Domain Randomization in Robotic Control

Yasaman Golshan
Politecnico di Torino
s328324@studenti.polito.it

Sara Asadi Khomami
Politecnico di Torino
s328885@studenti.polito.it

Francesco Giuseppe Gillio
Politecnico di Torino
s305909@studenti.polito.it

***Abstract*—Nowadays, a significant challenge in the field of Reinforcement Learning is its difficulty in being applied effectively to robotics. This is primarily because of the complexities involved in learning within real-world environments and accurately simulating physics. In this report, we investigate several algorithms and additionally explore an advanced technique of Domain Randomization. Our aim is to understand how this method addresses challenges encountered by previous approaches and to highlight its advantages and limitations.**

I. Introduction

Since the goal of this project is to evaluate various proposed solutions to the Sim-to-Real challenge in Reinforcement Learning, this section outlines the key concepts of this learning framework. The code for the implemented methods is available in the project repository [1].

A. Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning dedicated to the training of agent that sequentially interact with the environment to make optimal decisions with the objective of maximizing a cumulative numerical reward, R_t. Unlike supervised learning, where model is trained on a static dataset of labeled examples, RL involves agent collects observations on the environment's current state, S_t, which guides the actions A_t it takes. These actions are followed by a particular strategy known as the policy $\pi(\cdot)$. This learning paradigm is inspired by behavioral psychology and has applications in various domains, such as robotics, game playing, autonomous driving, and financial trading.

This process involves a delicate balance between exploration, where the agent tries new actions to discover their effects, and exploitation, where the agent chooses actions that are known to yield high rewards.

Reinforcement Learning can address very complex tasks, making it an effective learning framework for robots. However, training in the real world can be time-consuming, expensive, and potentially hazardous. To mitigate these issues, training can be accelerated using a simulator, which approximates the environment's dynamics. The discrepancy between real-world dynamics and simulation is known as the reality gap, which can cause the model to learn only the simulator's approximate dynamics and perform poorly in the real environment. The challenge of accurately transferring experience from simulation to the real world is called Sim-to-Real. This challenge involves training policies that can adapt to the real environment while being trained in a different one.

In the context of Sim-to-Real, this project focuses on Domain Randomization (DR), which enhances policy robustness by exposing the agent to varied situations during training. By randomly varying the environment's parameters, a collection of slightly different environments is created for the simulator. This approach helps prevent overfitting to the simulation. This report discusses two versions of DR: Uniform DR in Section III and Adaptive DR in Section IV.

For this project, the randomized parameters were the masses of body parts. Due to practical constraints, we did not transfer the experience to the real world but simulated this process through Sim-to-Sim transfer. The source environment is where training occurs, and the target environment is where the policy is transferred. The two environments differ only in the mass of the torso.

B. Simulation Environment

The environment considered for this project is the Hopper environment from the Gym API [2], which features a one-legged robot consisting of a torso, thigh, leg, and foot (see Figure 1). The goal for the Hopper is to learn to hop forward without falling. Movement is achieved by applying torques to the three hinges connecting the body parts. Once the Hopper has fallen, that would consider as a failure.

The article focuses on a setup where the environment is a simulated flat world containing the robot. In this source environment, the robot's body parts have the following masses: torso at 2.53 kg, thigh at 3.93 kg, leg at 2.71 kg, and foot at 5.09 kg. In the target environment, the only difference is that the torso weighs 3.53 kg. The agent controls the robot based on the observations received about the robot's positions and velocities and decides on the torques to apply at the joints to maintain balance and move forward.

- The reward structure consists of three parts:
  1) **Healthy Reward**: The robot earns a reward of +1 for each time step it remains active.
  2) **Forward Reward**: A reward of hopping forward is measured as weight $\times (x_{t+1} - x_t) / \mathrm{dt}$, where dt is the time between actions. This reward is positive if the hopper moves forward in the positive x-direction.
  3) **Control Cost**: Cost for penalizing the hopper if it takes actions that are too large.

The **total reward** returned is reward = healthy reward + forward reward − control cost.

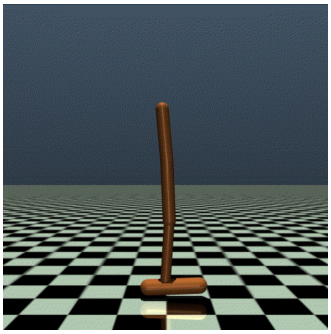

Fig. 1: Hopper environment

II. Methodology

Reinforcement Learning algorithms can be broadly classified into two main types: model-free and model-based. In this section, we focus on model-free algorithms, which aims to acquire an effective behavior policy through trial and error interaction with a black box environment [3].

A. Reinforce

REINFORCE [4] is a policy optimization algorithm (Algorithm 1), which utilizes an estimate of the total reward received throughout an entire episode to calculate the policy gradient. In our approach, a Multi-Layer Perceptron with 3 layers and the tanh activation function were used for training the policy.

Algorithm 1 REINFORCE

Input A differentiable policy parameterization $\pi_\theta(A|S)$
Algorithm parameter Step size $\alpha > 0$
Initialize The policy parameters $\theta \in \mathbb{R}^d$ at random.
for Loop forever (for each episode) do
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ following π_θ
  $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
  $\theta \leftarrow \theta + \alpha \cdot \frac{1}{T} \sum_{t=0}^{T} \gamma^t G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$
end for

For making improvement, A common used method is to subtract a baseline value from the return to reduce the variance of gradient estimation while keeping the bias unchanged. However, this method relies on choosing an appropriate baseline, which can be challenging in complex environments. Whitening offers an alternative approach, by normalizing the returns of each episode step by subtracting the mean and dividing by the standard deviation across all time steps within the episode [5]. This process essentially removes correlations between features, aiming to improve learning efficiency and potentially lead to better policy performance.

We achieved these results(TABLE I) by comparing different versions of the REINFORCE algorithm. Each version was trained and tested in the source environment over 25,000 episodes, with a 50-episode test to evaluate performance.

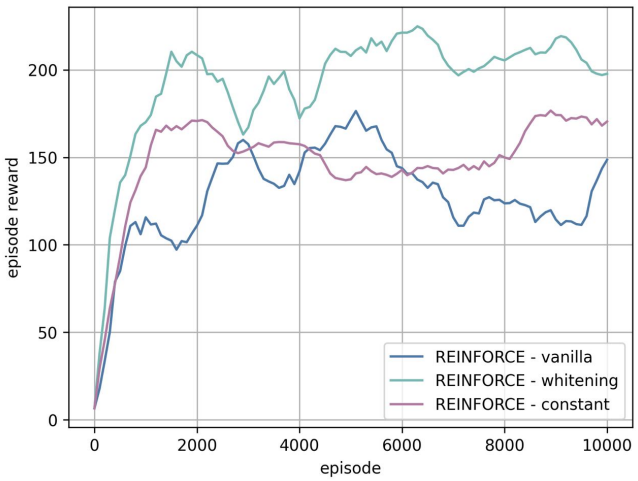

Fig. 2: RINFORCE for 10000 in 5 environment

The table shows the mean returns from these experiments, illustrating how each algorithm variation affected performance.

Baseline	Return
-	147
constant(20)	169
withening	213

TABLE I: REINFORCE versions comparison

B. Actor-Critic

Actor-Critic methods represent a powerful paradigm that leverages the combined strengths of policy gradient and value-based approaches. These methods employ two distinct but interconnected components: the actor and the critic.

- **Critic**: The critic updates the parameters ($\mathbf{w}$) of the value function, which can be either:
  - **Action-value function (Q-value)**: $Q(s, a; \mathbf{w})$ represents the expected future reward, starting from state s and taking action a, parameterized by $\mathbf{w}$.
  - **State-value function (V-value)**: $V(s; \mathbf{w})$ represents the expected future reward starting from state s, parameterized by $\mathbf{w}$.
- **Actor**: The actor updates its policy parameters ($\boldsymbol{\theta}$) in the direction suggested by the critic's value estimate. This update aims to maximize the expected future reward based on the current policy. [7]

Upon implementation, however, we found that the results resembled those of the Reinforce algorithm. High variance in performance gradients can arise due to the Q-value. Actor-Critic methods address this by introducing the advantage function(A). Defined as $A(s, a) = Q(s, a; w) - V(s; w)$, it removes the baseline state-value (V) from the action-value (Q), focusing the critic's feedback on advantageous actions. This simplifies learning for the actor, leading to more efficient policy updates. This is the approach called Advantage Actor-Critic (A2C) algorithm [7].

Algorithm 2 ACTOR-CRITIC

Input A differentiable policy parameterization $\pi_\theta(A|S)$
Input A differentiable state-value function $\hat{V}_w(S)$
Parameters: Step sizes $\alpha^\theta > 0, \alpha^w > 0$
Initialize The policy parameters $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$
for Loop forever (for each episode) do
  Initialize S_0 (first state of episode)
  $I \leftarrow 1$
  while S_t is not terminal (for each time step t) **do**
    $A \sim \pi_\theta(\cdot|S_t)$
    Take action A, observe S_{t+1}, R
    $\delta \leftarrow R + \gamma \hat{V}_w(S_{t+1}) - \hat{V}_w(S_t)$
    $w \leftarrow w + \alpha^w \delta \nabla_w \hat{V}_w(S_t)$
    $\theta \leftarrow \theta + \alpha^\theta I \delta \nabla_\theta \ln \pi_\theta(A|S_t)$
    $I \leftarrow \gamma I$
    $S_t \leftarrow S_{t+1}$
  end while
end for

Hyperparameters	REINFORCE
Learning Rate	{**1e-3**, 7.5e-4, 5e-4, 2.5e-4, 1e-4}
Gamma	{0.97, **0.99**}
Hidden	{**64**, 128, 256}

TABLE II: Grid Search for REINFORCE

Hyperparameters	A2C
Learning Rate	{**1e-3**, 7.5e-4, 5e-4, 2.5e-4, 1e-4}
Gamma	{0.97, **0.99**}
Hidden	{**64**, 128, 256}
Batch Size	{8, 16, **32**, 64}
Critic Coefficient	{0.25, **0.5**, 0.75}
Entropy Coefficient	{0.0, **0.25**, 0.5, 0.75}

TABLE III: Grid Search for A2C

Hyperparameters	PPO
Learning Rate	{1e-3, 7.5e-4, 5e-4, **2.5e-4**, 1e-4}

TABLE IV: Grid Search for PPO

C. Proximal Policy Optimization

Proximal Policy Optimization (PPO) is an algorithm that trains a stochastic policy in an on-policy manner. PPO balances exploration and exploitation and features two variants: PPO-Penalty and PPO-Clip. For our implementation, we focus on PPO-Clip, which uses a clipping mechanism in the objective function to prevent large deviations from the previous policy, promoting stable and effective learning dynamics. PPO-Clip does not include a KL term or hard constraints, relying instead on clipping to maintain policy stability [9].

For our implementation, we utilize the Stable Baselines3 library and conduct a grid search to optimize the learning rate for enhanced performance. To establish performance bounds, we trained the PPO on the source environment and tested it on the target environment to define a lower bound. For the upper bound, we trained and tested it on the target environment. These bounds provide a basis for meaningful comparisons in the subsequent domain randomization sections. The results can be seen in fig 3 .

III. Domain Randomization

Domain randomization (DR) represents a technique for enhancing the robustness and generalization of policies trained in simulated environments. The objective of domain randomization involves introducing enough simulated variability at training time, enabling the model to generalize to real-world data during testing [10]. By exposing the agent to a variety of different scenarios during training, DR helps bridge the sim-to-real gap, ensuring the learned policy performs effectively in real-world settings. This section provides an overview of two variants of Domain Randomization: Uniform Domain Randomization (UDR) and Domain Randomization Optimization IDentification (DROID).

A. Uniform Domain Randomization

Uniform Domain Randomization (UDR) enhances the robustness of an agent's policy by exposing it to variety of environmental conditions during training. In this project, we implemented UDR on the link masses of the Hopper robot, specifically targeting the masses of the thigh, leg, and foot, while fixing the torso mass at a value different from the target environment by -1 kg. The torso mass accounts the discrepancy between the simulation environment and the real-world.

To implement UDR, we designed uniform distributions for each of the three link masses. During each training episode, the values of these masses were drawn from their respective distributions. Specifically, for each link i, the environment initializes the boundaries of the uniform distribution $\mathcal{U}_\phi$ and samples the value of the physical parameter θ_i at the beginning of each episode:

$$\theta_i \sim \mathcal{U}_\phi((1-\phi)\cdot\theta_i^{(0)}, (1+\phi)\cdot\theta_i^{(0)})$$

where $\theta_i^{(0)}$ represents the original mass of the i-th link of the Hopper robot, and ϕ the variation factor. Here, $\mathcal{U}_\phi$ represents a continuous uniform distribution between $(1-\phi)\cdot\theta_i^{(0)}$ and $(1+\phi)\cdot\theta_i^{(0)}$. This approach forces the agent to adapt its policy in order to achieve satisfactory performance across environments with varying dynamics. UDR aims to train the agent to maximize its reward under different conditions, ensuring resilience to environmental variations. As the choice of distribution acts as a hyperparameter, we manually tested various distributions to determine the optimal variation factor ϕ that yields the best performance in the target environment.

Hyperparameters	UDR
Variation Factor (ϕ)	{0.25, **0.5**, 0.75}

TABLE V: Grid Search for UDR

B. Domain Randomization Optimization IDentification

Domain Randomization Optimization IDentification (DROID) constitutes an advanced approach in reinforcement learning aimed at refining simulation environments to enhance the robustness and transferability of learned policies. In contrast to traditional Uniform Domain Randomization, which introduces stochastic variations to simulation parameters uniformly, DROID focuses on iteratively adjusting these parameters to closely approximate real-world dynamics [14] [15]. This section presents a comprehensive overview of DROID, detailing its objectives, methodology, and computational framework.

1) Objective: DROID attempts to address the sim-to-real gap inherent in reinforcement learning by optimizing the fidelity of simulation environments. By iteratively adjusting physical parameters such as masses within the simulation, DROID aims to minimize the discrepancy between the distributions of trajectories observed in simulation ($\mathcal{D}_{\text{sim}}(\theta)$) and those in the real world ($\mathcal{D}_{\text{real}}$). This optimization process significantly enhances the adaptability and generalization capabilities of reinforcement learning policies, enabling them to perform effectively across varying real-world conditions.

2) Problem Formulation: The algorithm efforts to minimize the Wasserstein distance W between the real-world trajectory distribution $\mathcal{D}_{\text{real}}$ and the simulated trajectory distribution $\mathcal{D}_{\text{sim}}(\theta)$. Initially, DROID sets θ to an initial set of physical parameters $\theta^{(0)} = \{\theta_{\text{torso}}, \theta_{\text{thigh}}, \theta_{\text{leg}}, \theta_{\text{foot}}\}$ that govern the simulation dynamics. This setup establishes an initial simulation data distribution $\mathcal{D}_{\text{sim}}(\theta^{(0)})$ from which iterative improvements begin. The algorithm proceeds through M iterations, refining θ to progressively minimize the discrepancy:

Parameter Update: For each parameter θ_i in θ:

- Perturb θ_i and update the simulation environment.
- Compute the Wasserstein distance $W(\mathcal{D}_{\text{real}}, \mathcal{D}_{\text{sim}}(\theta_i))$ between the real-world and updated simulated distributions.
- Calculate the gradient ∇_i of the distance metric with respect to θ_i, facilitating parameter adjustments.
- Clip parameter values to maintain stability and prevent divergence.

Distribution Update: Update $\mathcal{D}_{\text{sim}}(\theta)$ based on the refined parameters after each iteration, ensuring that the simulation increasingly approximates real-world conditions.

The iterative refinement process outlined in Algorithm 4 ensures that DROID systematically adjusts simulation parameters to minimize the discrepancy between simulated and real-world trajectory distributions. This approach enhances the realism of the simulation, thereby improving the effectiveness and adaptability of learned policies in practical applications.

3) Mathematical Foundation: The Wasserstein distance $W(\mathcal{D}_{\text{real}}, \mathcal{D}_{\text{sim}}(\theta_i))$ measures the discrepancy between the real-world trajectory distribution $\mathcal{D}_{\text{real}}$ and the simulated trajectory distribution $\mathcal{D}_{\text{sim}}(\theta)$ [16].

$$W(\mathcal{D}_{\text{real}}, \mathcal{D}_{\text{sim}}(\theta)) = \inf_{\gamma \in \Gamma(\mathcal{D}_{\text{real}}, \mathcal{D}_{\text{sim}}(\theta))} \mathbb{E}_{(s,a)\sim\gamma}[c(s,a)],$$

$\Gamma(\mathcal{D}_{\text{real}}, \mathcal{D}_{\text{sim}}(\theta))$ represents the set of joint distributions $\gamma(s, a, s', a')$ with marginals $\mathcal{D}_{\text{real}}$ and $\mathcal{D}_{\text{sim}}(\theta)$, and $c(s, a)$ denotes the cost function, quantifying the dissimilarity between state-action pairs (s, a) in the real and simulated environments. This formulation enables Wasserstein distance to consider not only statistical moments but also the spatial arrangement of trajectories, crucial for accurately assessing and minimizing the discrepancy between simulated and real-world dynamics. By minimizing $W(\mathcal{D}_{\text{real}}, \mathcal{D}_{\text{sim}}(\theta))$, DROID drives its iterative optimization process to refine simulation parameters, aligning two-dimensional trajectory distributions and enhancing simulation fidelity. This approach aims to closely align the simulated environment with real-world dynamics, thereby improving the adaptability of reinforcement learning policies across diverse real-world conditions.

Algorithm 3 DROID

```
Initialize parameters θ = θ^(0)
Collect D_real = {(s_i, a_i)}_{i=1}^N
Collect D_sim(θ) = {(s'_j, a'_j)}_{j=1,θ}^N
for m = 0, 1, 2, ..., M do
    Compute base loss b = W(D_real, D_sim(θ))
    for each θ_i ∈ θ do
        Perturb θ_i ← θ_i + η
        Collect D_sim(θ_i) = {(s'_j, a'_j)}_{j=1,θ_i}^N
        Compute loss L_i = W(D_real, D_sim(θ_i))
        Compute gradient ∇ = ∇_{θ_i} L_i
        Clip parameters θ_i ← clip(θ_i, 0.01, 10.0)
    end for
    Update simulation data D_sim(θ) ← {(s'_j, a'_j)}_{j=1,θ}^N
end for
```

The systematic adjustment of simulation parameters in DROID aligns with the foundational objective of reducing the domain gap between simulation and reality in reinforcement learning.

In the fine-tuning process, we searched through various learning rates η to optimize performance. Among the several values $\{1e{-}3, 5e{-}3, 5e{-}4, 1e{-}4\}$, an optimal configuration returns a learning rate η set at $1e-4$. This choice appears pivotal as it balances the gradient descent efficiency with stability, crucial for the iterative adjustment of simulation parameters.

Figure 3 presents a comparison of the performance trends of the PPO model in the target environment across different

Hyperparameters	DROID
Learning Rate (η)	{1e-3, 5e-3, 5e-4, **1e-4**}

TABLE VI: Grid Search for DROID

Fig. 3: Comparison of PPO model performance trends in the target environment across different training configurations.

training configurations. The graph shows the average rewards for three scenarios: training in the source environment (lower bound), training in the target environment (upper bound), training in the source environment with Uniform Domain Randomization (UDR), and training in the source with DROID configuration. Each configuration results from parallel training in three different environments for 25,000 episodes with different seeds. The points on each trend show the average reward over 50 evaluation episodes in the target environment, sequentially every 100 training episodes in the respective scenario. The results demonstrate that using UDR leads to a higher average reward than the configuration without randomization, indicating performance improvement in the target environment due to UDR. Furthermore, the graph illustrates that training with the DROID configuration outperforms standard Uniform Domain Randomization. DROID achieves this by optimizing parameter randomization, significantly reducing the discrepancy in sim-to-real transfer. This improvement underscores DROID's effectiveness in enhancing performance and reducing the domain gap between simulation and real-world conditions.

IV. Conclusion

In conclusion, this report delves into the complexities of applying Reinforcement Learning (RL) to robotics, focusing specifically on addressing the Sim-to-Real challenge. The study aims to investigate the efficacy of Domain Randomization (DR) techniques in enhancing policy robustness and transferability across different environments. Through the exploration of various RL algorithms such as REINFORCE, Actor-Critic, and Proximal Policy Optimization (PPO), our research highlights their respective strengths and applications in training agents to perform tasks in simulated environments. The study evaluates each algorithm based on its ability to optimize policies effectively within the context of a Hopper robot tasked with hopping forward. The core contribution of this report lies in the comprehensive analysis of two DR techniques: Uniform Domain Randomization (UDR) and Domain Randomization Optimization IDentification (DROID). UDR involves varying the masses of the robot's body parts uniformly, whereas DROID introduced an iterative optimization process to tailor simulation parameters for closer alignment with real-world dynamics. UDR demonstrated its effectiveness in enhancing policy resilience by exposing the agent to diverse environmental conditions during training. This approach aimed to mitigate the "reality gap" between simulation and real-world scenarios, thereby improving the agent's adaptability. On the other hand, DROID represented an advanced strategy that iteratively adjusted simulation parameters to minimize the discrepancy between simulated and real-world trajectories. By optimizing simulation fidelity, DROID significantly enhanced policy performance and transferability, as evidenced by superior results compared to traditional UDR.

Overall, this study underscores the importance of DR techniques in bridging the gap between simulated and real-world environments for reinforcement learning in robotics. Future research could explore further refinements to DR methodologies and their integration with state-of-the-art RL algorithms, ultimately advancing the field towards more robust and practical robotic applications.

References

[1] Yasaman Golshan, Sara Asadi, and Francesco Giuseppe Gillio. Domain Randomization in Robotic Control. URL: https://github.com/305909/gym-hopper.

[2] Greg Brockman et al. "Openai gym". In: arXiv preprint arXiv:1606.01540 (2016).

[3] Nachum, Ofir, et al. "Bridging the gap between value and policy based reinforcement learning." Advances in neural information processing systems 30 (2017).

[4] Sutton, Richard S., and Andrew G. Barto."Reinforcement learning: An introduction." MIT press, 2018.

[5] Fork Tree. "Understanding Baseline Techniques for REINFORCE." Medium, Oct. 2019. URL: https://medium.com/@fork.tree.ai/understanding-baseline-techniques-for-reinforce-53a1e2279b57..

[6] OpenAI. "Proximal Policy Optimization." OpenAI Spinning Up, 2018. URL: https://spinningup.openai.com/en/latest/algorithms/ppo.html#references.

[7] Lilian Weng. Policy Gradient Algorithms. Apr. 2018. URL: https://lilianweng.github.io/posts/2018-04-08-policy-gradient/

[8] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013). ————

[9] OpenAI. "Proximal Policy Optimization." OpenAI Spinning Up, 2018. URL: https://spinningup.openai.com/en/latest/algorithms/ppo.html#references.

[10] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World." arXiv, Mar. 20, 2017

[11] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[12] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[13] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.
[14] Lilian Weng, "Domain Randomization for Sim2Real Transfer." May 5, 2019. URL: https://lilianweng.github.io/posts/2019-05-05-domain-randomization/.
[15] Ya-Yen Tsai, Hui Xu, Zihan Ding, Chong Zhang, Edward Johns, Bidan Huang, "DROID: Minimizing the Reality Gap using Single-Shot Human Demonstration", Feb 22, 2021. URL: https://arxiv.org/abs/2102.11003.
[16] Mohammed Amin Abdullah, Aldo Pacchiano, Moez Draief, "Reinforcement Learning with Wasserstein Distance Regularisation, with Applications to Multipolicy Learning", Feb 12, 2018. URL: https://arxiv.org/abs/1802.03976.