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

Daniel Seita
Computer Science Division
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 – in other words, exact how a human would learn how to play. While there has been substantial follow-up work on how to improve the performance of agents in such games, there has been very little work that incorporates human guidance in the process. In this paper, we report our progress about an idea for using human expert gameplay on Atari 2600 games to boost the performance of Deep Reinforcement Learning agents. Specifically, we focus on the Deep Q-Network algorithm, and during the exploration stage for Q-Learning, we explore how substituting the random exploration with human actions impacts gameplay. We report on progress for two Atari 2600 games (Breakout and Space Invaders) and show the potential for this idea to eventually improve the performance of DQN agents.**

I. Introduction

The now-popular technique of deep learning can be used for challenging tasks in reinforcement learning, where the job of an AI agent 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 may exhibit complicated, long-term rewards. For instance, [12] combined model-free reinforcement learning with deep learning techniques to develop an AI agent capable of learning how to play several 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 rely on Monte Carlo Tree Search [2], including the well-publicized AlphaGo AI agent [16].

Nonetheless, despite the progress advanced by neural networks, many questions still remain about how exactly neural networks learn, and it is still unclear if this underlying "process" is at all similar to the way that humans would learn. One way to explore this question would be to try and directly incorporate learning from demonstrations to boosting the performance of agents.

In this report, a human expert plays games, Breakout and Space Invaders, and we augment the learning process of neural network agents with human data to accelerate training to get fast, high-quality policies. This step involves two main steps. The first is to train a classifier to map from game frames to actions based on human data. The second step is to incorporate the classifier during the exploration phase of the neural network agent, when it is following an ϵ-greedy policy. Rather than have the "ϵ cases" correspond to *random* actions, the AI agent can use those cases to follow the *human action*.

We report on the results of our classifier and the AI agents. We show that standard convolutional neural networks (CNNs) can often identify the correct actions for humans to take, but that combining this inside a DQN agent does not generally improve performance that much, though there are several obvious steps to take for future work. Ultimately, we hope to better understand the human learning and deep learning processes that enable the corresponding agents to successfully play Atari games and hope to eventually boost the DQN process with human 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 $x_1, x_2, \dots, x_t$ encountered during game play[1]. The optimal action-value function Q obeys the *Bellman equation* identity:

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

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 (84×84)-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 θ, and it is θ that the Q-Learning algorithm must learn.

In practice, [12] uses a variant of online Q-Learning (with an ϵ-greedy policy for exploration) with two key ideas: experience replay for breaking the correlation among data

[1]Technically, [12] reports that states are sequences of game frames *and* actions: $x_1, a_1, x_2, \dots, a_t$. When doing Q-Learning, however, their code only considers four consecutive frames and does not take into account actions other than the current one under consideration.

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. In [2], they augment training using data collected *offline* through the use of Monte-Carlo tree search planning. The "offline player," while substantially better than DQN, cannot play in real time, but can be used to improve DQN's performance. The work of [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 overoptimistic value 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 the 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]. Otherwise, however, 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].

The aim of this work is to resolve the gap between DQN (and more generally, Deep Reinforcement Learning) and Learning from Demonstrations by augmenting the DQN algorithm with data on human gameplay. We also hope to better understand the connection between human learning versus deep learning.

III. Problem Statement and Idea

The chief goal of this project is to improve the performance of DQN by inserting a classifier trained on human actions as part of the ϵ-greedy policy practiced as a result of Q-Learning. Along the way, we present auxiliary results regarding the potential for training a classifier directly on human gameplay data.

A. Algorithm Details

To ensure sufficient exploration of the state space, both standard Q-Learning and standard DQN follow ϵ-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.

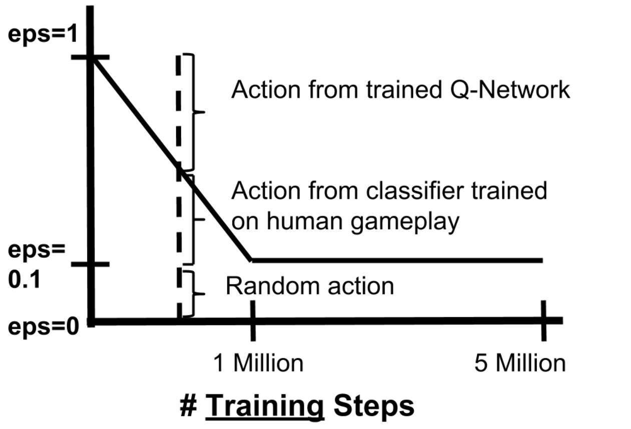
eps=1
Action from trained Q-Network
Action from classifier trained on human gameplay
eps= 0.1
Random action
eps=0
1 Million
5 Million
# Training Steps

Fig. 1: The overall picture for our Human-Guided DQN algorithm with regards to ϵ decay during Q-Learning. During the exploration stage, instead of playing random actions with probability ϵ, 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 **TODO TODO**, we adjust the number of steps taken to investigate the impact of a longer exploration period.

Our objective is to provide potentially better state exploration by utilizing human data. Rather than choose an action with probability ϵ, which will be very high in the beginning, why not choose the action that a human would take in this situation? 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 a faster 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 ϵ 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: 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

TABLE I: Classifier Performance on Breakout

Reg.	λ	Train	Valid	Reg.	λ	Train	Valid
L_1	0.00005			L_2	0.00005		
L_1	0.0001			L_2	0.0001		
L_1	0.0005			L_2	0.0005		
L_1	0.005			L_2	0.001		
L_1	0.001			L_2	0.005		
L_1	0.05			L_2	0.05		
L_1	0.01			L_2	0.01		

TABLE II: Classifier Performance on Space Invaders

Reg.	λ	Train	Valid	Reg.	λ	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	0.005	65.9	65.8	L_2	0.005	81.0	**72.7**
L_1	0.001	74.4	73.4	L_2	0.001	95.1	68.2
L_1	0.05	28.5	29.0	L_2	0.05	64.7	64.0
L_1	0.01	28.5	29.0	L_2	0.01	75.9	**72.7**

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.[2] 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 the standard preprocessing steps performed in [12], such as frame skipping and taking four consecutive (but non-skipped) frames to form a state. (For additional details, see Appendix I.) 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, and has the number of outputs equal to the number of actions chosen. As mentioned in Section III-A, however, we filter the actions so that those which happen infrequently or a fixed amount of times per game 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.[3]

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 enable it to use the classifier during the training process. Again, our code is open-source on GitHub.[4]

IV. Results: Human Gameplay

For additional details, see Appendix II-A.

IN PROGRESS

A. Classifier Performance

Discuss Breakout, Space Invaders results here, put a table here with the tuned parameters?.

The results are shown in Table I.

The results are shown in Table II.

The distribution of actions is displayed in Figure 2.

[2]This is in contrast to the methodology from [12], where human experts had only two hours of training.

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

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

Fig. 2: TODO how about this, we try and plot the distribution of actions for the classifiers on the two games? Just have this be a very small figure but side by side ... so it still only takes up half a column.

B. Classifier Investigation

Now provide example images? Results are shown in Figure 3.

V. Results: Modified DQN

Figure 4

Figure 5.

For additional details, see Appendix II-B.

VI. Conclusions

In this work, we have made efforts to use Learning from Demonstration techniques to boost the performance of DQN agents on the problem of playing Atari games. We collected many hours of human expert gameplay data on Breakout and Space Invaders, and provided extensive evidence to show that a convolutional neural network can, to a large extent predict the action of the human expert given a sequence of game frames. A DQN agent then used this classifier as part of our human-guided DQN algorithm. While the DQN results do not improve using our algorithm, we believe there is still potential for algorithmic improvements in this work. In future work, we will first run the algorithm with a larger number of exploration steps to better see the effects of the DQN agent. Second, we will try the concept of *human experience replay* and combine experience replay from the agent's exploration with that of a human. Third, our more elaborate goal is to shift gears and work on training attention models [11], [21], the idea being that for these games, there are only a few important signals that matter for score (e.g., is the ball near

Fig. 3: This will represent examples of game frames (i.e. sequence of 4 game frames) for classifier, just like I did in the presentation. I'll want the full width (i.e. two columns) for this, with good and bad examples from each game. More details should be provided in a table with tuned values.

Fig. 4: This will be another full-page figure. Here I'll hope to have the plots comparing my results with DQN results, with both action-value and rewards. Use moving averages. I may need a second one of these.

Fig. 5: TODO hopefully this will be the SI with human but with longer exploration periods.

the paddle in Breakout *and* moving downwards?), and this can be trained into an attention model. We will explore these directions of work in the coming months.

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.

Appendix I
Preprocessing Details

We modify ALE so that a human player can play the games. This will create several directories inside (...) which is a folder that will have the `actions.txt`, `rewards.txt` (per frame, not cumulative), and a `screenshots` folder that contains *every* frame in the game, which is 60 per seconds. That's a lot of data, and it's in the raw RGB format as well, in `(210,160,3)` dimensions. Then I (personally) will play Pong or Breakout (one simple game to start) for, I guess, maybe 5-6 hours to gather a crapload of data. I can do the same for other games.

Next, with all this raw data, we have to process it somehow. I will have another script that can do all of this at once. My guess is to pass it a parameter k (say $k = 4$) so that it combines k consecutive frames together, thus reducing the number of files by a factor of k (but each file itself is larger by a factor of k so the overall size is the same).

Now we can get to our real contributions. The first one will be to develop a simple classifier from game frames to actions. There are several issues we have to get around regarding what frames to skip, though. Here is what I think I know:

- The ALE environment will process all frames (so 60fps really means 60 frames per second!) by default, so if I want to use all frames, I can do that!
- Frame skip? This means skipping frames at the ALE level, so with "game ticks" which are the *real*, single-frame case, we end up only considering $\{t_0, t_4, t_8, \dots\}$ to be used for φ. See John Schulman's description after this.
- Phi history? Recall that most say they use $\varphi(s)$ to be the "last four frames" so the input to the CNN is $84 \times 84 \times 4$ (I *assume* the actions are ignored.) Ah, yes! Look at[5] John Schulman's explanation: *You can stack the frames returned by the environment That's the same thing done in the other papers that use the Atari domain, such as deepmind's paper. I.e., they stack frames (t, t-4, t-8, t-12), not (t,t-1,t-2,t-3).* This really helps! So now I know that φ is the last m (where $m = 4$ usually) game frames, *when only considering the non-skipped frames*. Got it.[6]
- OpenAI environments automatically repeat actions for 2, 3, or 4 frames (at random). So instead of $\{t_0, t_4, t_8, \dots\}$, we could get $\{t_0, t_3, t_7, t_9, \dots\}$. I see! This introduces more stochasticity. Maybe this no longer needs the human starts condition?
- The original DQN paper (I think both NIPS and NATURE versions?) used frame skipping with max values attached to *consecutive* frames, and this time consecutive really means consecutive! It includes skipped frames. Ouch! In other words, the input to the neural network is, I believe:

```
[max(T-1, T), max(T-3, T-4), max(T-7, T
```

  So one of those above is a $\varphi(s_T)$ that is our state. Looks like no actions here. And also, these are the units of information that get stored in the experience replay ... along with actions and rewards, which I *guess* are the action and reward at the same frame as T, but not sure, seems like it would be better to have it a few frames *after* T.
- Details on the database for experience replay: these should be of the form (s, a, r, s'), but what exactly are the components?

What does this mean in practice for me? When I play the games, I get screenshots, actions and rewards *every* time step. So how do I gather the data?

- First, we develop a series of indices which we want to use, i.e. generate via numpy an array that looks like:

[5]https://github.com/openai/gym/issues/275

[6]All of this should be in a blog post someday.

```
[0,3,7,9,11,15,19,22,26,...]
```

and then only extract the data from those indices. That will automatically reduce the number of frames by a factor of 2-4 (the reason why I use 2,3,4 is because that's what OpenAI uses). Then subsample actions and rewards using those exact indices (should be OK, rewards and actions come right after those state indices). Then after this is done, downsample the remaining screenshots to 84x84 pixels. This *will* be game-dependent!

- After reading open-sourcer DQN code, it indeed looks like $\varphi(s_t)$ and $\varphi(s_{t+1})$ (i.e. s and s' in normal MDP notation) share (non-skipped) frames. That is, for the example above, I will need to make $\varphi(s_0) = (x_0, x_3, x_7, x_9)$ where x_i denotes the screenshots at *time-step* i, *including* skipped frames Then $\varphi(s_1) = (x_3, x_7, x_9, x_{11})$. The action a and reward r for this particular sample are chosen according to action a_9 and r_9 at time-step 9 according to the last one in $\varphi(s_0)$. All this will be repeated to form $(\varphi(s_t), a_t, r_t, \varphi(s_{t+1}))$.
- And *that* is the dataset which we can put in for the experience replay rule. I will save all of this in separate folders, but combining all the games together. This is following the reference which uses human experience replay.
- OK, but what about training a classifier from images to actions? For this, I believe we want to gather all the $(\varphi(s_t), a_t)$ and simply use that. Alternatively, I can try $(\varphi(s_t), a_{t+k})$ for some *small* value k (remember, there's frame skipping, and I feel like this won't make that much of a difference). Then I train on that to get a classifer. How to use it? During the exploration phase where we're following random actions with probability ϵ, simply follow what the classifier tells us to do. This is *different* from the human experience replay technique (but it can be combined with it!).

APPENDIX II
EXPERIMENT DETAILS

TODO

A. *Human Gameplay Classifier*

TODO

B. *Deep Q-Network With Human Data*

TODO