



Ben-Gurion University of the Negev

The Faculty of Engineering Sciences

The Department of Software and Information Systems Engineering

Enhancing the Robustness of Deep Reinforcement Learning Models Using Prediction Intervals

Thesis submitted in partial fulfillment of the requirements
for the Master of Sciences degree

Amir Loewenthal

Under the supervision of **Dr. Gilad Katz** , **Prof. Asaf Shabtai**

September 2022



Ben-Gurion University of the Negev

The Faculty of Engineering Sciences


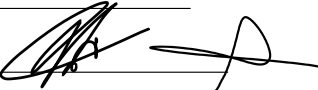

The Department of Software and Information Systems Engineering

Enhancing the Robustness of Deep Reinforcement Learning Models Using Prediction Intervals

Thesis submitted in partial fulfillment of the requirements
for the Master of Sciences degree

Amir Loewenthal

Under the supervision of **Dr. Gilad Katz , Prof. Asaf Shabtai**

Signature of student:  Date: 01.10.22
Signature of supervisor:  Date: 2/10/22
Signature of chairperson of the
committee for graduate studies:  Date: 17/10/2022

September 2022

Enhancing the Robustness of Deep Reinforcement Learning Models Using Prediction Intervals

Amir Loewenthal

Master of Sciences Thesis

Ben-Gurion University of the Negev

2022

Abstract

Deep learning and deep reinforcement learning (DRL) algorithms have become state-of-the-art solutions in multiple domains. However, these algorithms are also highly vulnerable to noise. For this reason, improving the robustness of DRL algorithms in noisy environments is highly consequential in a multitude of real-world problems. In this study, we propose a novel DRL approach that builds upon the recent advances in the use of prediction intervals (PIs) to improve the robustness of neural networks in regression tasks. We present an approach for integrating PIs in DRL architectures and develop an approach for utilizing the PIs to identify noisy environments and re-calibrate the agent's policies accordingly. Our evaluation demonstrates

that our approach significantly improves the performance of the DRL agent in highly-noisy environments.

Contents

List of Figures	v
List of Tables	vii
1 Introduction	1
2 Related Work	4
2.1 Adversarial Attacks on DRL	4
2.2 Robustness of DRL agents	6
2.3 Prediction Intervals	8
2.4 Deep Q-Networks (DQN)	9
3 Research Goals	10
3.1 Hypothesis	10
3.2 Research Questions	10
4 Method	12

4.1	The IPIV-DQN architecture	12
4.2	IPIV-DQN 's Reward Function	14
4.2.1	The PI Loss	14
4.2.2	The DQN Reward Function	15
4.3	Prediction Intervals-Based Action Selection	17
5	Experimental Results	21
5.1	Datasets and Evaluation Metrics	21
5.2	Baselines	22
5.3	Experimental Setup	22
5.4	Results	25
6	Discussion and Conclusions	28
6.1	Analyzing IPIV-DQN 's ability to detect noise	28
6.2	Analyzing PI size as a function of time and noise level	30
6.3	Conclusions	30
	Bibliography	32

List of Figures

4.1	An illustration of IPIV-DQN architecture. The agent receives a state input from the environment, and this input can be perturbed by noise. After that, the agent calculates for each action a specific Q-value and a prediction interval for the Q-value. In response to these prediction intervals, the agent decides on an action and updates the environment.	13
5.1	Bar plot of the DQN agents' reward with different noise magnitudes.	24
5.2	Performance percentage of our variants in comparison to a Vanilla-DQN agent. Our policies outperform the Vanilla-DQN by 5-20 percents depends on the noise magnitude.	24
5.3	A comparison between an IPIV-DQN agent with the backup method to an IPIV-DQN agent with the rolling average method on an example trajectory with a noise magnitude of 0.2. The rolling average agent achieves a reward of 210 while the backup agent achieves a reward of 240.	27

5.4	A comparison between an IPIV-DQN agent with the backup method to an IPIV-DQN agent with the rolling average method on an example trajectory with a noise magnitude of 0.3. The rolling average agent achieves a reward of 142 while the backup agent achieves a reward of 180.	27
-----	--	----

List of Tables

5.1	The hyperparameters used in our experiments. We use the PastAvgWindow parameter to specify the window size for calculating the Q values using the rolling average method. EpisLimit indicates how many consecutive steps our PIS is above the AIS when using the backup method. AIS is the acceptable interval size on which we decide to use our extended methods.	23
5.2	Average rewards pm standard deviation over ten episodes on two baselines and IPIV-DQN. We report the rewards with different magnitudes of noise. In addition to the three action selection methods, we present the results of IPIV-DQN without the consideration of the AIS, these results are under No variants. Noise magnitudes on which our method outperforms the baselines are in bold.	25

5.3	An analysis of IPIV-DQN 's ability to detect noisy samples, and of the manner by which it responds to them. The 'Traj. The noise detection' column refers to the percentage of trajectories (games) in which noise was detected. The '# Steps until AIS' refers to the average number of steps prior to the PIs exceeding the AIS in size. '% Total noise detection' refers to the percentage of time steps in which backup/rolling average were used.	26
-----	--	----

Chapter 1

Introduction

Deep learning (DL) algorithms have achieved state-of-the-art (SOTA) results in multiple domains in recent years, including image recognition [1], natural language processing [2], and generative models [3]. Despite their popularity and high performance, DL algorithms are particularly sensitive to both adversarial attacks and noisy data [4]. These problems also apply to the field of deep reinforcement learning (DRL), which consists of deep architectures that are often applied to complex, multi-step problems such as video games [5–7], robotics, [8, 9], and autonomous driving [5] .

While multiple studies aim to improve the robustness of DRL approaches to adversarial attacks [10–13], less attention has been given to dealing with noisy data. Noisy data, which is very common in real-world scenarios, is different from adversarial samples in that it is easily detectable but still highly disruptive. Existing approaches to handle noisy data using a mathematical approach, either by calculating bounds of the prediction with noise [14] or

by considering noise within the basis of the algorithm [15, 16].

In this study, we propose IPIV-DQN , a novel approach for addressing the challenge posed by noisy data in DRL algorithms. We follow the intuitive notion that noisy data increases the DRL model’s uncertainty in its actions, and therefore seek to build upon recent advances in the field of uncertainty modeling in neural networks, specifically prediction intervals (PIs). While they are derived in various ways, PIs produce not only a predicted value but also an upper and lower bound. The size of the intervals between the bounds measures the model’s level of uncertainty, so noisy data is therefore likely to result in larger intervals. In this sense, PIs enable us both to detect and address noisy data.

We build upon the recent work of [17], which proposed IPIV—an elegant method for jointly producing both the upper and lower bounds of the confidence interval, as well as an exact prediction within the interval. The approach proposed in this study differs from previous approaches which provide upper and lower bounds accurately but have difficulties in providing specific value prediction [18, 19]. Another advantage of IPIV in a DRL setting is its modularity, which means that it can be easily implemented on top of any DRL algorithm.

Our approach is the first to integrate PI-based solutions in a DRL setup. Instead of using a single PI for the entire method, we create a PI for each individual action. This setup enables us to not only produce values that predict each action’s contribution but also analyze the size of the PI and determine the model’s confidence in the prediction. Using this information,

we are able to develop strategies that enable IPIV-DQN to mitigate the effects of noisy data.

The contributions of our proposed approach are as follows:

- We present IPIV-DQN , a novel approach that integrates prediction intervals into standard DRL architectures. As shown in our evaluation, IPIV-DQN is highly effective in reducing the negative impacts of noise on the performance of DRL algorithms.
- We demonstrate that our approach can also be used not only to detect or mitigate the effects of noise, but also to enable the DRL algorithm to recognize whether its decision-making process is impacted by the noise, and to what degree.

Chapter 2

Related Work

2.1 Adversarial Attacks on DRL

In recent years many papers presented adversarial attack mechanisms against DRL methods. The main goal of those attacks was to perturb some of the agent's states.

Huang et al[20] showed how existing adversarial example crafting techniques can be used to degrade the test-time performance of trained policies, by attacking with adversarial examples at every timestamp of the episode. The adversarial examples were crafted with the FGSM[21] method. The effectiveness of the attack is compared between three different deep RL algorithms: DQN, TRPO, and A3C.

Later, many papers try to reduce the number of steps in an episode that need to be attacked while keeping the attack result to improve the efficiency of such attacks.

Lin et.al [22] offered the Strategically timed attack. The attack is performed in specific timesteps on which one action is highly preferred over the other actions. In this paper, Lin et.al also presented the Enchanting attack which aims to lure the agent into a specific state. This is done by splitting into two subtasks, the first is planning a sequence of actions that will make the agent arrive at the target state. and second is craft adversarial examples so that the agent will take the planned actions. Both attacks were tested on DQN and A3C algorithms.

Sun et.al [23] presents two adversarial attack techniques to stealthily and efficiently attack the DRL agents. The first attack is called Critical Point Attack, at this method the adversary first samples observations from the target environment and builds a prediction model to predict the following state transitions of the target, then the adversary assesses the damage of each attack strategy and chooses the optimal one. The second attack is called antagonist attack, this attack is domain free and can be applied to different tasks. In the antagonist attack, the adversary trains DNN with rewards that are negative to the agent’s rewards, and the antagonist attack on the states is determined by the trained policy. Both attacks use the C & W method [24] to generate adversarial perturbations. Kos et.al [13] offered a way to reduce the number of injections that are needed to successfully attack the agent based on its value function, the perturbation occurs only when the value of the current frame is above a chosen threshold.

While all the attacks described above are applied at testing time, adversarial attacks can be applied at training time as well, such attack is presented at

[25]. The attack is applied to the DQN algorithm by training an adversary DQN model that aims to minimize the agent’s reward. The attack success reduces the agent’s reward on a game learning DQN.

Adversarial attacks are extreme cases of the noises. Therefore such methods were considered during the experimental phase of this thesis. A future work of this thesis will include experiments against adversarial examples.

2.2 Robustness of DRL agents

A known method for improving the robustness of DRL agents is adversarial training. The authors of [13] proposed adversarial training as a method to improve DRL robustness. By training their A3C agent in environments where they injected random noise and FGSM perturbations, they were able to the resilience their DRL agent to those kinds of perturbations. The approach proposed in [12] also used adversarial training to improve the robustness of the DDQN and DDPG model. However, in this work, the authors used samples that were generated by gradient-based attacks.

The approach presented in [11] offers a more efficient way of adversarial training called adversarial guided exploration (AGE). This technique is based on an epsilon greedy algorithm and was tested on DQN trained for a cartpole environment. [26] proposed the ARPL algorithm, which involves the use of adversarial examples during training to improve the robustness of policy learning. The algorithm was tested on a TRPO system.

Instead of training the neural network on adversarial examples directly, the

work presented in [10] used an additional policy network. The algorithm, called "RS-DQN", splits the standard DQN architecture into a policy network S and an additional network Q . The Q network is trained as a regular DQN network, and the algorithm uses policy distillation [27] to train the policy network S from Q while using adversarial examples in S 's network training. The agent then takes actions based on Q and uses S for exploration. The algorithm was implemented and compared to many DQN variations on five different Atari games, and outperformed the regular DQN networks under adversarial attacks scenarios.

Another approach for improving the robustness of DRL algorithms focuses on the algorithms' action selection methods. Methods of this kind attempt to modify the Q -function (the ranking of the actions based on their expected rewards) or the reward function. These modifications are designed to better handle noise in the data or adversarial attacks. In [14], the authors proposed calculating a lower bound to the state action value and then choosing the optimal action on that bound. This method was integrated into an existing DQN algorithm and was tested on collision avoidance scenarios.

In [15, 16] the authors explore methods for improving the robustness of DRL agents against adversarial perturbations over the observation space. These works propose a state-adversarial Markov decision process (SA-MDP), a theoretical approach for improving the robustness of DRL agents. Due to the addition of noise to the MDP's definition, they were able to modify the Bellman equations and the loss function of DRL agents to be more robust against adversarial attacks. The evaluation consisted of a variety of DRL

algorithms, where the algorithm they used for DQN—SA-DQN—is used in the evaluation of this study.

2.3 Prediction Intervals

Prediction intervals (PIs) are a method that quantifies the uncertainty within regression problems. By using PI-based approaches, regression models generate prediction intervals for each sample. Multiple studies have proposed ways for the generation of PIs, either by post-processing approaches [28] or by adding terms to the loss function of the model and generating the prediction intervals during training [18, 29].

In order to implement PIs in RL methods, we would like to generate both the PI and the specific value within it. While there are multiple approaches for achieving this goal [30–32], the large majority of which return the middle of the PI as the value. Because we consider flexibility in scoring to be essential for a DRL algorithm, we chose to implement our solution using Integrated Prediction Intervals and specific Value predictions (IPIV) [17].

Contrary to other PI methods, IPIV does not increase the complexity of the model. Another advantage of the IPIV architecture is its ability to be implemented on top of any trained neural networks. This is an important advantage, as we can apply the IPIV algorithm on top of existing DQN networks.

2.4 Deep Q-Networks (DQN)

Reinforcement learning algorithms aim to solve the problem of an agent that interacts with an environment over time. The environment is represented by states, within those states there are initial states and terminal states. An episode is a sequence of steps that starts with an initial state and ends with a terminal state. At time step t , the agent receives as an input a state s_t from the environment and needs to select an action a_t . Then, based on the agent's action, the environment transitions to a new state s_{t+1} and the agent receives a reward r_t for its selected action. The reward of an episode is defined as the sum of rewards for each step in the episode. The goal of the agent is to learn a policy $\pi(a_t|s_t)$, a function that determines what action to perform, given a state, that maximizes the expected cumulative reward of an episode.

One of the most common algorithms in the field of RL is Q-learning. Q-learning learns the Q-function $Q(s, a)$, which measures the overall expected reward, under the assumption that an agent is currently in state s and performs an action a . Using the predictions of the Q-function, the agent then selects an action based on the maximal Q-value in a given state. Deep RL (DRL) uses deep neural networks as an approximator for the learning part of the reinforcement learning agents. Deep Q Network (DQN) [5] is a DRL algorithm that learns the Q function with weights θ_Q of a neural network $Q(s, a; \theta_Q)$. Given a state s the network will approximate the Q value for each action a in discrete action space. The chosen action by the agent's policy will be the action with the maximal Q-value based on the network output.

Chapter 3

Research Goals

The main goal of this thesis is to improve the robustness of existing DRL methods against noisy state-space observations. We propose the use of prediction intervals in the DRL domain as a method to improve robustness.

3.1 Hypothesis

We hypothesize that the use of prediction intervals estimation within the architecture of neural networks can make them more robust to noise by (1) Detecting the noise, and; (2) Take action based on the noisy environment without harming the performance of the network.

3.2 Research Questions

This thesis will consider the following research questions

- What is the best way to integrate the prediction intervals into the architecture
- Should they prove effective, should the prediction intervals prioritize inclusion (i.e., PICP) or accuracy (MPIW and RMSE)
- How to balance our desire for selecting top-performing actions (i.e., high Q-values) and "safe" actions (i.e., small prediction intervals)

The questions will be answered by introducing IPIV-DQN , a novel architecture which combines IPIV, a PI approach for regression problems with a DRL agent. For a given state S_0 , the architecture will compute a Prediction Interval for every possible state-action value. Then, the agent will choose the top-performing action based on both the value prediction and the PI of each action.

Chapter 4

Method

4.1 The IPIV-DQN architecture

The proposed architecture is presented in Figure 4.1. Following the terminology used in [17], IPIV-DQN consists of three components: an *input*, a *backbone*, and *PI heads*. The input is the representation of the current state, while the backbone can be any deep learning-based implementation of a DRL agent—both on or off-policy. This versatility is an important advantage of our proposed approach. In our experiments, we use the popular Double deep Q network (DDQN) architecture [5] as a backbone.

The PI heads output the score assigned to each action (in our case, the Q-values), as well as its prediction interval. Each PI head models a single action, and the representation of action a_i consists of three values (i.e., outputs of the PI head): the upper bound U_i , the lower bound L_i , and the auxiliary head v_i . The auxiliary head produces the actual Q-value of a_i using the

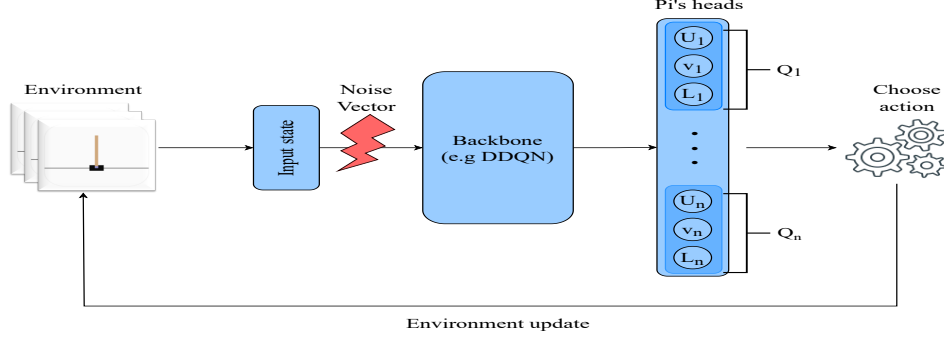


Figure 4.1: An illustration of IPIV-DQN architecture. The agent receives a state input from the environment, and this input can be perturbed by noise. After that, the agent calculates for each action a specific Q-value and a prediction interval for the Q-value. In response to these prediction intervals, the agent decides on an action and updates the environment.

following equation:

$$Q_i = v_i \cdot U_i + (1 - v_i) \cdot L_i \quad (4.1)$$

where v_i is within the range $[0,1]$. This representation has two advantages. First, the actual Q-value can be positioned anywhere within the PI, while at the same time never producing values that are outside of it. Secondly, by expressing v_i as a function of U_i and L_i , we ensure that the three outputs are jointly trained and coordinated in their output¹.

When IPIV-DQN does not detect any problems (i.e., performance-degrading levels of noise) in the input, the selection of the actions at each time step is determined solely by Q_i . The backpropagation process also follows the standard convention, where only the PI heads of the chosen actions are being

¹For a full analysis of the advantages of this architecture we refer the reader to [17]

updated. In the next section, we describe how we integrate the loss function of our PI heads into the DRL agent’s reward function.

4.2 IPIV-DQN ’s Reward Function

4.2.1 The PI Loss

An effective prediction interval needs to satisfy two conflicting goals. First, it needs to contain as many samples as possible within it. This goal is measured by a metric called *prediction interval coverage probability* (PICP), and its value is the percentage of samples that fall within the interval:

$$PICP = \frac{1}{n} \sum_{i=1}^n k_i \quad (4.2)$$

where $k_i = 1$ if $y_i \in (L_i, U_i)$, and $k_i = 0$ otherwise. The second goal is to generate a bound that is as tight (i.e., small) as possible. This goal is measured by a metric called *Mean prediction interval width* (MPIW), and it is calculated by averaging the size of the bound produced for all samples falling within their respective interval (denoted by c):

$$MPIW_{capt_a} = \frac{1}{c} \sum_{i=1}^n (U_{i_a} - L_{i_a}) * k_i \quad (4.3)$$

By combining the two metrics presented in Equations 4.2 and 4.3, we create

the PI-loss (note that this loss is calculated individually for each PI head):

$$L_{PI_a} = MPIW_{capt_a, \theta} + \sqrt{n} * \lambda \Psi(1 - \alpha - PICP_{\theta}) \quad (4.4)$$

where Θ denotes the network weights, Ψ is a quadratic penalty function, n is the batch size and λ is a hyperparameter controlling the relative importance of width vs. coverage. The parameter α is user-defined and is used to determine the desired level of coverage (i.e., PICP). Based on this parameter, the architecture attempts to optimize the MPIW metric while staying as close as possible to the specified PICP Value. Next, we demonstrate how we integrate the PI loss into the DRL agent's reward function.

4.2.2 The DQN Reward Function

The loss function used to update the parameters of the standard DQN algorithm is defined as follows (see [5]):

$$MSE = (r + \gamma_a^{max} Q(s', a, \theta) - Q(s, a, \theta))^2 \quad (4.5)$$

where s, s' are the current and the next state respectively, and r is the reward derived from the current action. We denote this loss as l_{DQN} and rewrite it in a more succinct form as

$$l_{DQN}(y_i, Q_i) = (y_i - Q_i)^2 \quad (4.6)$$

where y_i is the updated Q-value of each action. In Equation 4.1 we showed

that Q_i can be expressed as a function of the outputs of our PI head. We, therefore, rewrite l_{DQN} as

$$L_{v_a} = \frac{1}{n} \sum_{i=1}^n l_{DQN}(y_{i_a}, v_{i_a} * U_{i_a} + (1 - v_{i_a}) * L_{i_a}) \quad (4.7)$$

where n is the number of samples, and a is the action selected by the model. Note that the loss for each action is calculated separately. Finally, we integrate the two functions—those of the DQN and the PI—as follows:

$$L_{IPIV-DQN} = \sum_{\forall a \in A} \text{sign}(L_{v_a}) * (\beta * L_{PI_a} + (1 - \beta) * L_{v_a}) \quad (4.8)$$

Equation 4.8 binds all the previous equations together and describes the complete IPIV-DQN reward function. The equation generates a weighted sum of the PI loss and the DQN loss. The weight of each component is determined by the hyperparameter β . Note that this update will only be performed for the selected action (i.e., to its corresponding PI head) and backbone architecture.

In conclusion, our model is trained in a similar way to any other DQN model. The novelty of our approach lies in the output layer, where instead of single Q-value output for each possible action, the model produces a PI. The reward function focuses both on optimizing the Q-values for each action in a given state, while also keeping the PI of each action as small as possible.

4.3 Prediction Intervals-Based Action Selection

Our prediction intervals are designed to achieve two goals. The first goal is *the detection of noise and/or perturbations in the input*. More specifically, our goal is not the identification of all forms of noise, but those cases that negatively affect our model’s decision-making ability. The second goal is *the creation of methods that utilize our PIs for more robust decision-making*.

The first step in the application of our approach is the setting up of the *acceptable interval size* (AIS) for the analyzed dataset. We set the value of the AIS after the training of our approach is complete, and set its value to be the average interval size for all possible actions, when the trained model is applied to noiseless data. We find that this setup is both effective and easy to compute (because we can use the interval sizes of the last X batches that were used in the training of our model).

Once the AIS is set, for each time step there can be two possible use-cases. *In the first use case*, the PIs of all actions are within the AIS. In this is the case, IPIV-DQN simply selects the action for the current time step based solely on the actions’ Q-values. More formally, the selection of the action is performed by solving

$$a = \operatorname{argmax}_i (v_i * U_i + (1 - v_i) * L_i) \quad (4.9)$$

The rationale of this use case is straightforward: our model did not detect

any unusual patterns in the data, and can therefore select the action whose Q-value is highest.

In the second use case, the size of at least one of our action PIs is larger than the AIS. This serves as an indication that the current input is sufficiently noisy to degrade our model’s decision-making abilities. In these circumstances, we propose three variants of our approach (all evaluated in Section 5):

1) IPIV-DQN with backup - In this setup we continue to use our approach after the detection of noise (i.e., Equation 4.9), relying on the PIs to provide stability in a noisy environment. We hypothesize that riskier actions—actions for whose outcomes our model has less information—would have larger PIs, and as a result lower Q-values.

While our evaluation supported our hypothesis, we found that long sequences of a noisy input can eventually degrade the performance of our model compared to the standard DQN-based DRL agent. The reason for this relatively reduced performance is as follows: after multiple noisy inputs, the DRL agent is further and further away from its desired position. In such cases, sharp changes of course are often needed. Our analysis found that the PIs sometimes delay the making of these sharp changes because our model does not use the upper bound of the interval (i.e., the most optimistic outcome of the action), but rather a value within it.

To address the need for flexibility while also maintaining the benefits of our PIs, we implemented the following approach: if the size of at least one PI is greater than the AIS for a *consecutive predefined number of time steps*,

we “turn off” our PIs and use the standard DQN mechanism to select the actions. Once there is even a single time step where all PIs are smaller than the AIS, we reset the counter and return to using our PIs. This setup enables our approach to alternate between making more robust, PI-supported, decisions, and taking risks when it recognizes its situation is more problematic and requires a sharp change of course.

2) IPIV-DQN with rolling average - While our PIs can improve the DRL agent’s decision-making process, there is always the risk that the noise in a given time step will distort the input in a way that will derail the PIs. To reduce the negative effect of these problematic inputs, we apply a *rolling average*: in every time step where the size of the PIs exceeds the AIS, the Q-value of action i at time step t will be determined using the following formula:

$$a_i^t = \operatorname{argmax}_i (0.5 * \frac{U_i^{t-1} + L_i^{t-1}}{2} + 0.5 * Q_i^t) \quad (4.10)$$

The rationale of the proposed approach is that the Q-value of actions seldom changes drastically in consecutive time steps. Therefore, a rolling average that takes into account the PIs of actions in previous time steps can help stabilize the behavior of our DRL agent.

3) IPIV-DQN with rolling average + backup - This approach combines the two presented above: once a PI becomes larger than the AIS, we use Equation 4.10 to calculate the Q-value of each action. If the situation persists for a predefined number of time steps (the same number as in the rolling

average setup), we turn off our PIs. We hypothesize that combining these two approaches will further stabilize IPIV-DQN 's performance.

Chapter 5

Experimental Results

5.1 Datasets and Evaluation Metrics

We evaluate IPIV-DQN on the classic CartPole control problem, using the implementation openAI Gym [33] implementation. In this environment, the agent’s goal is to maintain a balanced pole upright ($\pm 12^\circ$) on a horizontally moving cart. The state vector is 4-dimensional, and contains the cart’s position, cart velocity, pole angle, and pole angular velocity. The action space of the agent contains two actions: $\mathcal{A} = \{\text{move_cart_left}, \text{move_cart_right}\}$. Additional information can be found in openAI Gym GitHub¹.

The evaluation metric of the cartpole problem is simple: for every time-step where the pole remains upright, the agent receives a 1-point reward, with a maximal reward of 500. When the pole falls down the game terminates, but no negative rewards are incurred.

¹<https://github.com/openai/gym>

5.2 Baselines

We compare IPIV-DQN to the following two baselines:

- **The standard DQN algorithm** This algorithm is a natural baseline, because our proposed approach is implemented on top of it. We implemented this baseline using the Keras-rl library for python [34].
- **SA-DQN [15]** A recent work that achieved state-of-the-art results for several RL algorithms in general, and DQN in particular.

5.3 Experimental Setup

- We evaluate our baselines on five different magnitudes of noise to the input state of the agent. The noise is injected into the input in the following manner: given a state s_i in time step i , the noisy state s_{i-pert} is chosen uniformly from the space that is far from s_i by σ .

$$s_{i-pert} \sim UNIF(s_i - \sigma, s_i + \sigma) \quad (5.1)$$

- We evaluate our approach in six different noise configuration. The amount of noise we inject into the input is defined by $\sigma \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$. A value of $\sigma = 0.1$, for example, indicates that 10% of the input is randomly replaced by noise.
- In all our experiments, the first 50 time-steps are a warm-up period which is noise-free. This setup is designed to enable the models to

Table 5.1: The hyperparameters used in our experiments. We use the *PastAvgWindow* parameter to specify the window size for calculating the Q values using the rolling average method. *EpsLimit* indicates how many consecutive steps our PIS is above the AIS when using the backup method. AIS is the acceptable interval size on which we decide to use our extended methods.

Name	Type (model / eval)	Value
α	model	0.15
β	model	0.9
λ	model	15
Learning rate	model	$1e^{-4}$
γ	model	0.99
<i>PastAvgWindow</i>	eval	30 Eps
<i>AIS</i>	eval	40
<i>EpsLimit</i>	eval	3 Eps

reach a large range of states prior to the introduction of noise. Once we begin injecting the noise, however, we do so at every time step (i.e., the evaluated algorithms do not have time to “recover”).

- For each noise level, we ran 10 experiments, each with a different seed for the random noise. All results reported in this study are the average performance of all runs.
- All experiments were run on a private cluster using NVIDIA GeForce GTX 1080 Ti GPU. We used the weights and biases platform [35] to manage and monitor the experiments.
- All the hyper-parameters used in our experiments are reported in Table 5.1.

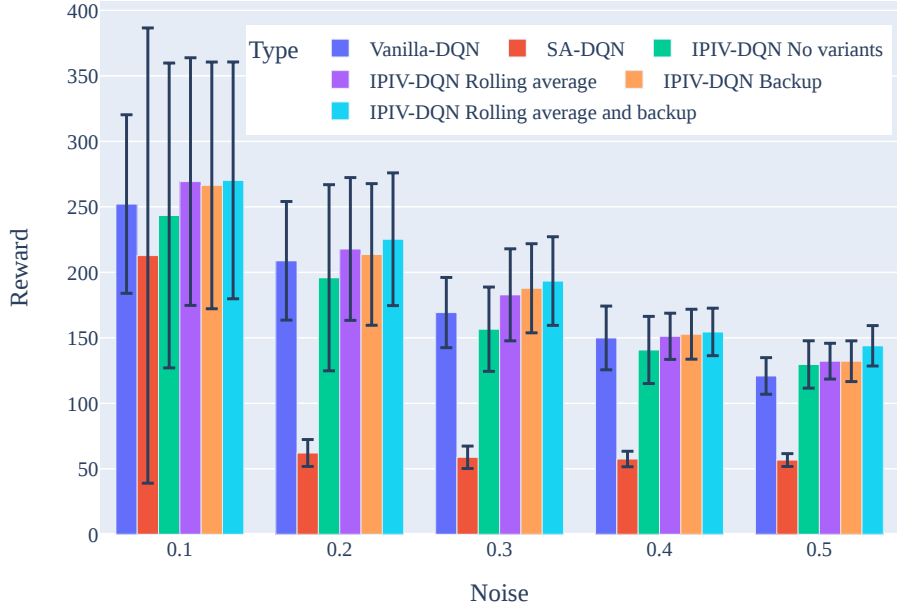


Figure 5.1: Bar plot of the DQN agents' reward with different noise magnitudes.

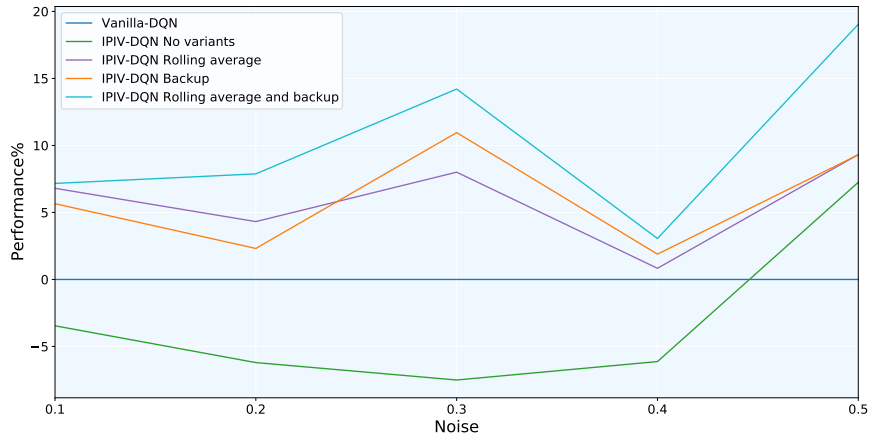


Figure 5.2: Performance percentage of our variants in comparison to a Vanilla-DQN agent. Our policies outperform the Vanilla-DQN by 5-20 percents depends on the noise magnitude.

Table 5.2: Average rewards pm standard deviation over ten episodes on two baselines and IPIV-DQN. We report the rewards with different magnitudes of noise. In addition to the three action selection methods, we present the results of IPIV-DQN without the consideration of the AIS, these results are under No variants. Noise magnitudes on which our method outperforms the baselines are in bold.

Method	Noise reward					
	0.0	0.1	0.2	0.3	0.4	0.5
Vanilla-DQN	343.93 \pm 43.54	252.12 \pm 68.16	208.8 \pm 45.27	169.27 \pm 26.81	149.9 \pm 24.33	120.88 \pm 13.99
SA-DQN	500 \pm 0	212.78 \pm 173.78	62.06 \pm 10.28	58.78 \pm 8.57	57.47 \pm 5.92	56.67 \pm 4.91
IPIV-DQN No variants	268.01 \pm 87.82	243.4 \pm 116.38	195.85 \pm 71.06	156.57 \pm 32.19	140.72 \pm 25.65	129.65 \pm 18.08
IPIV-DQN Rolling average	289.91 \pm 77.74	269.27 \pm 94.54	217.82 \pm 54.51	182.82 \pm 35.13	151.15 \pm 17.65	132.15 \pm 13.67
IPIV-DQN Backup	288.98 \pm 76.59	266.37 \pm 94.19	213.62 \pm 54.07	187.82 \pm 34.02	152.73 \pm 19.07	132.12 \pm 15.56
IPIV-DQN Rolling average and backup	288.98 \pm 76.59	270.18 \pm 90.41	225.25 \pm 50.67	193.32 \pm 33.84	154.48 \pm 18.13	143.9 \pm 15.43

5.4 Results

The results of our evaluation are presented in Table 5.2 and Figures 5.1,5.2. It is clear that while both the standard DQN and SA-DQN baselines outperform IPIV-DQN in a no-noise setup, the introduction of even limited amounts of noise makes significantly reduces the performance of these baselines. The top-performing variant of our approach—IPIV-DQN with rolling average and backup—outperforms the standard DQN algorithm by 7% in the 10% noise setup, with the improvement reaching 19% in the 50% noise setup.

Analysis of the different variants of our approach shows that both the use of backup and rolling averages to the performance of our approach. Both variants consistently outperform the standard DQN algorithm, and their performance is very close to each other both in terms of the rewards they obtain and in their score distributions (as evident by the standard deviation values, presented in Table 5.2).

An interesting question raised by the results concerns the relatively low performance of our approach in a no-noise setup. Based on our analysis, we

Table 5.3: An analysis of IPIV-DQN’s ability to detect noisy samples, and of the manner by which it responds to them. The ‘Traj. The noise detection’ column refers to the percentage of trajectories (games) in which noise was detected. The ‘# Steps until AIS’ refers to the average number of steps prior to the PIs exceeding the AIS in size. ‘% Total noise detection’ refers to the percentage of time steps in which backup/rolling average were used.

Noise	Traj. noise detection %	# Steps until AIS	% Total noise detection
0.1	67%	108.1 ± 94.62	$14\% \pm 14.33$
0.2	82%	78.44 ± 85.37	$16\% \pm 13.97$
0.3	90%	29.05 ± 40.93	$22\% \pm 19.97$
0.4	98%	16.3 ± 28.64	$24\% \pm 21.04$
0.5	100%	7.71 ± 11.33	$31\% \pm 20.69$

conclude that the reduced performance stems from our approach’s attempt to optimize two goals: in addition to maximizing the rewards obtained in the game, IPIV-DQN aims to optimize the size of the PIs. This additional constraint seems to limit our ability to perform well in some edge cases, which leads to reduced performance. It should be noted that *a simple solution to this problem would be to maintain two models*: a standard DQN model that will operate when no noise is detected, and an IPIV-DQN model that will act both as a detector of noise and a decision-maker when needed.

Finally, it is clear that despite achieving the highest results in the no-noise setup, SA-DQN performs worst in all other setups. This was unexpected, as this approach was specifically designed to operate in noisy and adversarial domains. The reason for the approach’s reduced performance in our setup is that it is designed to counter *deterministic noise*, meaning that it expects specific samples to be perturbed in the same way each time they are encountered. This assumption, which doesn’t hold in our setup, proves to be detrimental to SA-DQN.

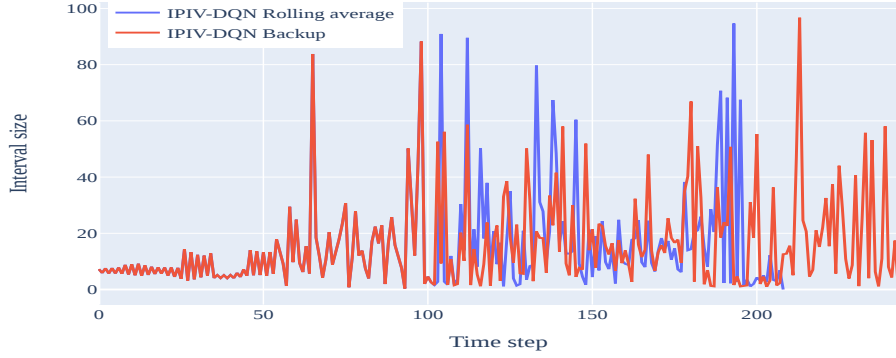


Figure 5.3: A comparison between an IPIV-DQN agent with the backup method to an IPIV-DQN agent with the rolling average method on an example trajectory with a noise magnitude of 0.2. The rolling average agent achieves a reward of 210 while the backup agent achieves a reward of 240.

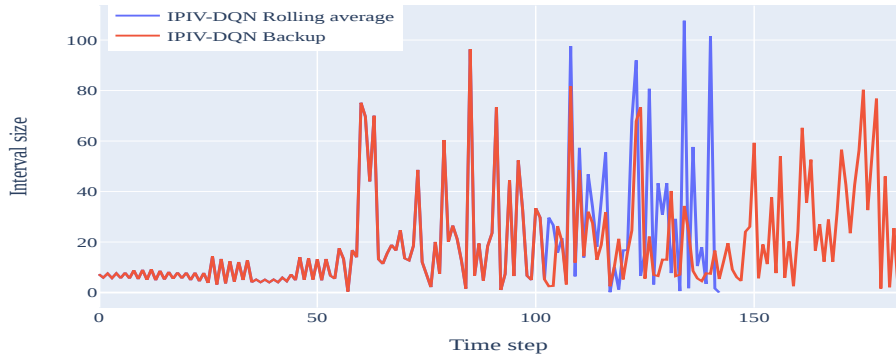


Figure 5.4: A comparison between an IPIV-DQN agent with the backup method to an IPIV-DQN agent with the rolling average method on an example trajectory with a noise magnitude of 0.3. The rolling average agent achieves a reward of 142 while the backup agent achieves a reward of 180.

Chapter 6

Discussion and Conclusions

In this section, we analyze aspects of our approach that are relevant to its ability to detect and effectively respond to noisy data.

6.1 Analyzing IPIV-DQN 's ability to detect noise

Our goal is to analyze IPIV-DQN 's ability to identify noise in its input. This ability is important in itself since high accuracy in this task can make our approach. It is important to note that IPIV-DQN does not define noise objectively (that is, whether the data contains noise or not). Instead, our approach's definition of noise is a function of its effect on the DRL agent's ability to have high degrees of certainty regarding the PIs of actions. For this reason, we expect IPIV-DQN to identify a smaller number of (its definition of) noisy instances in the use-cases we inject lower levels of noise into each

sample (keep in mind that noise is randomly injected to *every* sample after $t = 50$).

The results of our analysis are presented in Table 5.3. As expected, the percentage of trajectories (i.e., games) in which IPIV-DQN identifies noise is correlative with the level of injected noise. However, our analysis found two interesting aspects of the way our approach deals with noise. The first aspect is with regard to the number of time steps that pass before noise is detected (i.e., the size of the PI exceeds the AIS). For the 10% noise setup, this number is 108, which means that our PIs remain immune to the injected noise for a relatively long amount of time (the randomness of the noise, however, results in a high standard deviation for this statistic). For the 50% noise setup, however, the average number of steps is only 7.7, meaning that PIs exceed the AIS almost immediately.

The second interesting aspect of our analysis is the percentage of time steps in which our approach used the proposed techniques to overcome the noise, i.e., the backup and the rolling average. Even in the noisiest setup—50%—these techniques were only used on average in a third of the time steps. These results provide another indication of the robustness of our proposed approach because our proposed PIs proved sufficient on their own in more than two-thirds of the analyzed trajectories.

6.2 Analyzing PI size as a function of time and noise level

As shown in Figures 5.1 the backup and rolling average variants of our approach have comparable performance. Combining the two variants, however, yields additional improvement. We analyzed the behavior of the two variants and found that while the performance of the two variants is similar, each stabilizes the PI heads at different points throughout the trajectories.

We demonstrate our observation using two examples: for the 20% and 30% random noise setups, we identify a trajectory for which the two variants—backup and rolling averages—achieved similar results. These two trajectories are presented in Figures 5.3 and 5.4. While initially, the two approaches are highly correlated, they later diverge in their performance. Moreover, it is clear that *a)* often one variant manages to keep the size of its intervals small while the other does not, and; *b)* each approach has rapid fluctuations in its interval size. Given the large differences in behavior, it is understandable how our full approach manages to leverage the strengths of both variants.

6.3 Conclusions

In this study, we present IPIV-DQN , a novel approach for the integration of prediction intervals into DRL algorithms. Our approach produces a PI for each action evaluated by the model, thus improving its ability to perform in noisy environments. Evaluation of the CartPole data demonstrates that

IPIV-DQN is highly effective in mitigating the effects of noise. Moreover, our approach enables users to determine the effects of noise on the performance of their DRL agent.

For future work, we intend to evaluate our approach to additional types of DRL algorithms (e.g., policy gradients). Additionally, we will explore ways to integrate our approach into the Transformer architecture, possibly by including the training of the PI heads as an auxiliary task.

Bibliography

- [1] Guo, Yanming, Liu, Yu, Oerlemans, Ard, Lao, Songyang, Wu, Song, and Lew, Michael S. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, 2016.
- [2] Otter, Daniel W, Medina, Julian R, and Kalita, Jugal K. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 32(2):604–624, 2020.
- [3] Salakhutdinov, Ruslan. Learning deep generative models. *Annual Review of Statistics and Its Application*, 2:361–385, 2015.
- [4] Yuan, Xiaoyong, He, Pan, Zhu, Qile, and Li, Xiaolin. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.
- [5] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin A. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL <http://arxiv.org/abs/1312.5602>.

- [6] Wu, Yuxin and Tian, Yuandong. Training agent for first-person shooter game with actor-critic curriculum learning, 2016.
- [7] Firoiu, Vlad, Whitney, William F, and Tenenbaum, Joshua B. Beating the world’s best at super smash bros. with deep reinforcement learning. *arXiv preprint arXiv:1702.06230*, 2017.
- [8] Gu, Shixiang, Holly, Ethan, Lillicrap, Timothy, and Levine, Sergey. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.
- [9] Fan, Tingxiang, Cheng, Xinjing, Pan, Jia, Long, Pinxin, Liu, Wenxi, Yang, Ruigang, and Manocha, Dinesh. Getting robots unfrozen and un-lost in dense pedestrian crowds. *IEEE Robotics and Automation Letters*, 4(2):1178–1185, 2019.
- [10] Fischer, Marc, Mirman, Matthew, Stalder, Steven, and Vechev, Martin T. Online robustness training for deep reinforcement learning. *CoRR*, abs/1911.00887, 2019. URL <http://arxiv.org/abs/1911.00887>.
- [11] Behzadan, Vahid and Hsu, William. Analysis and improvement of adversarial training in dqn agents with adversarially-guided exploration (age), 2019.
- [12] Pattanaik, Anay, Tang, Zhenyi, Liu, Shuijing, Bommannan, Gautham, and Chowdhary, Girish. Robust deep reinforcement learning with adversarial attacks, 2017.

- [13] Kos, Jernej and Song, Dawn. Delving into adversarial attacks on deep policies, 2017.
- [14] Lütjens, Björn, Everett, Michael, and How, Jonathan P. Certified adversarial robustness for deep reinforcement learning. In *Conference on Robot Learning*, pages 1328–1337. PMLR, 2020.
- [15] Zhang, Huan, Chen, Hongge, Xiao, Chaowei, Li, Bo, Liu, Mingyan, Boning, Duane, and Hsieh, Cho-Jui. Robust deep reinforcement learning against adversarial perturbations on state observations. *Advances in Neural Information Processing Systems*, 33:21024–21037, 2020.
- [16] Zhang, Huan, Chen, Hongge, Boning, Duane, and Hsieh, Cho-Jui. Robust reinforcement learning on state observations with learned optimal adversary. *arXiv preprint arXiv:2101.08452*, 2021.
- [17] Simhayev, Eli, Katz, Gilad, and Rokach, Lior. Integrated prediction intervals and specific value predictions for regression problems using neural networks. *Knowledge-Based Systems*, page 108685, 2022.
- [18] Pearce, Tim, Brintrup, Alexandra, Zaki, Mohamed, and Neely, Andy. High-quality prediction intervals for deep learning: A distribution-free, ensembled approach. In *International conference on machine learning*, pages 4075–4084. PMLR, 2018.
- [19] Tagasovska, Natasa and Lopez-Paz, David. Single-model uncertainties for deep learning. *Advances in Neural Information Processing Systems*, 32, 2019.

- [20] Huang, Sandy H., Papernot, Nicolas, Goodfellow, Ian J., Duan, Yan, and Abbeel, Pieter. Adversarial attacks on neural network policies. *CoRR*, abs/1702.02284, 2017. URL <http://arxiv.org/abs/1702.02284>.
- [21] Goodfellow, Ian J., Shlens, Jonathon, and Szegedy, Christian. Explaining and harnessing adversarial examples, 2015.
- [22] Lin, Yen-Chen, Hong, Zhang-Wei, Liao, Yuan-Hong, Shih, Meng-Li, Liu, Ming-Yu, and Sun, Min. Tactics of adversarial attack on deep reinforcement learning agents. *CoRR*, abs/1703.06748, 2017. URL <http://arxiv.org/abs/1703.06748>.
- [23] Sun, Jianwen, Zhang, Tianwei, Xie, Xiaofei, Ma, Lei, Zheng, Yan, Chen, Kangjie, and Liu, Yang. Stealthy and efficient adversarial attacks against deep reinforcement learning, 2020.
- [24] Carlini, Nicholas and Wagner, David A. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016. URL <http://arxiv.org/abs/1608.04644>.
- [25] Behzadan, Vahid and Munir, Arslan. Vulnerability of deep reinforcement learning to policy induction attacks. In Perner, Petra, editor, *Machine Learning and Data Mining in Pattern Recognition*, pages 262–275, Cham, 2017. Springer International Publishing. ISBN 978-3-319-62416-7.
- [26] Mandlekar, A., Zhu, Y., Garg, A., Fei-Fei, L., and Savarese, S. Adversarially robust policy learning: Active construction of physically-

- plausible perturbations. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3932–3939, 2017. doi: 10.1109/IROS.2017.8206245.
- [27] Rusu, Andrei A., Colmenarejo, Sergio Gomez, Gulcehre, Caglar, Desjardins, Guillaume, Kirkpatrick, James, Pascanu, Razvan, Mnih, Volodymyr, Kavukcuoglu, Koray, and Hadsell, Raia. Policy distillation, 2016.
- [28] Keren, Gil, Cummins, Nicholas, and Schuller, Björn. Calibrated prediction intervals for neural network regressors. *IEEE Access*, 6:54033–54041, 2018.
- [29] Khosravi, Abbas, Nahavandi, Saeid, Creighton, Doug, and Atiya, Amir F. Lower upper bound estimation method for construction of neural network-based prediction intervals. *IEEE transactions on neural networks*, 22(3):337–346, 2010.
- [30] Salem, Tárík S, Langseth, Helge, and Ramampiaro, Heri. Prediction intervals: Split normal mixture from quality-driven deep ensembles. In *Conference on Uncertainty in Artificial Intelligence*, pages 1179–1187. PMLR, 2020.
- [31] Kivaranovic, Danijel, Johnson, Kory D, and Leeb, Hannes. Adaptive, distribution-free prediction intervals for deep networks. In *International Conference on Artificial Intelligence and Statistics*, pages 4346–4356. PMLR, 2020.

- [32] Romano, Yaniv, Patterson, Evan, and Candes, Emmanuel. Conformalized quantile regression. *Advances in neural information processing systems*, 32, 2019.
- [33] Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. Openai gym. *CoRR*, abs/1606.01540, 2016. URL <http://arxiv.org/abs/1606.01540>.
- [34] Plappert, Matthias. keras-rl. <https://github.com/keras-rl/keras-rl>, 2016.
- [35] Biewald, Lukas. Experiment tracking with weights and biases, 2020. URL <https://www.wandb.com/>. Software available from wandb.com.

שיפור העמידות של מודלי למידת חיזוק עמוקה

באמצעות שימוש ברווחי סמך

אמיר לבנטל

עבודת גמר לתואר מוסמך למדעי הטבע

אוניברסיטת בן-גוריון בנגב

2022

תקציר

אלגוריתמי למידת חיזוק עמוקה משיגים תוצאות מרשימות במגוון תחומים. עם זאת, אלגוריתמים אלו רגישים מאוד לרעש. לכן, שיפור העמידות של אלגוריתמי למידת חיזוק עמוקה לסביבות רועשות הוא קריטי על מנת להתמודד עם בעיות מה"עולם האמיתי". במחקר זה אנחנו מציגים שיטה חדשה מבוססת למידת חיזוק עמוקה אשר מסתמכת על החידושים האחרונים בתחום השימוש ברווחי סמך על מנת לשפר עמידות של רשתות נוירונים בבעיות רגרסיה. אנו מציגים גישה בה משלבים את רווחי הסמך בארכיטקטורה של למידת חיזוק עמוקה, כאשר השימוש ברווחי הסמך הוא על מנת לזהות רעשים בסביבה ושיפור קבלת ההחלטות של האלגוריתם בהתאמה לכך. הניסויים אותם ביצענו מעידים כי השימוש בשיטה שלנו משפר באופן מובהק את הביצועים של אלגוריתמי למידת חיזוק עמוקה בסביבות רועשות במיוחד.



אוניברסיטת בן-גוריון בנגב
הפקולטה למדעי ההנדסה
המחלקה להנדסת מערכות תוכנה ומידע

שיפור העמידות של מודלי למידת חיזוק עמוקה באמצעות שימוש ברווחי סמך

חיבור זה מהווה חלק מהדרישות לקבלת התואר מוסמך למדעי
הטבע (M.Sc.)

אמיר לבנטל

בהנחיית ד"ר גלעד כץ, פרופ' אסף שבתאי

01.10.22	תאריך:	חתימת הסטודנט:
2/10/22	תאריך:	חתימת המנחה:
17/10/2022	תאריך:	חתימת יושב ראש ועדת הוראה:

ספטמבר 2022



אוניברסיטת בן-גוריון בנגב
הפקולטה למדעי ההנדסה
המחלקה להנדסת מערכות תוכנה ומידע

שיפור העמידות של מודלי למידת חיזוק עמוקה באמצעות שימוש ברווחי סמך

חיבור זה מהווה חלק מהדרישות לקבלת התואר מוסמך למדעי
הטבע (M.Sc)

אמיר לבנטל

בהנחיית ד"ר גלעד כץ, פרופ' אסף שבתאי

ספטמבר 2022