IEEE Robotics and Automation Letters (RAL) paper presented at the
2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Invariant Transform Experience Replay: Data Augmentation for Deep Reinforcement Learning

Yijiong Lin[1], Jiancong Huang[1], Matthieu Zimmer[2], Yisheng Guan[1], Juan Rojas[3], Paul Weng[2,4]

***Abstract*— Deep Reinforcement Learning (RL) is a promising approach for adaptive robot control, but its current application to robotics is currently hindered by high sample requirements. To alleviate this issue, we propose to exploit the symmetries present in robotic tasks. Intuitively, symmetries from observed trajectories define transformations that leave the space of feasible RL trajectories invariant and can be used to generate new feasible trajectories, which could be used for training. Based on this data augmentation idea, we formulate a general framework, called Invariant Transform Experience Replay that we present with two techniques: (i) Kaleidoscope Experience Replay exploits reflectional symmetries and (ii) Goal-augmented Experience Replay which takes advantage of lax goal definitions. In the Fetch tasks from OpenAI Gym, our experimental results show significant increases in learning rates and success rates. Particularly, we attain a 13, 3, and 5 times speedup in the pushing, sliding, and pick-and-place tasks respectively in the multi-goal setting. Performance gains are also observed in similar tasks with obstacles and we successfully deployed a trained policy on a real Baxter robot. Our work demonstrates that invariant transformations on RL trajectories are a promising methodology to speed up learning in deep RL. Code, video, and supplementary materials are available at [1].**

I. INTRODUCTION

Deep Reinforcement Learning (RL) has demonstrated great promise in recent years [2], [3]. However, despite being shown to be a viable approach in robotics [4], [5], deep RL still suffers from significant low sample efficiency in practice—an acute issue in robot learning.

A natural approach to deal with this issue is to better exploit the actual samples generated during learning. Indeed, this is one of the motivations behind experience replay (ER) [6] and hindsight experience replay (HER) [7]. In ER, interactions with the environment are stored in a replay buffer and can then be reused multiple times for training. HER extends this idea to multi-task settings in order to take advantage of failed trajectories (*i.e.*, sequences of interactions between the robot and its environment). Its basic principle is to construct successful trajectories from failed ones by changing the unachieved (original) goals to artificial goals achieved by the failed sequences.

Corresponding author: Paul Weng.
[1]Y. Lin, J. Huang and Y. Guan are with School of Electromechanical Engineering, Guangdong University of Technology, P.R. China, {2111701025,jiancong.huang}@mail2.gdut.edu.cn, ysguan@gdut.edu.cn.
[2]M. Zimmer and P. Weng are with UM-SJTU Joint Institute, Shanghai Jiao Tong University, P.R. China, {matthieu.zimmer,paul.weng}@sjtu.edu.cn.
[3]J. Rojas is with School of Mechanical and Automation Engineering, Chinese University of Hong Kong, P.R. China, rojas70@gmail.com.
[4]P. Weng is with Department of Automation, Shanghai Jiao Tong University, P.R. China.

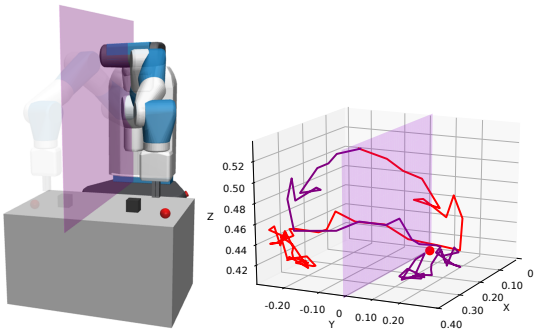


Fig. 1. Left: Kaleidoscope Experience Replay leverages natural symmetry. Feasible trajectories are reflected on the plane xoz. The latter can itself be rotated by some θ_z along axis $\vec{z}$. Right: A symmetrical trajectory (purple) is reflected from the observed trajectory (red) via the purple plane xoz. The red point denotes the robot base in the right plot.

In this work, we advance in this direction by using symmetries to generate novel feasible artificial trajectories for training from observed ones. We use *symmetry* in its mathematical sense. In our context, it is any transformation that leaves the space of feasible trajectories invariant. If many such transformations are used, one observed trajectory, which is costly to collect, can cheaply produce many artificial samples for training, leading to much more efficient algorithms in terms of true samples, which is an important factor in robotics.

As a basic illustration of such transformation consider a robotic manipulation task (see the left of Fig. 1 for illustration) where the robot may interact with some objects. The reflection with respect to the purple plane induces a transformation that maps any sequence of interactions recorded during learning to a new feasible trajectory, which can then also be used for training (see the right of Fig. 1 depicting the reflection applied to the path of the gripper). This transformation is naturally also applied to the goal achieved by the considered trajectory (and any other state relevant information, *e.g.*, object positions). The intuition is that if that trajectory has achieved a goal g, then its transformed trajectory defines a feasible sequence of controls that achieves the symmetrical reflection of g. Interestingly, such a transformation preserves any contact the robot may have with its environment if the transformation is applied to all the objects and obstacles in the robot's workspace.

The idea of using reflections and more general symmetries

(see Related Work in Sec. II) to expand the original training data is the basis of many data augmentation techniques in deep learning, but has been scarcely investigated in deep RL to the best of our knowledge.

In this paper, we propose a general framework for data augmentation in deep RL, which extends ER and HER, called Invariant Transform Experience Replay (ITER) where a transformation can be applied either on trajectories entering the replay buffer or on those sampled from it. To make ITER concrete, we present it with two different such transformations. Each of them could potentially be used separately, leading to two independent data augmentation techniques.

The first technique, Kaleidoscope Experience Replay (KER), is based on reflectional symmetry. It generalizes our previous example (Fig. 1) by using multiple different reflective hyperplanes.

The second technique, Goal-augmented Experience Replay (GER), is a direct generalization of HER: any hindsight goal g generated by HER can be instead replaced by a random goal sampled from within a small ball centered around g to obtain another successful goal. This idea takes advantage of tasks where success is defined as reaching a final pose within a distance of the goal set by a threshold (such tasks are common in robotics).

The paper is organized as follows: Sec. II introduces related work on increasing data efficiency. Sec. III presents the background (deep RL and HER) for our work. Sec. IV details our general framework with two invariant transform data augmentation techniques. Sec. V describes experimental results on OpenAI Gym Fetch tasks [8], which demonstrate the effectiveness of our propositions and show significant improvements in learning speed and success rates; particularly for robotic manipulation tasks with and without obstacles (see Fig. 6 and Fig. 10). Sec. VI discusses concerns of interest, and Sec. VII concludes.

II. RELATED WORK

HER [7], [9] has been extended in various ways. Prioritized replay was incorporated in HER to learn from more valuable episodes with higher priority [10]. In [11], HER was generalized to deal with dynamic goals. In [12], a variant of HER was also investigated where completely random goals replace achieved goals and in [13], it was adapted to work with on-policy RL algorithms. All these extensions are orthogonal to our work and could easily be combined with ITER. We leave these for future work.

Symmetry has been considered in MDPs [14] and RL [15]–[19]. It can be known a priori or learned [17]. In this work, we assume the former, which is reasonable in many robotic tasks. A natural approach to exploit symmetry in sequential decision-making is by aggregating states that satisfy an equivalence relation induced by some symmetry [14], [15]. Another related approach takes into account symmetry in the policy representation [19]. Doing so reduces representation size and generally leads to faster solution times. However, the state-aggregated representation may be difficult to recover, especially if many symmetries are considered simultaneously. Still another approach is to use symmetry during training instead. One simple idea is to learn the Q-function by performing an additional symmetrical update [16]. Another method is to augment the training data with their reflections [18]. A dihedral group with finite invariant elements has been leveraged to implement symmetry on the state representation of board position in Go [3]. In this paper, we generalize further this idea and extend it to propose a general and theoretically-founded framework for data augmentation where different kinds of symmetry (not only reflections) can be considered.

Though the data augmentation has been used extensively and with great success in machine learning [20] and in deep learning [21], there has been little attention drawn to data augmentation in the RL domain until recently, inspired by image augmentation techniques to ease the over-fitting problem when using large neural networks [22], [23]. Interestingly, symmetries can also be exploited in neural network architecture design [24]. However, in our case, the integration of symmetry in deep networks will be left as future work.

III. BACKGROUND

In this work, we consider robotic tasks that are modeled as multi-goal Markov decision processes [25] with continuous state and action spaces: $\langle \mathcal{S}, \mathcal{A}, \mathcal{G}, T, R, p, \gamma \rangle$ where $\mathcal{S}$ is a continuous state space, $\mathcal{A}$ is a continuous action space, $\mathcal{G}$ is a set of goals, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the unknown transition function that describes the environmental dynamics, $R(s, a, s', g)$ is the immediate reward when an agent reaches state $s' \in \mathcal{S}$ after performing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ if the goal were $g \in \mathcal{G}$. Finally, $p(s_0, g)$ is a joint probability distribution over initial states and original goals, and $\gamma \in [0, 1]$ is a discount factor. In this framework, the robot learning problem corresponds to an RL problem that aims at obtaining a policy $\pi : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$ such that the expected discounted sum of rewards is maximized for any given goal.

Due to the continuity of the state-action spaces, this optimization problem is usually restricted to a class of parameterized policies. In deep RL, the parameterization is defined by the neural network architecture. To learn such continuous policies, actor-critic algorithms [26] are efficient iterative methods since they can reduce the variance of the estimated gradient using simultaneously learned value functions. DDPG (Deep Deterministic Policy Gradient) [27] is a model-free off-policy deep RL algorithm that learns a deterministic policy, which is desirable in robotic tasks. In DDPG, the transitions are collected into a replay buffer to later update the action-value function in a semi-gradient way and the policy with the deterministic policy gradient [28]. Because the policy has to adapt to multiple goals, as in HER, we rely on universal value functions [25]: the classic inputs of the value function and the policy of DDPG are augmented with the desired goal.

When the reward function is sparse, as assumed here, the RL problem is particularly hard to solve. In particular, we consider here reward functions that are described as follows:

$$R(s, a, s', g) = \mathbf{1}[d(s', g) \leq \epsilon_R] - 1 \quad (1)$$

where $\mathbf{1}$ is the indicator function, d is a distance (*e.g.*, between object position in s' and goal g), and $\epsilon_R > 0$ is a fixed threshold.

To tackle this issue, HER is based on the following principle: any trajectory that failed to reach its goal still carries useful information; it has at least reached the states of its trajectory path. Using this natural and powerful idea, memory replay can be augmented with the failed trajectories by changing their goals in *hindsight* and computing the new associated rewards.

In the robotic tasks solved by HER, the states are generally defined as $s = (s_{gri}, s_{obj}, s_{rel})$ with its components defined as follows: s_{gri} is a 8-dimensional vector containing the absolute position of the gripper $(x_{gri}, y_{gri}, z_{gri})$, its linear velocity $(x'_{gri}, y'_{gri}, z'_{gri})$, the distance and relative velocity between the gripper's fingers d_{fin}, d'_{fin} respectively. Then, s_{obj} is a 12-dimensional vector that consists of the pose of the object $(x_{obj}, y_{obj}, z_{obj}, \alpha_{obj}, \beta_{obj}, \gamma_{obj})$ and its twist $(x'_{obj}, y'_{obj}, z'_{obj}, \alpha'_{obj}, \beta'_{obj}, \gamma'_{obj})$. s_{rel} is a 3-dimensional vector representing the position of object with respect to the target position $(x_{rel}, y_{rel}, z_{rel})$. Actions are defined as $a = (x_a, y_a, z_a, d_{gri})$ where (x_a, y_a, z_a) represent the new position that the gripper should reach at the next time step and d_{grip} is the desired distance between the two fingers of the gripper. Finally, goals are defined as $g = (x_g, y_g, z_g)$ specifying the target positions of objects.

IV. INVARIANT TRANSFORMATIONS FOR RL

To reduce the number of interactions with the real environment, we propose to generate artificial training data from observed trajectories collected during the robot's learning. Some care is needed to choose a transformation to be applied on actual data to generate artificial ones, otherwise the training would be too biased. To that regard, we consider symmetries (*i.e.*, any invariant transformations) in the space of feasible trajectories.

Consider a trajectory τ of length h with goal $g \in \mathcal{G}$ as $\langle g, (s_0, a_1, r_1, s_1, a_2, r_2, s_2, \ldots, s_h)\rangle$ where $s_0 \in \mathcal{S}$, $\forall i = 1, \ldots, h$, $a_i \in \mathcal{A}$, $s_i \in \mathcal{S}$, and $r_i = R(s_{i-1}, a_i, s_i, g)$. We assume that all trajectories have a length not larger than $H \in \mathbb{N}$, which is true in robotics (*i.e.*, the length of each manipulation task is not infinite). The set of all trajectories is denoted $\overline{\Gamma} = \cup_{h=1}^{H} \mathcal{G} \times \mathcal{S} \times (\mathcal{A} \times \mathbb{R} \times \mathcal{S})^h$.

A trajectory τ is said to be *feasible* if for $i = 1, \ldots, h$, $T(s_{i-1}, a_i, s_i) > 0$. The set of feasible trajectories is denoted $\Gamma \subseteq \overline{\Gamma}$. A trajectory τ of length h with goal g is said to be *successful* if $R(\tau) > R_{\min}$ where $R(\tau) = \sum_{i=1}^{h} \gamma^{i-1} R(s_{i-1}, a_i, s_i, g)$ and $R_{\min}$ is a fixed problem-dependent threshold. In the context of sparse rewards with $R_{\min} = -1$, a successful trajectory is one that reached the goal. The set of successful trajectories is denoted $\Gamma^+ \subseteq \Gamma$.

We can now define the different notions of symmetries that we use in this paper. A *symmetry* of Γ is a one-to-one mapping $\sigma : \overline{\Gamma} \to \overline{\Gamma}$ such that $\sigma(\Gamma) = \Gamma$ where $\sigma(\Gamma) = \{\tau \in \Gamma \mid \exists \tau' \in \Gamma, \sigma(\tau') = \tau\}$. In words, a symmetry of Γ leaves the space invariant, *i.e.*, it maps feasible trajectories to feasible ones. As we only apply symmetries to feasible trajectories, we directly consider their restrictions to Γ and keep the same notation, *i.e.*, $\sigma : \Gamma \to \Gamma$.

A *decomposable* symmetry is a symmetry σ such that there exist one-to-one mappings $\sigma_{\mathcal{G}} : \mathcal{G} \to \mathcal{G}$, $\sigma_{\mathcal{S}} : \mathcal{S} \to \mathcal{S}$, and $\sigma_{\mathcal{A}} : \mathcal{A} \to \mathcal{A}$ that satisfy for any $\tau \in \Gamma$:

$$\sigma(\tau) = \langle g', (s'_0, a'_1, r'_1, s'_1, a'_2, r'_2, s'_2, \ldots, s'_h)\rangle \quad (2)$$

where $\tau = \langle g, (s_0, a_1, r_1, s_1, \ldots, s_h)\rangle$, $g' = \sigma_{\mathcal{G}}(g)$, $s'_0 = \sigma_{\mathcal{S}}(s_0)$, $\forall i = 1, \ldots, h$, $a'_i = \sigma_{\mathcal{A}}(a_i)$, $s'_i = \sigma_{\mathcal{S}}(s_i)$, and $r'_i = R(\sigma_{\mathcal{S}}(s_{i-1}), \sigma_{\mathcal{A}}(a_i), \sigma_{\mathcal{S}}(s_i), \sigma_{\mathcal{G}}(g))$. In words, a decomposable symmetry is a simple mapping that applies transformations separately on states, actions, and rewards.

A *reward-preserving* symmetry $\sigma : \Gamma \to \Gamma$ is a symmetry such that for any trajectory $\tau \in \Gamma$, the rewards appearing in τ are exactly the same as those in $\sigma(\tau)$ in the same order. In words, the value of rewards in a trajectory is not changed by a reward-preserving symmetry. The previous definitions of symmetries can naturally be applied to the set of successful trajectories Γ^+ as well.

Besides, note that any number of symmetries induces a group structure (*i.e.*, they can be composed). Given n symmetries and a trajectory, one could possibly generate up to $2^n - 1$ new trajectories[1] (see Sec. IV-A for specifics). This property is useful if one only knows a fixed number of symmetries for a given problem, because a recursive application of those symmetries could lead to an exponential increase of trajectories that could be used for training.

As a general approach to increase data efficiency in deep RL, one can leverage the symmetries of Γ or Γ^+ for data augmentation. As an illustration, we propose ITER (Invariant Transform Experience Replay), a general architecture for data augmentation in deep RL, which we instantiate with two techniques for concreteness:

- Kaleidoscope experience replay (KER, Sec. IV-A) is based on reward-preserving decomposable symmetries of Γ and is applied to observed trajectories before they are stored in the replay buffer.
- Goal-Augmented Experience Replay (GER, Sec. IV-B) is based on reward-preserving decomposable symmetries of Γ^+, but can be applied to all feasible trajectories in the same fashion as HER. It is applied to trajectories sampled from the replay buffer.

In this general framework, other symmetries (*e.g.*, rotation, translation) could be used instead or in conjunction of KER or GER. Besides, these two approaches are orthogonal to each other, and could be used separately. An overview of our architecture is illustrated in Fig. 2. Our framework requires an off-policy RL algorithm since it changes the distribution of the agent's experience. Furthermore, note that our two methods preserve any contact that may occur between the robot and any object it may encounter (table included) as

[1]Though in our implementation, we use $2n$ 1 to avoid computational costs with an exponential increase.

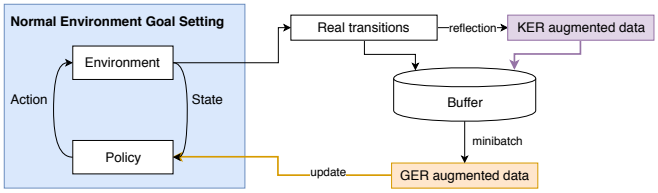


Fig. 2. ITER framework overview: observed and symmetrically transformed transitions are stored in the replay buffer. Sampled minibatches are then augmented with GER before updating the policy.

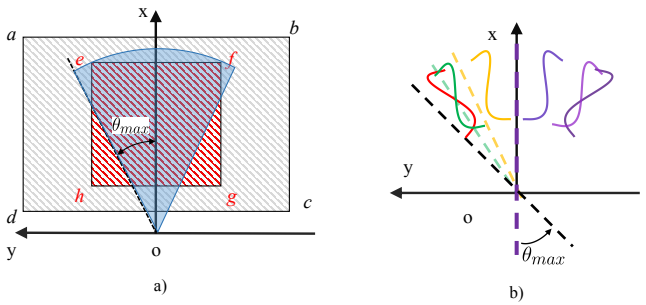


Fig. 3. a) Grey represents the valid workspace (the table surface). Red represents the object and goal's possible initial positions. Blue represents valid areas for symmetry hyperplanes (lines in this 2-dimensional visualization) that KER applies to reflect any observed trajectory. b) 2D illustration of how KER reflects an observed trajectory when $n_{\text{KER}} = 3$.

long as a symmetry is applied to all the objects and obstacles in the robot's workspace. Therefore, our approach also works in any contact-rich robotic task, including problems where some obstacles may limit the movements of objects or the robot. When the poses of obstacles are given in each state but not fixed across episodes, the agent can learn the effects of contact. For example, the agent can avoid obstacles or leverage contact to reach a goal (*e.g.*in the pushing task it may learn to push an object and let the obstacle stop the moving object).

A. Kaleidoscope Experience Replay (KER)

KER uses reflectional symmetry. Consider a 3D workspace with a bisecting plane xoz as shown in Fig. 1. If a trajectory is observed in the workspace (red in Fig. 1), the symmetry associated to xoz would then yield a new feasible trajectory reflected on this plane. More generally, the xoz plane may be rotated by some angle θ_z along axis $\vec{z}$ and still define an invariant symmetry for the robotic task.

We can now precisely define KER, which amounts to augmenting any observed trajectory with a certain number of random decomposable symmetries. To generate feasible symmetrical trajectories, one must choose a maximum valid angle θ_{max} for generating symmetries in any specific robotic manipulation task as shown in Fig. 3 a). Value θ_{max} is a hyperparameter that one can enlarge to expand the number of symmetrical trajectories (leading to more general policy training). Note, however, that it is also possible that a limited number of reflections lead to trajectories that consists of sections where the robot manipulator is outside the workspace. In these cases, such trajectories are not included in the replay buffer to ensure that the reflection is invariant.

State, action, and goal vectors consist of position, orientation, linear velocity, and angular velocity elements that can be reflected through symmetry. Position and linear velocity are represented through variables (x, y, z), whilst variables (α, β, γ) are used to represent the orientation and angular velocity elements (as recalled in Sec. III). The center of the robot's base coincides with the origin. Whenever, a robot's shoulder is offset from the origin and we need to consider the parallel sagittal plane, then we translate all coordinates to this new plane. Since the distance between fingers and their corresponding relative velocity are scalar, they remain unchanged after symmetry. Thus, we only consider reflecting (x, y, z) and (α, β, γ) to KER-augmented elements $(x_{sym}, y_{sym}, z_{sym})$ and $(\alpha_{sym}, \beta_{sym}, \gamma_{sym})$. Note that each plane in the 3-dimensional Cartesian space can be leveraged to yield one symmetry. Formally, each plane ψ is only associated with one symmetry σ^{ψ}. The number of reflections used in KER is controlled by hyperparameter n_{KER}. If $n_{\text{KER}} = 1$, KER directly applies σ^{xoz} to reflect the new trajectory in relation to states, actions, and goals in terms of (x, y, z) and (α, β, γ) elements as shown below:

$$\begin{aligned}\sigma^{xoz}((x, y, z)) &= (x, -y, z) \\ &= (x_{sym}, y_{sym}, z_{sym})\end{aligned} \tag{3}$$

$$\begin{aligned}\sigma^{xoz}((\alpha, \beta, \gamma)) \\ &= (-\alpha, \beta, -\gamma) \\ &= (\alpha_{sym}, \beta_{sym}, \gamma_{sym})\end{aligned} \tag{4}$$

If $n_{\text{KER}} > 1$, there are two stages. In the first stage, KER generates a set of rotated symmetric planes $\Psi = \{\psi^z_{\theta_j} \mid \theta_j \in \Theta\}$ which are rotated along the $\vec{z}$-axis by a set of uniformly sampled angles $\Theta = \{\theta_j \mid \theta_j \sim (0, \theta_{\text{max}}], j = 1, 2, ..., n_{\text{KER}} - 1\}$. The set of associated decomposable symmetries for those planes is denoted as $\Sigma^{\Psi} = \{\sigma^{\psi_{\theta_j}} : \Gamma \to \Gamma \mid \psi \in \Psi, j = 1, 2, ..., n_{\text{KER}} - 1\}$ (here for conciseness we use ψ_{θ_j} to represent $\psi^z_{\theta_j}$). The decomposition of $\sigma^{\psi_{\theta_j}}$ is:

$$\begin{aligned}\sigma^{\psi_{\theta_j}}((x, y, z)) &= Rot_z(\theta_j)[\sigma^{xoz}(Rot_z^{-1}(\theta_j)(x, y, z)^T)]^T \\ &= (x_{sym}, y_{sym}, z_{sym})\end{aligned} \tag{5}$$

$$\begin{aligned}\sigma^{\psi_{\theta_j}}((\alpha, \beta, \gamma)) &= Car(Rot_z(\theta_j)Eul(\dot{\alpha}, \dot{\beta}, \dot{\gamma})) \\ &= (\alpha_{sym}, \beta_{sym}, \gamma_{sym})\end{aligned} \tag{6}$$

where:

$$(\dot{\alpha}, \dot{\beta}, \dot{\gamma}) = \sigma^{xoz}(Car(Rot_z^{-1}(\theta_j)Eul(\alpha, \beta, \gamma)))$$

with $Eul : \mathbb{R}^3 \to SO(3)$ which maps a 3D-vector (α, β, γ) of Euler angles to a rotation matrix $Rot \in SO(3) \subset \mathbb{R}^{3\times3}$ (where $SO(n)$ denotes the special orthogonal group of dimension n) under right-handed coordinate frame, and $Car : SO(3) \to \mathbb{R}^3$ is the inverse mapping of Eul following the x-, y-, and z-axes rotation sequence (Cardano sequence) [29]. $Rot_z(\theta)$ is a standard rotation matrix for rotation of θ about the z axis. At the end of the first stage, we obtain a set

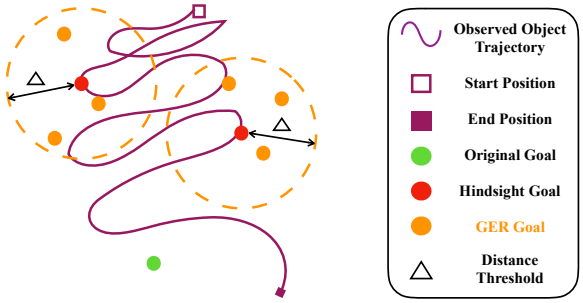


Fig. 4. An illustration for GER.

of trajectories $\overline{\Gamma}_1$ consisting of an observed trajectory and its symmetrical trajectories generated from Σ^{Ψ}. From this set, infeasible trajectories (*e.g.*, out of workspace) are filtered out to define a set of feasible trajectories Γ_1. Then KER applies σ^{xoz} to Γ_1 according to Eqn. 3 and Eqn. 4, and then yields another set of feasible symmetrical trajectories Γ_1' (all these trajectories are feasible since our workspace is symmetrical with respect to xoz plane). Finally, the trajectories in the set $\Gamma_2 = \Gamma_1' \cup \Gamma_1$ are stored into the replay buffer in each episode. In general, the maximum number of new trajectories generated is $2n_{\text{KER}} - 1$. Consider Fig. 3 b), here KER first samples two symmetry hyperplanes (the green and yellow lines) rotated about the z-axis with origin o with uniformly sampled angles with range $(0, \theta_{max}]$. Then, we reflect the observed (red) trajectory across the two planes to generate two new trajectories (yellow and green). Finally, KER reflects both the observed and reflected trajectories about the x-axis to generate three new purple trajectories.

Note that instead of storing the reflected trajectories in the replay buffer, random symmetries can also be applied to sampled minibatches from the buffer. This approach was tried previously for single-symmetry scenarios [18]. However, the approach is more computationally taxing (as transitions are reflected every time they are sampled) and leads to lower performance, which is due to a lower diversity in the minibatches as discussed in Sec. VI.

B. Goal-Augmented Experience Replay (GER)

GER exploits any reward function formulation (see Eqn. 1) that defines a successful trajectory as one whose end position is within a small radial threshold (a ball) centered around the goal. Thus, when the robot obtains a trajectory, we can consider it successful for any goal within a ball centered at each state of that trajectory. Based on this observation, GER augments trajectories by replacing the original goal with a random goal sampled uniformly within that ball. Here is an example in pushing task. As shown in Fig. 4, when HER chooses some hindsight goals (red points) to replace the original goal (green point) for experience replay, we can further sample more goals (orange points) within the circle with radius Δ that also satisfy the success condition for hindsight goal replacements, and we call those artificial goals (red and orange points) as GER-goals. This ball can be formally described as $B(s_h, \Delta) = \{g \in \mathcal{G} \mid d(s_h, g) \leq \Delta\}$ where s_h is the state reached in the observed trajectory and $\Delta \leq \epsilon_R$ is a threshold.

Formally, GER is based on reward-preserving decomposable symmetries of Γ^+ where $\sigma_{\mathcal{S}}$ and $\sigma_{\mathcal{A}}$ are identity mappings and σ_G is randomly chosen, conditional to a trajectory τ reaching some state s_h, in the following set: $\{\rho : \mathcal{G} \to \mathcal{G} \mid \forall g \in \mathcal{G},\ \rho(g) \in B(s_h, \Delta)\}$. Interestingly, such symmetries, when viewed as mappings from Γ to Γ^+ can be applied to the whole set of feasible trajectories to generate successful trajectories, which we do in our architecture. In this sense, GER is a generalization of HER and can be implemented in the same fashion. n_{GER} is a hyperparameter that controls the ratio between the numbers of original goals $g \in \mathcal{G}_{Ori}$ and GER goals $g \in \mathcal{G}_{GER}$ used in minibatch for policy training: $\frac{|\mathcal{G}_{GER}|}{|\mathcal{G}_{Ori}|} = n_{\text{GER}}$. Note that in order to take full advantage of realized goals, in our definition, when $n_{\text{GER}} = 1$, HER is a special form of GER with $\Delta = 0$, which means the GER-goals are all the hindsight goals chose right on the state of any observed trajectories.

V. Experiments and Results

A. Environment

To evaluate our method, a simulated 7-DOF (degrees of freedom) Fetch arm with a two-fingered parallel gripper is trained with DDPG on the pushing, sliding, and pick-and-place tasks from OpenAI Gym [8]. The state and action are defined according to Sec. III. The rewards are sparse and binary. If successful (the goal is achieved within an error distance) the agent gains a 0 reward, otherwise -1. With regards to goal replay, [7] proposed four strategies for selecting the replayed goal. Their experimental results show that the *future* strategy performed best. As such, we use the same goal sampling strategy in all of our experiments. In our case, this strategy will select k random states s_h (that will be set at the center of the ball for sampling random goals in GER experiments as shown in Fig. 4) that come from the same episode as the transition being replayed and were observed afterwards. In our experiment, we use the same k as in [7], which is 8.

Our method is evaluated on three simulated manipulation tasks described below (and introduced in [7]) and shown in Fig. 5. For all tasks, a movable object is initialized randomly on a table.

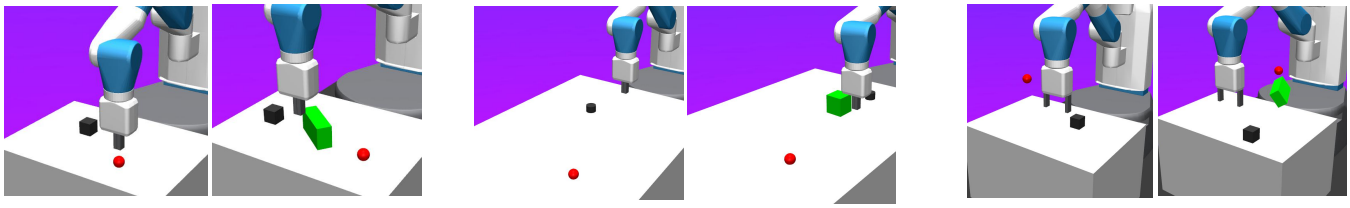
1) Pushing: The robot's aim is to move the object to a desired position on the table.

2) Sliding: The robot's aim is to slide the object to a goal position on the table (the goal position is outside the robot's workspace). The robot must learn to contact the object with enough momentum such that it reaches its goal (considering friction).

3) Pick-and-place: The robot's aim is to move the object to a desired position in space.

Note that for the pushing and sliding tasks, the fingers are blocked to prevent the agent from learning to grasp. All hyperparameter values are set equal to those presented in [7].

Learning with Obstacles: Our method also succeeds in more complex environments; namely, those with obstacles. In



a) Pushing b) Sliding c) Pick-and-Place

Fig. 5. Evaluation on robotic tasks without obstacles [7] (left) and with obstacles (showed as green bricks in right). Goals are represented by red balls.

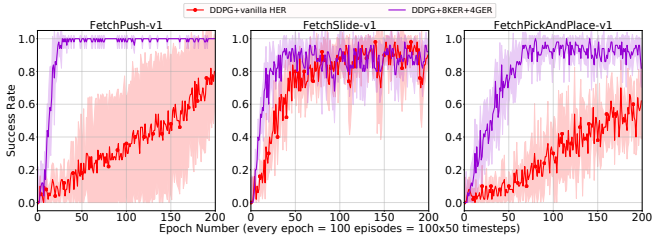


Fig. 6. Comparison of vanilla HER and ITER with 8 KER symmetries and 4 GER applications on obstacle-free tasks.

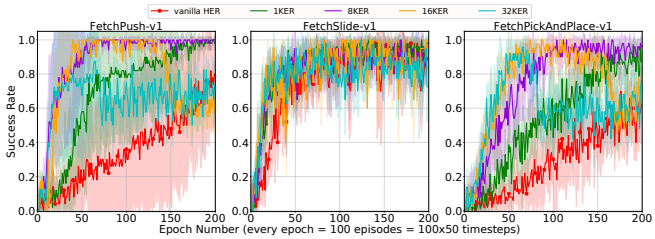


Fig. 7. Comparison of different n_{KER} for KER with a single GER on obstacle-free tasks.

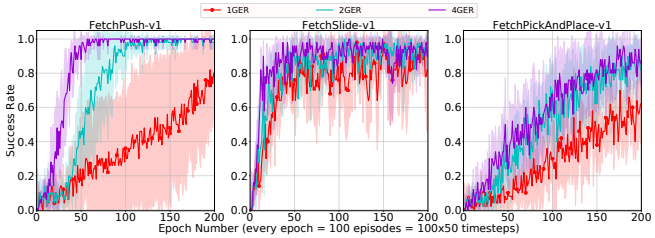


Fig. 8. Comparison of different n_{GER} for GER without KER on obstacle-free tasks.

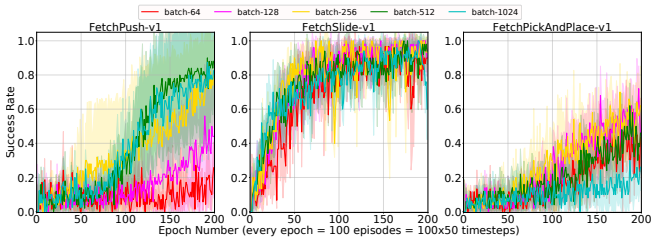


Fig. 9. Comparison of vanilla HER with different minibatch sizes from 64 to 1024.

each episode, a static brick-like obstacle is randomly placed in the robot workspace (see Appendix A [1] for details). The state space dimensionality increases to 31 and additionally includes the obstacle pose. In such scenarios, the robot must learn how to manipulate a movable object to achieve a goal by possible interactions with the obstacle—a much harder learning process.

B. Training Setting

The training setting is conducted according to [7]. Hyperparameter values are unchanged unless otherwise stated. We train policies on a single machine with 1 CPU core and generate experiences by using 2 rollouts.

An epoch is defined as a fixed-size set of successive episodes. Since a trajectory (also defined as a single episode) can be considered *successful* (Sec. IV), we can compute the *success rate* over an epoch by counting the number of successful episodes. To highlight the difference in learning rate ability between ITER and HER, we use an eighth of the number of episodes (100 episodes per epoch instead of 800) in each training epoch.

Note that during learning, the discount factor, the structures of the policy network and the Q-value network (input, output, activation functions, and hidden layers), the policy's exploration strategy, the optimization method, the learning rates, the soft update ratio, and the replay buffer size are all equal to [7]. Policy performance during testing is shown in Figs. 6-10. That is, exploration is disabled, rendering the policy fully deterministic. After each learning epoch, the testing success rate is computed over 10 episodes. We display an average over 5 random seeds for each curve.

Finally, note that a successful episode is defined as having an object reach a final position within distance ϵ_R of the goal. Namely, 5cm for pushing and pick-and-place, and 20cm for sliding. To sample the GER goals, we used 2D-balls in the pushing and the sliding tasks, and 3D-balls in the pick-and-place task (Δ equals ϵ_R).

We design experiments to demonstrate the effectiveness of our approach and to answer the following questions.

- How does ITER (GER+KER) perform compared to HER on obstacle-free robotic tasks?
- How much does KER contribute to ITER's performance? How many n_{KER} should be used?
- What is the contribution of GER to the performance of ITER? What is the impact of n_{GER} ?
- Does ITER (GER+KER) improve performance even with an obstacle in the robot workspace?
- Could we deploy a well-trained policy learned from

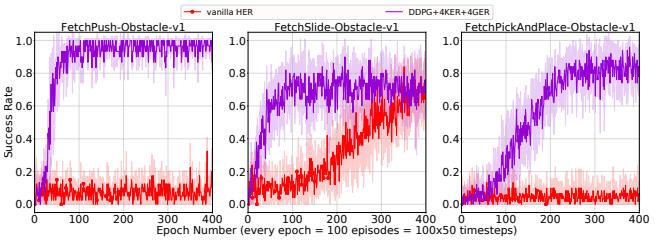


Fig. 10. Comparison of vanilla HER and ITER with 4 KER symmetries and 4 GER applications on tasks with obstacles.

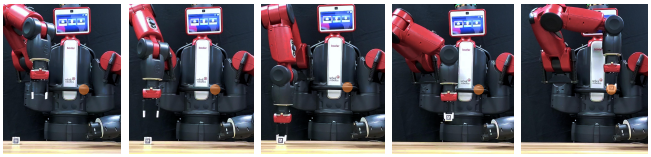


Fig. 11. A real Baxter robot running a pick-and-place policy trained via ITER (the goal is located at the orange ball).

ITER to the real robot without any finetuning?

C. Does ITER improve performance with respect to HER?

Experimental results show that when ITER uses $n_{\text{KER}} = 8$ and $n_{\text{GER}} = 4$ it significantly outperforms the data efficiency of HER across tasks in obstacle-free tasks (Fig. 6). In this experiment, we achieve a $13\times$, $3\times$, and $5\times$ speedup over HER for pushing, sliding, and pick-and-place tasks respectively. Note that the sliding task is very challenging as it is only determined by a few contacts (generally one) between the gripper and the object. The limited number of contacts limits the performance gain of ITER over HER.

D. How many symmetries should we use in KER?

In this experiment, only a single GER application with a zero threshold Δ is used (*i.e.*, HER). We observe a monotonic performance increase with respect to the number of random symmetries n_{KER} as illustrated in Fig. 7. We also note that there are performance drops for larger n_{KER} (see Appendix B [1] to explain this phenomena).

E. Does GER improve performance?

In this experiment, KER is not used and we only vary the number of GER applications. As with KER, performance improves as more GERs are applied until a ceiling is reached. Results are shown in Fig. 8.

Given that GER changes the size of the minibatch, we performed a controlled experiment where we increased the size of the minibatch in vanilla HER. More specifically, the size of the minibatch is set to $256 * n_{\text{GER}}$. Theoretically, with stochastic gradient descent, learning may speed up as the size of the minibatch increases, since the approximation of the minibatch gradient is more precise according to the Law of Large Numbers. However, we found that enlarging the minibatch size in HER did not always improve the performance in the aforementioned tasks—some times even deteriorated it (Fig. 9). We believe that with larger minibatches the network converges to sharp minimizers leading to poorer generalization performance [30]. In contrast, GER does not suffer such a degradation as it augments the data by introducing some noise. GER improves the learning performance despite a larger minibatch size. By observing Figs. 7 and 8, we can conclude that KER and GER both contribute to ITER in similar proportions.

F. How does ITER perform in tasks with obstacles?

The experimental results shown in Fig. 10 manifest that HER cannot resolve pushing and pick-and-place tasks within 400 epochs due to the more complex dynamics introduced by obstacles. In contrast, ITER yielded highly efficient learning, converging to a satisfying performance in around 80 and 230 epochs respectively. In the sliding task, ITER converges in around 100 epochs whilst HER converges after 400 epochs. These experiments prove the effectiveness of our method even in contact-rich environments (videos available in [1]).

G. Real robot deployment

Similarly to HER [7], we show that a policy trained with ITER can be transferred to a real robot. A Rethink Baxter dual-armed humanoid was used for evaluation. First, ITER trained policies in a MuJoCo simulation with Baxter. Then we directly applied a well-trained policy from simulations to the real Baxter without fine-tuning. Object poses were detected by Alvar markers (see Appendix C [1] for learning plots and other details).The real Baxter successfully achieved pick-and-place in 46 out of 50 trials as shown in Fig. 11 (videos available in [1]).

VI. DISCUSSION

A. Transformations to the replay buffer's input or output?

One interesting question concerns how the new data should be used. We could either populate the replay buffer with the artificial transitions or apply the transformations to a minibatch sampled from the replay buffer. If the transformations are applied after, the diversity of the minibatch could be limited because all the new artificial transitions come from the same source. On the other hand, if the transformations are applied before, then we do not fully exploit the information contained in this transformation because we only sample observed states within the same trajectory for several times. In our experiment, we notice that applying KER before and GER after works better in practice.

B. Performance drop with KER

In the KER experiments, we noticed an unexpected performance dropped after running around certain numbers of epochs. This drop showed up earlier as the number of symmetries n_{KER} increased. We first thought the algorithm was overfitting the new artificial goals at the expense of the real goals. However, in an experiment not shown here (see Appendix B [1] for details), we observed that the performance drop occurs also with HER, regardless of whether ITER is used or not. DDPG seems to suffer from this instability after seeing a certain amount of data (actual or artificial).

VII. CONCLUSIONS

We proposed ITER, a general framework for data augmentation in deep RL, which we instantiated with two novel techniques KER and GER in both simple and complex dynamical environments. KER exploited reflectional symmetry in the feasible workspace while creating invariant RL trajectories. GER, as an extension of HER, is specific to goal-oriented tasks where success is defined with a threshold distance and generalizes hindsight goals. These techniques greatly accelerate learning and improved success rates as demonstrated in our experiments. As mentioned before, ITER could be formulated with other kinds of transformations (*e.g.*, translation, rotation) as long as they satisfy the properties (*e.g.*, reward-preserving symmetries on feasible trajectories) we introduced. We leave the investigation of such other symmetries in ITER for future work. ITER's accelerated learning enabled satisfactory learning performance in only 250k timesteps (2 hours of interaction time) for pick-and-place–a 500% improvement compared to HER's. And we presented that policies learned under ITER can also endow the real robot with the pick-and-place ability without further finetuning. ITER also resolved environments with obstacles in a highly-efficient manner, while vanilla HER fails to solve some tasks.

ACKNOWLEDGMENT

This work is supported by GD Dept. of Science & Tech. [2019A050510040], by the NSF of China [61950410758, 61750110521, 61872238], and the Shanghai NSF [19ZR1426700].

REFERENCES

[1] Y. Lin, J. Huang, M. Zimmer, Y. Guan, J. Rojas, and P. Weng, "Invariant transform experience replay: Data augmentation for deep reinforcement learning supplement," Tech. Rep., 2020. [Online]. Available: http://www.juanrojas.net/iter/

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and Others, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and others, "Mastering the game of Go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[4] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 2018, pp. 651–673.

[5] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1808.00177

[6] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[7] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5049–5059.

[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[9] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba, "Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research," Tech. Rep., 2 2018.

[10] R. Zhao and V. Tresp, "Energy-Based Hindsight Experience Prioritization," in *coRL*, 2018. [Online]. Available: http://arxiv.org/abs/1810.01363

[11] M. Fang, C. Zhou, B. Shi, B. Gong, J. Xu, and T. Zhang, "DHER: Hindsight Experience Replay," in *ICLR*, 2019.

[12] A. Gerken and M. Spranger, "Continuous Value Iteration (CVI) Reinforcement Learning and Imaginary Experience Replay (IER) For Learning Multi-Goal, Continuous Action and State Space Controllers," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 5 2019, pp. 7173–7179. [Online]. Available: https://ieeexplore.ieee.org/document/8794347/

[13] P. Rauber, A. Ummadisingu, F. Mutz, and J. Schmidhuber, "Hindsight policy gradients," *arXiv preprint arXiv:1711.06006*, 2017.

[14] M. Zinkevich, M. Zinkevich, and T. Balch, "Symmetry in Markov decision processes and its implications for single agent and multi agent learning," *IN PROCEEDINGS OF THE 18TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING*, pp. 632–640, 2001.

[15] M. Kamal and J. Murata, "Reinforcement learning for problems with symmetrical restricted states," *Robotics and Autonomous Systems*, vol. 56, no. 9, pp. 717–727, 9 2008.

[16] A. Agostini and E. Celaya, "Exploiting Domain Symmetries in Reinforcement Learning with Continuous State and Action Spaces," in *ICMLA*, 2009.

[17] A. Mahajan and T. Tulabandhula, "Symmetry Learning for Function Approximation in Reinforcement Learning," *arxiv preprint 1706.02999*, 2017.

[18] Ł. Kidziński, S. P. Mohanty, C. F. Ong, Z. Huang, S. Zhou, A. Pechenko, A. Stelmaszczyk, P. Jarosik, M. Pavlov, S. Kolesnikov *et al.*, "Learning to run challenge solutions: Adapting reinforcement learning methods for neuromusculoskeletal environments," in *The NIPS'17 Competition: Building Intelligent Systems*. Springer, 2018, pp. 121–153.

[19] F. Amadio, A. Colome, and C. Torras, "Exploiting Symmetries in Reinforcement Learning of Bimanual Robotic Tasks," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1838–1845, 4 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8637816/

[20] H. S. Baird, "Document Image Defect Models," in *Structured Document Image Analysis*. Springer, 1992, pp. 546–556.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, p. 2012.

[22] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *arXiv preprint arXiv:2004.14990*, 2020.

[23] I. Kostrikov, D. Yarats, and R. Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," *arXiv preprint arXiv:2004.13649*, 2020.

[24] R. Gens and P. Domingos, "Deep symmetry networks," in *Advances in Neural Information Processing Systems*, 2014, pp. 2537–2545.

[25] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International Conference on Machine Learning*, 2015, pp. 1312–1320.

[26] V. R. Konda and J. N. Tsitsiklis, "Actor-Critic Algorithms," *Neural Information Processing Systems*, vol. 13, pp. 1008–1014, 1999.

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *International Conference on Learning Representations*, 2016.

[28] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," *Proceedings of the 31st International Conference on Machine Learning*, pp. 387–395, 2014.

[29] P. Corke, "Robotics, vision and control," *Springer Tracts in Advanced Robotics*, vol. 118, 2017.

[30] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," 2016.