Adversarial Control of Neural Network Policies

Thomas C.H. Lux*
Virginia Tech
tchlux@vt.edu

Reid Bixler
Virginia Tech
reidbix@vt.edu

Colin Shea-Blymyer
Virginia Tech
c0lin@vt.edu

ABSTRACT

Neural Networks are beginning to control many safety critical systems with physical presence. However, evaluating the robustness of such physical systems is difficult, and requires significant time, space, resources, and a legal framework to allow for such tests. In this paper, we present a system to evaluate the robustness of a self-driving car within a simulated physical system. This approach shows the difficulty of adversarial control on such systems, as well as presenting an application off which future work may be built.

KEYWORDS

Neural Network Policy, Adversary, Image Processing

ACM Reference Format:
Thomas C.H. Lux, Reid Bixler, and Colin Shea-Blymyer. 2017. Adversarial Control of Neural Network Policies. In *Proceedings of Github*. ACM, New York, NY, USA, 6 pages.

1 INTRODUCTION

1.1 Problem

We aim to demonstrate that it is possible to make perceptual modifications to the inputs of real-time control algorithms which produce undesired behaviors, allowing for adversarial control. Deep neural network policies [13, 14] are being researched for their viability in increasingly complex reinforcement learning tasks [4, 5, 19, 27]. It is known that adversarial examples can be used to cause incorrect actions, however we pose the use of similar techniques to take total control of the policy decisions. This work would provide a framework in which an intruder could manually control a neural network policy by modifying its perception of the environment.

1.2 Motivation

Neural network policies are on the verge of controlling numerous critical applications across human society. Self-driving cars [28], autonomous helicopters [18], and even medical treatment [8] are all among the safety-critical domains where neural network research has made great strides. However, as with any safety-critical technology, the reliability of such learning algorithms is a major concern. Further, it has been shown that neural network policies are vulnerable to adversarial attacks [2]. Little work, however, has been performed on adversarial control. Such work may be able to pave the way towards developing a framework for protecting deep neural network architectures from such attacks in the future.

1.3 Related Work

Recent work has attempted to generate adversarial images for neural network policies [2]. The authors utilize well-known adversarial image generation techniques in order to identify minimal input perturbations that cause the neural network policy to change its decision for a particular frame. This work does not however consider creating adversarial images that are robust to image transformations such as scale, rotation, and perspective change. Another recent work [1] identifies adversarial images that are robust to modest transformations in scale, rotation, and perspective, but not on the order of magnitude that would be experienced by a self-driving car. The robust examples are most sensitive to changes in perspective, which (along with scale) is most common in a moving vehicle. The work presented here seeks to merge the attempts of [2] and [1] in order to try and generate robust adversarial images that are capable of attacking neural network policies.

Three of the most promising reinforcement learning (RL) algorithms studied to date are deep Q-networks (DQN) [17], trust region policy optimization (TRPO) [16], and advantage actor-critic (A2C) [24]. Previous works have assessed the vulnerability of deep neural network architectures to adversarial examples [12, 23]. Further recent research has addressed similar vulnerabilities in the reinforcement learning domain [2, 10]. Techniques such as the Fast Gradient Sign Method (FGSM) [9] are capable of quickly generating input perturbations that can cause a neural network policy to produce an undesired action. These perturbations are generated analytically and in most cases are imperceptible, or nearly imperceptible, to human observers. In FGSM, we are given raw input to the neural network x, a distribution over possible class labels y, network parameters Θ, and a loss function $J(\Theta, x, y)$. Then the best adversarial perturbation η, can be computed as:

$$\eta = \epsilon \ \text{sign}(\nabla_x \ J(\Theta, x, y))$$

Where ϵ is a bound on the magnitude of the perturbation. There are available online code repositories designated to the task of generating adversarial examples with this method among others [20, 21]. In order to generate adversarial perturbations for any neural networks, we require a well-defined loss function J. This loss function can be computed directly from knowledge of the neural network in a white-box attack, but in practice it is not realistic to assume full knowledge of the internal state of the target. In order to orchestrate a black-box attack, previous works have assumed a target uses one of a set of learning algorithms and trained a local model to be an approximation of the target [10]. After creating a local approximation, FGSM is used as it would be in a white-box attack. Even when different training algorithms are used by the target and the attacker, the adversarial perturbations often work. This may seem counterintuitive, but previous research has deemed this the transferability property and suggested that the cause is similar underlying feature models across different deep learning architectures [26]. In the context of our work, it suggests that the ability to generate effective adversarial perturbations for one neural network policy may provide a framework for deceiving any policy that might be in use by a target.

*Authors contributed equally.

By combining the ability to generate adversarial examples with a more specific criteria of control, this work will serve as an immediate extension to that done in [10] as well as [21]. This will further refine the ability of an adversary to maintain constant, deliberate control over the neural network policy subject to various constraints in the modifications made.

2 APPROACH

In order to orchestrate thorough adversarial control, we first must make sure the policies are accessible for study. We use an open source implementation of FGSM [20] against NVIDIA's driving architecture [3] as the neural network policy. Our target environment for control will be Udacity's Self-Driving Car Simulator that uses multiple cameras as sensors and standard turn controls expected in an automobile (explained in Section 1.5). Using our own trained policy, we execute adversarial attacks against the self-driving car in the simulator. Our adversarial attacks are white-box attacks that make no attempt to hide the adversarial perturbations. These attacks will be orchestrated via filters placed in front of the vehicle's cameras, as a billboard, sign, or sheet of paper may be used in real life. We hope to obstruct as few of the automobile's cameras as possible while also ensuring the attacks are minimally sensitive to scale, perspective, and rotation. In order to demonstrate control, we modify the environment of the self-driving car in order to make it deliberately steer off the road.

2.1 Adversarial Algorithms

One avenue of attack we investigated was the use of the L2 method developed by Carlini and Wagner [6]. This algorithm uses gradient descent to minimize the change in an image that changes that image's classification to the desired class based off of the L2 metric. This attack often finds adversarial examples that other attacks do not, and which work on defenses that are robust to other attacks. While these attacks generally have less distortion than others, this attack method is slower in comparison. Here, the classification of the neural network architecture we are investigating will classify an image into a turning angle. Further, the algorithm uses box constraints to ensure that the image it generates is valid for its application. Unfortunately, however, this algorithm relies on a white-box model with a softmax output layer. While we could conceivably transfer the model we were performing our attacks against to another with a softmax output layer (thereby avoiding the issues arising from the softmax requirement, and the white-box requirement), the work by Papernot, et al. suggests that this would be beyond the scope of our goals [22]. To transfer their models, and generate synthetic data, Papernot et al. had to query the base model upwards of 300 times in the simplest case. Further, they required direct access to the classifier's output. Neither of these requirements would be simple to fulfill in our context, as self driving car policies would be well-guarded and defended.

2.2 The Unity Simulator

Udacity's self-driving car simulator is a straightforward application of the Unity Game Engine. A majority of the code for running the simulator is game-engine specific and not in the scope of this research. Importantly, there exist 2 modes that the simulator can run in: 'Training' and 'Autonomous'. The Training mode can only be used after creating a self-driving model for a car (i.e. Udacity does not provide their own models). The Autonomous mode has 2 settings: 'Autonomous' and 'Manual'. The Autonomous mode will have the simulated car respond to driving inputs (steering angle) via a web-port connection to Unity. In autonomous mode the car will not move until it receives inputs via the port, or until the user goes into 'Manual' mode by pressing a steering key (WASD or arrow keys). The car will prioritize manual commands to those sent over the port, allowing users to make corrections during self-driving when necessary.

This simulator provides two tracks for training and testing a self-driving car. Users can create their own tracks, but doing so requires significant effort and an extensive knowledge of the Unity Game Engine. Our work focuses on the Lake Track (see source code) for training and testing the self-driving neural network policy. Extending the same performance results to other tracks is left to future work.

There are a large number of scripts, textures, models, and peripheral files provided within Udacity's Unity project. In particular, three scripts are important for this research: *CommandServer.cs*, *UISystem.cs*, and *CarController.cs*. The *CommandServer.cs* C-Sharp script is the primary interface via which all Machine Learning projects must interface. Unity opens up web-sockets listening for specific commands such as *steer* or *manual*. A *steer* command is expected to provide a steering angle and throttle (a.k.a. acceleration) which Unity then triggers in the self-driving car. In return, Unity communicates the car's current viewpoint to any listening machine learning models. The *UISystem.cs* updates the heads up display (HUD) for the user in the simulator. *CarController.cs* contains all of the code related to how the car moves in the simulator. This includes identifying the appropriate revolutions per minute (RPM) of the engine, the gear (RPM:speed ratio), torque on the wheels, gravity force applied while driving, and other specifics related to accurately simulating the movement of a car.

2.3 Generation of Adversarial Images via Blackbox Optimization

All current adversarial attack implementations utilize gradient-based optimization schemes in order to identify adversarial images that can trick the neural network policy. However, these techniques are not able to generate adversarial images if the underlying policy is structured with a noisy or undefined gradient. A more robust, yet expensive, mechanism for generating adversarial examples is the the use of gradient-free optimization techniques. This work attempts to identify adversarial images that are robust to transformations by applying multiple random transformations to any given adversarial image when evaluating adversary performance. The average performance of these randomly transformed images generates a stochastically noisy gradient with respect to the pixel intensity changes in the adversarial image, which cannot be optimized by previous gradient-based techniques.

This paper utilizes two gradient-free (black box) optimization techniques to attempt generation of adversarial images. Adaptive Memory Programming for Global Optimization (AMPGO) [7] is an optimization technique that does not require a gradient. AMPGO

augments an arbitrary local minimization algorithm with tunneling. Tunneling avoids previously identified local-minima in order to perform global optimization in a multi-minimum search space. Internally, the implementation of AMPGO used here does local minimization with L-BFGS-B [29] and a discrete approximation of the local gradient [11]. Adaptive Normal minimization, or modified simulated annealing [15], uses only random perturbations along single dimensions to search for optimal solutions. This algorithm never approximates a gradient, allowing it to perform unbiased searches through a domain. We suspect that the performance of AMPGO is considerably damaged by the noisiness of the gradient with respect to changing pixels in the adversarial image.

In summary, the structure of our adversarial attack generation is as follows for each desired attack (right and left turns):

(1) Generate an initial random image
(2) Randomly transform the adversarial image 100 times with a neutral colored background, compute the average turning angle produced by the self-driving car
(3) Repeat Step 2, modifying the adversarial image according to the selected derivative-free optimization algorithm to identify the largest magnitude turn that can be produced

3 DATA

Udacity's Self-Driving Car Simulator [5] allows us to train a self-driving model for experimentation with adversarial attacks. It would be preferable to use a pretrained model for Udacity's simulator, available on GitHub [25]. A pretrained model can eliminate the need for training time and also have a consistent baseline to attack. However, the available pretrained model does not successfully drive around the track without making errors. This is solved by running the simulator in 'Training Mode' and manually driving in order to gather more data for the model. The model was retrained on the newly collected training data consisting of a series of images and steering angles produced by a human driver. For reproducibility, we provide the source training data and the more performant final model. This model is based on NVIDIA's End-to-End Deep Learning network architecture [3], consisting of 9 layers (1 normalization, 5 convolutional, and 3 dense), but with some slight modifications (1 lambda layer, 1 dropout layer, ELU for activation function). This model can drive the course without error, which provides a foundation to test an adversary's ability to attack and control the model by modifying the camera's input in the simulator.

3.1 Modifications to Udacity's Simulator

We made a number of modifications to not only train our model, but also in order to simplify the adversarial interface with the simulator. The first and most important modification was to create 2 entirely new and separate scenes which were carbon copies of the 2 tracks that Udacity provided which would then become the 'Adversarial' tracks. We added the ability to choose between having the Adversarial or non-Adversarial track loaded via the Menu Screen with a simple little checkbox which would switch scenes dependent on if the user wanted the Adversarial track or not. Specifically, the Adversarial tracks were no different than the Autonomous tracks except for overlaying an image in front of the car's viewpoint. We confirmed that modifying this image in real-time (i.e. updating the image entirely) was possible in this context and would definitely affect how the car drove. For training purposes, we modified the *CommandServer.cs* script to essentially run the given machine learning model on both the perturbed image and the unperturbed image (this was done by modifying the alpha values of the image from 0 to 1) and then pass the expected inputs (steering angle and throttle) along with the current frame number to our adversarial trainer.

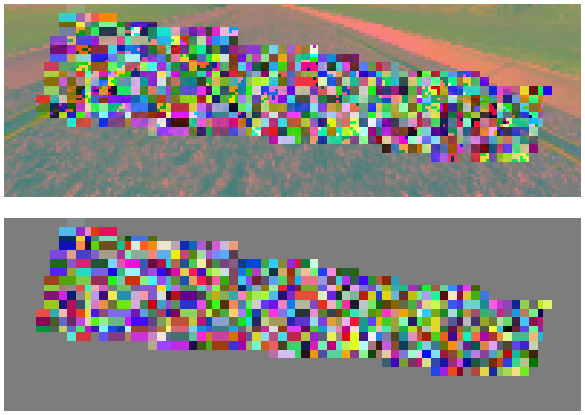

Figure 1: A demonstration of how the difference between attempt 1 (top), where images contain background information, and attempt 2 (bottom) with the background information removed.

The other modifications that were made to this project were within the *UISystem.cs* script. These were fairly simple modifications in order to reset the car to the original starting position and also to pass along the car's current map-coordinates via a text file. The car's position could be reset via 2 methods: adding content to a text file or by pressing the 'Enter' key. We wanted to be able to reset the car's position in the case of our adversarial training ever going wrong in order to have an easy restart rather than having to reload the whole simulator again. The car position was passed to a text file for an attempted adversarial attack on a reinforcement learning model.

3.2 Training the Neural Network

The original open-source self-driving model was not able to successfully drive the car uninterrupted around the selected test track. In order to generate a better driving model, we utilized the 'Training' mode provided by the simulator. The minimum amount of training data that produced a performant model was:

- 9 laps (forward direction)
- 1 lap (backward direction)
- 6 passes through starting turn (forward direction)

The single pass backwards around the track was necessary to make the model understand right turns, as the track consists almost entirely of left turns. The extra passes through the starting turn were necessary because some latent visual features in that region often caused the model to incorrectly steer off the right side of the track.

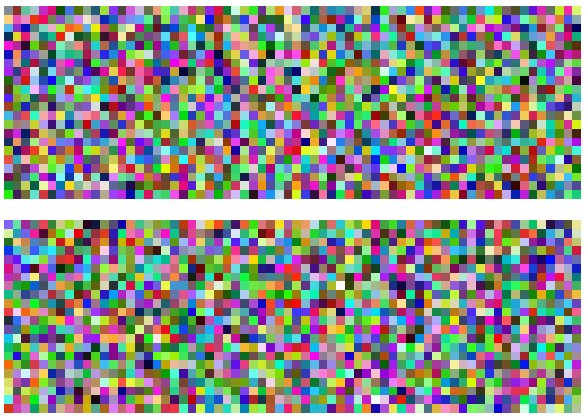

Figure 2: The images that produced the most robust left (top) and right (bottom) turns when transformed randomly and placed into the view of the neural network policy. Note that the seed for each turn was a neutral image. The images look identical, but produce noticably different distributions of turns when randomly transformed into the view of the neural network policy.

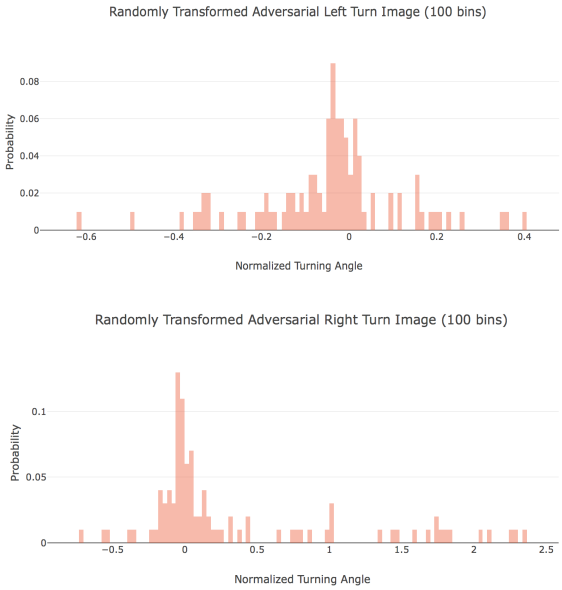

Figure 3: The 100-bin probability mass function (PMF) of the different turning angles produced by transformed adversarial left-turn (top) and right-turn (bottom) images. Notice that the mode does not shift so much as the outliers of turns produced.

The modified simulator, final trained model, as well as all the training images and associated steering angles are available online at *https://github.com/tchlux/Adversarial_Control*.

4 RESULTS

Gradient-free optimization algorithms require significantly more computations in order to converge on adversarial examples. This is expected, because in the most naive case (L-BFGS-B in AMPGO) an approximation to the gradient is generated using 2D computations, where D is the dimension of the input vector. The adversarial vectors (images) generated in this work are $22 \times 66 \times 3$ and are randomly transformed 100 different ways, making for 871, 200 policy evaluations at every gradient approximation (where gradient-based techniques would require only 100). This $\times 9000$ increase in expense is likely the cause of poor performance by AMPGO. No results are presented for AMPGO, because it never achieved more optimal solutions that the initial solution given ten hours of compute time on an 8-core 4.0 GHz AMD RX8 Processor at near full usage. Adaptive Normal however was capable of generating moderately improved adversarial examples that were robust to 100 random image transformations involving [1,3] times scaling, [-20,20] degrees of rotation, and [-70,70] degrees perspective change. The respective reasons for these ranges are: roadside signage could potentially scale to fill nearly all of a camera's input (the selected adversarial image size is 1/3 of a all camera input), the maximum degree hill on a road in the U.S. is 20 degrees (allowing signage to be rotated by that amount), and passing signs remain clear in camera images nearly until the sign is not visible at 90 degrees.

The first attempt, seen as the top image in Figure 1, was not successful in generating adversarial examples. The increased noise provided by variation in background images made the black box optimization algorithms unable to find any solutions that produced consistent turns in either direction under numerous random transformations. In order to simplify the problem, we reduce the background image to a neutral color (see bottom image in Figure 1). Once reducing the problem as such, the Random Normal optimization algorithm identified adversarial examples that produced turns in the left and right directions semi-consistently. Figure 2 displays the images generated for producing left and right turns respectively. Note that these images look largely like random noise because the optimization algorithms were not able to identify any modifications to the initial (random seeded) image that improved the distribution of turns produced. As mentioned in the discussion, the noisy resulting images are likely a side effect of insufficient compute time.

In Figure 3, the range of turns produced by different transformations of the left and right turn adversarial images from Figure 2 can be seen. The distributions demonstrate that under random transformations of the magnitudes chosen, the mode of turning angles produced will have a mode of 0, even for the best adversarial images. When the noisy backgrounds are included, the random process is so noisy that the distribution mean never shifts from 0. The outliers of the distribution are largely where the turning effect is noticed. These outliers occur when the selected random transformation happens to be one that takes up a majority of camera-input, also when making the vehicle turn is easiest.

4.1 Discussion

This research presents an attempt to use targeted adversarial attacks on the neural network policy of a self-driving car in order to orchestrate adversarial control. This work demonstrates that

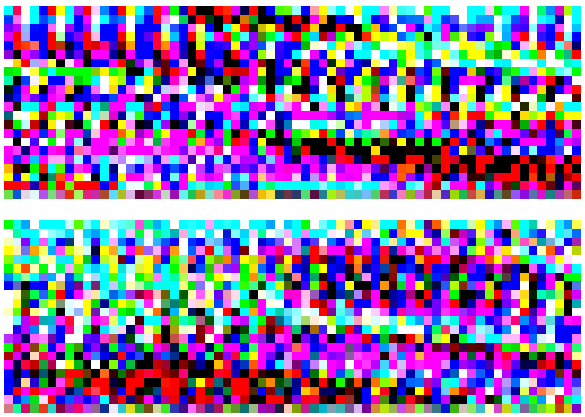

Figure 4: Two images that maximize the turning angle produced to the left (top) and the right (bottom). These untransformed images are so effective, they cause the neural network policy to produce turning angles outside the range of allowable values [-1,1].

without proper defenses, it may be possible for adversaries to take total control of a self-driving car via some mixture of signage and ulterior perceptual modifications. In that regard, this work demonstrates the difficulties associated with generating robust adversarial images in a noisy perceptual environment as well as using gradient-free techniques to do such. It can be inferred from the noisiness of the adversarial images produced that the optimization techniques were not allowed enough compute time to converge on the best adversarial images. An important research direction is adapting existing attack techniques to work on randomly transformed images by identifying new mechanisms for estimating the gradient as well as side-stepping the need for a softmax layer. Such work would make libraries similar to CleverHans considerably more effective for the task of generating robust adversarial images.

Convergence aside, the gradient-free optimization techniques were able to generate images that produced marginally different distributions of turning angles under a series of random transformations. These results are promising and suggest that it is possible to generate even more powerful adversarial images given enough compute time and the appropriate optimization strategy. In order to improve convergence, it may be better to incrementally increase the possible space of scale, rotation, and perspective changes allowed during transformation. A gradual increase in variety of transformation may result in faster convergence and an initially smaller search space. To demonstrate that this process may in fact produce different, more optimal results, Figure 4 shows what the optimal untransformed left and right turn images look like. Notably, these images generate turns that are outside the range of normalized turning angles that the neural network policy was trained on (and what the simulator allows), [-1,1]. The left turn image produces a turning angle of -2.5 and the right turn image produces a turning angle of 3.5.

Other important factors that would likely be important for generating truly robust adversarial images are real-world reproducibility and subtlety. The highly noisy images generated by the optimization algorithms in this work would likely prove difficult to print and place into the real world accurately. Not only that, but they would be easily identifiable by a human observer as out-of-place. It is arguable that these could be cast as 'artwork', but riders in self-driving cars would quickly realize otherwise. Traffic authorities could swiftly respond to such blatant advances on self-driving car technology. Future works should consider the 'uniqueness' of the robust adversarial images by using some techniques similar to generative adversarial networks, where it is also necessary that an adversarial example not be recognizable as different from standard traffic signage.

5 CONCLUSION

In this paper, we have demonstrated the utility and limitations of designing attacks against neural network policies in systems with physical presence using simulations. Difficulty arises in producing images that are adversarial across multiple scales, rotations, and perspectives. Further, as each image - stationary in the world - is viewed with moving scenery behind it, this makes optimization across all possible backgrounds a very challenging task. However, this suggests that systems with physical presence and the capability to move in relation to an adversarial example may be more robust to attacks. We hope that this use of simulated worlds to test the robustness of systems with physical presence will advance, allowing for a strong evaluation between the training stages, and roll-out to the real world.

5.1 Future Work

Our work on simulated attacks has left us with multiple avenues for future work. Many of these concepts were seminal to our work, but were left aside due to their inherent difficulty or computational complexity. One goal we have in mind is to move past a static modification to a system's perception and towards modification of the world itself. In the work we have presented here, this might manifest as an adversarial street sign, or billboard. Further, we would be interested in optimizing constrained ranges of skew, rotation, and scale for images. We believe this may result in more generally applicable adversarial examples, while remaining computationally tractable. In particular, road signs and billboards travel a specific path of transformations, not randomly generated ones. Using a specific set of transformations would likely improve convergence as well as the strength of turns produced by final adversarial images. Another area of future research includes other paradigms of deep learning. One paradigm of particular interest to us is deep reinforcement learning, as little adversarial research on such physical-presence systems has been performed. Finally, as we use a well realized simulated world, we are interested in investigating the challenges that additional perceptual input generates as well as the increased level of noise in real-world environments.

REFERENCES

[1] Anish Athalye and Ilya Sutskever. 2017. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397* (2017).
[2] Vahid Behzadan and Arslan Munir. 2017. Vulnerability of Deep Reinforcement Learning to Policy Induction Attacks. *arXiv preprint arXiv:1701.04143* (2017).

[3] Mariusz Bojarski, Ben Firner, Beat Flepp, Larry Jackel, Urs Muller, and Karol Zieba. 2016. End-to-End Deep Learning for Self-Driving Cars. (17 Aug. 2016). https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/

[4] Greg Brockman, @jietang, and Jonas et al. Schneider. 2017. OpenAI Gym. (16 June 2017). https://github.com/openai/gym

[5] Aaron (Udacity) Brown. 2017. Udacity's Self-Driving Car Simulator GitHub. (7 Feb. 2017). https://github.com/udacity/self-driving-car-sim

[6] Nicholas Carlini and David Wagner. 2016. Towards Evaluating the Robustness of Neural Networks. (2016).

[7] Abraham Duarte, Rafael Martí, and Fred Glover. 2007. Adaptive memory programming for global optimization. *Valencia, Spain: University of Valencia* (2007).

[8] Pablo Escandell-Montero, Milena Chermisi, José M Martínez-Martínez, Juan Gómez-Sanchis, Carlo Barbieri, Emilio Soria-Olivas, Flavio Mari, Joan Vila-Francés, Andrea Stopper, Emanuele Gatti, et al. 2014. Optimization of anemia treatment in hemodialysis patients via reinforcement learning. *Artificial intelligence in medicine* 62, 1 (2014), 47–60.

[9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[10] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284* (2017).

[11] Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2001–. SciPy: Open source scientific tools for Python. (2001–). http://www.scipy.org/ [Online; accessed <today>].

[12] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).

[13] Sergey Levine and Pieter Abbeel. 2014. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*. 1071–1079.

[14] Sergey Levine and Vladlen Koltun. 2014. Learning complex neural network policies with trajectory optimization. In *International Conference on Machine Learning*. 829–837.

[15] Thomas Lux. 2016. Convergence Rate Evaluation of Derivative-Free Optimization Techniques. In *International Workshop on Machine Learning, Optimization and Big Data*. Springer, 246–256.

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

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

[18] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. 2006. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*. Springer, 363–372.

[19] OpenAI. 2017. Dota 2 Adversarial Machine Learning. (11 Aug. 2017). https://blog.openai.com/dota-2/

[20] Nicolas Papernot, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Fartash Faghri, Alexander Matyasko, Karen Hambardzumyan, Yi-Lin Juang, Alexey Kurakin, Ryan Sheatsley, Abhibhav Garg, and Yen-Chen Lin. 2017. cleverhans v2.0.0: an adversarial machine learning library. (2017). https://github.com/tensorflow/cleverhans

[21] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, and Patrick McDaniel. 2016. cleverhans v1. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768* (2016).

[22] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. B. Celik, and Ananthram Swami. 2016. Practical Black-Box Attacks against Machine Learning. (2016).

[23] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 506–519.

[24] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 1889–1897.

[25] Naoki Shibuya. 2017. Car Behavioral Cloning GitHub. (12 May 2017). https://github.com/naokishibuya/car-behavioral-cloning

[26] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[27] Oriol Vinyals. 2016. DeepMind and Blizzard to release StarCraft II as an AI research environment. (4 Nov. 2016). https://deepmind.com/blog/deepmind-and-blizzard-release-starcraft-ii-ai-research-environment/

[28] Yurong You, Xinlei Pan, Ziyan Wang, and Cewu Lu. 2017. Virtual to Real Reinforcement Learning for Autonomous Driving. *arXiv preprint arXiv:1704.03952* (2017).

[29] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)* 23, 4 (1997), 550–560.