# Parameter Selection for the Deep Q-Learning Algorithm

**Nathan Sprague**
Department of Computer Science
James Madison University
Harrisonburg, VA 22801
spragunr@jmu.edu

## Abstract

Over the last several years deep learning algorithms have met with dramatic successes across a wide range of application areas. The recently introduced deep Q-learning algorithm represents the first convincing combination of deep learning with reinforcement learning. The algorithm is able to learn policies for Atari 2600 games that approach or exceed human performance. The work presented here introduces an open-source implementation of the deep Q-learning algorithm and explores the impact of a number of key hyper-parameters on the algorithm's success. The results suggest that, at least for some games, the algorithm is very sensitive to hyper-parameter selection. Within a narrow-window of values the algorithm reliably learns high-quality policies. Outside of that narrow window, learning is unsuccessful. This brittleness in the face of hyper-parameter selection may make it difficult to extend the use deep Q-learning beyond the Atari 2600 domain.

**Keywords:** Reinforcement learning, deep Q-Learning, feature learning

## 1 Introduction

The last several years have seen dramatic progress in the application of deep neural network architectures to problems in both supervised and unsupervised learning. The recently introduced deep Q-learning algorithm [6, 7] represents the first convincing application of deep feature learning to a non-trivial reinforcement learning task. Deep Q-Networks are able to approach or exceed human performance on a range of Atari 2600 games. Policies are learned completely from scratch based on pixel-level input.

The success of deep Q-learning in the Atari domain is a potentially important result. Scaling up reinforcement learning algorithms to handle continuous and high-dimensional tasks has been a longstanding challenge. Powerful feature learning algorithms are likely to be essential to making progress in this area. The success of deep Q-learning is also somewhat surprising. The algorithm has no theoretical convergence guarantees. On the contrary, it is well recognized that Q-learning has a tendency to be unstable when coupled with non-linear function approximation [8, 4]. These factors make it particularly important to reproduce the deep Q-learning results.

The goal of the work described here is twofold: first to reproduce the results of the original deep Q-learning workshop paper [6] by developing an open-source implementation that can be used as a starting point for future research[1], and second, to systematically explore the impact of several key hyper-parameters on the success of the algorithm.

## 2 Methods

The implementation described here uses the Arcade Learning Environment [2] as an interface to an Atari 2600 emulator. The deep Q-learning implementation is built on top of Theano [3, 1] and uses the neural network code developed for Sander Dieleman's galaxy zoo Kaggle competition entry [5].

The implementation follows the published description as closely as possible. The network architecture, image preprocessing, exploration schedule etc., all match the published specifications. As in the original paper, weight updates are handled using RMSProp with a batch size of 32.

The RMSProp algorithm [9] involves scaling weight updates on a per-weight basis according to a running average of the square of the gradient. The following two equations describe the update rules for tracking the average gradient values and updating weights.

$$r_{t,w} = \rho\, r_{t-1,w} + (1-\rho)\left(\frac{\partial L}{\partial w}\right)^2 \tag{1}$$

$$w_{t+1} = w_t + \alpha \frac{1}{\sqrt{r_{t,w}}} \frac{\partial L}{\partial w} \tag{2}$$

The two hyper-parameters that appear in the equations above are the decay rate $\rho$ and the step size parameter $\alpha$. The results presented below will explore the impact of these two parameters along with the discount rate $\gamma$

An additional factor that is not described in the original paper is the approach that was taken to weight initialization. For all of the results presented below, the bias weights are initialized to .1 and all other initial weight values are drawn from $\mathcal{N}(0, .0001)$.

## 3 Results

Results are reported below for three of the seven games mentioned in the original paper: Breakout, Seaquest and Enduro. Successful policies are learned for each game. Table 1 compares the best average reward received for each game with the corresponding results from [6].

Figure 1 illustrates the impact of hyper-parameter selection on the success of the deep Q-learning algorithm. Several trends are apparent in the data. First, it appears that the larger value of $\rho$ tends to lead to better learning results, particularly for Enduro and Seaquest. Although the results are not presented here, a number of unsuccessful preliminary tests were performed on Breakout with $\rho = .9$.

Second, at least among values examined here, it appears that no single set of hyper-parameters is optimal across all games. For example, the best hyper-parameter settings for Enduro results in no learning for Seaquest. It is possible to find settings that work reasonably well across multiple games, but any one choice will be a compromise.

[1] `https://github.com/spragunr/deep_q_rl`

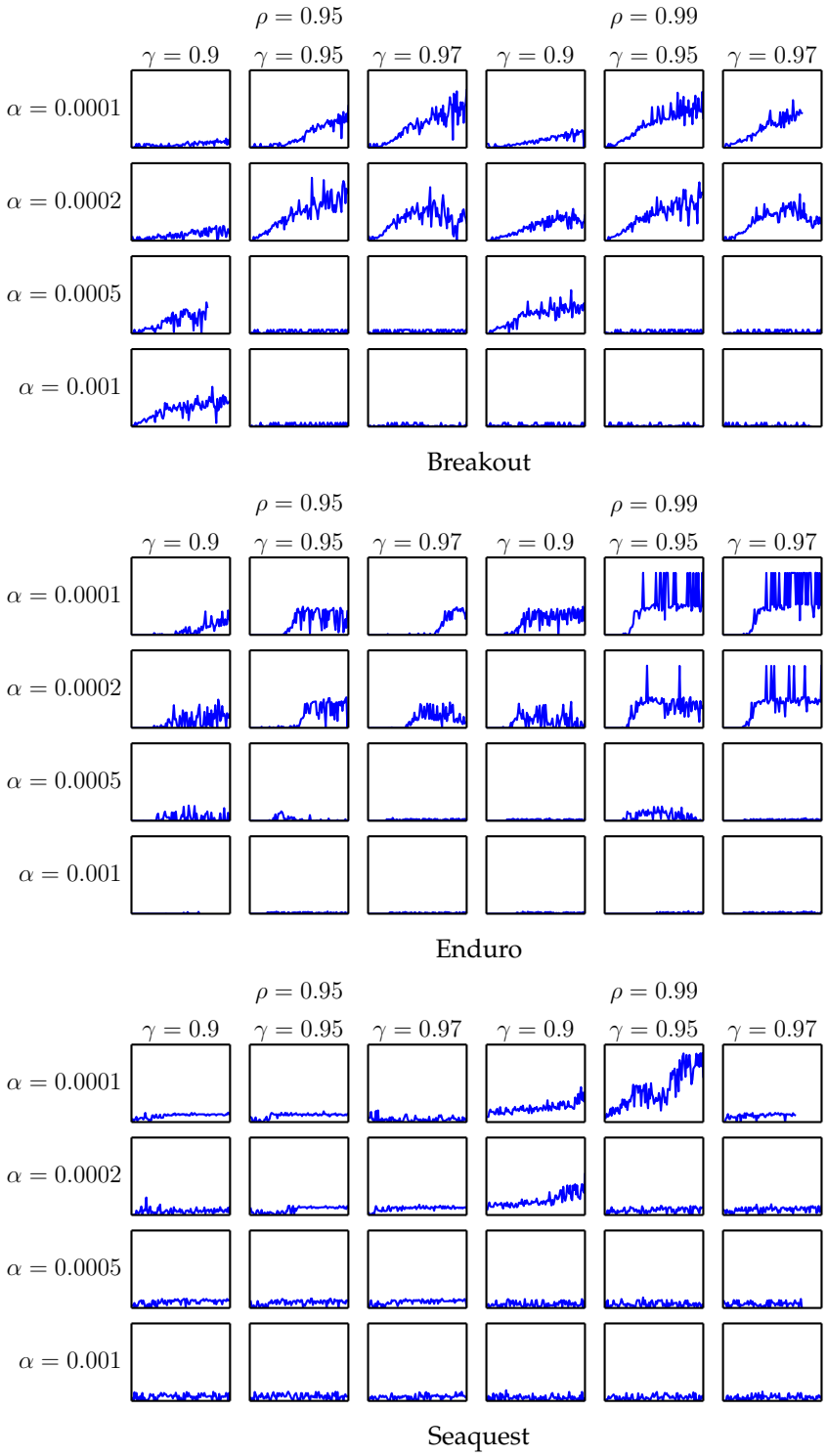


Figure 1: Each graph represents one training run with the indicated parameter settings for the indicated game. Training takes place over 100 epochs where each epoch represents 50,000 actions. Learning is evaluated after every epoch by testing the learned policy for 10,000 steps with an $\epsilon$-greedy policy using $\epsilon = .05$. The average per-game reward is plotted for each epoch. The lower bounds for all y-axes are 0. The upper bounds are 200 for Breakout, 1000 for Enduro, and 2500 for Seaquest. (Note that the incomplete data in some of the graphs represent jobs that had not completed at the time of submission.)

The most striking observation is that some games show significantly more sensitivity to hyper-parameter selection than others. In particular, 21 of the 24 different settings for Seaquest show no noticeable learning progress. Only one setting results in learning that is comparable to reported results.

| | Breakout | Enduro | Seaquest |
|---|---|---|---|
| DQN [6] | **168** | 470 | 1705 |
| DQN (current results) | 162 | **804** | **2228** |

Table 1: Maximum average total reward for learned policies. The results labeled "DQN [6]" represent the best policies discovered using an unspecified, but constant, set of parameters. The results labeled "DQN (current results)" represent the maximum values across all of the graphs in Figure 1.

Note that the graphs in Figure 1 represent a single training run for each parameter setting. It has been our experience that learning results tend to be reasonably consistent for a given set of hyper-parameters, but further experiments would be necessary to quantitatively determine how hyper-parameter values impact the variability of learning.

## 4 Conclusion

Even in the realm of supervised learning, the problem of tuning hyper-parameters for gradient-based optimization of neural networks is notoriously difficult. The situation becomes worse when these networks are used in conjunction with value function estimation. In supervised learning progress can be tracked by observing the value of the loss function or the error on a validation set. In reinforcement learning the value of the loss function is not a reliable indicator of progress: the value function is a moving target, so increases in the loss function may represent changes in the magnitude of the estimated value function rather than a lack of progress in learning. The only reliable way to monitor progress is to periodically evaluate the learned policy. This process is noisy, slow, and computationally expensive.

Given these challenges, it is particularly desirable that a reinforcement learning algorithm that incorporates deep neural networks be relatively robust to hyper-parameter selection. The results reported here suggest that improving the robustness and reliability of deep Q-learning may be a valuable avenue for future research.

These results were prepared in reference to the original deep Q-learning paper [6]. The authors of that paper have recently published an extended set of results obtained using a slightly modified version of the algorithm [7]. The updated version of the algorithm periodically copies the network weights so that the target Q-values are calculated using weights that are held constant across many updates. The authors report that this modification improves the stability of the algorithm. It will be a focus of future work to determine how this modification impacts the algorithm's sensitivity to hyper-parameter selection.

## 5 Acknowledgments

An initial version of the code described here was posted on-line in September 2014. A number individuals have contributed suggestions and improvements since then. I am particularly indebted to Alejandro Dubrovsky for his contributions.

## References

[1] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 06 2013.

[3] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[4] Justin Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, pages 369–376, 1995.

[5] Sander Dieleman. My solution for the galaxy zoo challenge. `http://benanne.github.io/2014/04/05/galaxy-zoo.html`, 2014.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis

Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[8] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*. Citeseer, 1993.

[9] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.