Towards Generalizing Deep Reinforcement Learning Algorithms For Real World Applications

Daniel Ruiru[1,2][0000−0003−3197−8821], Nicolas Jouandeau[2][0000−0001−6902−4324], and Dickson Owuor[1][0000−0002−0968−5742]

[1] Strathmore University, Nairobi, Kenya
{druiru,dowuor}@strathmore.edu
[2] Université Paris 8: Saint-Denis, France
{dr,n}@up8.edu

Abstract. Generalization is a major problem in reinforcement learning (RL), as agents struggle in environments outside their training sets. Often caused by overfitting during the training phase, this issue limits the application of RL in the real world. This paper tries to solve the generalization problem by using a domain randomization technique during the training period. Using two real-world problems; the financial market and agriculture (crop production), this paper trains classical deep reinforcement learning algorithms (DQN and PPO) in different and randomized environments (MDPs). Agents trained in randomized environments generalize better than those trained in single environments (baseline agents). This conclusion is based on the results, in which the agents trained in the randomized environments achieve higher cumulative rewards.

Keywords: Deep Reinforcement Learning · Markov Decision Process · Deep Q Network · Policy Optimization.

1 Introduction

Deep Reinforcement Learning (DRL) is today one of the most prominent subdomains of artificial intelligence. One of its most exciting possibilities is the use of its algorithms to solve complex, real world issues such as resource allocations problems in finance, agriculture and computing [1]. This DRL potential comes from performance of reinforcement learning (RL) methods which can be improved using function approximators such as neural networks [2]. Neural networks are often preferred for this role because of the straight forward way of adjusting a gradient [3]. This improvement is much needed in the RL domain because of its persistent critique of limited adaptability in the real world [4]. While RL has shown remarkable success in solving complex problems like games, it fails to achieve competitive results in most worldly problems. RL depends on unbiased interactions with environments and countless trial and error attempts, requirements that are often not achieved [5]. Also, unlike games which are based on controlled environments, the real world has unique challenges that are usually open-ended [6] and non-stationary [7]. These attributes make it harder to get good results with conventional RL methods.

By combining RL with Deep Learning, DRL provides a starting point for solving many real world problems. First, by providing a means to engage vast amounts of environments and secondly, by facilitating efficient and countless interactions with the said environments [8]. As such, today, it is possible to train DRL policies that operate in a variety of state and actions spaces as defined by Markov Decision Processes (MDPs) [9]. While most of the learned policies are restricted to the training environments (poor generalization), this ability to work in different MDPs is an important building block for reasonable generalization.

However, as it stands, most RL/DRL solutions are specific to the training environments. This limitation is well documented within the DRL field. For instance, there has been countless studies that have shown this generalization problem using adversarial perturbations in state observations of policies and neural networks [10]. Other scholars like [11] have reviewed adversarial perturbations in the DRL domain, while simple review studies like those done by [12] have analyzed adversarial attacks in deep policies. Majorly, most of these studies have focused on this issue by exploring the fundamental problems of function approximations, action value functions and estimation biases of states [13] while still dealing with new architectural designs [14]. This brief overview of the generalization problem highlights two critical issues found in DRL policies; one, the typical use of deep neural networks as function approximators, which inherits their intrinsic issues [15]. Two, the inability to completely explore entire MDPs in high-dimensional problems.

These critical drawbacks are then worsened by the problem of over fitting. Most DRL policies suffer from overfitting because they are only trained on single environments. Without solutions to any of these fundamental issues, the generalization problem will always exists as highlighted by [16] who thought RL policies specialization in training sets is their major fault when deployed in different environments. As a possible solution, this paper tries to improve DRL generalization by exposing algorithms to different MDPs during the training process. The two DRL algorithms used during the experiments were randomly exposed to different MDPs at each training episodes with their results analyzed in this paper. Two test environments were used, the financial markets (profits and losses) and agriculture (application of fertilize and impact on winter wheat production).

The process of conducting the suggested experiments led to two contributions from this paper:

1. Development of a financial trading environment (including; action space, state space, and reward function) that exposes any reinforcement learning algorithms to different MDPs during training. The frequency of MDP changes in every episode can be defined by the user, including the restrictions of repeat and reshuffle.
2. Experiments on two real world problems where baseline and domain randomization tests were done.

.

2 Background

To properly define the approaches used in this paper to achieve generalization, it is important to outline the basic components of RL and DRL. In particular, the algorithms and how they interact with an environment. An RL and DRL algorithm $\mathbb{A}$ learns a policy π by interacting with a Markov Decision Process (MDP) [15]. MDPs are a vital aspect of RL as they provide a framework for solving decision-making problems that have random or controlled decision makers [17]. These decision makers tend to make sequential decisions that MDPs help to define by evaluating actions taken during current states and environments[18]. In this paper, MDPs from different but related contexts are used to improve performance by randomly using them in the training episodes.

In a standard RL formulation an agent sequentially engages an environment. This process is modeled as a tuple problem (formal definition of an MDP) having a state (S), action (A), transition function (P), reward (R) and discounting factor (Υ). This interaction typically starts when the agent begins from a state s_0 which is part of a possible start states (S_0) by taking an action $a\ \varepsilon\ A$, at given time step $t\ \varepsilon\ N$ [19]. What follow is a transition function (T) which defines the probability of moving to state s' having taken action a from state s [20]. Thereafter, a reward function R outlines the agent's reward following a transition. Ultimately, the agent's objective is to maximize the accumulated reward, discounted by a factor Υ.

A policy π that facilitates the agent's movement is important to mention. Policy π helps to map states into actions (π: $S \rightarrow A$), a process that fully highlights the behavior of the agent. In the end, the typical objective of an RL problem is to maximize the expected cumulative reward R (equation 1).

$$R = \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)\right] \tag{1}$$

Where:

- R is expected return,
- Υ is the discount factor,
- $r(s_t, a_t)$ is reward a time t when being in state s_t by taking action a_t ,
- T is the total time-steps.
- All this sum run over time-steps t=0 to T-1

2.1 Related Works on Generalization

Generalization has been a long-standing issue in RL that has led to different specifications and solutions. Often, most measures have focused on the learning process in an attempt to capture the abilities of agents operating in different environments and making efficient use of samples [21], including those in policy-based methods [22]. Such measures have dominated learning theories in sample complexities as experiments try to capture efficiency in the learning processes

[23], more so when samples have information on environment dynamics [24]. Other notable works have attempted to apply supervised learning dynamics by having train and test paradigms in training trajectories or state spaces [25]. Overfitting has also been studied in a similar manner, where RL agents have been found to overfit to their training environments [26]. Such studies have provided benchmarks for testing RL in the real world [27]. When formularized, these benchmarks have showed the importance of having a wide range of testing environments to limit overfitting. In all, generalization in most of these works has been captured as a means to avoid over-fitting in particular training environment [28]. This broad definition has implied a need to sample from diverse environments as a starting point for generalization.

However, this sampling requirement can vary based on the definition of generalization. For instance, when generalization is defined as improved performance in off-policy states [29], this diverse sampling is not necessary as it pushes RL into the standard methods of supervised learning. Similarly, generalization techniques such as adding stochasticity in policies, having random steps, and adding human play steps push RL agents into off-policy states which diversify the training data without any sampling requirement [30].

2.2 Methods of Generalization

Generalization in DRL and RL in general is led by two approaches; modification to the training algorithm and modification of the MDP.

Modifying the Training Algorithm Also referred to as algorithmic generalization, this approach has methods that modify the training algorithm through techniques that use optimizers, regularizer, and updates to the rules of the policy [15].

A holistic definition of these methods considers a training algorithm $\mathbb{A}$ which gets an MDP as input and produces a policy π. Given a typical MDP definition (tuple, with 5 elements), an algorithmic generalization method G_A is produced by a function $F : A \rightarrow \mathbb{A}$ that runs the training algorithm $F(\mathbb{A})$ in the said MDP.

Modifying the MDP - Domain Randomization The other major category of generalization tries to change the MDP directly by either modifying the learning environment or the training data [31]. This transformation affects the interaction between the training algorithm and the learning environment by having an overall objective of increasing the samples engaged.

This approach is also called domain randomization as it involves varying the parameters of the environment during training to increase the robustness of the learned policy [15]. Domain randomization is implemented during training where the selected environment parameters are adjusted randomly during the episodes [2]. The changes done alter the state transition dynamics, reward processes and

in some instances the agent's interaction with the environment [32]. Domain randomization modifications can be summarized as:

1. Randomizing the state space - where the state space is changed to reflect environmental conditions as highlighted by equation 2.

$$S = \{s \mid s \in S_0 \text{ with random perturbations from } P\} \tag{2}$$

   With S_0 being the base state space and P representing random perturbations of an environmental variable.
2. Randomizing the action space - here the action space is modified by changes in environmental dynamics as formulated by equation 3.

$$A = \{a \mid a \in A_0 \text{ with random variation from } P\} \tag{3}$$

   Where A_0 is the original action space and P is the noise in the environment.
3. Randomization of transition probability - the state transition model is varied to simulate uncertainty in the real world (equation 4).

$$P(s' \mid s, a) = P_0(s' \mid s, a) \times N(\mu, \sigma) \tag{4}$$

   With $P_0(S'|s, a)$ representing the original transition model and $N(\mu, \sigma)$ is the added noise to simulate environmental dynamics.
4. Randomization of the reward function - finally, the reward function can be randomized as influenced by environmental changes (equation 5).

$$R(s, a) = R_0(s, a) + R_{\text{rand}}(s, a) \tag{5}$$

   $R_0(s, a)$ being the base reward and $R_{rand}(s, a)$ is the randomness element.

Therefore, a possible path for generalization through domain randomization can be achieved using the following basic steps:

- Start randomization
- Training on different (varied) environments
- Variations are introduced to the environments (i.e. changes to state space, action space, transition and reward)
- Policy learning stage where the agent is exposed to diverse training scenarios
- Transfer the learning to an unknown environment

3 Methodology

3.1 Motivating Examples

Robust performances for either offline and online RL techniques will begin to emerge when algorithms gain some level of generalization. This demand for enhanced capabilities motivated the use of the two real world problems; the financial market and agriculture. Both of these issues have ever changing conditions

that limits the application of other machine learning techniques. For instance, while a financial instrument like Gold or Euro/U.S. Dollar may have an evident trading pattern, unlimited factors can drastically shift its price action [33]. Similarly, in agriculture, the yields from a particular crop can drastically change from one season to another. Developing a model that adapts to immediate conditions is thus necessary as defined by the existing environmental parameters.

This requirement defines a Markov Property where the future state only depends on the present state. Also defined as memorylessness, this property is considered in the motivating examples where formally, if S_t represents a state of a process at time t, the Markov property is given by

$$P(X_{t+1} \mid X_t, X_{t-1}, \ldots, X_0) = P(X_{t+1} \mid X_t) \tag{6}$$

Which means that the conditional probability of moving to a new state X_{x+1} depends solely on the present state X_t. While, this assertion is true, it borrows from the understanding that the past and future are conditionally independent, given the present[34]. This because a *State* as defined in an MDP determines what happens next and is understood to be a function of the history [35]. With history being a sequence of observations, actions and rewards (equation 7).

$$H_t = O_1, R_1, A_1, ...A_{t-1}, O_t, R_t \tag{7}$$

Formally, therefore;

$$State(S_t) = f(H_t) \tag{8}$$

Defining The Environments For the financial market, Gymnasium (formerly Gym) environments were designed using six financial instruments (Euro/U.S. Dollar(EURUSD), British pound/U.S. dollar (GBPUSD), U.S. dollar/Chinese Yuan (USDCNY), U.S. Dollar/Japanese Yen (USDJPY), Silver (XAGUSD) and Gold (XAUUSD)). This environment was named BeshaGym and the dataset for all the 6 instruments was from 2004 to 2024.

BeshaGym unlike existing trading environments provides a means to randomize MDPs during training, which is a significant contribution of this paper. Moreover, the randomization can be changed based a user's needs.

For the second environment, CropGym was used. CropGym is a reinforcement learning platform developed by the Wageningen University in Netherlands for optimizing DRL agents for the application of nitrogen fertilizer in crops [36].

The objective of the BeshaGym was to create an environment where a DRL agent trades the financial market to maximize profit. The agent would have access to a window size of 50 past time steps (i.e. the agent observed the last 50 time steps at every trading period) to base its decisions. The reward (R_t) from the BeshaGym environment was based on the ability to take profitable trades be it either through buys (long) or sells (shorts). A starting trading balance of 10,000 (representing $10000) was also initialized in the environment.

CropGym on the other hand, as defined by [36] is a highly configurable RL environment built on the Python Crop Simulation Environment (PCSE) to help

$$\mathrm{R}_t = \begin{cases} -(spotprice - lastprice) \times 10000 & \text{if, Short \& action = Sell} \\ +(spotprice - lastprice) \times 10000 & \text{if, Long \& action = Buy} \\ 0 & \text{otherwise} \end{cases}$$

develop RL agents for implementing crop growth simulation models. In this paper, the experiments done were on the application of nitrogen fertilizer on Winter Wheat [36]. DRL agents were to apply a discrete amount of nitrogen fertilizer to balance crop yield and environmental impact. The reward (R_t) of this environment borrows from the LINTUL-3 model which is a subset of PCSE general model where growth parameters like TNSOIL (amount of nitrogen in the soil), LAI (leaf area index), NUPPT (uptake of nitrogen in soil) and WSO (weight of storage organ - the grain) are simulated. The DRL agent thus tries to maximize the difference in WSO for policy using nitrogen fertilizer and those that do not use the fertilizer (zero nitrogen policy). There is also an added cost β that mimics the price of nitrogen fertilizer (equation 9).

$$\mathrm{R}_t = (WSO_t^{\pi} - WSO_{t-1}^{\pi}) - (WSO_t^{0} - WSO_{t-1}^{0}) - \beta N_t \tag{9}$$

DRL Agents For the DRL agent, they were based on Deep Q Network (DQN) (Figure 1) [37] and Proximal Policy Optimization (PPO) (Figure 2) [38] algorithms. These algorithms were sourced from the Stable-Baselines3 (SB3) library which provides simple and effective implementations. For DQN, SB3 builds on top of the fitted Q-iteration (FQI) which helps stablize learning with neural networks. Conversely, PPO combines ideas from A2C [38] and TRPO [21] by minimizing updates for new policy from old policy.

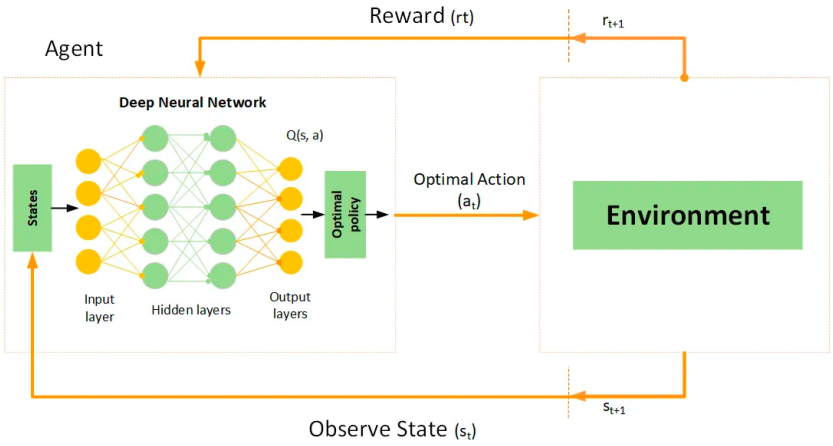


Fig. 1: High-Level Diagram of Deep Q Network [39]

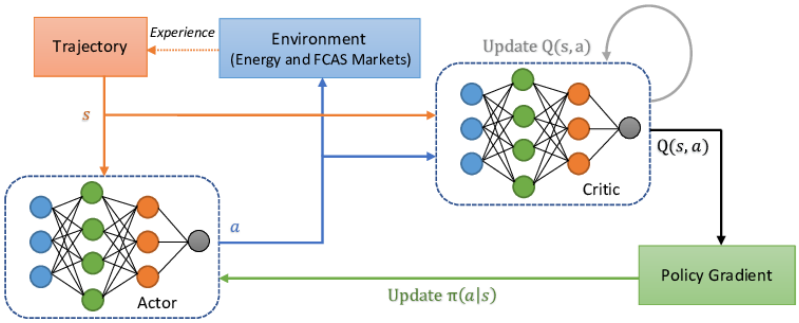


Fig. 2: High-Level Diagram of Proximal Policy Optimization [40]

3.2 Definitions

To improve generalization, the experiments conducted were formally setup with a domain randomization approach. A randomization strategy was used while training the agents in different MDPs across the training episodes. These MDPs were generated by creating diverse Gymnasium environments using different CSV datasets which encoded different training scenarios. As such, each episode involved training an agent on an environment from a different dataset which effectively varied the environment's dynamics, promoting generalization by simulating real world situations.

Key Items Markov Decision Process (MDP): An MDP as outlined earlier was defined by 5 elements; S set of states, A set of actions, P state transition probability, $R(s, a)$ reward function and Υ a discounting factor.

Environment Distribution ξ: Distribution over environments where every environment $M_i = (S_i, A_i, P_i, R_i, \Upsilon)$ was drawn from the distribution ξ. In this case, the environments were drawn from the CSV data, having different state transition dynamics and rewards designs.

Policy π_θ: Agent's policy which was parametrized by θ. It defined the action selection strategy when given a state s.

Training Episodes: At each training episode t, an environment M_i was sampled from the environment distribution ξ. Thus, an agent was interacting with environment M_i during a given episode t.

Objective: The overall goal for the agent was to maximize expected cumulative reward across the training episodes.

Mathematical Formulation Training Process: Across all the training episodes T, an agent interacted with environments M as sampled from ξ

$$\mathrm{M}_i \sim \xi, i = 1, 2, ..., T \tag{10}$$

MDP Dynamics: Agent interacts with each environment M_i during episodes t followed the standard RL setup

$$\mathrm{s}_t \sim S_i, a_t = \pi_0(s_t, r_t = R_i(s_t, at), s_{t+1} \sim P_i(s_{t+1}|s_t, a_t) \tag{11}$$

Expected Return: Agent's objective was to maximize the expected cumulative reward which represents the expected return $J(0)$ over multiple environments.

$$J(\theta) = \mathbb{E}_{M_i \sim E}\left[\sum_{t=0}^{T} \gamma^t r_t\right] \tag{12}$$

4 Experiments

All the algorithms were developed using PyTorch because of its flexibility and seamless integration with CUDA (easy acceleration with GPU). The baseline agents were also implemented using Stable Baseline3 library. Most of the training and comparison of agents was done on a personal computer with the following specifications: 11th generation Intel core i7-11850H x 16 processor and a NVIDIA T600 GPU with 32 gigabyte of RAM and 1 terabyte of storage space.
The key hyperparameters for the algorithms used are highlighted in table 1.

4.1 Running Experiments

In the two test environments, BeshaGym (the financial market) and CropGym (nitrogen application on winter wheat), two general experiments were conducted; ***Baseline Experiment*** and ***Domain Randomization Experiment***.

Baseline Experiment: In this first experiment, the DQN and PPO algorithms were trained on the environments without any domain randomization. This classical deployment was done to establish a baseline to test the effectiveness of domain randomization.

For BeshaGym, only one environment was used and it was sourced from the XAUUSD dataset. This meant, the algorithms were exposed to the same environment across all the training episodes. Conversely, for CropGym, a single region was defined in the training location. CropGym connects to environmental and geographical data providers like the NASA Power project which makes it easier to source the necessary training data [41]. Therefore, like BeshaGym, the training environment was consistent across all the training episode for the first set of experiments.

Domain Randomization Experiment: Thereafter, the second experiment revolved around domain randomization as highlighted in section 3 above. For BeshaGym, the implementation of domain randomization was practically done by randomizing the selection of CSV datasets (i.e. EURUSD, GBPUSD, USDCNY, USDJPY, XAGUSD and XAUUSD). To ensure agents had access to all CSV files, and hence the resultant environments, the randomization had a no-repeat

Table 1: Default Hyperparameters for DQN and PPO

Hyperparameter	**BeshaGym**		**CropGym**	
	DQN	**PPO**	**DQN**	**PPO**
learning_rate	1e-2	1e-2	1e-4	2.5e-4
buffer_size	200000	–	–	–
batch_size	64	64	64	256
gamma	0.995	0.995	0.99	0.999
exploration_fraction	0.2	–	0.1	–
exploration_initial_eps	1.0	–	1.0	–
exploration_final_eps	0.06	–	0.01	–
target_update_interval	2000	–	–	–
train_freq	5	–	–	–
learning_starts	2000	–	–	–
max_grad_norm	20	20	10	0.7
n_steps	–	–	–	512
ent_coef	–	–	–	0.01
clip_range	–	–	–	0.1
n_epochs	–	–	–	4
gae_lambda	–	–	–	0.99
vf_coef	–	–	–	0.5
net_arch	–	–	[256, 256]	pi/vf:[256,256]

restriction and reshuffle. That is, agents had access to unique CSV files for every 6 episodes and all files were reshuffled thereafter.

On the other hand, CropGym provides a rudimentary means to randomize the training location where an algorithm can be trained in different locations as encoded using source files or data from providers like the NASA POWER Project. The latter method was used where five (5) different locations were used to train the algorithms by providing their longitude and latitude to the Nasa Power parameter [41]. The five locations were Ol'Kalou, Shamata, Ishiara, Meru and Nanyuki (all major agricultural areas in Kenya). Similarly, the agents were exposed to unique locations for every 5 episodes with a shuffle done thereafter.

4.2 Evaluation

To evaluate the two sets of experiments, agents were tested in two environments. The first environment was found in the training set and the second was not i.e. a random environment. For BeshaGym, this saw evaluations done on environments sourced from XAUUSD and XAGUSD (found in the training set on both experiments) and EURGBP (not on the training set). CropGym on the other

hand saw evaluation of the agents on environments sourced from the Ol'kalou region (found in the training set) and Mwea region (not in the training set).

The evaluation was primarily done using the reward obtained from the environments. In BeshaGym, this reward influenced the Profit/Loss outcomes and in CropGym, it affected the crop growth. Moreover, TensorBoard logs were used, giving further evaluation results through training/validation loss, policy gradient loss, and value loss.

4.3 Results and Discussion

BeshaGym The results as depicted in both figure 3 and Table 2 below shows DQN outperformed PPO, both in baseline and domain randomization experiments.

Domain randomization results were far better than those of the baseline results on most of the runs. In particular for DQN, the agents exposed to randomized environments had higher cumulative reward as compared to the baseline agents.

From the evaluation results highlighted in Table 2, domain randomization (DR) typically led to better reward, hence better Profit/Loss (P/L). It is however important to emphasize that PPO performed poorly across the board in the BeshaGym environment. Additionally, it is important to note that the agents got higher reward during evaluation for environments that were part of the training set. For environments not in the training set, agents in the domain randomization experiments had higher reward as compared to the baseline agents.

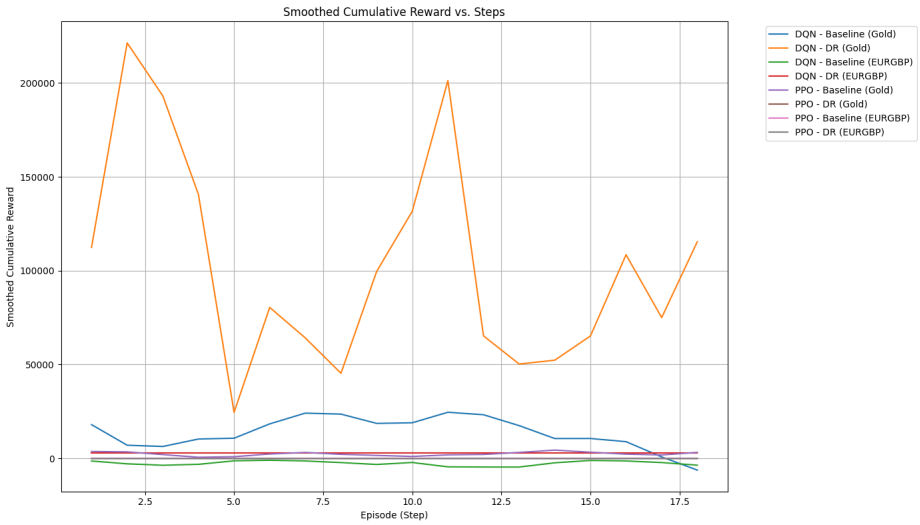


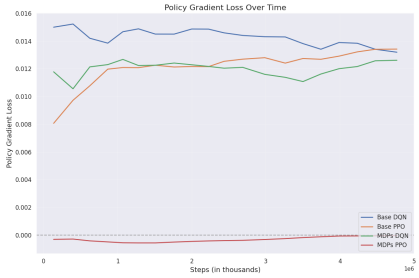
Fig. 3: BeshaGym Cumulative Reward

Table 2: BeshaGym Evaluations

	Cumulative Reward				
Evaluation Episodes	**Ep1**	**Ep2**	**Ep3**	**Ep4**	**Ep5**
			DQN		
Baseline (Gold)	41998.0	2769.9	9162.5	9162.5	701.0
DR (Gold)	18310.0	101000.0	218000.0	345000.3	16450.0
Baseline (EURGBP)	1548 .0	-711.8	-4988.9	-2976.9	-2979.0
DR (EURGBP)	2982.0	2982.0	2982.0	2982.0	2982.0
			PPO		
Baseline (Gold)	1254.9	4873.1	4999.1	540.8	529.8
DR (Gold)	-4.9	-4.9	8.9	0	0
Baseline (EURGBP)	0	0	0	0	0
DR (EURGBP)	-96.6	-91.6	-91.6	-62.5	-60.5

These results were further emphasized by the tensorboard logs, where for DQN the training loss decreased more smoothly in the domain randomization experiments (Figure 4a). For the agents in the baseline experiments, their training loss had random spikes across various training iterations, although, it was able to decrease towards the end. In comparison, the agents in domain randomization experiments saw a smooth decline in the training loss, across various training iterations. For PPO, both baseline and domain randomization agents saw the training loss decrease smoothly across the training period, however, both failed to get reasonable changes on the policy gradient loss (Figure 4b) (even with hyper-parameter tuning) which could explain the poor performance of PPO in the financial market environment.

(a) Training Loss: Baseline Vs Domain Randomization



(b) Policy Gradient Loss: Baseline Vs Domain Randomization

Fig. 4: Training metrics

CropGym The training reward for agents in the domain randomization experiments was higher than that of the agents in baseline experiments as shown in Figure 5. Generally, the cumulative reward for the agents in the randomized environments was higher than that of the baseline experiments. This outcome meant that the trained policy for both DQN and PPO performed better in balancing the application of fertilizer to meet the needs of crop yield (good WSO) while minimizing any environment effects. From Figure 4, the cumulative reward of baseline PPO was lower than all other experiments. Baseline DQN had a better performance, including competing with the best agent (domain randomized DQN) in some steps. However, its cumulative reward fell towards the end which highlights the importance of exposing agents to different environments.

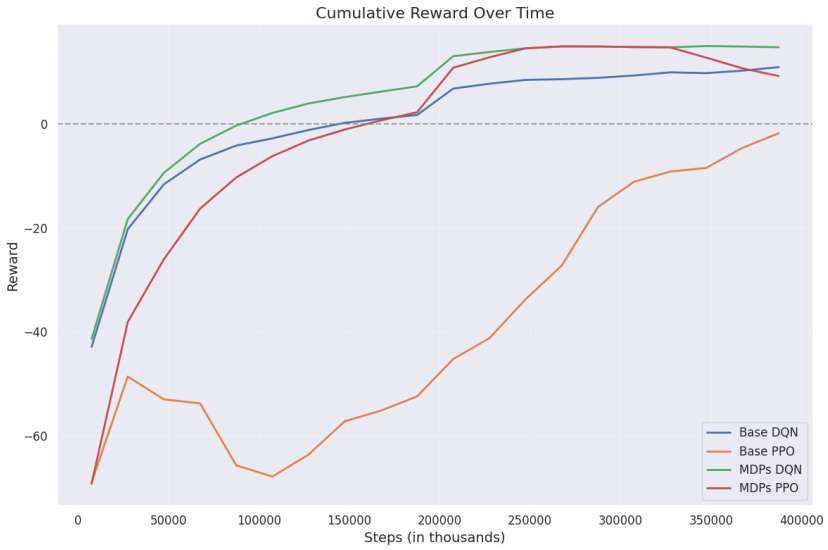


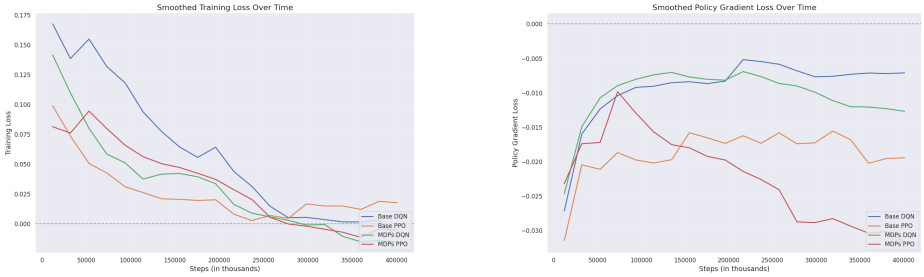
Fig. 5: CropGym Cumulative Reward: Baseline Vs Domain Randomization

On top of the cumulative reward, a subtle difference in WSO results was also observed. The WSO (weight of the storage grain/organ) saw a bigger growth for the agents in the domain randomization experiments. This is observed in Table 3 where the agents exposed to different environments had bigger WSO values.

These results were also confirmed by the tensorboard logs where in both algorithms, the training reward and loss had better outcomes for the randomized experiments (Figure 6). Higher training rewards were observed in the second set of experiments, as compared to the baseline experiments. The training loss also decreased in a smoother fashion with minimal spikes, showcasing a better training process for the second set of experiments.

Step	Base DQN	Base PPO	MDPs DQN	MDPs PPO
20500	38.90	36.62	6.21	6.11
62500	44.91	56.49	79.47	62.57
104500	29.60	65.58	80.39	33.25
146500	35.17	44.92	50.38	53.5
188500	25.15	28.69	34.33	43.94
230500	29.77	28.22	40.68	43.75
272500	36.21	28.22	39.89	43.57
314500	21.44	20.60	54.17	64.22
356500	27.76	33.22	35.69	56.35
398500	23.36	29.39	31.94	38.71

Table 3: WSO for Baseline and Domain Randomization



(a) Training Loss: Baseline Vs Domain Randomization

(b) Policy Gradient Loss: Baseline Vs Domain Randomization

Fig. 6: Training metrics

5 Conclusion

In this paper, an attempt was made to improve the generalization of deep reinforcement learning algorithms. Using domain randomization, DQN and PPO agents were deployed in two real world scenarios where the objectives were to increase profits and crop yields. These agents had improved performance when trained on different and randomized environments as compared to when they were trained on single (specific) environments.

Future works can try to address the performance problem of PPO in the financial market environment by either adjusting the environment parameters or the training specifications. To further test the effectiveness of domain randomization, different randomization techniques can be compared. Moreover, domain randomization can be compared with other techniques of generalizing DRL algorithms. For instance, modifying the training algorithms through regularization techniques, reward shaping and redefining replay buffer among many other methods. Additionally, combining these classical DRL algorithms with Transformers

and Recurrent Neural Network can be done with the same objective of trying to improve generalization.

Acknowledgments. This work was performed using HPC resources from GENCI-IDRIS (Grant 2025-[AD010614071R2]).

The researchers thank Strathmore University and LIASD Lab (University Paris 8) for invaluable academic and institutional support during the research of this paper.

The lead author gratefully acknowledges the French Embassy in Kenya for its generous support and facilitation of his doctoral studies.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Dai. T, Arulkumaran. K, Gerbert. T, Tukra. S, Behbahani. F, and Bharath. A. Analysing deep reinforcement learning agents trained with domain randomisation. *Neurocomputing*, 493:143–165, July 2022.
2. Kang. C, Chang. W, and Choi. J. Balanced domain randomization for safe reinforcement learning. *Applied Sciences*, 14(21):9710, 2024. Published: 24 October 2024.
3. Wen. X, Yu. X, Yang. R, Chen. H, Bai. C, and Wang. Z. Towards robust offline-to-online reinforcement learning via uncertainty and smoothness. *Journal of Artificial Intelligence Research*, 81:481–509, November 2024.
4. Terven. J. Deep reinforcement learning: A chronological overview and methods. *AI*, 6(3):46, 2025. Published: 24 February 2025.
5. Mnih. V and et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.
6. Nweye. K, Liu. B, Stone. P, and Nagy. Z. Real-world challenges for multi-agent reinforcement learning in grid-interactive buildings. *Energy and AI*, 10:100–202, 2022. Published online: September 2022.
7. Dulac-Arnold. G, Levine. N, Mankowitz. D, and Li. J. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021. Published online: 22 April 2021.
8. Yutaka Matsuo, Yann LeCun, Maneesh Sahani, Doina Precup, David Silver, Masashi Sugiyama, Eiji Uchibe, and Jun Morimoto. Deep Learning, Reinforcement Learning, and World Models. *Neural Networks*, 152:267–275, August 2022.
9. Andriotis. C and Papakonstantinou. K. Managing engineering systems with large state and action spaces through deep reinforcement learning. *Reliability Engineering & System Safety*, 191:106483, November 2019.
10. Huang. S, Papernot. N, Goodfellow. I, Duan. Y, and Abbeel. P. Adversarial attacks on neural network policies. `https://arxiv.org/abs/1702.02284`, 2017. Published: 8 February 2017.
11. Korkmaz. E. Nesterov momentum adversarial perturbations in the deep reinforcement learning domain. In *Proceedings of the International Conference on Machine Learning (ICML) Workshop on Inductive Biases, Invariances and Generalization in Reinforcement Learning*, 2020. Accessed: 2025-05-28.
12. Kos. J and Song. D. Delving into adversarial attacks on deep policies. `https://arxiv.org/abs/1705.06452`, 2017. Published: 18 May 2017.
13. Van Hasselt. H, Guez. A, and Silver. D. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 2094–2100. AAAI Press, 2016. Published: February 2016.
14. Wang. Z and et al. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 1995–2003. PMLR, 2016. Accessed: 2025-05-28.
15. Ezgi Korkmaz. A survey analyzing generalization in deep reinforcement learning, 2024.
16. Packer. C, Gao. K, Kos. J, Krähenbühl. P, Koltun. V, and Song. D. Assessing generalization in deep reinforcement learning. `https://arxiv.org/abs/1810.12282`, 2018. Accessed: 2025-05-28.
17. Xianjia Wang, Zhipeng Yang, Guici Chen, and Yanli Liu. A reinforcement learning method of solving markov decision processes: An adaptive exploration model based on temporal difference error. *Electronics*, 12(19):4176, October 2023.

18. Xu. T, Zhu. H, and Paschalidis. I. Learning parametric policies and transition probability models of markov decision processes from data. *European Journal of Control*, 57:68–75, 2021.
19. Nguyen. H. S, Cruz. F, and Dazeley. R. Towards a broad-persistent advising approach for deep interactive reinforcement learning in robotic environments. *Sensors*, 23(5):2681, 2023.
20. Zhang. Z, Zou. Y, Lai. J, and Xu. Q. M2dqn: A robust method for accelerating deep q-learning network. In *Proc. 2023 15th Int. Conf. Machine Learning Comput.*, pages 116–120, 2023.
21. Witty. S et al. Measuring and characterizing generalization in deep reinforcement learning. *Applied AI Letters*, 2(4):e45, 2021. Published: November 2021.
22. Schulman. J et al. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897. PMLR, 2015. Accessed: 2025-05-28.
23. Kakade. S. M. *On the Sample Complexity of Reinforcement Learning.* Phd thesis, University College London, 2003. Accessed: 2025-05-28.
24. Strehl. A, Li. L, and Littman. M. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(11):2413–2430, 2009.
25. Nichol. A et al. Gotta learn fast: A new benchmark for generalization in rl. `https://arxiv.org/abs/1804.03720`, 2018. Published: 10 April 2018.
26. Zhang. C, Vinyals. O, Munos. R, and Bengio. S. A study on overfitting in deep reinforcement learning. `https://arxiv.org/abs/1804.06893`, 2018. Published: 18 April 2018.
27. Nouri. A, Littman. M, Li. L, Parr. R, Painter-Wakefield. C, and Taylor. G. A novel benchmark methodology and data repository for real-life reinforcement learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*. Association for Computing Machinery, 2009. Accessed: 2025-05-28.
28. Kolb. L, Panzer. M, and Gronau. N. Assessing generalizability in deep reinforcement learning based assembly: A comprehensive review. *Journal of Intelligent Manufacturing*, 35(1):1–19, 2024. Published: 25 December 2024.
29. Nair. A et al. Massively parallel methods for deep reinforcement learning. `https://arxiv.org/abs/1507.04296`, 2015. Published: 15 July 2015.
30. Hausknecht. M and Stone. P. The impact of determinism on learning atari 2600 games. In *Proceedings of the AAAI Workshop on Learning for General Competency in Video Games*, Austin, Texas, USA, January 2015. AAAI Press. Accessed: 2025-05-28.
31. Kenny Young, Aditya Ramesh, Louis Kirsch, and Jürgen Schmidhuber. The benefits of model-based generalization in reinforcement learning. *arXiv*, 2022. Preprint, accessed 2025-05-28.
32. Kaidanov. O, Al-Hafez. F, Suvari. Y, Belousov. B, and Peters. J. The role of domain randomization in training diffusion policies for whole-body humanoid control. `https://arxiv.org/abs/2411.01349`, 2024. Published: 2 November 2024.
33. Roostaee. M and Abin. A. Forecasting financial signal for automated trading: An interpretable approach. *Expert Systems with Applications*, 211:118570, 2023. Published: 2023.
34. Getoor. R and Sharpe. B. Markov properties of a markov process. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 55(3):313–330, 1981. Published: January 1981.

35. Sutton. R and Barto. A. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, 2nd edition, 2018. Published: November 13, 2018.
36. Overweg. H, Berghuijs. H, and Athanasiadis. I. Cropgym: A reinforcement learning environment for crop management, 2021.
37. Mnih. V et al. Playing atari with deep reinforcement learning. `https://arxiv.org/abs/1312.5602`, 2013. Published: 19 December 2013.
38. Schulman. J et al. Proximal policy optimization algorithms. `https://arxiv.org/abs/1707.06347`, 2017. Accessed: 2025-05-28.
39. Amin. S. Deep q-learning (dqn). `https://medium.com/@samina.amin/deep-q-learning-dqn-71c109586bae`, 2024. [Online]. Available: `https://medium.com/@samina.amin/deep-q-learning-dqn-71c109586bae`.
40. Anwar. M, Wang. C, de Nijs. F, and Wang. H. Proximal policy optimization based reinforcement learning for joint bidding in energy and frequency regulation markets. In *2022 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2022. Accessed: Mar. 04, 2025.
41. Stackhouse. P. Power | dav.., 2025. [Online]. Accessed: Jan. 15, 2025. Available: `https://power.larc.nasa.gov/`.