D-GAN: Autonomous Driving using Generative Adversarial Networks

Cameron Fabbri
Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
fabbr013@umn.edu

Jayant Sharma
Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
sharm546@umn.edu

Abstract

We propose a framework for learning a policy directly from data in the contex of behavioral cloning. We explore environments in which a reward function R is not known. Classically, Inverse Reinforcement Learning is used to extract R to then use with Reinforcement Learning to learn a policy π. We skip this step, and train an agent that matches the policy of the behavior given by a human such that the two are indistinguishable. A focus is put on the self-driving environment, however we note that this framework is general and can be applied to any simulation for which human experience is obtainable.

1 Introduction

The ability of autonomous agents to be able to play various types of games at or better than human performance has been a research interest for decades, possibly beginning with The Turk, an "autonomous" chess playing machine from the 18th century (later to be found a hoax). Games such as chess have a very large state space, making solving them very difficult or intractable. Modern video games have an even larger, possibly infinite state space, and definitely cannot be computed in a reasonable amount of time. Due to this, the community has turned to learning-based approaches. The goal then becomes finding the optimal method to train such an agent to learn how to play (and most importantly win) a game.

There are two obvious methods for training an agent to play a game. The first is a supervised training approach, where an agent is shown example state and action pairs provided by a human expert, and aims to generalize what action(s) should be taken in a given state, through a processes such as regression. This is commonly known as behavioral learning [19]. While effective this approach has several issues, the first being a need for an expert. For each game we want to train an agent to learn, an expert player must provide training data for. This is extremely tedious, and furthermore we have no way of knowing if even the best human player available has pushed the game to its limits, and is truly an "expert". The second method aligns closely to how humans learn to play games — without direct supervision. This field has become known as reinforcement learning [23], and revolves around learning a policy in order to maximize a cumulative reward. With this, we hope to extract a policy that exceeds human performance, without any human teaching or intervention. A brief review of reinforcement learning is given in Section 2.

Behavioral cloning has seen less attention in the reinforcement learning community, partially due to the fact that a main goal is to outperform a human, so cloning the play style of one may not necessarily achieve this. In our work, we put a focus on game environments in which there is no clear winner or loser, and where an end goal may not even be in mind. The goal is to perform behavioral cloning using Generative Adversarial Networks (GANs), specifically for the task of driving a car. In this setup, we assume a policy exhibited by a human player π_P and attempt to approximate that policy π_θ. In other words, we aim to train an agent that is able to drive a car such that it is indistinguishable

from a human controller. While there may be sub-policies that we may want to learn such as "stay on the road", or "don't hit pedestrians", that depends on the human behavior we are attempting to clone. We believe that this work may provide steps towards understanding human behavior and how certain properties may be reproduced.

Learning a generative model is natural for this task. A regression model will inherently be constrained towards the data points seen in the dataset, and there is no guarentee that the solution space will not be partially disjoint. A generative model on the other hand, puts no constraints on the solution space towards specific instances seen in the training data. Due to GANs learning a distribution, they are able to learn a much more diverse set of possible solutions, and provides a perfect framework for this task.

2 Background

This section provides a brief background on reinforcement learning (RL) and some related work. Readers are directed to *e.g.* [23, 25, 26, 20, 3] for a more comprehensive review and in depth details on RL. Section 2.3 provides a brief review of Generative Adversarial Networks, which we choose to use as our generative model.

Nomenclature

- $\mathcal{E}$ Environment
- a Action
- s State
- r Reward
- γ Discount Factor
- π Policy

2.1 Reinforcement Learning

An agent in some environment $\mathcal{E}$ following policy $\pi(a_t|s_t)$ interacts with the environment over time, taking action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$ in order to maximize some defined cumulative reward R. The result of taking action a_t is the transition $s_t \rightarrow s_{t+1}$, and the receiving of some immediate reward r_t. The goal of RL is to learn an optimal *policy* π^* by using these observed rewards in order to maximize the expected total reward. A policy takes as input a state and returns an action to perform, and the *optimal policy* is a policy which maximizes the expected return $R_t = \sum_{t=0}^{T} \gamma^t r_{t+1}$, where T is the length of an episode (when the game terminates). In RL, our goal is to find some optimal policy π^*. The following model-free reward-driven RL approaches describe the current state of the art in control.

Q-Learning Q-learning [25, 26] is a form of model-free RL with the goal of learning an action-utility function, commonly known as a Q-function, which returns the expected utility of taking action a in state s following policy π: $Q^\pi(s, a) = \mathbb{E}[R|s, a, \pi]$. In this framework, an agent chooses an action in a state, and observes the outcome of taking that action in terms of a reward. Following the Bellman Equation, we can learn Q^π in order to approximate Q^*:

$$Q^\pi(s, a) = \mathbb{E}_{s_{t+1}}[r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi(s_{t+1}))]$$

A Q-function can be represented by a neural network with weights θ_Q. In order to learn these weights, we can define a loss over the estimated Q values:

$$L = \mathbb{E}[||(r_t + \gamma Q^\pi(s_{t+1}, a_{t+1})) - Q(s, a)||_2]$$

Differentiating this loss with respect to θ_Q resolves in a gradient which can be computed and optimized via gradient descent. Usually this is done after every time-step as an agent interacts with an environment.

The work of [18] showed that by using a Convolutional Neural Network to approximate Q^* they were able to train an agent to play various Atari games using this method. It was shown, however, that this method would overestimate action values in certain scenarios. Towards improving this, the work of [24] introduced Double Q-learning, which decouples the selection and evaluation of actions.

While viable for discrete action spaces, it was not until the work of [13] before continuous action spaces could be considered.

Policy Gradient While Q-learning methods store a value function, the use of policy gradient methods instead search directly for the optimal policy π^*. In this framework, the policy is represented explicitly by its own function approximator, and is updated using gradient *ascent* with respect to policy weights in order to maximize the expected reward. The REINFORCE algorithm [28] showed a way to provide an estimation of the gradient, but usually some baseline $b(s_t)$ is subtracted to reduce the variance of the estimate. The final gradient direction is then

$$\nabla_\theta \, log \, \pi(a_t|s_t)(R_t - b(s_t))$$

The work of [23] showed that the gradient can be estimated by an approximate action-value or advantage function by using the agent's experience. This is proven to converge to a locally optimal policy.

Actor-Critic Combining policy search and value iteration methods yields the actor critic model [10]. In this framework, the policy acts as the actor, and the value function acts as the critic. The baseline mentioned earlier in policy gradient methods is computed using the value function. Actor critic methods have seen huge success in areas such as learning to play atari games. Using an asynchronous approach, [17] showed that they were able to train multiple agents simultaneously in order to update a "master model", and were able to outperform previous learning algorithms, all while training *faster* on a CPU. This was soon shown to work even faster when trained on a GPU [4].

2.2 Inverse Reinforcement Learning

Given an optimal policy π^*, Inverse Reinforcement Learning (IRL) aims to recover the reward function R. Instead of using rewards as a means of punishment in order to learn a policy, IRL observes a policy in order to learn what the end goal may be. A comprehensive review of IRL along with its connection to Generative Adversarial Networks can be found in [5]. We show in Section 2.4 how this relates to our method.

2.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [6] represent a class of generative models based on a game theory scenario in which a generator network G competes against an adversary, D. The goal is to train the generator network to generate samples that are indistinguishable from the true data $\mathbb{P}_r$ by mapping a random input variable $z \sim \mathbb{P}_z$ to some x. This mapping can be represented as $G(z; \theta_g)$, where G is a neural network with weights θ_g, and z is a random variable sampled from some distribution, *e.g.* $z \sim \mathcal{N}(0, 1)$. The discriminator, $D(x; \theta_d)$, is represented by a second neural network with weights θ_d, and outputs a scalar representing the probability that a sample x came from $\mathbb{P}_r$ rather than from G. The two networks are trained simultaneously, with D being trained to maximize the probability of correctly distinguishing whether or not a sample came from $\mathbb{P}_r$ or from G, and G being trained to fool D by minimizing $log\,(1 - D(G(z)))$. This can be represented by the following objective function:

$$\min_G \max_D \mathbb{E}_{x \sim \mathbb{P}_r}[log\, D(x)] + \mathbb{E}_{z \sim \mathbb{P}_z}[log\,(1 - D(G(z)))] \tag{1}$$

where $\mathbb{P}_z$ is a prior on the input noise (*e.g.*, $z \sim \mathcal{N}(0, 1)$) and $\mathbb{P}_r$ is the true dataset. It is shown in [6] that this amounts to minimizing the Jenson-Shannon divergence between two distributions.

Conditional GANs Conditional GANs (cGANs) [15] introduce a simple method to give varying amounts of control to the image generation process by some extra information y (e.g a class label). This is done by simply feeding y to the generator and discriminator networks, along with z and x, respectively. The new objective function then becomes:

$$\min_G \max_D \mathbb{E}_{x,y \sim \mathbb{P}_r}[log\, D(x, y)] + \mathbb{E}_{z \sim \mathbb{P}_z, y \sim \mathbb{P}_r}[log\,(1 - D(G(z, y)))] \tag{2}$$

cGANs have been shown to work well on image-to-image translation problems, where the generator is conditioned on an image [9, 12, 29]. In our approach, we condition on an image, but instead generate an action.

GAN Variations GANs have been the most popular and sucessful generative model for the past few years, and have been getting a lot of attention in the research community. Despite their theroretical findings, in practice they are extremely difficult to train. A major issue is that of the vanishing gradient, where the discriminator does not offer suitable gradients for the generator to train from. This was further explored by the work of [1], who proved that as the discriminator improves, the gradients given to the generator get worse. A least squares approach was done by [14], which adds a least squared penalty, pulling the generated data closer to the true distribution. Most impressive has been work done on using the Wasserstein distance as a metric, as first seen in [2]. The main idea behind the Wasserstein GAN (WGAN) is that the Wasserstein distance provides much stronger gradients to the generator than the Jenson Shannon does. This comes with a price, however, namely that the discriminator must be a K-Lipschitz function for some K. [2] does this by clipping the weights of the discriminator after each gradient update in order for the parameters to lie in a compact space. This turns out to greatly cripple the discriminator in terms of capacity and the speed at which it learns. The work of [7] improves the WGAN by instead penalizing the norm of the gradient with respect to its input. This follows the property that a differentiable function is 1-Lipschitz if and only if it has gradients with norm equal to or less than 1 everywhere [7]. Further improvements have been recently seen as well [27, 16]. To keep training times low, we stick with the original formulation, however consider these variations for future work.

2.4 Generative Adversarial Imitation Learning

Most similar to our approach is the work of [8] who directly extract a policy from data. The task of learning a policy from expert behavior *without* access to any reinforcement signal or interaction from the expert has several approaches. The use of IRL in order to extract a cost function to then use to learn a policy is one approach, but can be slow and has many steps. Instead, [8] shows they can learn a policy directly from expert data in a model-free way. In their adversarial framework, they aim to train a generator in order to generate an occupancy measure p_π such that it is indistinguishable from the true occupancy measure p_{π_E} exhibited by an expert. The work of [11] applies this directly to the self driving car approach. They use a combination of "LIDAR-like" beams and hand crafted features (speed, vehicle length, lane offset, etc.) as input to their network. Our approach is similar, but we are able to train on images.

Our method aims to extract a policy directly from data as seen in [8]. That policy may not be optimal for the game environment, but it is the policy we wish to approximate, as our end goal is to simply perform behavioral cloning by performing realistic actions in a simulation that are indistinguishable from a human player. In this setting, there is not a direct reward function for us to use. One may argue that in our setting (self-driving) a reward function may be to stay on the road, or follow the rules of the road, etc. However, rules of the road are rarely followed in video games, and we may in fact want to go off-roading due to exploration, following a path, escaping the police, etc. This relaxation makes the reward function very nonlinear, and thus very difficult to approximate. For this reason, we do not take the Inverse Reinforcement Learning approach, and instead aim to extract the policy directly from data.

3 D-GAN

In this section we present Driving-GAN (D-GAN), an adversarial network able to approximate a policy from data and generate realistic actions conditioned on a frame from a simulation.

Approach Our approach is based on the idea that in many situations in a video game there is more than one "correct" or realistic action to take. Consider playing GTAV for the first time (driving), and trying to get a feel from the controls. Upon coming to a fork in the road, should you go left, or right? Because there is no incentive to go one way or the other, the choice is entirely up to the player. It is important to not confuse this with the problem of exploration in reinforcement learning. Here, we aren't trying to cover every part of the map in our exploration, we are exploring just for the sake of

exploring. The goal of D-GAN is to approximate human behavior in the sense of playing a video game.

While it may be argued that a discriminative model would be a better choice for this task, we argue that this would in fact constrain the solution space to that displayed in the training data. Consider the scenario of always turning right at a certain intersection during the training data. Although in other intersections in the training data the human has turned left, there is a very high probability that during testing when the agent comes to that intersection, it will turn right as well. This is because by constraining the solution space, the model exhibits a behavior of turning right when seeing that intersection. However, in a generative approach, such as with GANs, the model does not constrain itself to situations such as these. Instead, the model learns that at any intersection, a left or right turn is realistic, and we would expect it to vary its choices.

We train a policy π_θ to exhibit human-like behavior by rewarding it for successfully fooling a classifier. The discriminator network in our GAN framework acts as this classifier, and penalizes state-action pairs which are unrealistic. More formally, given a set of state-action pairs sampled from π_E by recording a human player $\mathcal{X}_E = \{(s_1, a_1), (s_2, a_2), ..., (s_n, a_n)\}$ our objective becomes:

$$\min_G \max_D \mathbb{E}_{(s,a)\sim\mathcal{X}_E}[log\ D(s,a)] + \mathbb{E}_{s\sim\mathcal{X}_E, z\sim\mathbb{P}_z}[log\ (1 - D(s, G(s,z)))], \tag{3}$$

where $\mathbb{P}_z$ is a prior on the input noise (*e.g.,* $z \sim \mathcal{N}(0,1)$).

Architecture We use GANs as our generative model, as discussed in Section 2.3. Our generator network is flipped from the usual GAN architecture design because instead of inputing a vector and outputting an image, we are inputing an image and outputting a vector. Below show the architectures for our generator and discriminator. In the output size, the batch is omitted for clarity. Here, a_n is the number of actions and f_n is the number of input frames. For the discriminator, the action vector is padded and concatenated onto the image channel-wise.

Generator

Operation	Kernel	Stride	Filters	Batch Norm	Activation	Output Size
Concatenation						$256 \times 256 \times 3f_n$
Convolution	5×5	2	32	Yes	ReLU	$128 \times 128 \times 32$
Convolution	5×5	2	64	Yes	ReLU	$64 \times 64 \times 64$
Convolution	5×5	2	128	Yes	ReLU	$32 \times 32 \times 128$
Convolution	5×5	2	256	Yes	ReLU	$16 \times 16 \times 256$
Convolution	5×5	2	512	Yes	ReLU	$8 \times 8 \times 512$
Convolution	5×5	2	512	Yes	ReLU	$4 \times 4 \times 512$
Convolution	5×5	2	512	Yes	ReLU	$2 \times 2 \times 512$
Convolution	5×5	2	512	No	ReLU	$1 \times 1 \times 512$
Fully Connected					TanH	a_n

Discriminator

Operation	Kernel	Stride	Filters	Batch Norm	Activation	Output Size
Concatenation						$256 \times 256 \times (3f_n + a_n)$
Convolution	5×4	2	64	No	Leaky ReLU	$128 \times 128 \times 64$
Convolution	5×4	2	128	Yes	Leaky ReLU	$64 \times 64 \times 128$
Convolution	5×4	2	256	Yes	Leaky ReLU	$32 \times 32 \times 256$
Convolution	5×4	2	512	Yes	Leaky ReLU	$16 \times 16 \times 512$
Convolution	1×1	1	1	Yes	Sigmoid	$16 \times 16 \times 1$

Due to the time series nature of the data, future work will encompass using Long Short Term Memory (LSTM) networks. LSTM units are a special type of building piece for Recurrent Neural Networks (RNNs) that are able to remember values over arbitrary periods of time. Because of the very short

Figure 1: A very cherry-picked example of our agent choosing to turn right when approaching an intersection.

interval between frames, the network may decide on one frame to turn right, but then decide on the next frame to turn left, causing a very jittery and unnatural looking agent. One quick fix (hack) for this is to execute actions across multiple input frames during testing. [1]

4 Experiments

Environment We use the video game Grand Theft Auto V (GTAV) for our simulation environment. With realistic graphics, modifications (mods) able to change the weather, hundreds of different types of cars and motorcycles (even planes and bicycles), and a multitude of landscapes including desert, city, and rural, it is the perfect sandbox environment for teaching an agent. In order for there to be enough GPU memory for both running the game and also performing inference in our network, we set the graphics to the lowest settings (which are still pretty good). While just a simulation, the work of [22] showed that they were able to improve the realism of synthetic images by introducing an "enhancer" network. This work could be applied here in order to improve the already very realistic graphics for training on close to real-world data. We leave that for future work.

Figure 2: Two screenshots from GTAV showing very different environment, weather, vehicles, and lighting conditions. The networks are trained using this viewpoint as input.

Capturing Data We capture data by recording a human play the game, driving around the map with no end location or goal in mind. For each frame, we capture the keys pressed during that frame. Possible actions are: W, A, S, D, WA, WD, DS, DA, NO_KEY, leaving us with a one-hot action vector $y \in \mathbb{R}^9$ (WASD correspond to arrow keys on a keyboard with W=up, S=down, etc.). An in game day takes 48 minutes in real time, meaning that by having a person play the game for an hour we can obtain data across varying lighting conditions. Changing the weather to rainy or cloudy, as well as driving the many different types of cars available can expand our dataset even further [2]. We would like to note that the training data is not perfect, and off-road driving, collisions, wrong side of the road, etc. did occur. Future work will focus on training multiple types of drivers to compare, *i.e.,* one following the rules of the road, one off road driver, one slow, etc.

[1] Our code can be found here: https://github.com/cameronfabbri/Adversarial-Agent

[2] A full list of vehicles, planes, bikes, and boats that are available in-game (without mods) can be found here: http://grandtheftauto.net/gta5/vehicles

Training Details Training was done on a GTX 1080. We used a batch size of 16, $f_n = 1$ frame per action as input, and the vanilla GAN loss. The simulation was running in windowed mode at a resolution of 800×600. Frames were grabbed directly from the screen and resized to 256×256. One frame of Gaussian noise sampled from $\mathcal{N}(-1, 1)$ was concatenated onto the image channel, resulting in an input $s \in \mathbb{R}^{256 \times 256 \times 4}$. We trained for 50 epochs which took about 12 hours. The images and one-hot action vectors were normalized to a range of [-1,1].

Figure 3: A more accurate depiction of our agent in action. A full video of our agent driving can be seen here: `https://youtu.be/9lFmWjGhagU`

Results Obtaining quantitative results for generative models is quite difficult. For the same reason we do not have or know the reward function R, we do not have a direct way of evaluating our agent quantitatively. On the other hand, qualitatively we can observe its behavior in the environment and compare with our knowledge of the human player. We can say with certainty that there is still work to be done. As seen in the video[1], the agent acts very erratically, and when stuck in a corner or against a wall is unable to recover. However, in some instances the agent is able to avoid obstacles such as other vehicles, and make turns onto other streets. This shows promise in our generative approach for policy approximation.

Because our generative model is producing a distribution over driver actions, it should model the distribution seen in the training set. Part of our future work includes finding an efficient way to measure this, given that the training and testing data would be different. A very nice test would to record short clips of the training data, as well as short clips of the model playing, then perform a large scale user study to ask participants to classify them as either real or generated. Successful results from this would be a classification of 50% (meaning our model was indistinguishable from a human player). We leave this to future work.

5 Conclusion

This paper presented an adversarial approach towards approximating a human policy given by a human. Applied to the self driving car setting in simulation, the goal was not to be a perfect driver but to exhibit human like qualities based on the training data. We were driven towards using generative models for behavioral cloning by the analyzing the task we wished to accomplish: train an agent to drive a car in simulation such that it is indistinguishable from a human player. While Inverse Reinforcement Learning could be used for this task, it is a two step process: 1) Learn the reward function R, and 2) Use Reinforcement Learning to learn a policy π in order to maximize the expected reward from R. This two step process is cumbersome and has long processing times. Our one-step method aims to extract a policy π directly from observation. Our contribution was the extension of using Generative Adversarial Networks as a model to directly approximate a policy from image data.

Future Work There are a lot of ways in which this work can be extended. GTAV contains more than just cars: bicycles, airplanes, buses, etc., all of which can be controlled. Because our method only requires state-action pairs for training in the form of image-keystroke, we should be able to learn policies for those forms of transportation as well. Furthermore, this can be applied to other games and simulations.

In terms of our method, we plan to test out the other GAN formulations as discussed in Section 2.3. We briefly experimented with the Improved Wasserstein Method [7], but found no improvement and a much longer training time, so it was discarded for the time being.

Acknowledgments

We would like to acknowledge the work of [21], who provided the groundwork and implementation for the messy portion of this project, mainly being able grab frames from the screen using Python and sending key presses to the simulation. The code at [1] has acknowledged [21] in the files necessary.

References

[1] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[3] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[4] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. 2016.

[5] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[7] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[8] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[9] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

[10] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[11] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating driver behavior with generative adversarial networks. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 204–211. IEEE, 2017.

[12] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint*, 2016.

[13] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[14] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. *arXiv preprint ArXiv:1611.04076*, 2016.

[15] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[16] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[17] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[19] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[20] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[21] Harrison Sentdex. Using python programming to play grand theft auto 5, 2017.

[22] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. Learning from simulated and unsupervised images through adversarial training. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 3, page 6, 2017.

[23] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[24] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pages 2094–2100, 2016.

[25] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[26] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[27] Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang. Improving the improved training of wasserstein gans: A consistency term and its dual effect. *ICLR*.

[28] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.

[29] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*, 2017.