Search for Robust Policies in Reinforcement Learning

Qi Li
University College London, 105 Gower Street, London, U.K.

Keywords: Reinforcement Learning, Model Free Policy Search, Robust Agents.

Abstract: While Reinforcement Learning (RL) often operates in idealized Markov Decision Processes (MDPs), their applications in real-world tasks often encounter noise such as in uncertain initial state distributions, noisy dynamics models. Further noise can also be introduced in actions, rewards, and the observations. In this paper we specifically focus on the problem of making agents act in a robust manner under different observation noise distributions for during training and for during testing. Such characterization of training and testing distributions is not common in RL as it is more common to train and deploy the agent on the same MDP. In this work, two methods of improving agent robustness to observation noise - training on noisy environments and modifying the reward function directly to encourage stable policies, are proposed and evaluated. We show that by training on noisy observation distributions, even if the distribution is different from the one in test, can benefit agent performance in test, while the reward modifications are less generally applicable, only improving the optimisation in some cases.

1 INTRODUCTION

A common formulation of Reinforcement Learning (RL) is to learn agents that act in an environment with previously unknown dynamics, and sometimes observation models, in such a way that optimises for some stationary reward signal. Traditional RL formulation operates on a single Markov Decision Process (MDP), where an agent is allowed to interact with the MDP in many episodes. This data collection process often gives variations in the initial state distribution - that is - the task given to the agent is expected to begin not at one particular state, but rather a set of states. In theory, with an adequate exploration policy, under the Markovian assumption and assumptions about observably and stationary dynamics and rewards, it shouldn't matter which states the agents begin during training, as the agent should learn a (probabilistic) optimal policy or value function for all possible states. In practice, especially for RL tasks involving high dimensions and continuous state or action spaces, this does not happen, as the agent is only expected to interact with the MDP a finite number of times. This finite sampling means that the agent usually performs well if it is initialized in the initial state distribution it was trained on, and not on states outside of that distribution. Such out-of-distribution issues is usually not a concern for most RL applications, as there is no clear difference between the "training" and "testing" initial state distributions.

For more complex tasks however, a similar mismatch might occur for the distribution of the state-action transition model from "training" to "testing." This distributional shift is natural to formulate when the MDP has stochastic transitions. During training, the agent might interact with the MDP under one transition distribution, but during test, the agent might be expected to act in an environment with a slightly different dynamics model. If the divergence between the dynamics model of train and test is too wide, then of course the agent will not perform well - it is like acting on a different MDP altogether. If the transition models differ in some structured manner, then one might be able to *adapt* the agent's behavior through informed exploration, or train the agent on a wide enough distribution during training such that the distribution of the testing dynamics is covered by the training distribution. Some subset of the former approach can be called Transfer Learning, and the latter technique is called Domain Randomization (DR). These techniques have been studied by many works in the past.

Different from initial state distribution and dynamics mismatch, this work focuses on another aspect of uncertainty that is of interest to RL, which is the potential noise in observations. The observation

Li, Q.
Search for Robust Policies in Reinforcement Learning.
DOI: 10.5220/0008917404210428
In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020) - Volume 2*, pages 421-428
ISBN: 978-989-758-395-7; ISSN: 2184-433X

noise studied in this work is assumed to be uncorrelated across time and independent of an agent's state or actions. Further, it is assumed the noise is drawn from the same, stationary distribution at all times. In this way, the observation noise is i.i.d. This observation noise is of interest to us, because it is a very common phenomenon when deploying RL agents in real applications. Observations in the real world are always noisy. This is the case for a physical sensor such as cameras and accelerometers, and it is also the case for digital "sensors," such as survey results (human subjects may not accurately respond) and web traffic (dropped packets, unreliable cookies).

The general formulation for noisy observations is the Partially Observable Markov Decision Process (POMDP). In POMPDP, the agent does not know the actual underlying state that it is in, and instead it relies on an observation model to maintain a distribution over possible states that it could be in. This distribution is then used for policy or value function optimization. POMDPs are very difficult to solve, because the uncertainty in maintaining an agent's state distribution widens quickly as a function of time. This makes planning in the general POMDP problem very inefficient, so it is hard to make model-based approaches to POMPDPs tractable in practice.

A model-free approach, while more difficult to provide optimality bounds, tends to be more achievable, and is the focus of this preliminary work. In short, we'd like to train an agent in a model-free manner that is *robust* to observation noise during "test" time.

There are many ways to accomplish this goal, and two approaches are presented:

1. Like training with a distribution over initial state distributions and a distribution over the dynamics model, the agent can be trained to act in an MDP with a distribution over observation noise. This "observation" randomization hopefully encourages the agent to take actions that is more "reversible" or "conservative," so that even if the observation is incorrect, the agent can recover from suboptimal actions in the future.
2. The reward function can also be augmented, or modified, with additional reward signals that explicitly encourage the policy the agent learns to be robust. One additional reward that is experimented with is adding a cost that corresponds to the standard deviation of episode rewards of an agent across many rollouts. This cost favors agents that show consistent behavior across many samples of the noisy observation environments. Another cost in the experiments with is the standard deviation of episode rewards of an agent with slightly perturbed parameters. This parameter-space perturbation encourages the optimization of more "stable" agents. This stability in the parameter space implicitly corresponds to a more stable local minima during optimization, which may correspond to a higher quality policy that is less likely to change given new agent interactions. The hypothesis is that an agent less likely to change during training is also an agent that is more robust to observation noise.

The following sections give a detailed description and background discussions, our policy search algorithms, and the results of the experiments that were performed to evaluate the proposed robustness modifications. By leveraging fast model-free parameter-space policy search algorithms applied to an easy-to-understand toy task (the CartPole), we are able to characterize the agents' behaviors across many different observation noise models in carefully designed train and test distributions. This allows us to gain a clear understanding of the performance and impact of the proposed modifications.

2 RELATED WORKS

One context where the robustness framework has appeared is on-line reinforcement learning (Singh et al., 1994). Previous work has also studied robustness in terms of input disturbance (action noise) (Morimoto and Doya, 2005) (Anderson et al., 2007) (Kretchmar et al., 2001) (Tessler et al., 2019) and modeling errors (dynamics noise or stochastic transitions) (Sami and Memon, 2018) (Rajeswaran et al., 2016) (Kinjo et al., 2018). Some Other works introduce some modifications on RL algorithms to improve the robustness of the agent in specific settings, such as in Partigame (Al-Ansari and Williams, 1999), multi-task learning (Teh et al., 2017), signal temporal logic (Aksaray et al., 2016) (Jones et al., 2015), and hierarchical options (Mankowitz et al., 2018).

For many works, robustness is framed with respect to an adversary that can affect the RL agent during training or testing. Many works have proposed ways to improve RL in such adversarial environments (Lim et al., 2013) (Gu et al., 2018) (Pattanaik et al., 2018) (Gu et al., 2018) (Abdullah et al., 2019). In particular, these works might focus on "distributional robustness," which is in general applicable to the dynamics noise case, where the agent is made to perform well even under distribution shifts in the stochastic transition model.

Another way to improve robustness of RL is to make the agent easy to transfer to new domains.

(Oguni. et al., 2014) studies a case to transfer learned transition and reward models, so the agent can potentially adapt better online to say an environment with a different noise distribution. (Killian et al., 2017) studies policy transfer in the case of MDPs with hidden parameters.

Unlike previous works that focus on input disturbance (action noise) or modeling errors (dynamics noise), this work is unique in that it explicitly considers distributional shifts in observation noise, and studies how might an agent learn to be robust to that.

3 ROBUST POLICY SEARCH

3.1 Markov Decision Processes

A mathematically idealized model, Markov decision processes (MDPs) constitutes a basic framework for dynamically controlling systems that could evolve in a stochastic way. An MDP could have discrete or continuous state and action spaces, and can have finite or infinite time horizons. An MDP has states $s_t \in \mathcal{S}$, actions $a_t \in \mathcal{S}$, and rewards $R_t \in \mathbb{R}$. The focus of this work is on the case of continuous states and actions, so $s_t \in \mathbb{R}^n$, and $a_t \in \mathbb{R}^m$. Importantly, the Markovian assumption made about MDPs states that the future state, conditioned on the current state and action, is independent of previous states and actions.

Reinforcement learning aims to find a policy $\pi : \mathcal{S} \to \mathcal{A}$, mapping from states to actions of an MDP, that maximizes the expected sum of discounted rewards:

$$\mathbb{E} \sum_{t=1}^{T} \gamma R_t \tag{1}$$

Where γ is a discount factor between 0 and 1. Here the expectation can be taken over the distribution over transitions if the dynamics is stochastic. Or, it can also be taken over the distribution of the policy if the policy is stochastic. In general, T is the time horizon of the task, and it can be infinite. However, even if the time horizon is infinite, a γ smaller than 1 effectively imposes a finite time horizon, as future rewards fall off exponentially.

Different from supervised learning, RL is not supervised with the optimal action, or actions, to take at each state. Instead, it is given a evaluative supervision - how good is an action or a sequence of actions. In addition, actions chosen by the RL agent will affect future states that the agent is in, hence it will affect future inputs into the model. The sequential decision nature of RL may lead to distributional shifts over time in the states and dynamics the agent encounters, making RL much harder to learn than supervised learning.

3.2 Policy Search in Parameter Space

There are many ways to search for a policy π in RL, and they roughly separate into Model-based or Model-free methods. Model-based methods first sample transitions from the MDP. Then it fits a function to model the dynamics of the environment, as well as the reward, to learn the MDP. With the MDP learned, standard MDP optimization algorithms can be used to find the optimal policy in the learned MDP. Model-based methods often suffer from distribution mismatch issues - the data collected during model learning may not correspond well to the data seen by the agent during execution. However, when the distribution shift is small, and when the learned model has low estimation bias, model-based RL tend to be more sample efficient than model-free ones.

Model-free methods can be further separated into on-policy and off-policy algorithms, as well as value-based and policy-based algorithms. This work focuses on the on-policy, policy-based variation. This type of policy search usually first represents the policy in some parameterized way, like a linear or Neural Network (NN) function. Denote these parameters as θ, and denote the policy with such parameters π_θ. Then, exploration is performed either in the action space, by adding noise to the policy outputs, or in the parameter space, by directly perturbing the policy parameters. The former method corresponds to Policy Gradients, while the latter is usually done by derivative-free optimization.

Derivative-free policy search in the parameter space is used, because it achieves comparable performance with policy gradients but with less variance and higher sample efficiency (Mania et al., 2018). Generally, a derivative-free policy search contains the following steps:

1. Sample policies
2. Evaluate each policy
3. Update optimiser
4. Keep track of the best policy so far until reaching a reward threshold or out of compute budget

Training policies can be done through either step-based or episode-based feedback. Episode based training waits for each episode to finish before starting another policy, whereas step based training gives a reward at each state-transition pair. Derivative-free policy search methods are usually episode-based, so the policy training feedback takes into account of the agent's cumulative performance across the entire task.

In our experiments, we evaluate and analyze 3 different derivative-free optimisers to search for agent policies in the parameter space:

3.3 Random Search (RS)

RS evaluates a random set of directions that the policy parameters can go into, and weights these directions by how well, in relative terms, the policies perform. At each training iteration, it first samples N numbers of $\Delta\theta_n$, then $\pi_{\theta+\Delta\theta_n}$ and $\pi_{\theta-\Delta\theta_n}$ are evaluated. Update formula is as following:

$$\theta_{t+1} = \theta_t + \frac{\alpha}{N}\sum_{n=1}^{N}(R(\pi_{\theta_t+\Delta\theta_n}) - R(\pi_{\theta_t-\Delta\theta_n}))\Delta\theta_n \quad (2)$$

where α is the learning rate.

3.3.1 Cross Entropy Method (CEM)

CEM uses Gaussian distribution to maintain a distribution over the policy parameters $\mathcal{N}(\mu_t, \Sigma_t)$. At each training iteration, CEM samples N numbers of $\theta_n \sim \mathcal{N}(\mu_t, \Sigma_t)$, then evaluates π_{θ_n}s. The update process is as the following:

1. Sort θ_ns by their rewards.
2. Choose the top N_{best} θ_ns
3.

$$\mu_{t+1} = \frac{1}{N_{\text{best}}}\sum_{n=1}^{N_{\text{best}}}\theta_n \quad (3)$$

$$\Sigma_{t+1} = \frac{1}{N_{\text{best}}}\sum_{n=1}^{N_{\text{best}}}(\theta_n - \mu_t)(\theta_n - \mu_t)^T \quad (4)$$

3.3.2 Genetic Algorithm (GA)

GA is a "population-based" algorithm. Unlike RS and CEM, it does not explicitly maintain a current policy or a current distribution of policies. Instead, it maintains a population of policies.

Initially, GA samples N numbers of parameters of the policy from $\theta_t^{(1)}$ to $\theta_t^{(N)}$. Then, at each training iteration:

1. Choose N pairs of previously sampled solutions, i.e $(\theta_t^{(i)}, \theta_t^{(j)})$, where $i \neq j$.
2. Create N new policies, with parameter values set to $\theta_n[k] = \beta_n\theta_t^{(i)}[k] + (1-\beta_n)\theta_t^{(j)}[k] + \varepsilon_n$

β_n is either 0 or 1, and it's sampled from a Bernoulli distribution with $p = 0.5$. ε_n is a noise parameter sampled from a Gaussian distribution with $\varepsilon \sim N(0, \sigma)$.

Unlike RS and CEM, GA can handle multi-modal object landscapes, because it is not forced to converge to a policy, it can maintain many policies that all perform well, even if they have very different parameters (Loughlin et al., 2001; Zhan et al., 2013).

3.4 Noisy MDPs and Robust Policies

Noise in RL is common for real world applications. Three types of noise models can be used to simulate realistic learning environment: 1) Action noise - for example, in robotics applications, a robot controller might not follow exactly what the commanded actions ask. 2) Reward noise - during training, the reward feedback is noisy. 3) Observation noise - an observation that an agent receives is always a perturbed version of the true observation. This paper studies how to make policy search robust to observation noise.

In reinforcement learning, a policy is said to be robust if it maximizes the reward when noise is introduced during the process. One way to measure robustness is to see how wide the variation, or the standard deviation, of the agent's performance is over multiple rollouts:

$$\sigma_R = \frac{1}{K}\sum_{k=1}^{K}(R_k - \mu_R)^2 \quad (5)$$

R_k means the total cumulative reward of the kth policy rollout. An agent with high μ_R and small σ_R is set to be more robust than the one with high $\sigma_R T$.

The standard deviation of rewards on multiple policies around a given policy is as:

$$\sigma_{\pi_\theta} = \frac{1}{K}\sum_{k=1}^{K}(R(\pi_{\theta+\delta\theta_k}) - \mu_R)^2 \quad (6)$$

$\delta\theta_k$ is a small perturbation on the true policy parameters, which is sampled from a Gaussian distribution. This measures how sensitive the policy is to parameter perturbations, and hence to future training data points.

Another way to measure an agent's robustness is by taking the ratio of rollout rewards under some observation noise distribution o_1, while the agent was trained on another o_2:

$$\phi_{12} = \frac{\mu_{R(o_1)}}{\mu_{R(o_2)}} \quad (7)$$

The higher this ratio is, the more robust the agent is to changes in the observation noise distribution.

The following experiments evaluate two ways of training that may help improve robustness of a policy:

1. First, the policy is trained in noisy environment, so it is expected a better performance during evaluation than a policy trained on environments with no observation noise.

2. Second, the training objective function of the optimiser is changed to optimise the robustness of the policy, while the reward of the environment stays the same. The modified reward function for the optimisers can be written as the following:

$$R' = R - \alpha_r \sigma_R - \alpha_\pi \sigma_{\pi_\theta} \quad (8)$$

The α_r and α_π terms are scaling hyperparameters.

4 EXPERIMENTS

4.1 Benchmark Tasks

The CartPole task is used to perform learning experiments. In a CartPole task, a pole is attached to a cart by an un-actuated joint, which can be moved to left or right along a track. So the action space of the task is discrete and binary (either left or right). An agent observes the current position and velocity of the cart, as well as the current angle and angular velocity of the pole. These 4 numbers make up the state space.

By moving the cart backwards and forwards, the pole can be controlled to stay upright. The goal of this task is to prevent the pole from falling over. The initial state of the task has the pole angled upright with 0 angular and linear velocity. A reward $+1$ is given when the pole stays upright. The performance of an agent is measured by the cumulative reward at the end of a fixed time horizon. If the time horizon is H, then the maximum total reward is also H. A time horizon of $H = 200$ is used in all experiments below.

The agent uses a linear policy consisting of a vector with 4 numbers: $\theta \in \mathbb{R}^4$. The action is determined by $\pi_\theta(s) = sign(\theta^\top s)$. If the output is 1, the agent moves right. If the output is -1, the agent moves left.

The CartPole task and the linear policy are chosen, because they are fast to simulate, compute, and optimise, allowing us to perform many ablation experiments to understand effects of the algorithm.

4.2 Observation Noise

To add observation noise, we do $s' = s + \beta\varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \Sigma_s)$. Σ_s is a diagonal covariance matrix, and its values are set to be the corresponding standard deviations of states encountered by a random agent. This ensures the magnitude of the noise added is in the appropriate scale for each dimension of the observation vector, which all have different units. β is a hyperparameter setting to scale the noise amounts.

Our experiments swept across 4 types of parameters to evaluate the performance of the agent with different modifications on the CartPole task:

1. Random seed - each of the following experiments were run with 3 different random seeds.
2. Training noise level - agents were trained in 4 different noise levels for $\beta \in [0, 0.25, 0.5, 1]$.
3. Training algorithm - agents were trained in the 3 derivative-free optimization algorithms explained above (RS, CEM, GA).
4. Reward modification - the reward function used by the optimiser was modified to explicitly account for both the standard deviation of rollout rewards and the standard deviation across policy parameter perturbations. Here we put four kinds of modification settings: $(\alpha_R, \alpha_\pi) \in [(0,0), (0.5,0), (0,0.5), (0.25,0.25)]$.

The CartPole environment is based on OpenAI gym [1]. Derivative-free optimisation was performed in Numpy with Python 3.6, and the implementation is fairly fast given the small size of the observation and action spaces. All trained policies are then evaluated on the 4 different types of observation noise levels ($\beta \in [0, 0.25, 0.5, 1]$). Totally, $3 \times 4 \times 3 \times 4 = 144$ policies were trained, and each of them were evaluated in 4 testing environments. The experiments were run by a computer with i7-8550U CPU at 1.80GHz, taking about 2 hours.

4.3 Results

Table 1: Robustness ratio ϕ across all algorithms and random seeds. The rows are testing noise, while the columns are training noise. For example, row 1 column 2 is ϕ_{12}, which gives the mean ratio of the total reward of the agent trained with 0.25 observation noise level and tested under 0 noise level over the rewards it had during training.

	0	0.25	0.5	1
0	1	1.15	1.38	2.36
0.25	0.91	1	1.25	2.14
0.5	0.73	0.83	1	1.71
1	0.42	0.48	0.58	1

Table 2: Robustness ratio ϕ of Genetic algorithm (GA). Both rows and columns are training noise as well as testing noise. For example, row 1 column 2 is ϕ_{12}, which gives the ratio of the reward of the agent trained with 0.25 observation noise level and tested under 0 noise level over the rewards it had with 0 training noise and 0 testing noise.

	0,0	0.25,0	0.5,0	1,0
0,0	1	1	1	1
0,0	0.78	1	0.99	0.98
0,0	0.62	0.77	0.9	0.91
0,0	0.36	0.39	0.56	0.61

[1] https://gym.openai.com/envs/CartPole-v1/

Table 1 shows the robustness ratio ϕs, where the entry in the ith row and jth column corresponds to the average ϕ_{ij} across all 3 random seeds and 3 optimisation algorithms. The row headers mean the testing noise level, while the column headers mean the training noise level. For example, row 1 column 2 is ϕ_{12}, which gives the ratio of the agent trained with 0.25 observation noise level and tested under 0 noise level.

Of course, an agent tested on the same observation noise distribution as it was trained on will perform the same, hence the 1s across the diagonals. It can be seen that across the board, testing performance decreases as the testing noise level increases, and increases as the training noise level increases. The agent tends to learn better and to be more robust when it is trained in a noisy environment.

Table 2 shows the robustness ratio ϕ of the Genetic algorithm (GA), where the entry in the ith row and jth column corresponds to ϕ_{ij}. Both row headers and column headers mean the training noise level and testing noise level. For example, row 1 column 2 is ϕ_{12}, which gives the ratio of the reward of the agent trained with 0.25 observation noise level and tested under 0 noise level over the rewards it had with 0 training noise and 0 testing noise.

It can be shown that training on noisy environments lead to better performance overall. For example, ϕ_{44} gives the robustness ratio of the agent trained on 1 noise level and tested on 0 noise level. The agent is able to gain a maximum reward of 0.61. While, ϕ_{41}, the robustness ratio showing an agent trained on 0 noise level and tested on 0 noise level, shows that the agent can only achieve a maximum reward of 0.36.

Figure 1 shows the testing rewards of different policies (RS ,CEM, GA). Each policy was trained on 0 noise and tested on the indicated noise level on the x-axis. Each bar and its error bar give mean rewards and standard deviation of rewards of all evaluations across seeds 0, 1 and 2.

It shows that training rewards decrease when observation noise rises from 0 to 1 for each algorithm. Overall, GA gives highest rewards in all three noise levels followed by Random search and Cross Entropy. However, the standard deviation of mean rewards for GA increases as the noise level increases, which means that the performance is less reliable and stable. Under noise level 0.25 and 1, RS and CEM have roughly the same mean rewards.

Figure 2 gives training curves for all 3 algorithms under different training observation noise levels. Each curve is accompanied by a shaded region that denotes the standard deviation across multiple rollouts, while the mean is computed across the random seeds.

For RS, as the test noise level increases, the

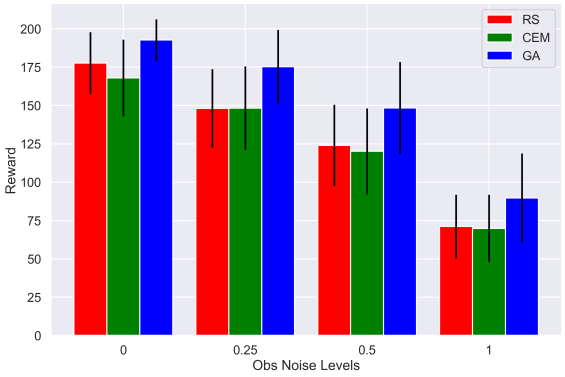

Figure 1: Performance of different algorithms trained with 0 observation noise and tested under different observation noise levels. Length of black bar means the standard deviation of mean rewards across 3 random seeds.

growth of the mean rewards gained by the agent goes down, and the final reward earned reduces. Although the performance of this policy is not good in the noisy environments, its variance is relatively smaller than the other algorithms.

For CEM, under 0, 0.25 and 0.5 noise levels, the mean rewards rise rapidly relative to that under 1 noise level. However, the mean standard deviation of these three mean rewards under the three noise levels are quite big, meaning that the rewards earning process is not very stable.

For GA the agent learns to achieve optimal rewards rapidly when noise level is 0. Under 0.25 and 0.5 noise levels, the agent learns and is able to achieve optimal rewards but with a high fluctuation during learning process. When noise level becomes to be 1, the agent learns slowly with high fluctuations and is not able to achieve optimization.

Figure 3 illustrates the mean and standard deviation of the mean reward that an agent earns in RS, CEM, and GA. It is trained under 0 noise level with different reward modification settings but evaluated under 0, 0.25, 0.5 and 1 noise levels across seeds 0, 1 and 2. In terms of reward modification settings, weights are added to the standard deviation of reward and to that of policy.

The motivation of adding a cost term to the standard deviation across episode rollouts is to encourage the agent to have a stable and consistent performance in noisy environments. The motivation of adding a cost term to the standard deviation of policy parameter perturbations helps the agent to behave roughly the same under new iterations and to decrease the possibility of optimisation to be 'getting lucky'.

For all algorithms, the mean rewards decrease as the test noise level rises. The standard deviation of the mean rewards ascends by the noise levels, mean-

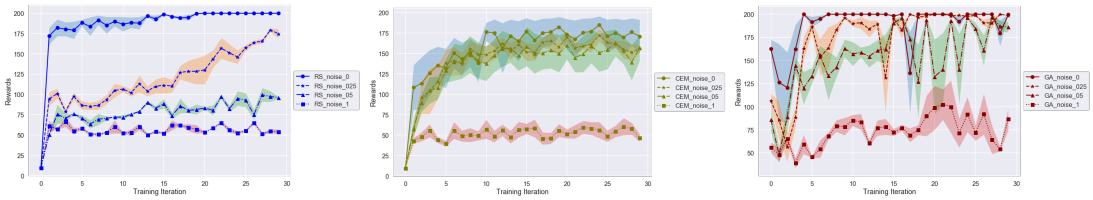

Figure 2: Comparison of training curves of three different policies for different training noise levels aggregated across 3 random seeds. CEM and GA perform better than RS at higher noise levels, but RS has the least reward variance.

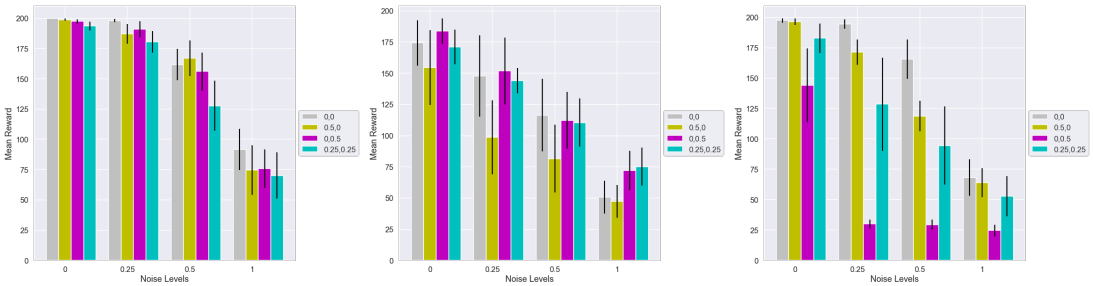

Figure 3: Performance of Different reward modifications for all optimisation algorithms (left = RS, middle = CEM, right = GA). All agents were trained at 0 noise level with the reward modifications, and the noise levels in the plot mean training noise levels. The first number in the legend tuples refer to α_R, the second is α_π. It can be observed that while the reward modifications improve performance by either higher mean or lower reward variance in some cases for CEM, in other cases it performs worse than having no reward modification at all.

ing that the consistence of the agent's performance for different iterations is reduced. Under 0 test noise level, the modification setting does not result much difference about the mean rewards which achieved by the agent from the first group of bars. The difference between rewards collected by an agent under different modification setting then rises as the noise level increases. For CEM, the difference between mean rewards collected by an agent under various modification settings is staggered at each noise level. Also, the standard deviation of mean rewards is huge for each modification setting, which shows that the agent may not give consistent behaviour in noisy environments or the policy may likely to change its results when an agent learns in new iterations. For GA, the modification setting with 0 weight on rewards and 0.5 weight on policy does not end up with an excellent rewards-collecting results compared to the other modification settings in noisy environments. And its mean rewards earned by the agent stay at a relatively low level once the noise participates. However, the standard deviation of the specific modification setting reduces quickly when noise is added.

In general, it is observed that the reward modifications can be beneficial for improving the performance of CEM (higher rewards and lower reward variance) in some noise levels, but it does worse than not having reward modifications for RS and GA.

5 CONCLUSIONS

The paper investigates the robust agent problem for Reinforcement Learning in the context of observation noise. Two main proposed methods were introduced - training in noisy environments and modifying optimisation objectives to encourage more stable learnt policies. Extensive experiments were performed on the CartPole problem via three derivative-free parameter-space policy search algorithms: Random Search, Cross Entropy Method, and the Genetic Algorithm. Several experiments are done to compare the performance in both noisy environment and non-noise environment. It is observed that the agent performance decreases further if the observation noise the agent tested on is more than the noise it was trained on, so that training in noisy environments helps to improve agent performance in noisy environments. The reward modifications, used to encourage optimisers to find stable policies are also analyzed. In the experiments, it is observed that while these modifications help CEM in some cases, in other cases they perform worse. It is likely that the decreased reward makes the agents overly conservative, as it also discourages policy exploration and obfuscates the true reward signal. Therefore, improving the robustness of an agent via training on noisier environments is preferred.

REFERENCES

Abdullah, M. A., Ren, H., Ammar, H. B., Milenkovic, V., Luo, R., Zhang, M., and Wang, J. (2019). Wasserstein robust reinforcement learning. *arXiv preprint arXiv:1907.13196*.

Aksaray, D., Jones, A., Kong, Z., Schwager, M., and Belta, C. (2016). Q-learning for robust satisfaction of signal temporal logic specifications. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 6565–6570. IEEE.

Al-Ansari, M. A. and Williams, R. J. (1999). Robust, efficient, globally-optimized reinforcement learning with the parti-game algorithm. In *Advances in Neural Information Processing Systems*, pages 961–967.

Anderson, C. W., Young, P. M., Buehner, M. R., Knight, J. N., Bush, K. A., and Hittle, D. C. (2007). Robust reinforcement learning control using integral quadratic constraints for recurrent neural networks. *IEEE Transactions on Neural Networks*, 18(4):993–1002.

Gu, Z., Jia, Z., and Choset, H. (2018). Adversary a3c for robust reinforcement learning.

Jones, A., Aksaray, D., Kong, Z., Schwager, M., and Belta, C. (2015). Robust satisfaction of temporal logic specifications via reinforcement learning. *arXiv preprint arXiv:1510.06460*.

Killian, T. W., Daulton, S., Konidaris, G., and Doshi-Velez, F. (2017). Robust and efficient transfer learning with hidden parameter markov decision processes. In *Advances in Neural Information Processing Systems*, pages 6250–6261.

Kinjo, K., Uchibe, E., and Doya, K. (2018). Robustness of linearly solvable markov games employing inaccurate dynamics model. *Artificial Life and Robotics*, 23(1):1–9.

Kretchmar, R. M., Young, P. M., Anderson, C. W., Hittle, D. C., Anderson, M. L., and Delnero, C. C. (2001). Robust reinforcement learning control with static and dynamic stability. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 11(15):1469–1500.

Lim, S. H., Xu, H., and Mannor, S. (2013). Reinforcement learning in robust markov decision processes. In *Advances in Neural Information Processing Systems*, pages 701–709.

Loughlin, D. H., Ranjithan, S. R., Brill Jr, E. D., and Baugh Jr, J. W. (2001). Genetic algorithm approaches for addressing unmodeled objectives in optimization problems. *Engineering Optimization*, 33(5):549–569.

Mania, H., Guy, A., and Recht, B. (2018). Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*.

Mankowitz, D. J., Mann, T. A., Bacon, P.-L., Precup, D., and Mannor, S. (2018). Learning robust options. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Morimoto, J. and Doya, K. (2005). Robust reinforcement learning. *Neural computation*, 17(2):335–359.

Oguni., K., Narisawa., K., and Shinohara., A. (2014). Reducing sample complexity in reinforcement learning by transferring transition and reward probabilities. In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,*, pages 632–638. INSTICC, SciTePress.

Pattanaik, A., Tang, Z., Liu, S., Bommannan, G., and Chowdhary, G. (2018). Robust deep reinforcement learning with adversarial attacks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2040–2042. International Foundation for Autonomous Agents and Multiagent Systems.

Rajeswaran, A., Ghotra, S., Ravindran, B., and Levine, S. (2016). Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*.

Sami, A. and Memon, A. Y. (2018). Robust optimal control of continuous time linear system using reinforcement learning. In *2018 Australian & New Zealand Control Conference (ANZCC)*, pages 154–159. IEEE.

Singh, S. P., Barto, A. G., Grupen, R., and Connolly, C. (1994). Robust reinforcement learning in motion planning. In *Advances in neural information processing systems*, pages 655–662.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4496–4506.

Tessler, C., Efroni, Y., and Mannor, S. (2019). Action robust reinforcement learning and applications in continuous control. *arXiv preprint arXiv:1901.09184*.

Zhan, Z.-H., Li, J., Cao, J., Zhang, J., Chung, H. S.-H., and Shi, Y.-H. (2013). Multiple populations for multiple objectives: A coevolutionary technique for solving multiobjective optimization problems. *IEEE transactions on cybernetics*, 43(2):445–463.