}$$

Implementation When implementing the memory, there are a few technical details to be resolved. To sample efficiently from the distribution of experiences, the tuples are stored in a priority queue based on a binary heap. The reason behind this is that a binary heap results in 0(1) performance when sampling the maximum element and updating a value is 0(log n). To prevent sampling from a distribution of N experiences, the power distribution is split into segments with equal probability. Mini batches are then sampled according to the segments.

Dueling Network Architecture

In environments with redundant actions, it is unnecessary to estimate the action value for each action. For many states, a more effective calculation of the q-values can be achieved by separating the estimation of the state value function from the state dependent action advantage function.

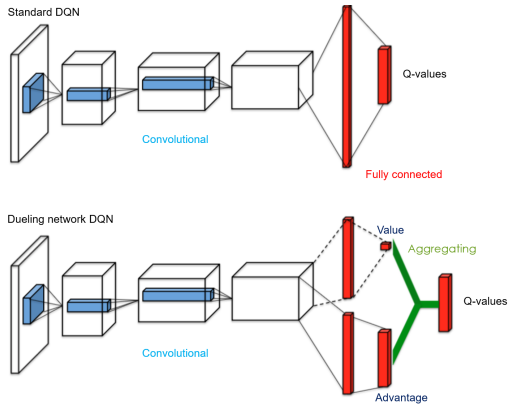

Table 1: Comparison of the original architecture vs the dueling network

The DNA splits the last aggregating layer in the network into two separate streams, as illustrated in figure 1. This architecture only changes the underlying estimation of the Q-values, the benefit is that the network is compatible with existing improvements of DQN, without changing the algorithm. The estimation of the Q-values can be rewritten as the sum of the advantage of an action with the value of the state as :

$$Q^{\pi}(s,a) = V^{\pi}(s) + A^{\pi}(s,a) \tag{14}$$

$$Q^{\pi}(s,a) = V^{\pi}(s) + (Q^{\pi}(s,a) - V^{\pi}(s)) \tag{15}$$

In the case of the optimal policy, Q becomes $Q(s,a^*) = V(s)$ thus the advantage function will be zero. Only good actions will affect the changes in value to Q, in the case of redundant actions, V dominates A and we get faster estimates of Q.

A naive Implementation of equation 10 would be to simply weight the scalar function V and the vector of actions in A :

$$Q^{\pi}(s,a;\ \theta,\alpha,\beta) = V^{\pi}(s;\ \theta,\beta) + A^{\pi}(s,a;\ \theta,\beta) \tag{16}$$

The problem with this approach is that the weighted functions are only an estimation, it would be wrong to assume that equation 12 represents the true values. Moreover, the estimations of the functions are not unique which results in bad performance. To resolve these issues,the advantage function is forced to output a zero in the best case :

$$Q^{\pi}(s,a;\ \theta,\alpha,\beta) = V^{\pi}(s;\ \theta,\beta) + (A^{\pi}(s,a;\ \theta,\beta) - \max_{a' \in A} A^{\pi}(s,a;\ ;\ \theta,\beta))I \tag{17}$$

However, in the original paper, they suggest to replace the max operator with an average :

$$Q^{\pi}(s,a;\ \theta,\alpha,\beta) = V^{\pi}(s;\ \theta,\beta) + (A^{\pi}(s,a;\ \theta,\beta) - \frac{1}{\mathbb{A}} \sum_{a'} A^{\pi}(s,a;\ ;\ \theta,\beta)) \tag{18}$$

Experiments

Because of the promising results when applying the DNA with improved and DDQN, Space Invaders and Atlantis are selected as the games in the Atari Environment. To determine if the algorithms are better then vanilla DQN, the average reward per episode of DDQN, prioritized replay and the dueling network is compared with vanilla DQN as baseline. The remaining results containing the max reward per episode, average Q-values and average loss are added in the appendix.

Parameters

The parameters for our experiments are similar to those in Mnih et al. (2015). However, to reduce the learning time, the size of the replay memory is reduced to 500000.

Setting

Prior to the results in this paper, several experiments were performed to reach similar results as in the original papers. Several attempts were made to reduce the learning time, for instance reducing the frame size resulted in no learning at all. In total there are 7 experiments to perform, as a consequence the experiments are only averaged at least 4 or 5 times. Due to the limited time in combination with the long queue times of hydra, proper testing is not performed on each experiment. The reason behind this is because the hydra cluster only allowed us to queue a limited amount of jobs in parallel.

Failed Experiment

The experiments involving the prioritized experience replay could not converge when learning. This might be due to the

fact that the memory size is reduced and the learning start is rather limited with the settings of the previous experiments. As a results , the algorithm lacks sufficient experiences with high TD error in the memory. Another issue with the replay memory is that the learning took exceptionally long compared to the other experiments. In Schaul et al. (2015) they sampled in segments from the distributions hereby reducing about 2%-4% procent of the runtime. Considering the long run times, it would have been a better idea to implement this in our application but we ran out of time to test this. Nevertheless, the limited results of the failed experiments are added in the paper.

Space Invaders

Double Q-learning As Figure 8 indicates, DDQN slightly outperforms the baseline in the Space Invaders environment. According to Schaul et al. (2015), DDQN should outperform the baseline with a relative percentage of around 50 procent. The reason behind the poor performance compared to the results in {Schaul et al. (2015) is because they tuned the hyper parameters.

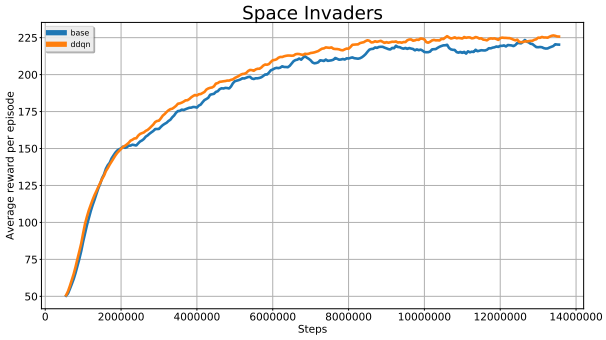

Figure 2:

Dueling Network Architecture In figure 2, it can be noticed that the dueling network outperforms the baseline. However, the gradients in our network are not clipped, which might indicate why the values drop near 1200000 steps.

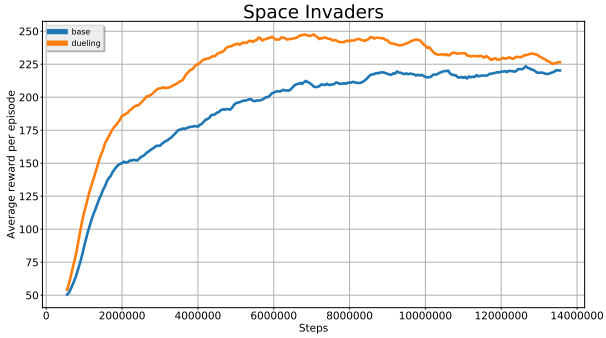

Figure 3:

DDQN and Dueling Network Architecture Combining DDQN with the dueling network results in a better performance compared to the baseline. These results indicate that DDQN stabilizes the learning when reaching the end of the episode. This results verifies that the value drops in the previous experiment is due to overestimations of the Q-function.

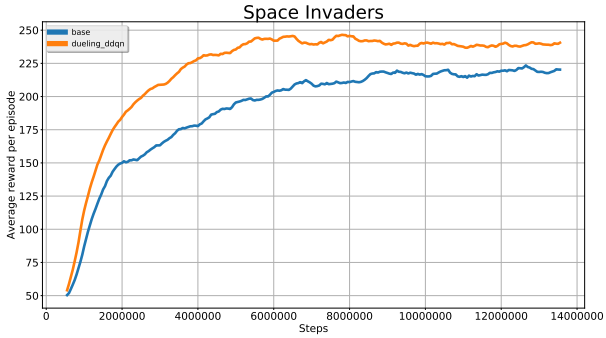

Figure 4:

Total Results The summary plot indicates the correlations between the results of each algorithm. In general, the effect of DDQN is only noticable in combination with the dueling network architecture. Proper hyper parameter tuning would lead to better results of DDQN without the dueling network.

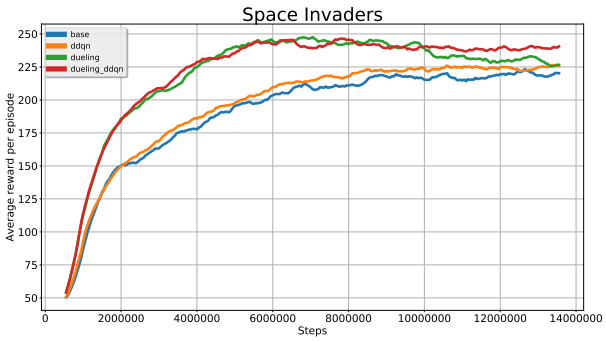

Figure 5:

Atlantis

Double Q-learning The figure illustrates performance of DDQN compared to the baseline.One thing to notice is that in Atlantis, DDQN needs at least 500000 steps of learning before it outperforms the baseline.This indicates that the overestimations happen when the baseline reaches it maximum value.

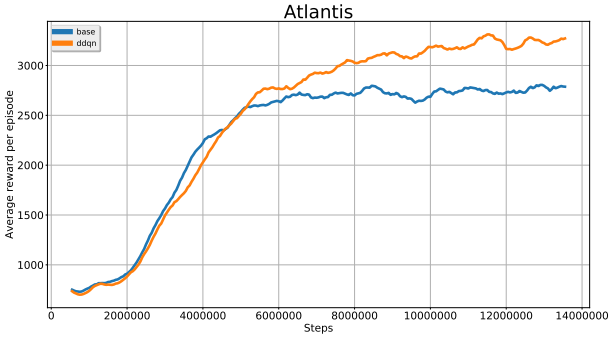

Figure 6:

Dueling Network Architecture It is shown in figure 7 that the dueling network exceptionally improves compared to the baseline. This assumption is further justified by the fact that the DNA already outperforms the baseline in the early phase of learning.

DDQN and Dueling Network Architecture As shown in figure 8, DDQN in combination with the dueling network

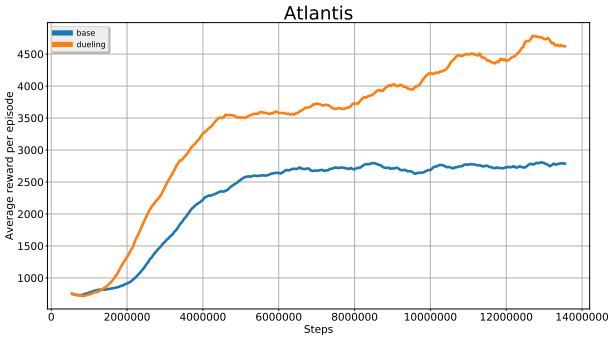

Figure 7:

outperforms the baseline. Furthermore, the stabilization of the overestimations around step 800000 allow the algorithm to reach the best overall performance in the Atlantis environment.

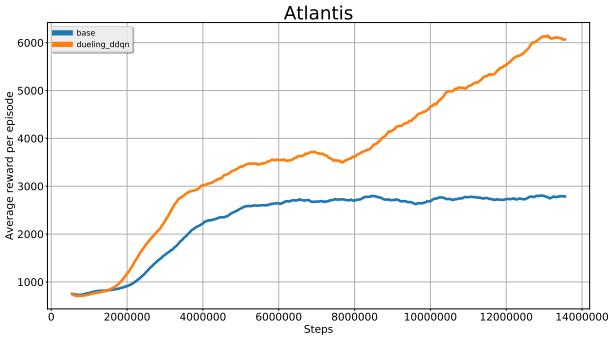

Figure 8:

Global As seen in the figure, DNN is efficient in the Atlantis environment and is even better when combined with DDQN.

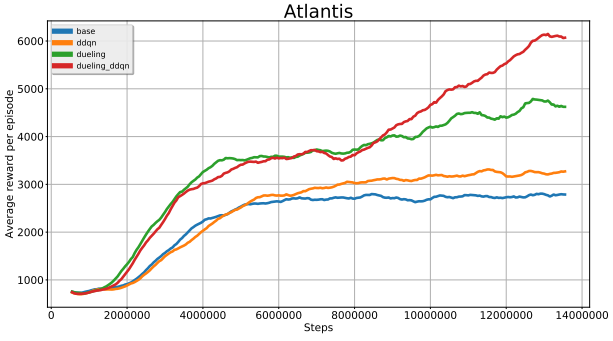

Figure 9:

Prior Experience Replay Figure 10 and 11 indicate the failed experiments when performing the prioritized experience replay. In Space Invaders, the algorithm seems to be stuck in a local minimum while Atlantis is not capable of any learning.

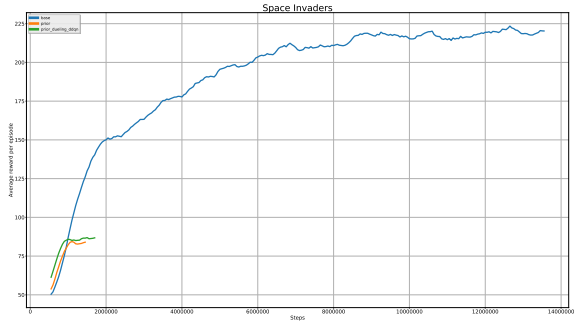

Figure 10:

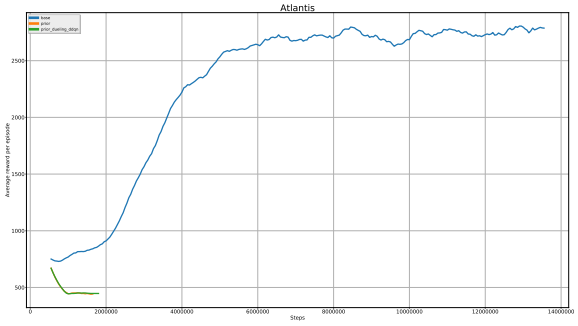

Figure 11:

Conclusion

In this paper, we have shown that the DNA increases the performance on two Atari Games. It can be noticed that both games suffer from overestimation, double Q-learning resolves this issue and in combination with the DNA reaches the best results.

appendix

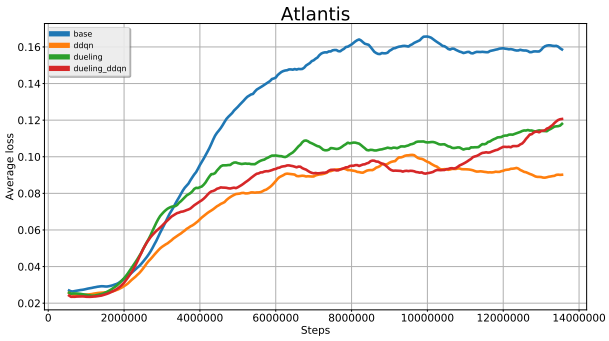

Figure 12:

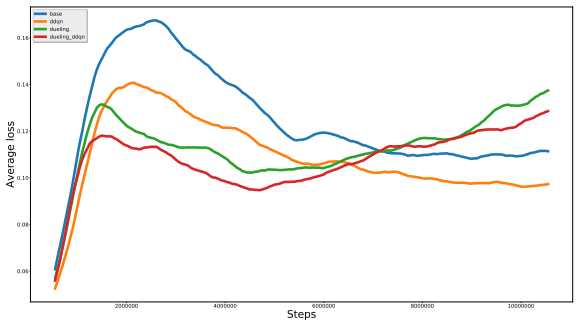

Figure 15:

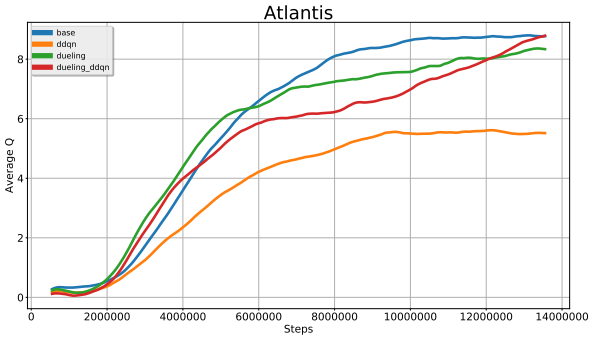

Figure 13:

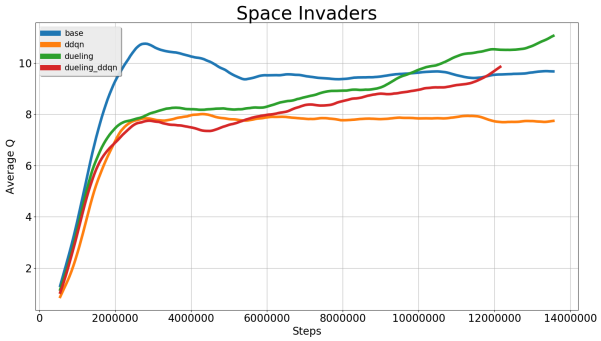

Figure 16:

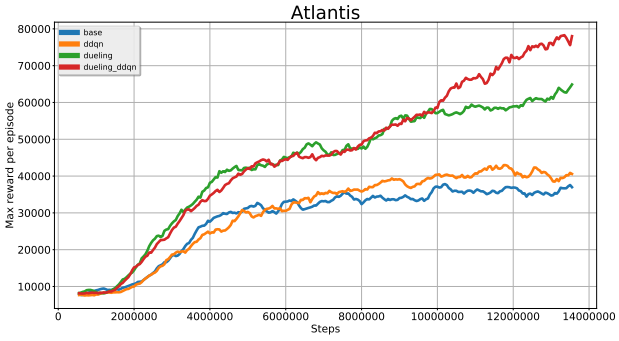

Figure 14:

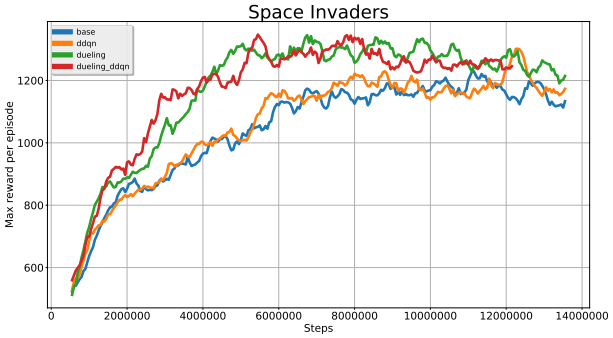

Figure 17:

References

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. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *CoRR*, abs/1511.05952.

van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461.

Wang, Z., de Freitas, N., and Lanctot, M. (2015). Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581.