Defending Against Adversarial Attacks on Policies Through Density Estimation

Alberto Villanueva[1], **Marcos Villacañas**[1], **Rubén Majadas**[1], **Javier García**[2], **Fernando Fernández**[1]

[1] Departamento de Informática, Universidad Carlos III de Madrid
[2] Departamento de Electrónica y Computación, Universidad de Santiago de Compostela
alvillan@inf.uc3m.es, mvillaca@pa.uc3m.es, rmajadas@pa.uc3m.es, franciscojavier.garcia.polo@usc.es, ffernand@inf.uc3m.es

Abstract

In recent years, reinforcement learning (RL) and, in particular, its "deep" variant, has been applied to tasks in the real world gradually. RL has shown unprecedented popularity, such as autonomous driving, robot control, solving complex video games. It was a matter of time before deep RL burst into finance and trading as well. Financial markets are simply too complex for non learning-based algorithms, as the state and action spaces are continuously expanding every second. With RL, however, we can learn autonomously complex trading strategies that can maximize profits in spite of the highly stochastic and non-stationary nature of financial markets. But if the field of finance and trading is benefiting from the power of deep RL algorithms, it has also inherited its vulnerabilities. Deep RL algorithms are well-known to be inherently vulnerable to manipulation by intentionally perturbed observations, rewards or actions, leading to unintended and potentially harmful results. This is particularly relevant in a financial context, where exploiting these vulnerabilities provides adversaries with the means to lead a company to millionaire losses. For this reason, in this paper we investigate a two-step defense mechanism not only able to detect these adversarial attacks, but also to recover from them. We show that our approach manages to achieve a nearly perfect defense in simple domains, and is proficient against several state-of-the-art attacking strategies.

Introduction

There has been an upward trend in recent years to use reinforcement learning (RL) in financial markets (Fischer 2018) and, as RL policies get applied to real world environments, a focus has to be placed on ensuring these policies are resilient against malign actors trying to interfere with the system. A perfect example of this is High Frequency Trading (HFT), a trading strategy which uses high speed algorithms in order to benefit from arbitrage opportunities. This way of operating constantly scans the Limit Order Book status, seeking to find mismatches of any size. As a result, hundreds of trades are closed every second placing value on, not only momentum, but also reliability. Recently, RL has gained popularity among policy training methods for HFT strategies (Briola et al. 2021). Thus, the motivation of this research is clear. Fighting adversarial attacks would help to prevent millionaire loses in this field. Recent advancements in adversarial machine learning have shown that agents based in neural networks are particularly sensitive to malignantly crafted small perturbations (Szegedy et al. 2014). These attacks have usually been applied to supervised learning tasks. However, they have also shown to be proficient when tackling RL policies (Kos and Song 2017).

In our work, we propose a two-step based defense system. First, we use experience tuple density estimation to identify when a new state has been perturbed. By using a non-supervised method, we avoid modeling specific attacking patterns, which aids in generalizing against any possible attack. Then, if a perturbation is detected in the state, the original state is recovered by using a k-nearest-neighbor approach on the experience tuple space. In contrast to previous defense methods, our approach avoids the use of neural networks, as defense systems based on neural networks have also shown to be sensitive to malignantly crafted perturbations (Carlini and Wagner 2017a). We show that our approach manages to achieve a nearly perfect defense in simple domains, and is proficient against several state-of-the-art attacking strategies.

Background

In this section, the concepts of RL and adversarial machine learning are reviewed.

Reinforcement Learning

RL environments are typically formalized by means of a Markov Decision Process (MDP) (Sutton and Barto 2018), which is a mathematical model of sequential decisions and a dynamic optimization method. It is represented by a tuple $< S, A, T, R >$ where *S* is the set of possible states, *A* is the set of possible actions available from each state, *T : S x A x S' → [0,1]* is the transition function where *T(s,a,s')* represents the probability of getting to state *s'* from state *s* when action *a* is performed, and $R : A \times S \rightarrow \Re$ is the reward function which maps a state *s* and an action *a* into a reward *R(s,a)*; *r* is used hereinafter to represent the stochastic reward result obtained from a distribution with mean *R(s,a)*.

In a Markov decision-making process, the transition probability and reward only depend on the current state and the action chosen. As a result, the objective is learning a policy π for every state to maximize the return of $J(\pi)$:

$$J(\pi) = \sum_{k=0}^{K} \gamma^k r_k \tag{1}$$

where γ affects how much future is considered from step *k* (discount factor, with $0 \leq \gamma \leq 1$) and r_k corresponds to reward received in step *k*. At this point, the value-function, which estimates the sum of rewards given a policy π, can be solved using Bellman's equation.

In Deep Q-Learning (Mnih, Kavukcuoglu, and Davi 2013), neural networks are used to estimate the action-value function, and the value over which the loss function takes place for the learning process, corresponds to the squared Bellman error, that is, the difference between the expected value and the predicted value.

$$\mathbb{E}_{s,a}[(r + \gamma \max_{a'} Q(s', a') - Q(s, a))^2] \tag{2}$$

Along to this, the learning takes place in a batch fashion, where instead of learning over one experience tuple at a time, the agent holds a memory of the last N experience tuples and, at every learning step, a sample is taken over the experience tuples in memory. These agents built using Neural Network take the name of Deep Q Networks (DQN) (Mnih, Kavukcuoglu, and Davi 2013).

Adversarial Machine Learning

In adversarial machine learning, adversarial examples are maliciously generated such that the machine learning model fails its task. Particularly, models based on neural networks have been shown to be specially vulnerable to such attacks.

These attacks can be classified as poisoning attacks, where adversarial examples are injected into the training phase, and exploratory attacks, where the adversarial examples are injected into the testing phase (Sethi and Kantardzic 2018). Furthermore, the previous classification can be enriched by taking into consideration a complementary perspective of the offensive strategy. Depending on whether the adversary has access to the model parameters, architecture or training data, or not, attacks are considered to be white-box or black-box, respectively. Even though having total knowledge to model's internal data for white-box attacks is game-changing capability, it is not common under real scenarios circumstances. From a realistic point of view, trying to exploit the model's vulnerabilities based on input/output pairs exclusively, as it is on black-box attacks, is closer to a down-to-earth approach. On adversarial machine learning tasks, the only thing that needs to be defined is how to create the adversarial example, however in adversarial RL, also it has to be decided when to create an adversarial example.

Adversarial example crafting Crafting an adversarial example x^δ for a model f from a given input x can be described as the optimization problem described in Equation 3, where l and l^δ are the labels of x and x^δ respectively and $\|\cdot\|$ denotes the p-norm distance. Here the perturbation can be defined as $\eta = x^\delta - x$ and the optimization problem can be described as finding the lowest perturbation η that manages to make the model f make a mistake when evaluating an instance x. In the area of classification, making a mistake would be equivalent to predicting the wrong label for x, and in the area of regression to predicting a value as far away as possible from the real value.

$$\begin{aligned} \min \quad & \left\|x^\delta - x\right\|_p \\ s.t. \quad & f(x^\delta) = l^\delta \\ & f(x) = l \\ & l \neq l^\delta \\ & x \in [0, 1] \end{aligned} \tag{3}$$

Adversarial attacks on policies Regarding RL in particular, with the intention of contemplating different approaches, the spotlight has been put on three recently proposed strategies which have been chosen to be tested against detection and recovery techniques presented hereunder.

- **Uniform attack**: this adversarial behavior consists on perturbing each state the agent observes, that is to say, attacking at every time step adding some noise (Huang et al. 2017). The perturbation injected can be created using FGSM (Szegedy et al. 2014) where the cost function needed to calculate the gradients $J(\theta, x, y)$ is the cross-entropy loss between y, which is the weighting over the possible actions, and the distribution that places all weight on the highest weighted action in y. (Huang et al. 2017). On the subject of our work, in order to approximate better to a real-life situation, it has been decided to not attack continually.
- **VF attack**: this attack injects a perturbation when the value function is greater than some threshold β. The reasoning is to perturb the agent on some crucial moments when it is close to reaching a reward. This can be seen as described in Equation 4. (Kos and Song 2017). Just like in uniform attack, to craft the adversarial examples, FGSM can be used (Szegedy et al. 2014).

$$\max_{a \in A} Q(s_t, a) > \beta \tag{4}$$

- **Strategically-Timed attack**: in this strategy, a perturbation is only incorporated to a normal state when there is a strong preference for an action over another one. Consequently, this attack inflicts more harm at the time the trained agent would strive to take a key action (Lin et al. 2019). The attack trigger is also set using a beta parameter, thus only when the difference between the highest and lowest Q values[1] exceeds beta, the attack will take place as can be seen in Equation 5. In addition, to avoid attacking excessively, a second parameter controls the maximum amount of times the agent can suffer an attack (Lin et al. 2019). To craft adversarial examples, it looks for an observation that can change the most preferred action to the least preferred one by using the Carlini & Wagner attack (Carlini and Wagner 2016).

$$\max_{a \in A} Q(s_t, a) - \min_{a \in A} Q(s_t, a) > \beta \tag{5}$$

[1] In the original work the difference between the preference to take an action is used. For policy gradient algorithms that is the probability to take an action and for value based methods it is the softmax function of the Q values

VF and Strategically-Timed attacks are white-box attacks, hence access to agent's Q values is required, while uniform attack is a black-box attack. However, it has been shown that, although less effective, an adversarial example can be transferred from one agent to another which could make white-box attacks work in a black-box environment (Huang et al. 2017).

Related work

To countermeasure these adversarial attacks, several defensive strategies have been proposed which can be divided into two distinct groups, proactive and reactive (Yuan et al. 2019). Proactive strategies protect the model from being affected from adversarial attacks before the model has been attacked. These include techniques such as adversarial training (Wu, Bamman, and Russell 2017; Dong et al. 2017; Goodfellow, Shlens, and Szegedy 2015; Tramèr et al. 2020; Huang et al. 2016), classifier robustifying (Bradshaw, de G. Matthews, and Ghahramani 2017; Abbasi and Gagné 2017), and network destilation (Papernot et al. 2016). Inside reactive strategies, two main types are discussed, adversarial detecting, in which the objective is to identify in test which of the given instances are adversarial, and input reconstruction, in which the objective is to reconstruct an adversarial example to the original example without the added noise so that the model can work normally with the clean example.

A plethora of different approaches have been taken to detect these adversarial attacks in the testing stage. A binary classifier sub-network can be trained with adversarial examples to distinguish them from the clean ones (Metzen et al. 2017; Gong, Wang, and Ku 2017) and another way is to add an additional output class to the networks output that corresponds to adversarial inputs (Grosse et al. 2017). Safety-net (Lu, Issaranon, and Forsyth 2017) uses an RDF-SVM that utilizes discrete codes computed from late stage ReLUs to detect adversarial examples, however similarly to the previous methods, it also requires to be trained with adversarial examples, which means that it has to model the attacker which is unlikely to generalize well to other processes to generate adversarial examples (Meng and Chen 2017).

Mag-Net (Meng and Chen 2017) uses an ensemble of 2 detectors, one is an autoencoder trained on the original clean dataset that predicts whether it is an adversarial example or not based on the reconstruction error, where an error higher than a threshold indicates an adversarial example. This performs well when the error is high enough but for smaller errors a second detector was added that measures the divergence between the logit of the input example and that of the autoencoded example, where a high divergence indicates an adversarial example, and finally, to ensemble them an adversarial attack is reported if any of the two detectors detect one.

Principal component analysis (PCA) whitening can be used on the input examples, and since adversarial ones have different coefficients for low-ranked principal components, a detector can be created from it (Hendrycks and Gimpel 2017). Furthermore, the authors say that a combination of detectors might be a better way to try to deal with adversarial detection.

A detector can also be built using a logistic regressor using as inputs the kernel density of the input example, calculated using the training set on the feature space of the last hidden layer, and the Bayesian uncertainty estimate of the network for said example (Feinman et al. 2017).

However, Carlini & Wagner (2017a) showed that most of these defenses are not as effective as previously analyzed, as they are still susceptible to their previous attack (Carlini and Wagner 2016) when the loss function is changed. Furthermore, they offer some guidelines on how to better approach detection defenses; such as using strong attacks for evaluation instead of single iteration ones, and that it should be resistant against white-box attacks too and not only gray or black box ones. They also showed the methods that performed the worst were the ones using another neural network for detection, because if an adversarial example can be made to fool one network, there can also be one that fools both (Carlini and Wagner 2017a,b).

Input reconstruction tries to transform the input data when it has been detected as having been attacked such that the output of the reconstructed input matches that of the unattacked one. Autoencoders can be used to perform input reconstruction (Meng and Chen 2017; Gu and Rigazio 2015) by learning from the training data. Another way is the approach taken by pixel defend (Song et al. 2017) where the training distribution probability of the novel example is estimated by using PixelCNN (van den Oord, Kalchbrenner, and Kavukcuoglu 2016) and then, a new value is generated for each pixel along each channel such that the probability distribution estimated of the new example is maximized and the difference between the original value and the new value is smaller than some value ϵ moving it closer towards the training dataset.

In the field of RL, on previous work on defensive mechanisms, the current state is predicted with the previous m states and m actions with which the result is then used to compare the states Q values with the received state Q values, where a high discrepancy indicates an adversarial attack has been produced, since the objective of the adversarial attack is to change the policy of the agent. Furthermore, an optimal action can be suggested based on the Q values of the predicted state and thus reconstruct the policy for the state (Lin et al. 2017).

Architecture - Proposal

The architecture for our defense system consists of two modules, the detection system, which detects if a state has been attacked, and the recovery system, which once a state has been detected as an attacked one, tries to recover the original values from the state. Both systems work by inferring knowledge over the experience tuple instead of just over the state to take into account the sequential nature of RL tasks, so after an action a is performed over the state s, the detection system analyzes the experience tuple $< s, a, s', r >$ to detect if the next state s' has been attacked or not.

To achieve this, both systems use experience tuples in the form of $\tau = < s_0, s_1, ..., s_n, \ a_0, a_1, ..., a_m, s'_0, s'_1, ..., s'_n, r >$, where s_i is the i-th feature of the state,

and a_i is the i-th feature of the action, where categorical actions or state features are one-hot encoded so that distance measures can be used, that are extracted by observing the agent's regular behavior on the testing phase and storing its experience tuples in the form of $\Gamma = \{\tau_0, \tau_1, ..., \tau_n\}$. To calculate similarity measures between any two experience tuples, ℓ_2 distance is used.

The overall process can be described as a number of steps where first the agent is trained (i), then Γ is extracted (ii) with which, a detection system (iii) and a recovery system (iv) are trained. During execution, the systems works as shown in Figure 1 where each new transition is given to the detector which predicts if it belongs to the distribution of Γ and, if it doesn't, it is given to the recovery system, which returns the reconstructed new state.

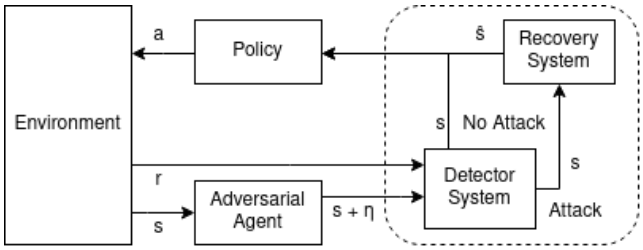

Figure 1: Defense Architecture Diagram

Detection System

When it comes to detecting an adversarial transition, our method compares the new perceived transition against the transitions in Γ, and if it deviates far enough from them, it is classified as an adversarial transition. This task can be seen as detecting outliers (adversarial examples) or estimating the underlying density function (DF) of Γ and labeling as adversarial transitions those with a low DF.

A number of different algorithms are used to estimate the DF, none of which use a Neural Network as the one used in pixel defense (Song et al. 2017) or in Visual Foresight (Lin et al. 2017), since Carlini and Wagner (2017a) found that defenses that used a Neural Network to defend from adversarial attacks happened to be the ones who performed the poorest as an attack could be designed to circumvent both neural networks, the predictor and the defender:

- **Kernel density** with a Gaussian kernel is used to estimate the underlying probability DF.
- **DBSCAN** (Ester et al. 1996) tries to identify the outliers in a set of data, however, in our case we already know a set of non-outlier data points, that being Γ, so instead we only calculate whether the new transition is an outlier or not in respect to Γ. Furthermore, instead of using a minimum number of samples needed inside the given radius, the number of samples inside the radius is used as the DF value.
- **KNN density estimation** is used to estimate the underlying DF, however, two different implementations of density estimation using KNN have been used:
  - The distance to the k nearest neighbors is taken, and a function is applied to them that returns a scalar such as the sum, the mean or maximum (which would simply be the distance to the k-th nearest neighbor).
  - The density is built according to the formula $\frac{n(x,a)}{NV(B(x,a))}$ where $B(x, a)$ is the hypersphere centered in x with radius a, $V(\cdot)$ represents a volume, N is the number of samples and $n(x, a)$ is the number of points within $B(x, a)$ (Zhao and Lai 2020). From now on, this one will be referred to as Hyper-KNN.
- **Gaussian Mixture** (Dempster, Laird, and Rubin 1977) estimates the underlying DF by fitting a mixture of Gaussian distributions to Γ and later using them to find the mixture probability.
- **K-Means** (Hartigan 1975) can be seen as a simplified version of Gaussian mixture, where the covariance matrix is fixed to being a scalar matrix, and the scalar controls the size of the cluster. The density metric used here is the inverse of the distance to the nearest centroid, as a higher number should correspond to a higher density. Furthermore, from this base DF two different approaches are taken:
  - No further changes are applied, and the DF depends solely on the distance to the nearest centroid. From now on this will be referred to as k-means global.
  - The DF is normalized with the distance to the furthest training sample assigned to the cluster the DF is based on. This makes it so that the DF takes into account contextual information from the cluster it is using, as it now depends on the size of the cluster.From now on this will be referred to as k-means local.
- **One class SVM** according to Schölkopf (Schölkopf et al. 2000) is used to establish a binary detection system. If the new transition falls between the origin and the calculated hyperplane, the sample is considered to be an adversarial transition.

Apart from one class SVM, every other method estimates a DF, but that is not enough to distinguish between adversarial and normal transitions, thus, a threshold has to be selected after which the transition is classified as adversarial. For that, the lowest DF value predicted from all transitions in Γ is used; that way, any new experience tuples with a lower DF than any in Γ is classified as an adversarial one. By selecting the decision threshold solely on normal experience tuples, it avoids the need to model any type of attacks, which ensures that it can not overfit the detection for that given attack.

Recovery System

The recovery system tries to increase the DF of the experience tuple that has been identified as an adversarial one, to do so, we once again step away from methods that use Neural Networks such as in Pixel Defense (Song et al. 2017) or visual foresight (Lin et al. 2017) for the same reasons as the ones described in the detection system, and so we propose a method that is independent of the detection system

and the learning agent, that repairs the next state by using the next state of the k nearest transitions in Γ, and weighting them by the distance to the original transition, where a k of 1 means that it takes the entire next state of the nearest neighbor. However, since the next state could be too corrupted due to adversarial injection, the recovery system calculated the distances in Γ', a subspace of Γ that does not include the next state. In order for the distances to have an actual meaning, the transition vector features have to be normalized, and discrete features have to be one-hot encoded if the set containing of all the possible values of the feature is not totally ordered.

$$\hat{s}' = \sum_{i}^{k} w_i s'_i \tag{6}$$

$$w_i = \frac{\|\tau' - \tau'_i\|_2}{\sum_{j}^{k} \left\|\tau' - \tau'_j\right\|_2}$$

Evaluation

This section evaluates the performance of the proposed defense in four different well-known domains from the OpenAI gym library: Acrobot, Cart pole, Mountain car and Taxi. But before presenting the results, the experimental setting is introduced.

Experimental setting

The proposed domains have been previously solved using tabular Q-Learning with an ϵ-greedy exploration-exploitation strategy. The agents are discretized along each dimension within the state limits in a number of bins[2], and are trained during H episodes with a limit of K steps per episode with the learning rate (α), and discount rate (γ) described in Table 2. Each of the training process will generate a policy π which will then be attacked and defended with the proposed system.

After the agents are trained, Γ is extracted (the algorithm Ball Tree (Dolatshah, Hadian, and Minaei-Bidgoli 2015) is used for efficient spacial indexing in the detector and recovery systems) from each agent by running them 500 episodes.

However, if a single adversarial example bypasses the defense system, it could throw off the trained policy off of the optimal path, which, if it is the only path the defense system has seen, would make it seem like every subsequent transition is adversarial (as it could be vastly different from what it has seen before). It is for this reason that instead of using the learned policy in a fully greedy way, an ϵ-greedy policy is used ($\epsilon = 0.1$) to prevent the transition sample from overfitting to the optimal path from the learned policy. That way, it accounts for slight deviations from the optimal path.

[2]These bin sizes are used to take into account the entire possible state space, however a lot of bins in Acrobot, Cart Pole and Mountain Car are empty as the states they describe are either unreachable, or so unlikely they are never visited. The number of bins where the learning actually takes place in is 14.3K, 13.7K and 7k for Acrobot, Cart Pole and Mountain Car respectively

For the adversarial perturbations, noise is generated at each dimension with a random value between 0 and $\frac{\delta}{\sqrt{d}}$, where d is the number of dimensions and δ is as shown in Table 1 for Acrobot, Cart Pole and Mountain Car, and then, the noise is added to the state normalized across the dimension limits such that $\| \eta \|_2 \leq \delta$. In the case of the Taxi environment, as the state space is discrete, the position of the agent is changed by 1 in the y or x coordinate. Only the position of the agent is attacked, as the other values, the passenger position and destination position, are given in the form of the index of a list that contain a set of possible positions, hence, a change of 1 in the index corresponds to a change of more than one in the actual positions, e.g. a change from 0 to 1 in the passenger position dimension (p) changes the position it is referring to, from (0,0) to (0, 4).

For the attacking strategies, three have been used; Uniform, Value Function and Strategically Timed attacks as described previously, whose parameters have been fine-tuned to achieve a compromise between the noise they inject and the damage they create, and are shown in Table 1.

Environment	δ	Uniform Frequency	VF β	ST β	max
Acrobot	0.10	0.5	-27	1.300	150
Cart Pole	0.10	0.5	40	0.800	50
Mountain Car	0.14	0.5	-15	1.300	50
Taxi	1.00	0.5	19	10.185	12

Table 1: Attack parameters

Results

Foremost, in Table 3 we showcase the average performance obtained by each agent across 200 episodes, how each attack affects the performance, and how much total perturbation (δ) it injects into an episode to achieve that loss in performance. The total perturbation injected is calculated by summing the ℓ_2 distance from the original state to the attacked state for each step in the episode. Here it can be seen how in the Cart Pole and Mountain Car environments, VF-attack and ST-attack achieve an equal or superior performance than Uniform attack while injecting much less perturbation. In the Taxi and Acrobot environments, ST-attack achieves a better performance with a similar or lower perturbation, but VF-attack achieves a higher performance at the cost of a lot more total perturbation. This is because with VF-attack, the states closer to the reward are attacked which prevents the agent from finishing the episode, and it enters a loop where it is constantly attacking the agent as the agent is always near the end of the episode but never really ending except if it performs the adequate action by pure chance, which explains the high variance.

Then Γ is normalized in one of two different ways, either using min max feature scaling normalization or z-score normalization. Afterwards, the detectors are trained using the parameters shown in Table 4. They are then evaluated by running the agent against each attack during 100 episodes

Domain	State	Bins	α	γ	ϵ	H	K
Acrobot	$\theta_1 \in [-\pi, \pi]$ $\theta_2 \in [-\pi, \pi]$ $\omega_{\theta_1} \in [-4\pi, 4\pi]$ $\omega_{\theta_2} \in [-9\pi, 9\pi]$	20 20 20 20	0.1	0.99	0.5	20000	500
Cart Pole	$p \in [-4.8, 4.8]$ $v \in [-4, 4]$ $\phi \in [-0.418, 0.418]$ $\kappa \in [-4, 4]$	10 50 50 50	0.1	0.99	0.3	20000	200
Mountain Car	$p \in [-1.2, 0.6]$ $v \in [-0.007, 0.007]$	100 100	0.1	0.99	0.3	300000	200
Taxi	$y \in \{0, 1, 2, 3, 4\}$ $x \in \{0, 1, 2, 3, 4\}$ $p \in \{0, 1, 2, 3, 4\}$ $d \in \{0, 1, 2, 3\}$	5 5 5 4	0.1	0.99	0.3	20000	200

Table 2: Train parameters for the victim policies

	No attack	Uniform	VF	ST
Acrobot	-140.96 $\pm$ 34.00	-171.13 $\pm$ 40.53 δ = 109.14 $\pm$ 28.07	-178.21 $\pm$ 52.59 δ = 131.28 $\pm$ 65.07	-180.20 $\pm$ 42.65 δ = 64.28 $\pm$ 18.43
Cart Pole	200.00 $\pm$ 0.00	116.74 $\pm$ 53.98 δ = 43.49 $\pm$ 19.13	50.77 $\pm$ 22.24 δ = 20.40 $\pm$ 11.79	52.80 $\pm$ 41.76 δ = 22.32 $\pm$ 9.47
Mountain Car	-98.98 $\pm$ 7.60	-149.87 $\pm$ 21.64 δ = 13.35 $\pm$ 2.40	-188.70 $\pm$ 23.42 δ = 9.27 $\pm$ 2.02	-154.43 $\pm$ 26.57 δ = 5.71 $\pm$ 2.39
Taxi	7.80 $\pm$ 2.56	-2.28 $\pm$ 9.92 δ = 9.95 $\pm$ 4.74	-51.40 $\pm$ 90.47 δ = 54.36 $\pm$ 82.50	-11.12 $\pm$ 14.88 δ = 9.80 $\pm$ 3.09

Table 3: Trained policy with and without attacks, and the total perturbation (δ) injected into an episode measured by ℓ_2 distance

where at each time step, if the state is detected as being an adversarial one, the original state is restored. This is done to decouple the results from the detector and the recovery systems and see how each of them behave independent of the other. The balanced accuracy is shown in Table 5 highlighting results of over 90% accuracy.

Detector	Parameters
Kernel Density	h: Scott's Rule
DBSCAN	ϵ: Scott's Rule
KNN-Hyper	a: Scott's Rule
KNN	k: 3, 5, 10, 15, 20, 30, 40, 50, 75, 100 f: sum, mean, max
Gaussian Mixture	k: 256, 512, 1024, 2048
K-means Global	k: 256, 512, 1024, 2048, 4096
K-means Local	k: 256, 512, 1024, 2048, 4096
One Class SVM	ν: 0.01, 0.05, 0.1, 0.5

Table 4: Detector parameters, where values separated by commas mean that all of those were tried and the best was selected

As can be seen in Table 5, the type of normalization has a strong impact on the performance of the detectors based on the environment; for every environment except for Cart Pole, z-score normalization has higher results for almost every detector, but in Cart Pole, min max normalization performs significantly better, having almost every detector a perfect detection score. The main reason for this can be that in a normal execution of Cart Pole with a perfect policy, every state it visits is really similar as it tries to keep the cart to the center of the screen and the pole as vertical as possible; this means that by doing min max normalization, if a single new state has a value lower than 0 or greater than 1 it is a strong indication of an adversarial attack. In contrast, in other environments, the states are a lot more varied, hence, a z-score normalization helps distinguish the most common states than the more uncommon. Furthermore, three detectors shine above others; DBSCAN, KNN-Hyper and Gaussian Mixture having a performance of above 90% in the best normalization method for every environment, however, this is while having a perfect recovery, so the performance also has to be analyzed with the recovery system.

To this end, the full defense system is now tested the same way as the detection system. Thereforere, now, when a new perceived transition is classified as an adversarial one, it is given to the recovery system which then returns the recovered new state. This recovery system is tried with multiple values for k; 1, 3, 5, 10, 15, 20, 40 and 50, and the best result is reported, although the method is not very sensitive to the different k values. Then the final reward achieved by the defense system (r_D) is measured against the reward obtained by the unaltered victim policy (r), using as a baseline the attacked reward (r_A), both of which can be seen in Table 1. In order to not just take into account the mean but also the

	Acrobot		Cart Pole		Mountain Car		Taxi	
normalization	**min max**	**z-score**	**min max**	**z-score**	**min max**	**z-score**	**min max**	**z-score**
UniformAttack								
Kernel Density	0.50	0.53	**1.00**	0.83	0.50	0.51	0.51	0.51
DBSCAN	0.65	**0.97**	**1.00**	0.63	0.63	**1.00**	**0.99**	**1.00**
KNN-Hyper	0.65	**0.98**	**1.00**	0.64	0.62	**1.00**	**0.99**	**1.00**
KNN	0.50	0.50	**1.00**	**0.98**	**1.00**	**1.00**	0.52	0.50
Gaussian Mixture	**1.00**	**1.00**	**1.00**	0.67	**1.00**	**1.00**	**1.00**	**1.00**
k-means global	0.50	0.60	**1.00**	**0.91**	**1.00**	**1.00**	-	**1.00**
k-means local	0.51	0.60	0.82	0.84	0.78	0.76	-	**0.93**
SVM	0.53	0.58	**1.00**	0.78	0.56	0.62	0.56	0.56
VF-Attack								
Kernel Density	0.50	0.75	0.50	0.50	0.51	0.53	0.61	0.53
DBSCAN	0.77	**0.98**	**1.00**	0.50	0.68	**1.00**	**0.99**	**1.00**
KNN-Hyper	0.77	**0.98**	**1.00**	0.50	0.68	**1.00**	**0.99**	**1.00**
KNN	0.50	0.52	**1.00**	0.71	**1.00**	**1.00**	0.64	0.50
Gaussian Mixture	**1.00**	**1.00**	**1.00**	0.52	**1.00**	**1.00**	**0.99**	**1.00**
k-means global	0.50	0.68	**1.00**	0.58	**1.00**	**1.00**	-	**1.00**
k-means local	0.50	0.52	0.61	0.58	0.72	0.72	-	**1.00**
SVM	**0.91**	0.89	**1.00**	0.50	0.77	0.78	0.72	0.69
ST-Attack								
Kernel Density	0.50	0.56	**1.00**	0.87	0.50	0.51	0.60	0.54
DBSCAN	0.63	**0.99**	**1.00**	0.67	0.66	**1.00**	**1.00**	**1.00**
KNN-Hyper	0.64	**0.98**	**1.00**	0.65	0.65	**1.00**	**1.00**	**1.00**
KNN	0.50	0.50	**1.00**	**0.97**	**1.00**	**1.00**	0.63	0.50
Gaussian Mixture	**1.00**	**1.00**	**1.00**	0.72	**1.00**	**1.00**	**1.00**	**1.00**
k-means global	0.50	0.59	**1.00**	**0.90**	**1.00**	**1.00**	-	**1.00**
k-means local	0.51	0.54	0.84	0.81	0.84	0.85	-	**0.93**
SVM	0.56	0.67	**0.99**	0.78	0.50	0.61	0.63	0.61

Table 5: Total balanced accuracy (using the transitions of 100 episodes) for the best parameter of each detector, using either min max feature scaling, or z-score normalization

variance of the rewards, Welch's t-test is used to compare both r_D and r_A to r, and finally this can be used to measure how the defense system affected the relative performance in a scale from 0 to 1 with the formula $\frac{t_D - t_A}{-t_A}$ where t_D is the t-score of r_d and r, and t_A is the t-score of r_A and r. The values can of course fall below 0 or raise above 1, where below 0 means the defense system creates a performance worse than the attack it is defending against, and above 1 a performance greater than the unaltered victim policy. These results can be seen in Table 6 where a performance over 90% is highlighted.

From these results, several conclusions can be drawn. Foremost, it can be seen that once again a distinction can be seen between different types of normalization depending on the environment, and said distinction corresponds to the one described before (min max normalization working better for Cart Pole and z-score normalization for the rest). Furthermore, it can also be seen how Gaussian mixture's performance drops significantly (specially in Cart Pole) compared to its performance in the detection tests, as with a perfect recovery it was achieving some of the highest results across all environments. This indicates that it was overfitting to the unaltered policy transitions, and as soon as the recovery introduced a slight error, the model did not recognize it as being valid. This could be because of a high number of clusters compared to the number of significantly different transitions, and lowering said number could lead to a better generalization.

In the Acrobot environment, a big difference can be seen from the different attacks, as with VF-attack results even better than the ones with the unaltered policy are obtained, however the detection accuracy was on par with the other attacks. The reason for the big difference in the reward, is because in Acrobot when the arm is near then top of the screen (which is when VF performs attacks) the arm already has velocity and a perfect recovery is not needed to finish the episode. On the other attacks, DBSCAN achieves the best results with a 62% and 74% recovered relative reward in uniform and ST attacks respectively.

In the Cart Pole environment, the performance is fully recovered in uniform and ST attacks, however, unlike in Acrobot, the defense does not manage to recover the performance against VF-attack. The reason for this is because, since VF-attack attacks performs multiple attacks in a row, the recovered state error starts accumulating until the detector starts failing. This was not that big of an issue in Acrobot because these repeated attacks happen towards the end of the episode, however, in Cart Pole the states at the beginning of the episode and at the end can have an equally high VF value, and so this carried error can start very early in the

	Acrobot		Cart Pole		Mountain Car		Taxi	
normalization	**min max**	**z-score**	**min max**	**z-score**	**min max**	**z-score**	**min max**	**z-score**
UniformAttack								
Kernel Density	-1.33	-0.90	**1.00**	-1.05	0.03	0.16	-0.13	0.15
DBSCAN	-1.08	0.62	**0.96**	-2.86	0.14	**1.01**	-0.60	0.64
KNN-Hyper	-1.47	0.47	**0.96**	-2.59	0.12	**0.99**	-0.30	0.64
KNN	-1.11	-0.41	**1.00**	-0.41	**1.01**	**1.00**	0.13	0.20
Gaussian Mixture	-0.43	0.34	-1.71	-2.35	**1.05**	**1.01**	-0.25	0.74
k-means global	-1.25	-0.54	**1.00**	-0.66	**1.04**	**1.03**	-	0.71
k-means local	-0.95	-0.11	0.15	-0.89	0.52	0.52	-	0.87
SVM	-1.02	-0.65	**1.00**	-1.90	0.07	0.16	0.39	0.17
VF-Attack								
Kernel Density	-1.38	**1.37**	-0.04	-0.15	0.02	0.18	0.16	0.75
DBSCAN	-0.69	**1.04**	-0.11	-0.16	0.70	**0.97**	0.38	0.88
KNN-Hyper	-0.71	**1.01**	-0.12	-0.21	0.71	**0.95**	0.44	0.74
KNN	-1.00	**1.48**	-0.02	-0.16	**1.02**	**1.01**	0.78	0.39
Gaussian Mixture	-0.24	**1.35**	-0.07	-0.07	**1.04**	**1.01**	0.35	**1.04**
k-means global	-1.22	**1.51**	-0.02	-0.17	**1.02**	**1.01**	-	0.82
k-means local	-1.20	**1.47**	0.12	0.09	0.58	0.75	-	**1.04**
SVM	-0.22	**1.35**	-0.06	-0.22	0.38	0.82	**0.97**	**0.96**
ST-Attack								
Kernel Density	-0.81	0.04	**1.00**	-0.11	0.14	0.08	0.12	0.22
DBSCAN	-0.76	0.74	**1.00**	-0.68	0.50	**1.00**	0.19	**0.93**
KNN-Hyper	-0.78	0.53	**0.99**	-0.72	0.47	**1.00**	0.17	0.88
KNN	-0.71	0.35	**1.00**	-0.09	**1.03**	**1.04**	0.17	0.22
Gaussian Mixture	-0.13	0.58	-0.29	-0.49	**1.03**	**1.04**	0.23	**0.95**
k-means global	-0.58	0.27	**1.00**	-0.05	**1.03**	**1.02**	-	0.88
k-means local	-0.73	0.37	0.62	0.07	0.63	0.63	-	**1.01**
SVM	-0.47	0.17	**1.00**	-0.24	0.48	0.47	0.60	0.59

Table 6: Relative recovered performance obtained by the defense using the best parameters

episode.

Finally, in the Mountain Car and Taxi environments multiple detectors achieve a performance comparable with that of the unaltered policy, but DBSCAN and Gaussian Mixture particularly show better results than other defenses across both domains.

Conclusion

In this work, a two-step defense system against adversarial attacks in RL is created; (i) a density estimation based approach to detect adversarial examples and (ii) a KNN approach to recover the original states. Different methods to estimate the density are used to show the viability of using density estimation on experience tuples to detect adversarial examples. None of the methods used relies on the use of neural networks, which also helps against attacks that could target the detecting network alongside the victim policy. It is also shown that choosing the detection threshold solely on the observed experience tuples of the victim agent is enough to successfully detect adversaries, avoiding the need to model any particular attack which prevents overfitting into any particular attack and helps to generalize. Furthermore, it is also shown how the normalization method impacts the system and how the nature of an environment affects which normalization is best to use.

The defense system is benchmarked against three state-of-the-art attacks across four well-known environments, managing to recover most of the lost performance for most attacks and environments, albeit it suffers from the carried recovery error in consecutive attacks that start early on an episode. A recovery system with a lower error would fix these problems and achieve a better performance.

Acknowledgments

This research was funded in part by JPMorgan Chase & Co. Any views or opinions expressed herein are solely those of the authors listed, and may differ from the views and opinions expressed by JPMorgan Chase & Co. or its affiliates. This material is not a product of the Research Department of J.P. Morgan Securities LLC. This material should not be construed as an individual recommendation for any particular client and is not intended as a recommendation of particular securities, financial instruments or strategies for a particular client. This material does not constitute a solicitation or offer in any jurisdiction.

References

Abbasi, M.; and Gagné, C. 2017. Robustness to Adversarial Examples through an Ensemble of Specialists. arXiv:1702.06856.

Bradshaw, J.; de G. Matthews, A. G.; and Ghahramani, Z. 2017. Adversarial Examples, Uncertainty, and Transfer Testing Robustness in Gaussian Process Hybrid Deep Networks. arXiv:1707.02476.

Briola, A.; Turiel, J.; Marcaccioli, R.; and Aste, T. 2021. Deep Reinforcement Learning for Active High Frequency Trading.

Carlini, N.; and Wagner, D. A. 2016. Towards Evaluating the Robustness of Neural Networks. *CoRR*, abs/1608.04644.

Carlini, N.; and Wagner, D. A. 2017a. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. *CoRR*, abs/1705.07263.

Carlini, N.; and Wagner, D. A. 2017b. MagNet and "Efficient Defenses Against Adversarial Attacks" are Not Robust to Adversarial Examples. *CoRR*, abs/1711.08478.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1): 1–38.

Dolatshah, M.; Hadian, A.; and Minaei-Bidgoli, B. 2015. Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces. *CoRR*, abs/1511.00628.

Dong, Y.; Su, H.; Zhu, J.; and Bao, F. 2017. Towards Interpretable Deep Neural Networks by Leveraging Adversarial Examples. *CoRR*, abs/1708.05493.

Ester, M.; Kriegel, H.-P.; Sander, J.; Xu, X.; et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, 226–231.

Feinman, R.; Curtin, R. R.; Shintre, S.; and Gardner, A. B. 2017. Detecting Adversarial Samples from Artifacts. arXiv:1703.00410.

Fischer, T. G. 2018. Reinforcement learning in financial markets - a survey. Technical report.

Gong, Z.; Wang, W.; and Ku, W.-S. 2017. Adversarial and Clean Data Are Not Twins. arXiv:1704.04960.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572.

Grosse, K.; Manoharan, P.; Papernot, N.; Backes, M.; and McDaniel, P. 2017. On the (Statistical) Detection of Adversarial Examples. arXiv:1702.06280.

Gu, S.; and Rigazio, L. 2015. Towards Deep Neural Network Architectures Robust to Adversarial Examples. arXiv:1412.5068.

Hartigan, J. A. 1975. *Clustering algorithms*. John Wiley & Sons, Inc.

Hendrycks, D.; and Gimpel, K. 2017. Early Methods for Detecting Adversarial Images. arXiv:1608.00530.

Huang, R.; Xu, B.; Schuurmans, D.; and Szepesvari, C. 2016. Learning with a Strong Adversary. arXiv:1511.03034.

Huang, S.; Papernot, N.; Goodfellow, I.; Duan, Y.; and Abbeel, P. 2017. Adversarial Attacks on Neural Network Policies. arXiv:1702.02284.

Kos, J.; and Song, D. 2017. Delving into adversarial attacks on deep policies. arXiv:1705.06452.

Lin, Y.-C.; Hong, Z.-W.; Liao, Y.-H.; Shih, M.-L.; Liu, M.-Y.; and Sun, M. 2019. Tactics of Adversarial Attack on Deep Reinforcement Learning Agents. arXiv:1703.06748.

Lin, Y.-C.; Liu, M.-Y.; Sun, M.; and Huang, J.-B. 2017. Detecting Adversarial Attacks on Neural Network Policies with Visual Foresight. arXiv:1710.00814.

Lu, J.; Issaranon, T.; and Forsyth, D. 2017. SafetyNet: Detecting and Rejecting Adversarial Examples Robustly. arXiv:1704.00103.

Meng, D.; and Chen, H. 2017. MagNet: A Two-Pronged Defense against Adversarial Examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, 135–147. New York, NY, USA: Association for Computing Machinery. ISBN 9781450349468.

Metzen, J. H.; Genewein, T.; Fischer, V.; and Bischoff, B. 2017. On Detecting Adversarial Perturbations. arXiv:1702.04267.

Mnih, V.; Kavukcuoglu, K.; and Davi. 2013. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602.

Papernot, N.; McDaniel, P.; Wu, X.; Jha, S.; and Swami, A. 2016. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, 582–597.

Schölkopf, B.; Williamson, R. C.; Smola, A.; Shawe-Taylor, J.; and Platt, J. 2000. Support Vector Method for Novelty Detection. In Solla, S.; Leen, T.; and Müller, K., eds., *Advances in Neural Information Processing Systems*, volume 12. MIT Press.

Sethi, T. S.; and Kantardzic, M. 2018. Data driven exploratory attacks on black box classifiers in adversarial domains. *Neurocomputing*, 289: 129–143.

Song, Y.; Kim, T.; Nowozin, S.; Ermon, S.; and Kushman, N. 2017. PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples. *CoRR*, abs/1710.10766.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. arXiv:1312.6199.

Tramèr, F.; Kurakin, A.; Papernot, N.; Goodfellow, I.; Boneh, D.; and McDaniel, P. 2020. Ensemble Adversarial Training: Attacks and Defenses. arXiv:1705.07204.

van den Oord, A.; Kalchbrenner, N.; and Kavukcuoglu, K. 2016. Pixel Recurrent Neural Networks. *CoRR*, abs/1601.06759.

Wu, Y.; Bamman, D.; and Russell, S. 2017. Adversarial training for relation extraction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 1778–1783.

Yuan, X.; He, P.; Zhu, Q.; and Li, X. 2019. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9): 2805–2824.

Zhao, P.; and Lai, L. 2020. Analysis of KNN Density Estimation. arXiv:2010.00438.