# Human-Guided DQN: Using Expert Human Gameplay Data on Atari 2600 Games for Deep Reinforcement Learning

Daniel Seita
University of California, Berkeley
Email: seita@berkeley.edu

*Abstract*— **Deep Reinforcement Learning is arguably the hottest and most popular subfield of Artificial Intelligence. In large part, this was popularized due to the success of agents in learning how to play Atari games from scratch, given only the input screen pixels and the game reward as input. While there has been substantial follow-up work on how to improve the Deep Q-Network (DQN) algorithm, there has not been much focus on how to utilize human guidance. In this paper, we report progress about an idea for using human expert gameplay on Atari games to boost DQN. During the exploration stage for Q-Learning, we substitute the random exploration with human actions. We investigate and discuss performance on two Atari games (Breakout and Space Invaders).**

## I. Introduction

Deep Learning can be used for challenging tasks in reinforcement learning, where the job of AI is not to perform "simple" classification as in [7], but to learn from high-dimensional, correlated data with a scalar reward signal that is noisy and exhibits complicated, long-term rewards. Most famously, [12] combined model-free reinforcement learning with Deep Learning techniques to develop an AI agent capable of learning how to play Atari 2600 games at a level matching or exceeding human performance. The AI only learned from the game frames and the score, just like how a human would learn. Similar techniques combine deep learning with Monte Carlo Tree Search [2], [16].

In this work, we combine Deep Reinforcement Learning and Learning from Demonstrations. In our setting, a human expert plays Atari games (Breakout and Space Invaders) and records the game frames and actions. Then, we experiment with an augmented version of DQN which utilizes the human gameplay data. This involves two main steps. The first is to train a classifier on the human data to map from game frames to actions. The second is to incorporate that classifier during the exploration phase of the DQN agent, when it is following an $\epsilon$-greedy policy. Rather than have the "$\epsilon$ cases" correspond to *random* actions, the AI can use those cases to follow the predicted *human action*.

We report on the results of our classifier and the revised AI agents. We show that standard convolutional neural networks (CNNs) can often identify the correct actions that the human expert took, but that combining this with a DQN agent does not generally improve performance. There are, however, several clear steps to engage for future work, and we hope to eventually be able to significantly boost the DQN algorithm by utilizing human gameplay data.

## II. Related Work

The Deep Q-Network (DQN) algorithm trains an AI agent using a variant of Q-learning [17]. In standard Q-Learning for solving a Markov Decision Process, one has state-action values $Q(s,a)$ for state $s$ and action $a$. This is the expected sum of discounted rewards for the agent starting at state $s$, taking action $a$, and from then on, playing optimally according to the action determined by the policy. With Atari games, the states are *sequences* of game frames $x_1, x_2, \ldots, x_t$ encountered during game play. The optimal action-value function $Q$ obeys the *Bellman equation* identity:

$$Q(s,a) = \mathbb{E}_{s'}\left[r + \gamma \cdot \max_{a'} Q(s',a') \mid s,a\right]. \tag{II.1}$$

The process of Q-Learning (or more generally, reinforcement learning) is to estimate the Q-values using the Bellman equation as an iterative update.

The states are extremely high dimensional; even with downsampling, one frame is an $(84 \times 84)$-dimensional input, and storing all $Q(s,a)$ values explicitly in a table is impractical. Therefore, the $Q(s,a)$ values are *approximated* by a neural network parameterized by its weights $\theta$, and it is $\theta$ that the Q-Learning algorithm must learn.

In practice, [12] uses a variant of online Q-Learning with two key ideas: experience replay for breaking the correlation among data points and a separate target network for generating the target terms in Equation II.1 to increase the algorithm's stability. The DQN trained with this variant of Q-Learning was able to excel at many Atari games, especially fast-paced games with simple rules such as Breakout. It was, however, weak on games such as Montezuma's Revenge, which requires substantial long-term strategy.

There has been a surge of follow-up work for training agents to play Atari games. For instance, [15] introduces prioritized experience replay to train DQN agents faster since the most important transitions (with respect to temporal difference error) would be considered more frequently. It is also possible to boostrap DQN [13] by borrowing techniques from the statistical method of boostrapping.

Th work of [20] presents a different neural network architecture specialized for reinforcement learning, and [19] proposes Double-DQN, which mitigates the problem of the "max" operator using the same values to both select and evaluate an action (thus leading to overly optimistic value

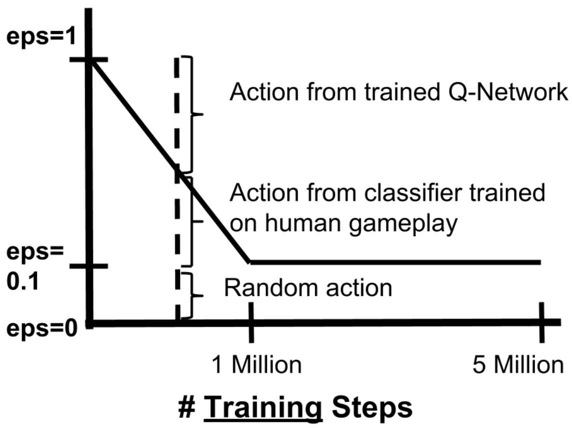


Fig. 1: The overall picture for our Human-Guided DQN algorithm with regards to $\epsilon$ decay during Q-Learning. During the exploration stage, instead of playing random actions with probability $\epsilon$, we perform the action chosen from our trained classifier with probability $\epsilon - 0.1$, up until 1 million steps, upon which time our classifier is ignored. Note that, as described in Section V, we sometimes adjust the number of steps taken to investigate the impact of a longer exploration period.

estimates). At the time of publication, it was the highest-quality DQN available, though it has since been surpassed by [10], which proposes asynchronous variants of DQN algorithms and uses an asynchronous actor-critic model to achieve state of the art Atari results. These results were finally surpassed by the *current* state of the art in [5].

While there has been much work concerning the technical aspects of DQN and its variants, there has been very little work on incorporating human aspects specifically to Atari games, the only major work of which is from [4]; we aim to rectify that with this research. Otherwise, this is a broader category of Learning from Demonstrations, a category which has been receiving more popularity including the seminal work of Maximum Entropy IRL [22] and DAGGER [14]. There has been more recent work about adjusting humans and the loss function [6], human supervision of robotic grasping [8], [9] along with that of cooperation with humans [3].

## III. Problem Statement and Idea

Our chief goal is to improve DQN by inserting a classifier trained on human actions as part of the $\epsilon$-greedy policy practiced by Q-Learning. In addition, we also hope to show that classifiers can successfully predict human actions.

### A. Algorithm Details

To ensure sufficient exploration of the state space, both standard Q-Learning and standard DQN follow $\epsilon$-greedy policies, where the action to take at state $s$ is selected to be $a = \arg\max_a Q(s,a)$ with probability $1-\epsilon$ and randomly selected otherwise. The code from [12] initializes at $\epsilon = 1.0$ and linearly anneals it down to $0.1$ after the first one million steps, and then fixes it thereafter.

Our objective is to provide potentially better state exploration by utilizing human data. Rather than choose an action with probability $\epsilon$, which will be high in the beginning, why not choose the action that a human would take? One hopeful outcome is that this will "bias" the state exploration space towards "better" areas, and then standard DQN would continue to build upon that positive exploration to obtain higher-quality rewards. In particular, we hope to see that this method provides improvement in the beginning of the exploration stage relative to standard DQN.

Figure 1 presents a picture of the overall pipeline. During the first million training steps where $\epsilon$ is linearly annealed, when the agent selects a random action, we usually (but not always) choose instead the action chosen by the classifier trained on human data. We leave a fixed probability of $\epsilon = 0.1$ to choose random actions, in part because certain games have actions which appear extremely infrequently (e.g., FIRE in Breakout during human gameplay occurs around five times per game) but will be executed via these random actions. We call our method *Human-Guided DQN*.

### B. Methodology and Implementation

There are three major steps for the experiments: getting human gameplay, developing a classifier to map from game frames to actions, and then plugging it into DQN.

*1) Human Gameplay:* To enable human gameplay, we modify the Arcade Learning Environment (ALE) [1] to enable a human to play. Each time step, we save the RGB game screen, the action taken, and the reward received. The human player is the author of this paper, who is an expert in Breakout and Space Invaders with roughly twenty hours and eight hours of prior gampeplay experience for these respective games.[1] We ultimately collected human gameplay data based on six hours of Breakout and five hours of Space Invaders. Due to the time-consuming nature of this work, we leave analysis on other Atari games to future work.

*2) Developing a Classifier:* With the data from the human gameplay, we apply all the standard preprocessing steps performed in [12], such as frame skipping and taking four consecutive (but non-skipped) frames to form a state. We then build a CNN using the same architecture as the Q-Network from [12], which uses three convolutional layers followed by two fully connected layers. The network has the number of output nodes equal to the number of actions chosen, thus allowing all $Q(\varphi_i, a_j)$ values to be determined for all actions $a_j$ during one pass with $\varphi_i$ as input. As mentioned in Section III-A, however, we filter the actions so that those which are designed to happen extremely infrequently are not considered (instead, they are played via the random actions or the standard Q-Network in DQN). For Deep Learning code, we use the Theano library [18]. Our classifier's code and supporting documents are open-source.[2]

*3) The DQN Algorithm:* Upon developing a classifier, we rigorously tuned it (see Section IV-A) to identify the strongest hyperparameters. We then modified a popular open-source implementation of DQN to load in the model weights into a new network (but with the same architecture) and to

[1] In [12], human experts had only two hours of training.

[2] `https://github.com/DanielTakeshi/Algorithmic-HRI`

TABLE I: Tuning Classifiers on Breakout

| Reg. | $\lambda$ | Train | Valid | Reg. | $\lambda$ | Train | Valid |
|---|---|---|---|---|---|---|---|
| $L_1$ | 0.00005 | **99.2** | 82.1 | $L_2$ | 0.00005 | **99.6** | 83.5 |
| $L_1$ | 0.0001 | 98.9 | 82.2 | $L_2$ | 0.0001 | 99.4 | 83.1 |
| $L_1$ | 0.0005 | 87.6 | **85.6** | $L_2$ | 0.0005 | **99.6** | 83.2 |
| $L_1$ | 0.001 | 85.0 | 84.4 | $L_2$ | 0.005 | 95.6 | 84.6 |
| $L_1$ | 0.005 | 85.1 | 83.9 | $L_2$ | 0.001 | 99.0 | 83.1 |
| $L_1$ | 0.01 | 77.6 | 76.0 | $L_2$ | 0.01 | 88.3 | **86.3** |
| $L_1$ | 0.05 | 33.7 | 36.8 | $L_2$ | 0.05 | 84.6 | 84.1 |

TABLE II: Tuning Classifiers on Space Invaders

| Reg. | $\lambda$ | Train | Valid | Reg. | $\lambda$ | Train | Valid |
|---|---|---|---|---|---|---|---|
| $L_1$ | 0.00005 | **96.3** | 67.5 | $L_2$ | 0.00005 | **97.8** | 66.0 |
| $L_1$ | 0.0001 | 94.7 | 68.2 | $L_2$ | 0.0001 | **97.8** | 66.9 |
| $L_1$ | 0.0005 | 76.5 | **74.5** | $L_2$ | 0.0005 | 96.5 | 68.4 |
| $L_1$ | 0.001 | 74.4 | 73.4 | $L_2$ | 0.001 | 95.1 | 68.2 |
| $L_1$ | 0.005 | 65.9 | 65.8 | $L_2$ | 0.005 | 81.0 | **72.7** |
| $L_1$ | 0.01 | 28.5 | 29.0 | $L_2$ | 0.01 | 75.9 | **72.7** |
| $L_1$ | 0.05 | 28.5 | 29.0 | $L_2$ | 0.05 | 64.7 | 64.0 |

enable it to use the classifier during the training process. Again, our code is open-source on GitHub.[3]

## IV. Results: Human Gameplay

### A. Classifier Performance

After collecting the human gameplay, we formed a dataset $\mathcal{D}$ consisting of state-action pairs $\mathcal{D} = \{\varphi_i, a_i\}_{i=1}^N$ encountered during human gameplay, where $\varphi_i$ consists of four $84 \times 84$ consecutive (non-skipped) grayscale images $\varphi_i = (s_{i-3}, s_{i-2}, s_{i-1}, s_i)$ and $a_i$ is the action chosen (by the human player) after observing game frame $s_i$.

During normal human gameplay, the distribution of actions is skewed. In Breakout, the NOOP action tends to be chosen far more often than LEFT or RIGHT, and the FIRE action is designed to occur only five times a game. We therefore do not incorporate FIRE in our Breakout classifier and we subsample NOOP actions. For Space Invaders, the FIRE actions (also including FIRE LEFT and FIRE RIGHT) occur more frequently, so we included them in the classifier, but still only kept a third of the NOOP actions. Thus, Breakout results in a balanced three-way classification task, while Space Invaders results in a (less-balanced) six-way classification task. Figure 2 shows the action distributions.

As described in Section III-B, we built a CNN following the architecture from [12]. We split the data into training, validation, and testing sets, and tuned weights via either $L_1$ or $L_2$ regularization based on validation set performance.[4]

Tuning results are shown in Tables I and II, with bold indicating the best settings. With low $\lambda$, the net gets arbitrarily high performance on the training data, but performs poorly otherwise. For both games, we keep only the best-performing net on the validation set. In Breakout, the best net achieves 86.3% accuracy on the validation set, and for Space Invaders, that figure is 74.5%, which is lower due to the more challenging classification task. The per-class results on the *test* set are shown in Tables III and IV, respectively. The results indicate that all three actions are fairly easy to detect for Breakout, but for Space Invaders, our classifiers are better at detecting LEFT and RIGHT actions.

[3] `https://github.com/DanielTakeshi/deep_q_rl`

[4] In this setting, it is probably OK to tune on the test set, but we decided to stick to best practices and tune on the validation set. For both games, the best-performing settings on the validation set also corresponded to the top settings for the test sets.

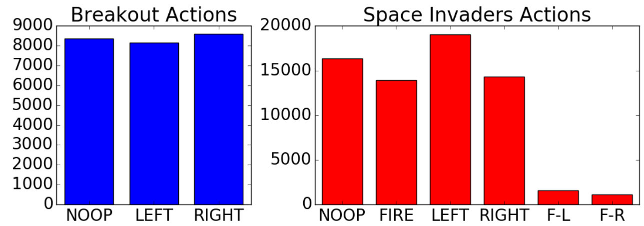


Fig. 2: The distribution of actions for Breakout (left) and Space Invaders (right) within all our data used for classification. This is the distribution obtained *after* preprocessing steps to eliminate most NOOP actions. For Breakout, we randomly select $x$ NOOP actions out of the full set, where $x$ is the average of the LEFT and RIGHT action counts. For Space Invaders, we kept one third of the NOOP actions. F-L and F-R represent "FIRE LEFT" and "FIRE RIGHT", respectively.

TABLE III: Per-Class Performance on Breakout

| | NOOP | LEFT | RIGHT |
|---|---|---|---|
| Correct | 1463 | 1492 | 1390 |
| Total | 1635 | 1784 | 1605 |
| Percent | 89.4 | 83.6 | 86.6 |

TABLE IV: Per-Class Performance on Space Invaders

| | NOOP | FIRE | LEFT | RIGHT | F-L | F-R |
|---|---|---|---|---|---|---|
| Correct | 2255 | 1660 | 2462 | 3378 | 34 | 83 |
| Total | 3310 | 2699 | 2817 | 3870 | 246 | 306 |
| Percent | 68.1 | 61.5 | 87.3 | 87.2 | 13.8 | 27.1 |

### B. Classifier Investigation

We further investigate the top-performing Breakout and Space Invaders classifier by visually inspecting example outputs. Figure 3 demonstrates examples of two states $\varphi$ for each of the two games, where the classifier *correctly* predicted the action the human player took with high confidence (as determined by the softmax probability distribution).

In Breakout, we see that the action LEFT is chosen in a state where the ball is moving towards the player and to the left. The NOOP action is chosen in a state where the ball was just hit and is moving up, thus meaning there is no need to move just yet. In Space Invaders, it is advantageous to clear out columns, and fortunately the classifier is able to detect this on our examples. The LEFT action is chosen in a state which has just one "alien" on the left hand column so that the agent can shoot it. The FIRE action is chosen in a state where the agent has already cleared a few aliens in a column but has not yet cleared it.

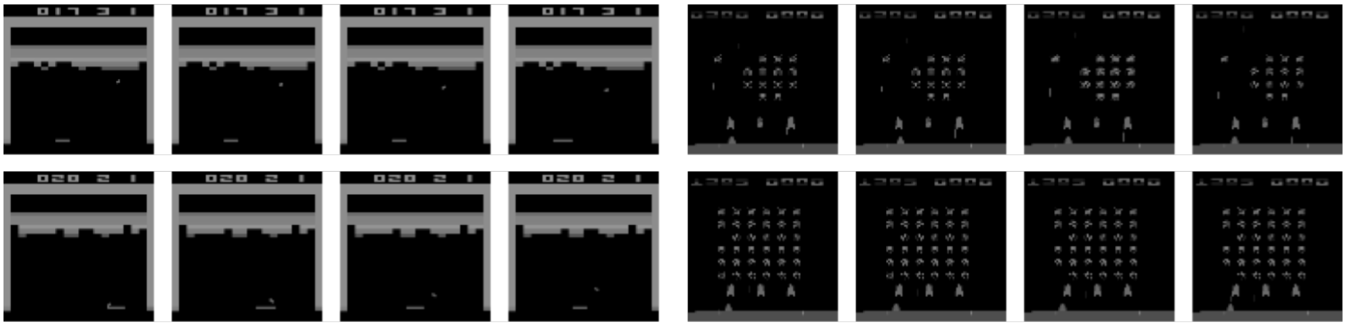

Fig. 3: Four examples of “states” $\varphi_t = (s_{t-3}, s_{t-2}, s_{t-1}, s_t)$ encountered during human gameplay, two for Breakout (left) and two for Space Invaders (right). Top left: the action LEFT is correctly predicted (99.7% confidence). Bottom left: the action NOOP is correctly predicted (99.1% confidence). Top right: the action LEFT is correctly predicted (99.1% confidence). Bottom right: the action FIRE is correctly predicted (91.9% confidence).

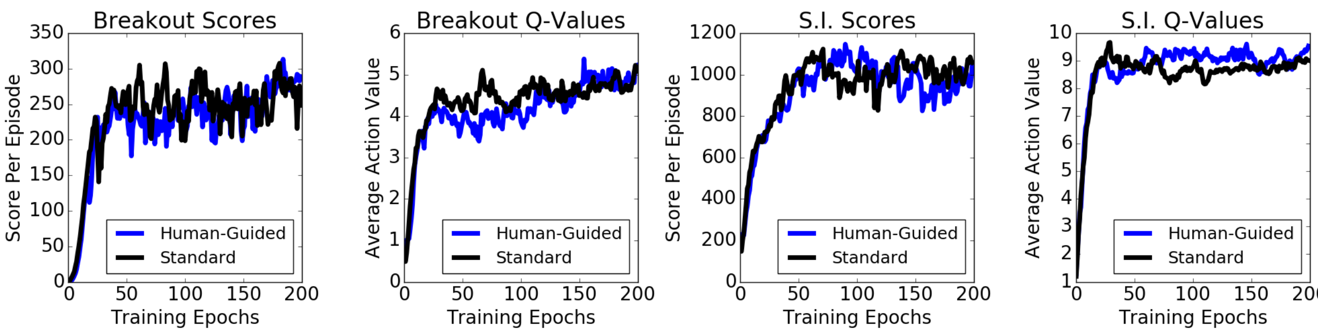


Fig. 4: Plots which compare the performance of standard DQN (black) versus Human-Guided DQN (blue) on Breakout (first two subplots) and Space Invaders (last two subplots). The metrics used are the average reward obtained per episode and the average $Q$-value encountered during gameplay. Since the outcome is extremely noisy, the reward per episode is a smoothed version, computed with $avg = 0.7 \cdot avg + 0.3 \cdot newscores$.

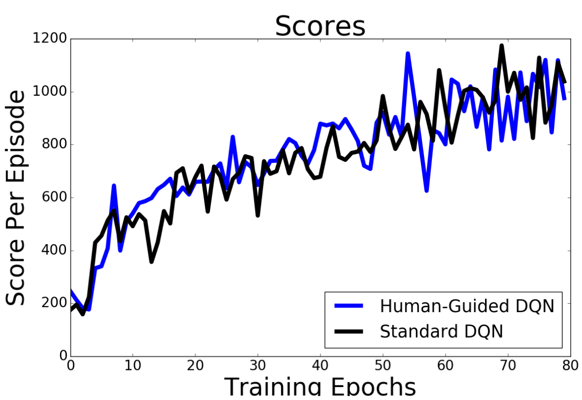


Fig. 5: Comparing average score per episode for standard DQN (black) versus Human-Guided DQN (blue) on Space Invaders. This is similar to the Space Invaders subplots of Figure 4, except that we do not do any smoothing, and that the exploration period is 10x longer and the number of epochs has been reduced from 200 to 80. (The exploration phase “ends” after epoch 40, which is when $\epsilon = 0.1$.)

## V. Results: Modified DQN

**TODO explain setup**

To evaluate the DQN algorithms, we use two standard metrics: the average reward per episode, and the average action value (i.e. $Q(s, a)$-value) encountered. The latter is often used since it tends to have smoother curves than the reward per episode. All experiments were run on a single computer with an NVIDIA TITAN GPU (with Pascal).

Figure 4 shows our results with respect to both games and both evaluation metrics. We see that there is no clear difference between the performance of our algorithm and standard DQN for both of the metrics. Each curve only represents one trial due to the heavy computational requirements (a single run takes approximately 35 hours) so future analysis should consider running additional trials to obtain error bars. It is worth noting that Breakout is one of the easiest games for DQN to learn from, so it may be difficult to improve performance while sticking with the DQN framework.

One factor which hinders the previous analysis[5] is that the exploration period is likely too short for us to discern any noticeable improvements. The exploration period consists of when $\epsilon > 0.1$, and for the previous settings, that ends after the fourth of 200 epochs. This does not give us enough data points, so we re-ran the Space Invaders experiment using an exploration period 10x longer. Due to computational limitations, we limited the runs to 80 epochs total.

Figure 5 shows the results with the improved experiment settings. We see that the results of the two algorithms are roughly equivalent, suggesting that using the human-guided classifier for the random actions does not hinder performance. The Human-Guided DQN is not superior than the default, however, so it remains an open question as to how to best utilize human gameplay to improve DQN.

[5] This was due to some extra settings inside the open source DQN code which were not obvious at first glance.

## VI. Conclusions

In this work, we have made efforts to use Learning from Demonstration techniques to boost the performance of DQN agents on Atari games. We collected many hours of human expert gameplay data on Breakout and Space Invaders, and provided evidence that a CNN can largely predict the human expert's actions. Our Human-Guided DQN utilized this classifier for exploration. Unfortunately, the results did not substantially improve, but we believe there is still untapped potential in this project. In future work, we will run Human-Guided DQN with longer exploration phases. Second, we will try using *human experience replay*: combining the usual experience replay with subsampled human gameplay data. Finally, a more elaborate goal is to work on training attention models [11], [21], the idea being that for these games, there are only a few important signals that matter for rewards, and this can be trained into an attention model, which hopefully can be used to improve DQN performance.

## References


[1] 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*, 2013.

[2] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3338–3346. Curran Associates, Inc., 2014.

[3] D. Hadfield-Menell, S. J. Russell, P. Abbeel, and A. Dragan. Cooperative inverse reinforcement learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3909–3917. Curran Associates, Inc., 2016.

[4] I. Hosu and T. Rebedea. Playing atari games with deep reinforcement learning and human checkpoint replay. *arXiv*, abs/1607.05077, 2016.

[5] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv*, abs/1611.05397, 2016.

[6] B. Kim, A. M. Farahmand, J. Pineau, and D. Precup. Learning from limited demonstrations. In C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, editors, *NIPS*, pages 2859–2867, 2013.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[8] M. Laskey, C. Chuck, J. Lee, J. Mahler, S. Krishnan, K. Jamieson, A. D. Dragan, and K. Y. Goldberg. Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations. *arXiv*, abs/1610.00850, 2016.

[9] M. Laskey, S. Staszak, W. Y.-S. Hsieh, J. Mahler, F. T. Pokorny, A. D. Dragan, and K. Goldberg. Shiv: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *ICRA*, pages 462–469, 2016.

[10] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1928–1937, 2016.

[11] V. Mnih, N. Heess, A. Graves, and k. kavukcuoglu. Recurrent models of visual attention. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2204–2212. Curran Associates, Inc., 2014.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[13] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4026–4034. Curran Associates, Inc., 2016.

[14] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 627–635, 2011.

[15] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations*, Puerto Rico, 2016.

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

[17] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[18] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[19] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 2094–2100, 2016.

[20] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1995–2003, 2016.

[21] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In D. Blei and F. Bach, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2048–2057. JMLR Workshop and Conference Proceedings, 2015.

[22] B. D. Ziebart, A. Maas, J. A. D. Bagnell, and A. Dey. Maximum entropy inverse reinforcement learning. In *Proceeding of AAAI 2008*, July 2008.