ADVERSARIAL GYM - A REINFORCEMENT-LEARNING FRAMEWORK FOR THE DESIGN AND EVALUATION OF ADVERSARIAL ATTACKS

A PREPRINT

Gilad Ben Or
ID: 036256527
benorgi@post.bgu.ac.il

January 4, 2024

ABSTRACT

Adversarial examples are maliciously manipulated examples that fool machine learning models. Decision-based (D.B.) attacks are a form of black box attacks that rely solely on the final decision of the model. In this paper, I am presenting a framework based on GYM environment that enables researchers to plan and test their D.B. attacks. This framework, together with CleverHans environment, may improve research cycles and open the mind to new algorithms, in particular reinforcement learning algorithm. In addition, I propose a novel deterministic algorithm and a reinforcement learning algorithm to fool MNIST classifier and test them both using this framework.
Availability and implementation: Adversarial Gym is freely available at `https://github.com/gbenor/Adversarial_RL`

***K*eywords** reinforcement-learning · adversarial · gym · boundary

1 Introduction

Many high-performance machine learning algorithms are susceptible to minimal changes in their inputs. Adversarial perturbations drawn attention from two directions. On the one hand, we are concerned about the integrity and security of machine learning algorithms deployed, such as autonomous cars or face-recognition systems. On the other hand, it is a exciting research opportunity to understand the difference between human and computer sensory information processing.

Decision-based attacks are a category of black box attacks which depend solely on the model's final decision. In terms of the structure of the inputs and outputs and certain examples classified by the model, the attacker has little knowledge regarding the the model to be attacked.

Previous works suggested algorithms for finding adversarial examples with minimal perturbation size and minimal calls of the model to be attacked. [Brendel et al., 2017] introduced a boundary attack that starts from a large adversarial perturbation and then seeks to reduce the perturbation while staying adversarial. [Chen et al., 2019] proposed a novel unbiased estimate of gradient direction at the decision boundary based solely on access to model decisions. Their algorithm requires significantly fewer model queries than several state-of-the-art decision-based adversarial attacks and achieves competitive performance in attacking several widely-used defense mechanisms.

In order to design and evaluate more attack algorithms, there is a real need for an environment that allows data interaction and results measurement. An existing, well known and powerful environment is CleverHans. CleverHans [Papernot et al., 2018] is a software library that provides standardized reference implementations of adversarial example construction techniques and adversarial training. The library may be used to develop more robust machine learning models and to provide standardized benchmarks of models' performance in the adversarial setting.

I looked at the same problems that CleverHans encountered and sought to address it in a different way. I have observed that the requirements of the testing environment are similar and may be reduced to those of the reinforcement learning environment. Reinforcement learning environment offers the opportunity to observe data , perform a single or sequence of actions and receive the corresponding score. Here, the attack algorithms are equivalent to the reinforcement learning policies.

The use of such an environment will improve the implementation of new algorithms and will be a standard tool for testing them. Thus, researchers may be able to work with state-of-the-art algorithms and methodologies, reinforcement learning in particular.

A few works used reinforcement learning tools in order to generate adversarial attacks. [Hyrum, 2017] implemented a reinforcement learning agent that was equipped with a set of functionality-preserving operations that may perform on PE file, and learnt through a series of games played against the anti-malware engine, which sequence of operations is most likely to result in evasion for a given malware sample

In this work, I developed a decision-based attack environment based on the OpenAI Reinforcement Learning Gym [Brockman et al., 2016] interface. The system fits for the MNIST dataset [LECUN,] and can quickly be expanded to support other datasets. I have researched and added support for custom actions, implemented two policies and trained a reinforcement learning agent to discover the optimal policy.

The contributions of this work are as follows:

- Introduce a research framework for decision-based attacks.
- Suggests custom actions for attack algorithms.
- Design, implementation and evaluation of two boundary attack algorithms.
- Training and evaluation of reinforcement learning agent in this environment.

2 Research Question

2.1 Actions

Actions are the methods of the Agent that allow it to interact and change the environment, and thus to be transferred between states. I would like to propose a set of actions that are good enough to compare the state-of-the-art approaches for blackbox attacks.

2.2 Reduced model calls

Blackbox attacks call models to determine their performance and to assess the next step. I would like to explore whether the proposed approach reduces the number of calls for each example.

2.3 Reinforcement learning vs deterministic algorithm

I wonder if the reinforcement learning agent manages, with the same set of actions, to overcome the performance of a deterministic algorithm.

3 Hypothesis

I hypothesize that I would be able to identify a series of actions that produce reasonable performance in terms of perturbation size and amount of API calls.
I believe that the reinforcement learning agent manages to overcome the performance of a deterministic algorithm because it would distill more information for the observation from the environment. However, its running time may be longer in some orders of magnitude.

4 Material and methods

4.1 Data

I used the MNIST [LECUN,] dataset for this work. The dataset was downloaded using Keras API without any additional prepossessing. The MNIST dataset has a training set of 60,000 examples, and a test set of 10,000 examples.

It is a subset of a larger set available from NIST. The original black and white images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

4.2 The model to be attacked

The model I would like to attack is the convolution neural network used during the course labs. It achieved accuracy performance of 0.975.

4.3 Evaluation environment

I chose OpenAI Gym [Brockman et al., 2016] as my evaluation environment. Gym is a toolkit for developing and comparing reinforcement learning algorithms. It makes no assumptions about the structure of the tested agent, and is compatible with any numerical computation library, such as TensorFlow or Theano. I selected this environment as it enables various algorithms to be fitted and tested. In specific, it is appropriate for both the deterministic method and the reinforcement learning algorithm.
In order to register a new environment, I implemented the following 4 methods: constructor, reset, render and step.

4.3.1 Constructor

In the environment's constructor I loaded the dataset, initiated the MNIST classifier (the model to be attacked) and checked that it performs well.

4.3.2 Reset

This method resets the environment to initial state and return first observation (see below). Here, it does the following things:

1. Choose a benign example (from the training or the testing sets, depending on the environment mode).
2. Generate two centers (see below).
3. Reset initial variables such as the initial step size, max perturbation and etc.

4.3.3 Step

This method commits an action and return a new observation, a reward, the is done signal and additional info (all explained below).

4.3.4 Observation

The observation is an environment-specific object representing the agent's observation of the environment. The observation consists of the following objects:

1. **current image** - the image of the current step.
2. **classifier prediction** - the classifier's prediction for the current image.
3. **perturbation norm** - the perturbation norm of the current image with respect to the source image.
4. **steps** - the index of the current step.

4.3.5 Action space

The following actions are available for the agent:

1. **Decrease step** - Decrease the step size according to the equation below.:
   step= step size > 0.4 : step size/2
   o.w.: max(0.05, step size- 0.05)
2. **Increase step** - Increase the step size according to the equation below.:
   step= step size > 0.4 : step size/2
   o.w.: max(0.05, step size - 0.05)

3. **Closet cluster** - Go one step in the direction of the nearest target cluster.

4. **Farthest cluster** - Go one step in the direction of the most distant target cluster.

5. **Original image** - Go one step in the direction of the original sample.

6. **New centers** - Generate new centers. In offline, the environment randomly generates 300 samples with the target label and select a pair of samples with the largest distance between them. The output of this process is 2 target samples which called centers. The agent calculates the distance to each center at each step. The closest target center called "closest cluster" and farthest one called "farthest center".

4.3.6 Reward

The reward has to take into account two factors:

1. **The classification of the adversarial example** - whether the classifier labels the examples as the target class.
   1: x$\in targetclass$
   $0 : x \notin targetclass$

2. **Perturbation size** - the agent gets negative reward if the perturbation is bigger than maximum perturbation P and positive reward if it under this limit.

A suggestion for the reward function is shown in (Equation 1):

$$reward = \alpha C + \beta \left(P - d\left(o, t \right) \right) \tag{1}$$

4.3.7 Is done signal

The is done signal stops the agent while it does the exploration-exploitation tasks. We will stop the agent if it performs more than L steps, or it reached the target class under the perturbation size constraint.

4.4 Deterministic agents

In this section I describe two deterministic agents that were evaluated in this work. The agents' policy was implemented using a graph. The agent moves from initial state and takes an action based on the current state and observation.

4.4.1 The triangle agent

The triangle agent used a policy as described in Algorithm 1 and Figures 1, 2. It used one source point and two target points to explore the boundary and then to move in a triangle track from point to point in order to reduce the perturbation. In each cycle, the agent decreases the step size.

Algorithm 1: Triangle agent policy

Result: Adversarial image
initialization;
while *perturbation* $\geq PERTURBATIONSIZE, steps \leq MAXSTEPS$ **do**
    go in the direction of the closet center until labeled as target;
    decrease step size;
    go in the direction of the source sample until labeled as source;
    (option) decrease step size;
    go in the direction of the farthest center until labeled as target;
    (option) decrease step size;
    go in the direction of the source sample until labeled as source;
    (option) decrease step size;
end

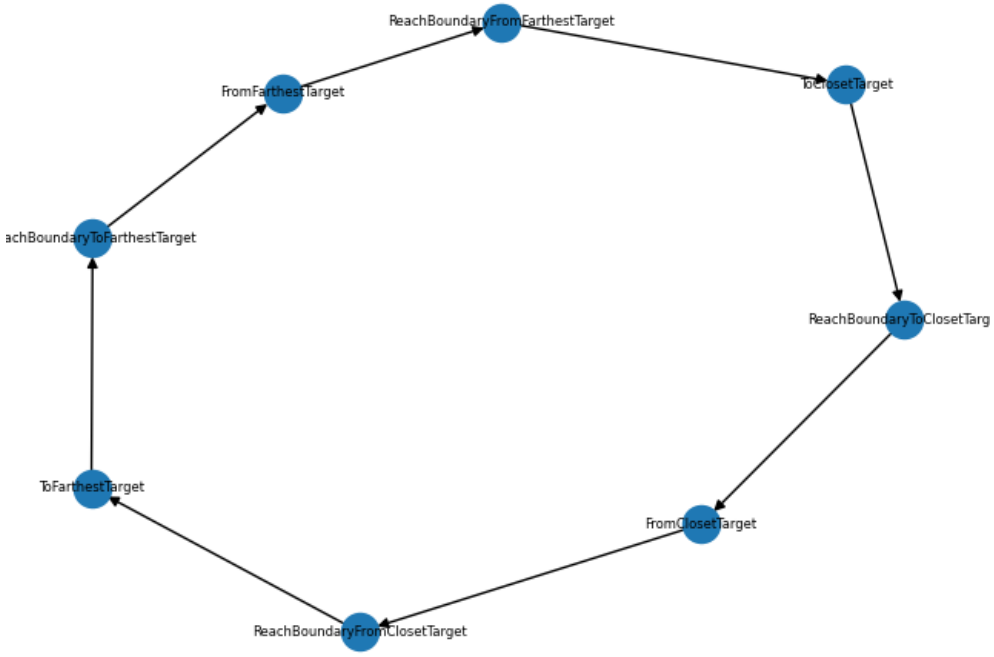

Figure 1: Graph diagram for algorithm 1

4.4.2 The alternating triangle agent

The alternating triangle agent is an enhancement of the triangle agent. It executes the triangle agent for K steps, takes the best observation and used it as the initial step for the next agent. It may keep the current step size or increase it back to the initial step size (algorithm 2).

Algorithm 2: Alternating triangle agent policy

Result: Adversarial image
initialization;
for *n in N* **do**
    Initiate thee agent (random state=k)
    Run the triangle agent for K steps.
    Pick the sample with the lowest perturbation size.
    (option) Increase the step size.
end

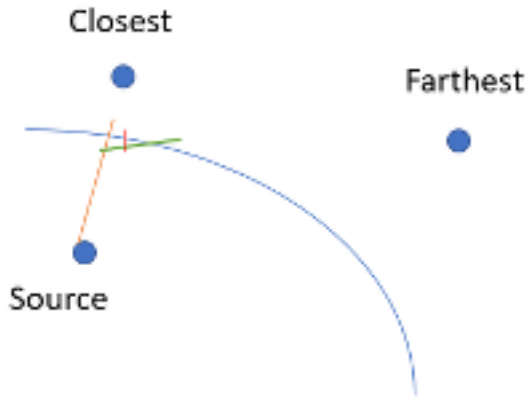

Figure 2: Illustration of algorithm 1. The orange line is the first sequence of steps in the direction of the closest center, the green line is the second sequence of steps in the direction of the farthest center and the red line is the third sequence of steps in the direction of the closest center again.

4.5 Reinforcement learning model

The purpose of reinforcement learning model is to teach an agent to take actions in an environment in order to maximize the cumulative reward. In this work, the agent learns how to take actions in order to generate an adversarial example for a target class while maximize the rewards which is relative to its success in generating an adversarial example with minimal perturbation size.

Figure 3 shows a widely used reinforcement learning framework. The agent accesses the environment, triggers it with an action, collects the reward and observes the new state of the environment.

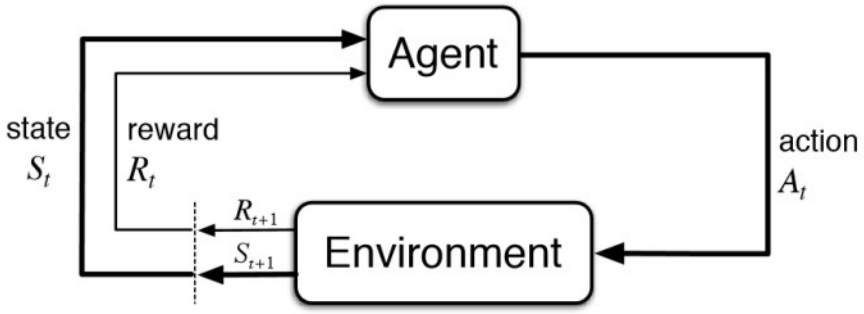

Figure 3: Reinforcement learning framework.

4.5.1 DQN agent

I used a DQN agent. The deep Q-network (DQN) algorithm is a model-free, online, off-policy reinforcement learning method. A DQN agent is a value-based reinforcement learning agent that trains a critic to estimate the return or future rewards. DQN is a variant of Q-learning. I choose the DQN agent because I have a positive experience with it, and because it has gained human-level power in several of the Atari games that are close to our task.

4.5.2 Keras-RL

I used the DQN implementation of Keras-RL. Keras-RL [Plappert, 2016] implements some state-of-the art deep reinforcement learning algorithms in Python and seamlessly integrates with the deep learning library Keras. Furthermore, keras-rl works with OpenAI Gym out of the box.

4.6 Metrics

I used 3 metrics to measure the performance of the algorithm:

1. Perturbation size - the perturbation size of the adversarial example.
2. Number of steps - the number of steps the algorithm performs. This measurement is highly correlated with the API calls requested by the algorithm.
3. Fitting effort - the effort needed to fit the algorithm into a new task.

4.7 Hyper-parameters

Table 1 describes the hyper-parameters which used to control the environment and the agents' learning process. Because the learning task is an intensive computing task, I used a single set of parameters and did not perform hyper-parameter optimization.

Table 1: Environment and agents hyper-parameters

Hyper parameter	**Value**	**Description**
MAX_STEPS	2000	Maximum number of steps for each episode
INITIAL_STEP_SIZE	2	The initial step size
MIN_STEP_SIZE	0.05	The minimal step size
MAX_STEP_SIZE	2	the maximal step size
STEPS_TO_IMPROVE	3	the number of steps to continue after reach the goal
SAMPLES_FOR_CALC_CENTERS	300	How many samples to use to calculate the centers
MAX_PERTURBATION	5	The maximal size of perturbation which gives posisitve reward
REWARD_COEF	10	The ratio between perturbation reard and label reward
DQN_LR	0.01	The size of the learning rate
WINDOW_LENGTH	4	The size of the experience replay window
TARGET_MODEL_UPDATE	500	The rate of updating the dqn target network

5 Results

This section presents the results of running the various algorithms on the environment. First, I present the results of the deterministic algorithm, then the results of the reinforcement learning algorithm, and finally, I present a comparison between the different methods.

For each algorithm I present the information for each target label that consists the following fields:

1. Target label - the algorithm goal.
2. nobs - number of experiments. I run 100 random tests per target label for all algorithms.
3. Min perturbation - the minimal perturbations that happened in the episodes.
4. Max perturbation - the maximum of all minimal perturbation over all the episodes.
5. Steps - the minimal and maximal steps needed to reach minimum perturbation. Note: these are not the steps needed to reach the Min perturbation and Max perturbation.

5.1 Deterministic algorithms

This section presents the results of the deterministic algorithms.

5.1.1 Triangle agent result

Table 2 present the results of the triangle agent.

Table 2: Triangle agent's result. The table explains the details about the minimal perturbations that happened in the episodes. Min perturbations is the minimum perturbations reached in all episodes. Max perturbations is the maximum perturbations over all the episodes' minimal perturbation.

		perturbation				**steps**			
target	**nobs**	**min**	**max**	**mean**	**variance**	**min**	**max**	**mean**	**variance**
0	100	3.79	7.84	5.42	0.97	2	1895	194.2	168529.84
1	100	2.0	11.33	6.23	2.37	1	1273	44.42	20352.17
2	100	2.0	9.01	5.35	1.89	1	1994	191.41	216859.19
3	100	3.59	10.72	5.79	1.81	2	1978	188.25	190226.86
4	100	3.55	10.04	5.78	1.71	2	1632	160.87	150310.03
5	100	3.07	9.6	6.2	1.84	2	401	19.24	2625.3
6	100	2.0	10.24	5.87	2.01	1	1987	207.7	195911.51
7	100	3.54	9.42	6.06	2.14	2	1979	192.28	211393.82
8	100	2.0	10.3	5.28	1.62	1	1940	192.8	192080.91
9	100	3.8	8.73	5.69	1.34	2	1930	77.56	74499.2

5.1.2 Alternating triangle agents

Table 3 shows the result of the alternating triangle agent, without resetting the step size as the centers are changed.

Table 4 shows the result of the alternating triangle agent, with resetting the step size to the initial step size value every time the centers are changed.

Table 3: Iterative centers algorithm results

		perturbation				steps			
target	**nobs**	**min**	**max**	**mean**	**variance**	**min**	**max**	**mean**	**variance**
0	100	3.52	6.57	5.15	0.56	2	1998	1240.96	337448.91
1	100	3.71	9.9	5.87	1.56	2	1924	840.9	413978.66
2	100	2.0	7.26	5.02	0.86	1	1999	894.11	456457.01
3	100	3.73	8.01	5.18	1.1	2	1993	1154.96	427066.36
4	100	3.25	7.96	5.24	1.07	2	1994	1147.4	426411.49
5	100	3.02	8.61	5.75	1.47	2	1810	240.7	131112.35
6	100	2.0	8.34	5.38	0.96	1	1991	1391.65	250875.6
7	100	3.52	8.02	5.41	1.28	2	1993	877.99	397813.36
8	100	2.0	7.18	4.91	0.75	1	1940	1086.72	437924.24
9	100	3.75	8.62	5.39	1.16	2	1989	1097.95	451468.78

Table 4: Iterative centers algorithm results with increasing the step size

		perturbation				steps			
target	**nobs**	**min**	**max**	**mean**	**variance**	**min**	**max**	**mean**	**variance**
0	100	3.23	6.53	5.07	0.51	3	1990	780.97	362951.63
1	100	3.8	9.16	6.06	1.36	2	1827	569.13	337126.05
2	100	3.42	9.49	4.93	0.9	2	1999	625.76	340929.54
3	100	3.33	10.6	5.11	1.35	2	1999	816.65	441403.28
4	100	3.33	8.78	5.21	1.21	2	1999	801.98	408205.86
5	100	3.79	9.09	5.86	1.72	2	1995	180.07	102842.85
6	100	2.0	7.51	5.3	0.91	1	1997	867.24	339526.18
7	100	3.02	9.55	5.43	1.67	2	1827	622.48	341692.9
8	100	3.3	7.03	4.95	0.87	2	1997	555.36	345146.5
9	100	3.36	7.9	5.05	0.94	2	1835	631.79	377226.27

5.2 Reinforcement learning

5.2.1 Fitting effort

It took 40 minutes to perform 10,000 learning steps on the i7 CPU Linux server. The full fitting task **for a single target**, lasted more than five days.

5.2.2 Full set of actions

I have been training the agent with a complete set of actions for 5 days. Every 50,000 training steps, I test its performance. I found that the agent was ignoring the observation from the environment, taking just one action and staying with it during the episode. In the beginning of the training phase, it was the "farthest center" action, then the "closest center" action, and then it changed to "change centers" action. I glanced at the reward table and found that the agent got almost no reward. Thus, it took the last action that would give him a reward and stick with it.

5.2.3 Minimal set of actions

I tried to train the agent with a limited set of actions that included the following: "CLOSET CLUSTER," "FARTHEST CLUSTER," "ORIGINAL IMAGE," "DECREASE STEP." Again, the agent nearly didn't change his preferred action and used the "CLOSET CLUSTER" action in most of the steps. Table 5 shows the performance of one of the targets.

Table 5: Reinforcement learning agent performance for target 1

		perturbation				**steps**			
target	**nobs**	**min**	**max**	**mean**	**variance**	**min**	**max**	**mean**	**variance**
1	100	4.0	10.0	7.3	1.8	2	5	3.65	0.45

5.3 Summary

Table 6 shows the mean perturbation size for each target label. The RL column displays just the result of a single target, owing to the amount of time it takes to train the agent.

Table 6: Summary of all 4 methods. The table shows the mean perturbation size for each target label

	Triangle	Alter. V1	Alter. V2	RL
0	5.42	5.15	5.07	
1	6.23	5.87	6.06	7.3
2	5.35	5.02	4.93	
3	5.79	5.18	5.11	
4	5.78	5.24	5.21	
5	6.2	5.75	5.86	
6	5.87	5.38	5.3	
7	6.06	5.41	5.43	
8	5.28	4.91	4.95	
9	5.69	5.39	5.05	

6 Discussion and Conclusions

All methods managed to find adversarial examples that fool the blackbox classifier. However, we need to look carefully at the results. There are examples that with few algorithm's steps achieve low perturbation size. These examples come probably from source labels very close to the boundary. Thus, few steps that hardly touch the example can create a fantastic result. I suggest giving these examples small weight in the analysis and concentrate on the "average perturbation" metric.

The average size of the perturbation is larger than that reported in other blackbox works. Current algorithm performance is not enough to attack real-world machine learning models. However, the results were generated without any hyperparameter optimization. I assume that hyperparameter optimization will dramatically boost the performance.

The proposed actions are sufficient for exploration and the discovery of adversarial examples. I believe that adding more actions to those supported in this work will improve the results even more.

The deterministic algorithms did not require training at all. This is a major benefit for them. In fact, they require less computing power and memory resources that is essential to the development of adversarial examples for real-world models.

The alternating algorithm showed better results than the others. It takes advantage of more samples from the training set, and it can touch more points around the boundary line. Again, I believe that with the right hyperparameter optimization, the results of this method can be improved even more.

The reinforcement learning method has shown the poorest results. This is due to the large amount of training needed to fit the agent. In the training phase, the agent almost didn't get any rewards, because it was very difficult to follow the criteria for a good example. When I made the requirement more flexible, the agent managed to generate an example. These were not good enough examples, however, and it was very easy to notice the perturbation. I suppose the observation space is too large, so it's going to be helpful to find a way to compact it, with PCA for example. In addition, a combination of a deterministic algorithm and a reinforcement learning agent, as suggested for future work, may be a promising direction.

The environment and framework that have been developed during this work are empowering the research work. They have the ability to be a platform for designing and evaluating attacks algorithms based on reinforcement learning building blocks. The effort that will be required to plan and analyze the attacks in the future would be substantially limited.

7 Future work

The first direction for future work is to extend the action space. Adding actions to the environment would make it easier to support more algorithms that can perform better.

Another interesting research direction would be to give the iterative policy to the reinforcement algorithm as its initial policy and to give it the task of refining it. The combination of the two method may achieve a better result with less computational power.

Finally, I would like to recommend that this framework be published as an open source project for the community. I think it complements the CleverHans environment. The use of both together can enhance the researcher's ability to develop and test new algorithms.

References

[Brendel et al., 2017] Brendel, W., Rauber, J., and Bethge, M. (2017). Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*.

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

[Chen et al., 2019] Chen, J., Jordan, M. I., and Wainwright, M. J. (2019). Hopskipjumpattack: A query-efficient decision-based attack. *arXiv preprint arXiv:1904.02144*, 3.

[Hyrum, 2017] Hyrum, S. A. (2017). Evading machine learning malware detection. *Black Hat USA*.

[LECUN,] LECUN, Y. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*.

[Papernot et al., 2018] Papernot, N., Faghri, F., Carlini, N., Goodfellow, I., Feinman, R., Kurakin, A., Xie, C., Sharma, Y., Brown, T., Roy, A., Matyasko, A., Behzadan, V., Hambardzumyan, K., Zhang, Z., Juang, Y.-L., Li, Z., Sheatsley, R., Garg, A., Uesato, J., Gierke, W., Dong, Y., Berthelot, D., Hendricks, P., Rauber, J., and Long, R. (2018). Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*.

[Plappert, 2016] Plappert, M. (2016). keras-rl. `https://github.com/keras-rl/keras-rl`.