Dueling Network Architectures for Deep Reinforcement Learning

Ziyu Wang ZIYU@GOOGLE.COM
Tom Schaul SCHAUL@GOOGLE.COM
Matteo Hessel MTTHSS@GOOGLE.COM
Hado van Hasselt HADO@GOOGLE.COM
Marc Lanctot LANCTOT@GOOGLE.COM
Nando de Freitas NANDODEFREITAS@GMAIL.COM
Google DeepMind, London, UK

Abstract

In recent years there have been many successes of using deep representations in reinforcement learning. Still, many of these applications use conventional architectures, such as convolutional networks, LSTMs, or auto-encoders. In this paper, we present a new neural network architecture for model-free reinforcement learning. Our dueling network represents two separate estimators: one for the state value function and one for the state-dependent action advantage function. The main benefit of this factoring is to generalize learning across actions without imposing any change to the underlying reinforcement learning algorithm. Our results show that this architecture leads to better policy evaluation in the presence of many similar-valued actions. Moreover, the dueling architecture enables our RL agent to outperform the state-of-the-art on the Atari 2600 domain.

1. Introduction

Over the past years, deep learning has contributed to dramatic advances in scalability and performance of machine learning (LeCun et al., 2015). One exciting application is the sequential decision-making setting of reinforcement learning (RL) and control. Notable examples include deep Q-learning (Mnih et al., 2015), deep visuomotor policies (Levine et al., 2015), attention with recurrent networks (Ba et al., 2015), and model predictive control with embeddings (Watter et al., 2015). Other recent successes include massively parallel frameworks (Nair et al., 2015) and expert move prediction in the game of Go (Maddison et al., 2015), which produced policies matching those of Monte Carlo tree search programs, and squarely beaten a professional player when combined with search (Silver et al., 2016).

In spite of this, most of the approaches for RL use standard neural networks, such as convolutional networks, MLPs, LSTMs and autoencoders. The focus in these recent advances has been on designing improved control and RL algorithms, or simply on incorporating existing neural network architectures into RL methods. Here, we take an *alternative but complementary approach* of focusing primarily on innovating a neural network architecture that is better suited for model-free RL. This approach has the benefit that the new network can be easily combined with existing and future algorithms for RL. That is, this paper advances a new network (Figure 1), but uses already published algorithms.

The proposed network architecture, which we name the *dueling architecture*, explicitly separates the representation of state values and (state-dependent) action advantages. The dueling architecture consists of two streams that represent the value and advantage functions, while sharing a common

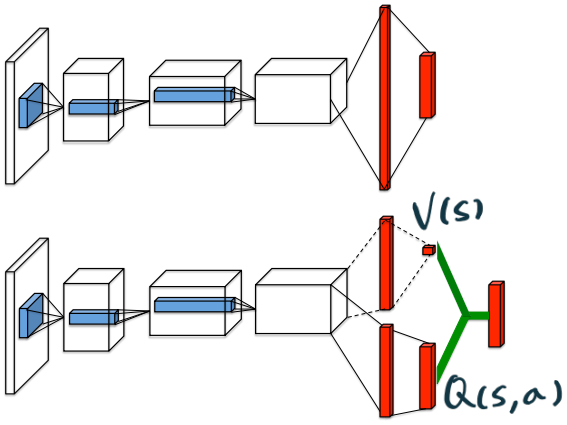


Figure 1. A popular single stream Q-network (**top**) and the dueling Q-network (**bottom**). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action; the green output module implements equation (9) to combine them. Both networks output Q-values for each action.

convolutional feature learning module. The two streams are combined via a special aggregating layer to produce an estimate of the state-action value function Q as shown in Figure 1. This dueling network should be understood as a single Q network with two streams that replaces the popular single-stream Q network in existing algorithms such as Deep Q-Networks (DQN; Mnih et al., 2015). The dueling network automatically produces separate estimates of the state value function and advantage function, without any extra supervision.

Intuitively, the dueling architecture can learn which states are (or are not) valuable, without having to learn the effect of each action for each state. This is particularly useful in states where its actions do not affect the environment in any relevant way. To illustrate this, consider the saliency maps shown in Figure 2[1]. These maps were generated by computing the Jacobians of the trained value and advantage streams with respect to the input video, following the method proposed by Simonyan et al. (2013). (The experimental section describes this methodology in more detail.) The figure shows the value and advantage saliency maps for two different time steps. In one time step (leftmost pair of images), we see that the value network stream pays attention to the road and in particular to the horizon, where new cars appear. It also pays attention to the score. The advantage stream on the other hand does not pay much attention to the visual input because its action choice is practically irrelevant when there are no cars in front. However, in the second time step (rightmost pair of images) the advantage stream pays attention as there is a car immediately in front, making its choice of action very relevant.

In the experiments, we demonstrate that the dueling architecture can more quickly identify the correct action during policy evaluation as redundant or similar actions are added to the learning problem.

We also evaluate the gains brought in by the dueling architecture on the challenging Atari 2600 testbed. Here, an RL agent with the same structure and hyper-parameters must be able to play 57 different games by observing image pixels and game scores only. The results illustrate vast improvements over the single-stream baselines of Mnih et al. (2015) and van Hasselt et al. (2015). The combination of prioritized replay (Schaul et al., 2016) with the proposed dueling network results in the new state-of-the-art for this popular domain.

1.1. Related Work

The notion of maintaining separate value and advantage functions goes back to Baird (1993). In Baird's original advantage updating algorithm, the shared Bellman residual update equation is decomposed into two updates: one for a state value function, and one for its associated advantage function. Advantage updating was shown to converge faster than Q-learning in simple continuous time domains in (Harmon et al., 1995). Its successor, the advantage learning algorithm, represents only a single advantage function (Harmon & Baird, 1996).

The dueling architecture represents both the value $V(s)$ and advantage $A(s,a)$ functions with a single deep model whose output combines the two to produce a state-action value $Q(s,a)$. Unlike in advantage updating, the representation and algorithm are decoupled by construction. Consequently, the dueling architecture can be used in combination with a myriad of model free RL algorithms.

There is a long history of advantage functions in policy gradients, starting with (Sutton et al., 2000). As a recent example of this line of work, Schulman et al. (2015) estimate advantage values online to reduce the variance of policy gradient algorithms.

There have been several attempts at playing Atari with deep reinforcement learning, including Mnih et al. (2015); Guo et al. (2014); Stadie et al. (2015); Nair et al. (2015); van Hasselt et al. (2015); Bellemare et al. (2016) and Schaul

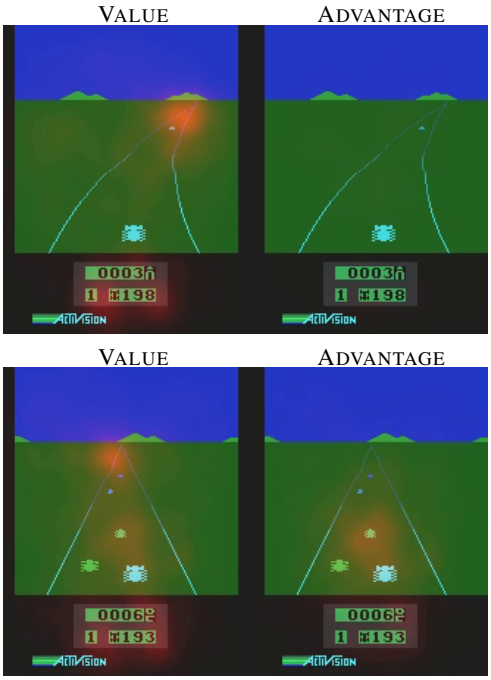


Figure 2. See, attend and drive: Value and advantage saliency maps (red-tinted overlay) on the Atari game Enduro, for a trained dueling architecture. The value stream learns to pay attention to the road. The advantage stream learns to pay attention only when there are cars immediately in front, so as to avoid collisions.

[1] https://www.youtube.com/playlist?list=PLVFXyCSfS2Pau0gBh0mwTxDmutywWyFBP

et al. (2016). The results of Schaul et al. (2016) are the current published state-of-the-art.

2. Background

We consider a sequential decision making setup, in which an agent interacts with an environment $\mathcal{E}$ over discrete time steps, see Sutton & Barto (1998) for an introduction. In the Atari domain, for example, the agent perceives a video s_t consisting of M image frames: $s_t = (x_{t-M+1}, \ldots, x_t) \in \mathcal{S}$ at time step t. The agent then chooses an action from a discrete set $a_t \in \mathcal{A} = \{1, \ldots, |\mathcal{A}|\}$ and observes a reward signal r_t produced by the game emulator.

The agent seeks maximize the expected discounted return, where we define the discounted return as $R_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau$. In this formulation, $\gamma \in [0, 1]$ is a discount factor that trades-off the importance of immediate and future rewards.

For an agent behaving according to a stochastic policy π, the values of the state-action pair (s, a) and the state s are defined as follows

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}\left[R_t \middle| s_t = s, a_t = a, \pi\right], \text{ and} \\ V^\pi(s) &= \mathbb{E}_{a\sim\pi(s)}\left[Q^\pi(s,a)\right]. \end{aligned} \tag{1}$$

The preceding state-action value function (Q function for short) can be computed recursively with dynamic programming:

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

We define the optimal $Q^*(s,a) = \max_\pi Q^\pi(s,a)$. Under the deterministic policy $a = \arg\max_{a'\in\mathcal{A}} Q^*(s,a')$, it follows that $V^*(s) = \max_a Q^*(s,a)$. From this, it also follows that the optimal Q function satisfies the Bellman equation:

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s, a\right]. \tag{2}$$

We define another important quantity, the *advantage function*, relating the value and Q functions:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s). \tag{3}$$

Note that $\mathbb{E}_{a\sim\pi(s)}\left[A^\pi(s,a)\right] = 0$. Intuitively, the value function V measures the how good it is to be in a particular state s. The Q function, however, measures the the value of choosing a particular action when in this state. The advantage function subtracts the value of the state from the Q function to obtain a relative measure of the importance of each action.

2.1. Deep Q-networks

The value functions as described in the preceding section are high dimensional objects. To approximate them, we can use a deep Q-network: $Q(s, a; \theta)$ with parameters θ. To estimate this network, we optimize the following sequence of loss functions at iteration i:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(y_i^{DQN} - Q(s,a;\theta_i)\right)^2\right], \tag{4}$$

with

$$y_i^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta^-), \tag{5}$$

where θ^- represents the parameters of a fixed and separate *target network*. We could attempt to use standard Q-learning to learn the parameters of the network $Q(s, a; \theta)$ online. However, this estimator performs poorly in practice. A key innovation in (Mnih et al., 2015) was to freeze the parameters of the target network $Q(s', a'; \theta^-)$ for a fixed number of iterations while updating the *online network* $Q(s, a; \theta_i)$ by gradient descent. (This greatly improves the stability of the algorithm.) The specific gradient update is

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

This approach is model free in the sense that the states and rewards are produced by the environment. It is also off-policy because these states and rewards are obtained with a behavior policy (epsilon greedy in DQN) different from the online policy that is being learned.

Another key ingredient behind the success of DQN is *experience replay* (Lin, 1993; Mnih et al., 2015). During learning, the agent accumulates a dataset $\mathcal{D}_t = \{e_1, e_2, \ldots, e_t\}$ of experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ from many episodes. When training the Q-network, instead only using the current experience as prescribed by standard temporal-difference learning, the network is trained by sampling mini-batches of experiences from $\mathcal{D}$ uniformly at random. The sequence of losses thus takes the form

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{U}(\mathcal{D})}\left[\left(y_i^{DQN} - Q(s,a;\theta_i)\right)^2\right].$$

Experience replay increases data efficiency through re-use of experience samples in multiple updates and, importantly, it reduces variance as uniform sampling from the replay buffer reduces the correlation among the samples used in the update.

2.2. Double Deep Q-networks

The previous section described the main components of DQN as presented in (Mnih et al., 2015). In this paper,

we use the improved Double DQN (DDQN) learning algorithm of van Hasselt et al. (2015). In Q-learning and DQN, the max operator uses the same values to both select and evaluate an action. This can therefore lead to overoptimistic value estimates (van Hasselt, 2010). To mitigate this problem, DDQN uses the following target:

$$y_i^{DDQN} = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta_i); \theta^-). \quad (6)$$

DDQN is the same as for DQN (see Mnih et al. (2015)), but with the target y_i^{DQN} replaced by y_i^{DDQN}. The pseudo-code for DDQN is presented in Appendix A.

2.3. Prioritized Replay

A recent innovation in prioritized experience replay (Schaul et al., 2016) built on top of DDQN and further improved the state-of-the-art. Their key idea was to increase the replay probability of experience tuples that have a high expected learning progress (as measured via the proxy of absolute TD-error). This led to both faster learning and to better final policy quality across most games of the Atari benchmark suite, as compared to uniform experience replay.

To strengthen the claim that our dueling architecture is complementary to algorithmic innovations, we show that it improves performance for both the uniform and the prioritized replay baselines (for which we picked the easier to implement rank-based variant), with the resulting prioritized dueling variant holding the new state-of-the-art.

3. The Dueling Network Architecture

The key insight behind our new architecture, as illustrated in Figure 2, is that for many states, it is unnecessary to estimate the value of each action choice. For example, in the Enduro game setting, knowing whether to move left or right only matters when a collision is eminent. In some states, it is of paramount importance to know which action to take, but in many other states the choice of action has no repercussion on what happens. For bootstrapping based algorithms, however, the estimation of state values is of great importance for every state.

To bring this insight to fruition, we design a single Q-network architecture, as illustrated in Figure 1, which we refer to as the dueling network. The lower layers of the dueling network are convolutional as in the original DQNs (Mnih et al., 2015). However, instead of following the convolutional layers with a single sequence of fully connected layers, we instead use two sequences (or streams) of fully connected layers. The streams are constructed such that they have they have the capability of providing separate estimates of the value and advantage functions. Finally, the two streams are combined to produce a single output Q function. As in (Mnih et al., 2015), the output of the network is a set of Q values, one for each action.

Since the output of the dueling network is a Q function, it can be trained with the many existing algorithms, such as DDQN and SARSA. In addition, it can take advantage of any improvements to these algorithms, including better replay memories, better exploration policies, intrinsic motivation, and so on.

The module that combines the two streams of fully-connected layers to output a Q estimate requires very thoughtful design.

From the expressions for advantage $Q^\pi(s,a) = V^\pi(s) + A^\pi(s,a)$ and state-value $V^\pi(s) = \mathbb{E}_{a\sim\pi(s)}[Q^\pi(s,a)]$, it follows that $\mathbb{E}_{a\sim\pi(s)}[A^\pi(s,a)] = 0$. Moreover, for a deterministic policy, $a^* = \arg\max_{a'\in\mathcal{A}} Q(s,a')$, it follows that $Q(s,a^*) = V(s)$ and hence $A(s,a^*) = 0$.

Let us consider the dueling network shown in Figure 1, where we make one stream of fully-connected layers output a scalar $V(s;\theta,\beta)$, and the other stream output an $|\mathcal{A}|$-dimensional vector $A(s,a;\theta,\alpha)$. Here, θ denotes the parameters of the convolutional layers, while α and β are the parameters of the two streams of fully-connected layers.

Using the definition of advantage, we might be tempted to construct the aggregating module as follows:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha), \quad (7)$$

Note that this expression applies to all (s,a) instances; that is, to express equation (7) in matrix form we need to replicate the scalar, $V(s;\theta,\beta)$, $|\mathcal{A}|$ times.

However, we need to keep in mind that $Q(s,a;\theta,\alpha,\beta)$ is only a parameterized estimate of the true Q-function. Moreover, it would be wrong to conclude that $V(s;\theta,\beta)$ is a good estimator of the state-value function, or likewise that $A(s,a;\theta,\alpha)$ provides a reasonable estimate of the advantage function.

Equation (7) is unidentifiable in the sense that given Q we cannot recover V and A uniquely. To see this, add a constant to $V(s;\theta,\beta)$ and subtract the same constant from $A(s,a;\theta,\alpha)$. This constant cancels out resulting in the same Q value. This lack of identifiability is mirrored by poor practical performance when this equation is used directly.

To address this issue of identifiability, we can force the advantage function estimator to have zero advantage at the chosen action. That is, we let the last module of the network implement the forward mapping

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a'\in|\mathcal{A}|} A(s,a';\theta,\alpha)\right) \quad (8)$$

Now, for $a^* = \arg\max_{a' \in \mathcal{A}} Q(s, a'; \theta, \alpha, \beta) = \arg\max_{a' \in \mathcal{A}} A(s, a'; \theta, \alpha)$, we obtain $Q(s, a^*; \theta, \alpha, \beta) = V(s; \theta, \beta)$. Hence, the stream $V(s; \theta, \beta)$ provides an estimate of the value function, while the other stream produces an estimate of the advantage function.

An alternative module replaces the max operator with an average:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha) \right). \quad (9)$$

On the one hand this loses the original semantics of V and A because they are now off-target by a constant, but on the other hand it increases the stability of the optimization: with (9) the advantages only need to change as fast as the mean, instead of having to compensate any change to the optimal action's advantage in (8). We also experimented with a softmax version of equation (8), but found it to deliver similar results to the simpler module of equation (9). Hence, all the experiments reported in this paper use the module of equation (9).

Note that while subtracting the mean in equation (9) helps with identifiability, it does not change the relative rank of the A (and hence Q) values, preserving any greedy or ϵ-greedy policy based on Q values from equation (7). When acting, it suffices to evaluate the advantage stream to make decisions.

It is important to note that equation (9) is viewed and implemented as part of the network and not as a separate algorithmic step. Training of the dueling architectures, as with standard Q networks (e.g. the deep Q-network of Mnih et al. (2015)), requires only back-propagation. The estimates $V(s; \theta, \beta)$ and $A(s, a; \theta, \alpha)$ are computed automatically without any extra supervision or algorithmic modifications.

As the dueling architecture shares the same input-output interface with standard Q networks, we can recycle all learning algorithms with Q networks (*e.g.*, DDQN and SARSA) to train the dueling architecture.

4. Experiments

We now show the practical performance of the dueling network. We start with a simple policy evaluation task and then show larger scale results for learning policies for general Atari game-playing.

4.1. Policy evaluation

We start by measuring the performance of the dueling architecture on a policy evaluation task. We choose this particular task because it is very useful for evaluating network architectures, as it is devoid of confounding factors such as the choice of exploration strategy, and the interaction between policy improvement and policy evaluation.

In this experiment, we employ temporal difference learning (without eligibility traces, *i.e.*, $\lambda = 0$) to learn Q values. More specifically, given a behavior policy π, we seek to estimate the state-action value $Q^\pi(\cdot, \cdot)$ by optimizing the sequence of costs of equation (4), with target

$$y_i = r + \gamma \mathbb{E}_{a' \sim \pi(s')} \left[Q(s', a'; \theta_i) \right].$$

The above update rule is the same as that of Expected SARSA (van Seijen et al., 2009). We, however, do not modify the behavior policy as in Expected SARSA.

To evaluate the learned Q values, we choose a simple environment where the exact $Q^\pi(s, a)$ values can be computed separately for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. This environment, which we call the *corridor* is composed of three connected corridors. A schematic drawing of the corridor environment is shown in Figure 3, The agent starts from the bottom left corner of the environment and must move to the top right to get the largest reward. A total of 5 actions are available: go up, down, left, right and no-op. We also have the freedom of adding an arbitrary number of no-op actions. In our setup, the two vertical sections both have 10 states while the horizontal section has 50.

We use an ϵ-greedy policy as the behavior policy π, which chooses a random action with probability ϵ or an action according to the optimal Q function $\arg\max_{a \in \mathcal{A}} Q^*(s, a)$ with probability $1 - \epsilon$. In our experiments, ϵ is chosen to be 0.001.

We compare a single-stream Q architecture with the dueling architecture on three variants of the corridor environment with 5, 10 and 20 actions respectively. The 10 and 20 action variants are formed by adding no-ops to the original environment. We measure performance by Squared Error (SE) against the true state values: $\sum_{s \in \mathcal{S}, a \in \mathcal{A}} (Q(s, a; \theta) - Q^\pi(s, a))^2$. The single-stream architecture is a three layer MLP with 50 units on each hidden layer. The dueling architecture is also composed of three layers. After the first hidden layer of 50 units, however, the network branches off into two streams each of them a two layer MLP with 25 hidden units. The results of the comparison are summarized in Figure 3.

The results show that with 5 actions, both architectures converge at about the same speed. However, when we increase the number of actions, the dueling architecture performs better than the traditional Q-network. In the dueling network, the stream $V(s; \theta, \beta)$ learns a general value that is shared across many similar actions at s, hence leading to faster convergence. This is a very promising result be-

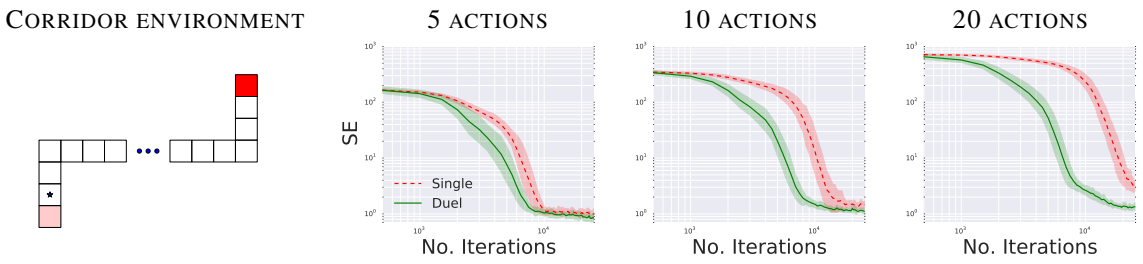


Figure 3. **(a)** The corridor environment. The star marks the starting state. The redness of a state signifies the reward the agent receives upon arrival. The game terminates upon reaching either reward state. The agent's actions are going up, down, left, right and no action. Plots **(b), (c)** and **(d)** shows squared error for policy evaluation with 5, 10, and 20 actions on a log-log scale. The dueling network (Duel) consistently outperforms a conventional single-stream network (Single), with the performance gap increasing with the number of actions.

cause many control tasks with large action spaces have this property, and consequently we should expect that the dueling network will often lead to much faster convergence than a traditional single stream network. In the following section, we will indeed see that the dueling network results in substantial gains in performance in a wide-range of Atari games.

4.2. General Atari Game-Playing

We perform a comprehensive evaluation of our proposed method on the Arcade Learning Environment (Bellemare et al., 2013), which is composed of 57 Atari games. The challenge is to deploy a single algorithm and architecture, with a fixed set of hyper-parameters, to learn to play all the games given only raw pixel observations and game rewards. This environment is very demanding because it is both comprised of a large number of highly diverse games and the observations are high-dimensional.

We follow closely the setup of van Hasselt et al. (2015) and compare to their results using single-stream Q-networks. We train the dueling network with the DDQN algorithm as presented in Appendix A. At the end of this section, we incorporate prioritized experience replay (Schaul et al., 2016).

Our network architecture has the same low-level convolutional structure of DQN (Mnih et al., 2015; van Hasselt et al., 2015). There are 3 convolutional layers followed by 2 fully-connected layers. The first convolutional layer has 32 8×8 filters with stride 4, the second 64 4×4 filters with stride 2, and the third and final convolutional layer consists 64 3×3 filters with stride 1. As shown in Figure 1, the dueling network splits into two streams of fully connected layers. The value and advantage streams both have a fully-connected layer with 512 units. The final hidden layers of the value and advantage streams are both fully-connected with the value stream having one output and the advantage as many outputs as there are valid actions[2]. We combine the value and advantage streams using the module described by Equation (9). Rectifier non-linearities (Fukushima, 1980) are inserted between all adjacent layers.

We adopt the optimizers and hyper-parameters of van Hasselt et al. (2015), with the exception of the learning rate which we chose to be slightly lower (we do not do this for double DQN as it can deteriorate its performance). Since both the advantage and the value stream propagate gradients to the last convolutional layer in the backward pass, we rescale the combined gradient entering the last convolutional layer by $1/\sqrt{2}$. This simple heuristic mildly increases stability. In addition, we clip the gradients to have their norm less than or equal to 10. This clipping is not standard practice in deep RL, but common in recurrent network training (Bengio et al., 2013).

To isolate the contributions of the dueling architecture, we re-train DDQN with a single stream network using exactly the same procedure as described above. Specifically, we apply gradient clipping, and use 1024 hidden units for the first fully-connected layer of the network so that both architectures (dueling and single) have roughly the same number of parameters. We refer to this re-trained model as *Single Clip*, while the original trained model of van Hasselt et al. (2015) is referred to as *Single*.

As in (van Hasselt et al., 2015), we start the game with up to *30 no-op* actions to provide random starting positions for the agent. To evaluate our approach, we measure improvement in percentage (positive or negative) in score over the better of human and baseline agent scores:

$$\frac{\text{Score}_{\text{Agent}} - \text{Score}_{\text{Baseline}}}{\max\{\text{Score}_{\text{Human}}, \text{Score}_{\text{Baseline}}\} - \text{Score}_{\text{Random}}}. \quad (10)$$

We took the maximum over human and baseline agent scores as it prevents insignificant changes to appear as

[2]The number of actions ranges between 3-18 actions in the ALE environment.

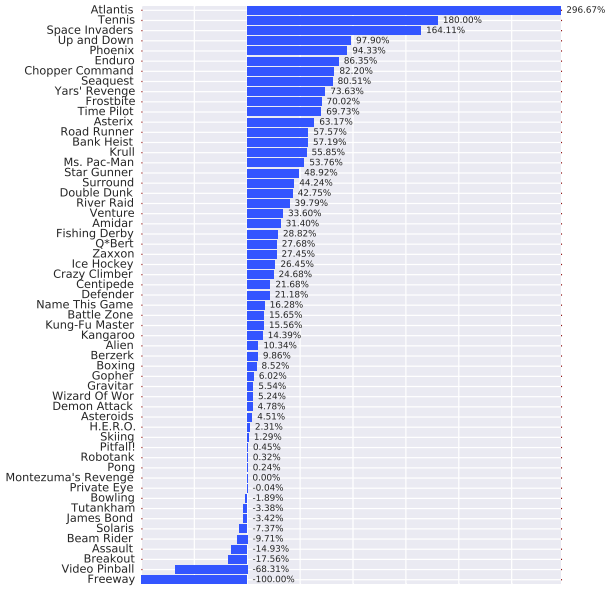


Figure 4. Improvements of dueling architecture over the baseline Single network of van Hasselt et al. (2015), using the metric described in Equation (10). Bars to the right indicate by how much the dueling network outperforms the single-stream network.

large improvements when neither the agent in question nor the baseline are doing well. For example, an agent that achieves 2% human performance should not be interpreted as two times better when the baseline agent achieves 1% human performance. We also chose not to measure performance in terms of percentage of human performance alone because a tiny difference relative to the baseline on some games can translate into hundreds of percent in human performance difference.

The results for the wide suite of 57 games are summarized in Table 1. Detailed results are presented in the Appendix.

Using this 30 no-ops performance measure, it is clear that the dueling network (Duel Clip) does substantially better than the Single Clip network of similar capacity. It also does considerably better than the baseline (Single) of van Hasselt et al. (2015). For comparison we also show results for the deep Q-network of Mnih et al. (2015), referred to as Nature DQN.

Figure 4 shows the improvement of the dueling network over the baseline Single network of van Hasselt et al. (2015). Again, we seen that the improvements are often very dramatic.

As shown in Table 1, Single Clip performs better than Single. We verified that this gain was mostly brought in by gradient clipping. For this reason, we incorporate gradient clipping in all the new approaches.

Table 1. Mean and median scores across all 57 Atari games, measured in percentages of human performance.

	30 no-ops		Human Starts	
	Mean	**Median**	**Mean**	**Median**
Prior. Duel Clip	**591.9%**	**172.1%**	**567.0%**	**115.3%**
Prior. Single	434.6%	123.7%	386.7%	112.9%
Duel Clip	**373.1%**	**151.5%**	**343.8%**	**117.1%**
Single Clip	341.2%	132.6%	302.8%	114.1%
Single	307.3%	117.8%	332.9%	110.9%
Nature DQN	227.9%	79.1%	219.6%	68.5%

Duel Clip does better than Single Clip on 75.4% of the games (43 out of 57). It also achieves higher scores compared to the Single baseline on 80.7% (46 out of 57) of the games. Of all the games with 18 actions, Duel Clip is better 86.6% of the time (26 out of 30). This is consistent with the findings of the previous section. Overall, our agent (Duel Clip) achieves human level performance on 42 out of 57 games. Raw scores for all the games, as well as measurements in human performance percentage, are presented in the Appendix.

Robustness to human starts. One shortcoming of the 30 no-ops metric is that an agent does not necessarily have to generalize well to play the Atari games. Due to the deterministic nature of the Atari environment, from an unique starting point, an agent could learn to achieve good performance by simply remembering sequences of actions.

To obtain a more robust measure, we adopt the methodology of Nair et al. (2015). Specifically, for each game, we use 100 starting points sampled from a human expert's trajectory. From each of these points, an evaluation episode is launched for up to 108,000 frames. The agents are evaluated only on rewards accrued after the starting point. We refer to this metric as *Human Starts*.

As shown in Table 1, under the Human Starts metric, Duel Clip once again outperforms the single stream variants. In particular, our agent does better than the Single baseline on 70.2% (40 out of 57) games and on games of 18 actions, Duel Clip is 83.3% better (25 out of 30).

Combining with Prioritized Experience Replay. The dueling architecture can be easily combined with other algorithmic improvements. In particular, prioritization of the experience replay has been shown to significantly improve performance of Atari games (Schaul et al., 2016). Furthermore, as prioritization and the dueling architecture address very different aspects of the learning process, their combination is promising. So in our final experiment, we investigate the integration of the dueling architecture with prioritized experience replay. We use the prioritized variant of DDQN (Prior. Single) as the new baseline algorithm, which replaces with the uniform sampling of the experi-

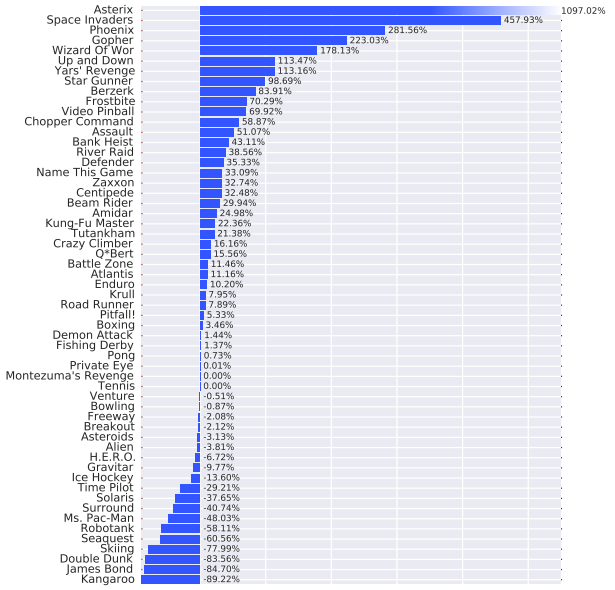


Figure 5. Improvements of dueling architecture over Prioritized DDQN baseline, using the same metric as Figure 4. Again, the dueling architecture leads to significant improvements over the single-stream baseline on the majority of games.

ence tuples by rank-based prioritized sampling. We keep all the parameters of the prioritized replay as described in (Schaul et al., 2016), namely a priority exponent of 0.7, and an annealing schedule on the importance sampling exponent from 0.5 to 1. We combine this baseline with our dueling architecture (as above), and again use gradient clipping (Prior. Duel Clip).

Note that, although orthogonal in their objectives, these extensions (prioritization, dueling and gradient clipping) interact in subtle ways. For example, prioritization interacts with gradient clipping, as sampling transitions with high absolute TD-errors more often leads to gradients with higher norms. To avoid adverse interactions, we roughly re-tuned the learning rate and the gradient clipping norm on a subset of 9 games. As a result of rough tuning, we settled on 6.25×10^{-5} for the learning rate and 10 for the gradient clipping norm (the same as in the previous section).

When evaluated on all 57 Atari games, our prioritized dueling agent performs significantly better than both the prioritized baseline agent and the dueling agent alone. The full mean and median performance against the human performance percentage is shown in Table 1. When initializing the games using up to 30 no-ops action, we observe mean and median scores of 591% and 172% respectively. The direct comparison between the prioritized baseline and prioritized dueling versions, using the metric described in Equation 10, is presented in Figure 5.

The combination of prioritized replay and the dueling network results in vast improvements over the previous state-of-the-art in the popular ALE benchmark.

Saliency maps. To better understand the roles of the value and the advantage streams, we compute saliency maps (Simonyan et al., 2013). More specifically, to visualize the salient part of the image as seen by the value stream, we compute the absolute value of the Jacobian of $\widehat{V}$ with respect to the input frames: $|\nabla_s \widehat{V}(s;\theta)|$. Similarly, to visualize the salient part of the image as seen by the advantage stream, we compute $|\nabla_s \widehat{A}(s, \arg\max_{a'} \widehat{A}(s,a');\theta)|$. Both quantities are of the same dimensionality as the input frames and therefore can be visualized easily alongside the input frames.

Here, we place the gray scale input frames in the green and blue channel and the saliency maps in the red channel. All three channels together form an RGB image. Figure 2 depicts the value and advantage saliency maps on the Enduro game for two different time steps. As observed in the introduction, the value stream pays attention to the horizon where the appearance of a car could affect future performance. The value stream also pays attention to the score. The advantage stream, on the other hand, cares more about cars that are on an immediate collision course.

5. Discussion

The advantage of the dueling architecture lies partly in its ability to learn the state-value function efficiently. With every update of the Q values in the dueling architecture, the value stream V is updated – this contrasts with the updates in a single-stream architecture where only the value for one of the actions is updated, the values for all other actions remain untouched. This more frequent updating of the value stream in our approach allocates more resources to V, and thus allows for better approximation of the state values, which in turn need to be accurate for temporal-difference-based methods like Q-learning to work (Sutton & Barto, 1998). This phenomenon is reflected in the experiments, where the advantage of the dueling architecture over single-stream Q networks grows when the number of actions is large.

Furthermore, the differences between Q-values for a given state are often very small relative to the magnitude of Q. For example, after training with DDQN on the game of Seaquest, the average action gap (the gap between the Q values of the best and the second best action in a given state) across visited states is roughly 0.04, whereas the average state value across those states is about 15. This difference in scales can lead to small amounts of noise in the updates can lead to reorderings of the actions, and thus make the nearly greedy policy switch abruptly. The dueling ar-

chitecture with its separate advantage stream is robust to such effects.

6. Conclusions

We introduced a new neural network architecture that decouples value and advantage in deep Q-networks, while sharing a common feature learning module. The new dueling architecture, in combination with some algorithmic improvements, leads to dramatic improvements over existing approaches for deep RL in the challenging Atari domain. The results presented in this paper are the new state-of-the-art in this popular domain.

References

Ba, J., Mnih, V., and Kavukcuoglu, K. Multiple object recognition with visual attention. In *ICLR*, 2015.

Baird, L.C. Advantage updating. Technical Report WL-TR-93-1146, Wright-Patterson Air Force Base, 1993.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Bellemare, M. G., Ostrovski, G., Guez, A., Thomas, P. S., and Munos, R. Increasing the action gap: New operators for reinforcement learning. In *AAAI*, 2016. To appear.

Bengio, Y., Boulanger-Lewandowski, N., and Pascanu, R. Advances in optimizing recurrent networks. In *ICASSP*, pp. 8624–8628, 2013.

Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

Guo, X., Singh, S., Lee, H., Lewis, R. L., and Wang, X. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *NIPS*, pp. 3338–3346. 2014.

Harmon, M.E. and Baird, L.C. Multi-player residual advantage learning with general function approximation. Technical Report WL-TR-1065, Wright-Patterson Air Force Base, 1996.

Harmon, M.E., Baird, L.C., and Klopf, A.H. Advantage updating applied to a differential game. In G. Tesauro, D.S. Touretzky and Leen, T.K. (eds.), *NIPS*, 1995.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436–444, 2015.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

Lin, L.J. *Reinforcement learning for robots using neural networks*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1993.

Maddison, C. J., Huang, A., Sutskever, I., and Silver, D. Move Evaluation in Go Using Deep Convolutional Neural Networks. In *ICLR*, 2015.

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.

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

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In *ICLR*, 2016.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438, 2015.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 01 2016.

Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Stadie, B. C., Levine, S., and Abbeel, P. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

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

Sutton, R. S., Mcallester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pp. 1057–1063, 2000.

van Hasselt, H. Double Q-learning. *NIPS*, 23:2613–2621, 2010.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double Q-learning. *arXiv preprint*

arXiv:1509.06461, 2015.

van Seijen, H., van Hasselt, H., Whiteson, S., and Wiering, M. A theoretical and empirical analysis of Expected Sarsa. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 177–184. 2009.

Watter, M., Springenberg, J. T., Boedecker, J., and Riedmiller, M. A. Embed to control: A locally linear latent dynamics model for control from raw images. In *NIPS*, 2015.

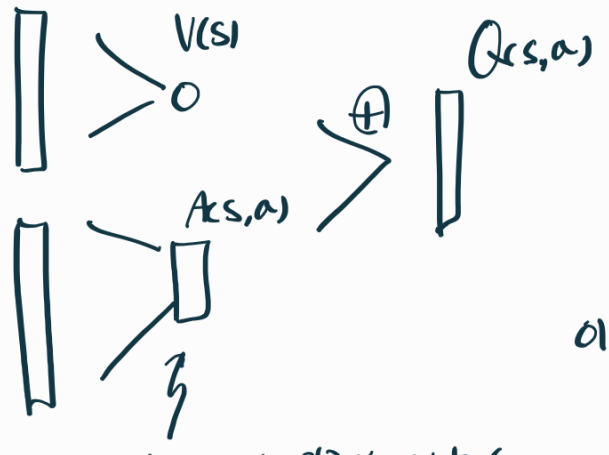
A. Double DQN Algorithm

Algorithm 1: Double DQN Algorithm.

input : $\mathcal{D}$ – empty replay buffer; θ – initial network parameters, θ^- – copy of θ
input : N_r – replay buffer maximum size; N_b – training batch size; N^- – target network replacement freq.
for *episode* $e \in \{1, 2, \ldots, M\}$ **do**
  Initialize frame sequence $\mathbf{x} \leftarrow ()$
  for $t \in \{0, 1, \ldots\}$ **do**
    Set state $s \leftarrow \mathbf{x}$, sample action $a \sim \pi_{\mathcal{B}}$
    Sample next frame x^t from environment $\mathcal{E}$ given (s, a) and receive reward r, and append x^t to $\mathbf{x}$
    if $|\mathbf{x}| > N_f$ **then** delete oldest frame $x_{t_{min}}$ from $\mathbf{x}$ **end**
    Set $s' \leftarrow \mathbf{x}$, and add transition tuple (s, a, r, s') to $\mathcal{D}$,
      replacing the oldest tuple if $|\mathcal{D}| \geq N_r$
    Sample a minibatch of N_b tuples $(s, a, r, s') \sim \text{Unif}(\mathcal{D})$
    Construct target values, one for each of the N_b tuples:
    Define $a^{\max}(s'; \theta) = \arg\max_{a'} Q(s', a'; \theta)$
    $y_j = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma Q(s', a^{\max}(s'; \theta); \theta^-), & \text{otherwise.} \end{cases}$
    Do a gradient descent step with loss $\|y_j - Q(s, a; \theta)\|^2$
    Replace target parameters $\theta^- \leftarrow \theta$ every N^- steps
  end
end

기존의 DQN은 network로 Q를 바로 뽑았고 학습 중인 network는 output이 Q인지 B를 제 학습한다.

Dueling DQN은 Q에 관한 property를 제공하여 학습 효율성을 증가시키고자 했다.



action의 개수 만큼의 node 수

$Q = V + A$ (stochastic)

Sampling을 통해 E[ ]를 구하면 $V = E[Q]$ 이다.

이 점을 이용해 ($E[Q] = V + E[A]$)

$= 0$ 이 되어야 함.

A의 값이 전달될 때 0에 가까울 수록 만들어서 전달한다. (결국은 A의 property 제공)

* deterministic일 땐 $V = \max_a Q$

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

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a' \in |\mathcal{A}|} A(s,a';\theta,\alpha)\right). \quad (8)$$