GENERALIZING REINFORCEMENT LEARNING TO UNSEEN ACTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

A fundamental trait of intelligence is the ability to achieve goals in the face of novel circumstances. In this work, we address one such setting which requires solving a task with a novel set of actions. Empowering machines with this ability requires generalization in the way an agent perceives its available actions along with the way it uses these actions to solve tasks. Hence, we propose a framework to enable generalization over both these aspects: understanding an action’s functionality, and using actions to solve tasks through reinforcement learning. Specifically, an agent interprets an action’s behavior using unsupervised representation learning over a collection of data samples reflecting the diverse properties of that action. We employ a reinforcement learning architecture which works over these action representations, and propose regularization metrics essential for enabling generalization in a policy. We illustrate the generalizability of the representation learning method and policy, to enable zero-shot generalization to previously unseen actions on challenging sequential decision-making environments. More training and testing videos can be found at [sites.google.com/view/action-generalization/](sites.google.com/view/action-generalization/)

1 INTRODUCTION

Imagine visiting your friend for the first time, and you decide to cook your favorite dish there. But since you have never been in their kitchen before, there could be certain tools you have never seen, like an odd-shaped sponge. However, by looking at its porous texture or observing its interaction with water, you can understand that this object can absorb liquid. Later during cooking when you want to clean the table, you can select that sponge since you can relate its absorbing characteristics with another tool you have used for cleaning. Just like in this scenario, our tasks often involve making selections from novel or unseen entities. When we encounter such choices, we examine them to first understand their functionality which informs our selection process while solving a task.

Can machines also understand previously unseen choices and subsequently use them for solving tasks? From a reinforcement learning perspective, this brings an interesting question of how to enable generalization of discrete action policies to solve tasks using unseen sets of actions. Prior work in deep reinforcement learning has explored generalization over environments (Cobbe et al., 2018; Nichol et al., 2018), and tasks (Finn et al., 2017; Parisi et al., 2018). However, action space generalization is relatively unexplored and is crucial for agents to be flexible in the face of novel circumstances, like selecting an unseen sponge for a known task of cleaning in above example.

In this work, our goal is to develop a framework that reflects the two phases of solving action generalization: (1) general understanding of unseen discrete actions from their characteristic information (like appearance or behaviors), and (2) training a policy to solve tasks by utilizing this general understanding. However, an action can have diverse behaviors and hence requires a collection of data (e.g. different viewpoints, videos or state trajectories of how it affects on environment) to sufficiently express this diversity. Hence, the primary challenge is to develop a generalizable unsupervised learning method which can extract an action’s characteristics from a dataset constituting its diverse effects. To this end, we propose to embed actions’ datasets by extending the work on hierarchical variational autoencoders (Edwards & Storkey, 2017).

The obtained embeddings reflect an action’s general utility, and can be used as action representations in the downstream task of reinforcement learning. However, conventional reinforcement learning
Figure 1: Generalizing the knowledge of solving a task to a new set of actions. (a) CREATE is a sequential environment where the task is to help the green ball reach the goal (blue) by selecting tools and deciding where to place them. (b) In Shape Stacking the goal is to stack a tall tower by selecting the right shapes and their placements. Scenario A depicts the training scenario when the agent learns to utilize a given set of actions to solve the task. Scenario B presents an unseen set of actions to the agent which is expected to generalize to solve the task zero-shot.

The main contributions of this paper are: (1) introducing the problem and a proposed solution to enable action space generalization in reinforcement learning, (2) representing an actions with a datasets reflecting its diverse characteristics, and employing a generalizable unsupervised learning approach to embed these datasets. (3) a method to use learned action representations in reinforcement learning, and regularization methods to enable learning of generalizable policies.

2 RELATED WORK

Generalization in reinforcement learning In typical deep reinforcement learning (RL) settings (Mnih et al. 2015; 2016; Lillicrap et al. 2015; Schulman et al. 2017), a policy or value network learns to act over an action space of fixed dimensionality. By taking states or observations as input to neural networks, these methods are able to generalize to unseen environment states drawn from a similar distribution as training (Cobbe et al. 2018; Nichol et al. 2018). Similarly, prior works have explored generalization in RL for unseen instructions (Oh et al. 2017), new sequences of subtasks (Andreas et al. 2017), manipulation of unseen tools (Fang et al. 2018; Xie et al. 2019), task demonstrations (Xu et al. 2017), and agent morphologies (Wang et al. 2018; Sanchez-Gonzalez et al. 2018; Pathak et al. 2019). In contrast, our framework enables zero-shot generalization of RL policies when the agent gets a previously unseen action set.

Unsupervised representation learning for downstream tasks Bengio et al. (2013) state representation learning of data makes it easier to extract useful information when building predictors. Prior
works show that such representations have been useful for a variety of downstream tasks, like classification and video prediction (Denton et al., 2017), visually representing objects for relational reasoning tasks (Steenbrugge et al., 2018), representing image-states for domain adaptation in RL (Higgins et al., 2017), and representing goals for better exploration (Laversanne-Finot et al., 2018) and sample efficiency (Nair et al., 2018) in RL. In this paper, we show how unsupervised representation learning over datasets (Edwards & Storkey, 2017) can be used for embedding discrete actions, and enable generalization in the downstream task of reinforcement learning.

Action Representations Using continuous representations of discrete actions, a policy can be trained through a combined Q-function over state and action representations (He et al., 2015), or in an actor-critic architecture by selecting the nearest neighbor action vector to the policy’s continuous output (Van Hasselt & Wiering, 2009; Dulac-Arnold et al., 2015). Unlike our work, these prior works assume access to ground truth action representations, which are usually not readily available. In other related work, action representations are learned implicitly through inverse model on a fixed action space to ease learning in large discrete action spaces (Chandak et al., 2019) or for intrinsic reward (Kim et al., 2019). In contrast, we do not have the assumption of fixed action space assumption as we learn action representations separately, and hence are able to incorporate new actions for the same policy. While Tennenholtz & Mannor (2019) pre-learn action representations explicitly using co-occurrence of actions in task-specific demonstrations, our generic embedding method applies to various modalities of datasets to represent actions, which are task-independent and hence suited for generalization to unseen actions.

Skill and Trajectory Embeddings In reinforcement learning, variational autoencoders (VAE) (Kingma & Welling, 2014) are often used for learning an abstraction for continuous entities like skills and state-action trajectories. Specifically, Co-Reyes et al. (2018) utilize a trajectory autoencoder for hierarchical RL, and Lynch et al. (2019) learn a latent space of trajectories and employ a goal-conditioned planner over it. Hausman et al. (2018) learn an embedding space of skills through a shared policy for different tasks, and utilize this space for solving other related tasks. In this paper, we extend the framework of hierarchical VAE (Edwards & Storkey, 2017; Achille et al., 2019) to trajectories, so as to embed even sequential datasets which are better indicative of action behavior. In general, an action can be a discrete skill choice, and an action’s behavior can be represented as the trajectory of effects it causes on the environment. Since individual trajectories are incapable of capturing the diverse effects of actions, we propose to use datasets for representing actions.

3 Generalization to Unseen Actions

Our approach is based on the intuition that when humans encounter previously unseen discrete entities, we examine them to understand their functionality through visual inspection or physical interaction, before deciding what to select for a task. Once the general functionality is inferred, these discrete objects can be used as actions in decision-making tasks, like selecting a tool for cooking or furniture assembly. In this paper, we incorporate these two phases (Figure 2) to enable agents to utilize previously unseen actions: (1) extracting representations of actions from datasets of unstructured information (e.g. image, videos), and (2) training a reinforcement learning policy to utilize these action representations with the joint objective of generalization and reward maximization.

In order to represent actions, we note that an action can have diverse behaviors like how it interacts with its environment. Further, there can be various ways an agent observes this dataset. In the sponge example, the action exhibits diverse properties like absorption or compression, and the agent can observe this through porous texture (image) or through interacting with it (states trajectory). Therefore, in its most general form, information about an action can be expressed in the form of a diverse collection of unstructured data like images, videos or trajectories. To learn action representations in an unsupervised and generalizable, we use a hierarchical VAE and extend it to sequence data like videos (Section 3.2). Next, we show how a policy is trained to use these action representations as input, and propose training objectives for enabling generalization (Section 3.3).

3.1 Preliminaries

For a learning agent, we denote the entire set of possible discrete actions as $\mathcal{A}$. For evaluation, we assume an episodic setting, where the agent only has a subset $A \subset \mathcal{A}$ of actions available to it. Each action $a \in \mathcal{A}$ has an associated dataset $D = \{x_1, \ldots, x_L\}$ of observable samples $x_n \sim P(x|a)$
Therefore, we aim to learn an action encoder to map each discrete action’s entire dataset to a continuous representation. For unsupervised learning of this encoder, we can use a variational autoencoder (VAE) with reconstruction objective (Kingma & Welling, 2014). However, since the available action sets are in general stochastic, we primarily consider stochastic policies in this paper. The performance of \( \pi_\theta \) is evaluated based on a discounted return \( R = \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t) \) where \( r \) is the reward function and \( T \) is the episode horizon. The aim is to train a policy which only has access to the known actions \( \mathcal{A}_K \) and its datasets, but generalizes to maximize reward on previously unseen actions.

### 3.2 Unsupervised Learning of Action Representations

We represent the diverse characteristics of an action with a dataset of observed information. To extract usable information from these action datasets, we propose an unsupervised representation learning method to learn action embeddings. Our key insight is that the common information underlying different samples of an action’s dataset best represents the general properties of that action.

Therefore, we aim to learn an action encoder to map each discrete action’s entire dataset to a continuous representation. For unsupervised learning of this encoder, we can use a variational autoencoder (VAE) with reconstruction objective (Kingma & Welling, 2014). However, since the input to VAE is in the form of a dataset, it should capture the information shared across multiple data samples. Therefore we encode both, the action datasets and the sample within each action’s dataset into a hierarchy of connected latent spaces.

Such a hierarchical VAE (HVAE) architecture has been explored by [Edwards & Storkey, 2017] for few-shot classification and clustering of datasets. We use it for the purpose of encoding action datasets and using them for generalization (Figure 2). HVAE is composed of an action VAE over datasets and an instance VAE over samples. The encoders and decoders of the instance VAE are conditioned on its parent action latent vector. For each action \( a \) and its associated dataset \( D = \{x_1, \ldots, x_L\} \), the action encoder \( q_a(c|D) \) is used to sample an action latent \( c \), while regularized by an action prior \( p_a(c) \). For each action sample \( x \in D \), the instance encoder \( q_a(z|x, c) \) is used to sample a latent \( z \) encoding the sample instance \( x \), while conditioned on \( c \). The prior distribution \( p_a(z|c) \) as well as the decoder \( p(x|z, c) \) are also conditioned on the action latent. For each action dataset, ELBO comprises of reconstruction over data samples and the two KL divergence terms (Edwards & Storkey, 2017).
We can train the parameters $\theta$. While solving tasks with new actions, humans first form a general interpretation of the behaviors. We further extend this framework to incorporate sequential data like state trajectories and videos, with embeddings (1). To overcome these challenges, we propose the following regularizing objectives to augment the ERM and help optimize the true risk in two ways: (a) it makes the data distribution used for training the policy expressive encoder for actions, since even seemingly new discrete actions can have characteristics lying in the distribution of behaviors of known actions. Hence, the hierarchy in HV AE makes it an expressive encoder for actions, since even seemingly new discrete actions can have characteristics which belong to the distribution of previously seen effects.

3.3 Training Policies to Generalize to Unseen Action Representations

While solving tasks with new actions, humans first form a general interpretation of the behaviors of actions, and then utilize it to take appropriate actions. Similarly, once our agent learns actions representations based on observed datasets (section 3.2), it should learn to utilize them for solving tasks. This involves not only extracting the task-specific information from the representations, but also doing so in a generalizable manner so that it can utilize previously unseen action representations.

Here we assume access to an embedder $\phi$, and hence the associated action representations $c_a = \phi(D_a)$ for each $a \in \mathcal{A}$. Our aim is to learn a policy $\pi_\theta(a|s, \mathcal{A}, c)$ which maximizes the expected reward under any set of available actions $\mathcal{A} \subset \mathcal{A}_{\text{seen}}$. We propose to utilize the action representations $c_a$ as inputs to the policy, which acts as a function approximator over action representations and states. Specifically, our policy consists of a utility function $f_\theta : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$, which maps a $d$-dimensional action embedding and a state to its utility. The probability distribution over actions is simply defined as the Softmax over the utilities of each available action $a' \in \mathcal{A}$.

$$
\pi_\theta(a|s, \mathcal{A}) = \frac{e^{f_\theta(s, c_a)}}{\sum_{a' \in \mathcal{A}} e^{f_\theta(s, c_{a'})}}
$$

(2)

We can train the parameters $\theta$ using policy gradient methods on $\pi$. Typically, RL policies act on various states and can extend to unseen states. However, our policy requires generalization to unseen action representations as well. Hence instead of the general reward maximization objective, our objective is to minimize the true risk of the policy:

$$
\text{Risk}(\pi_\theta) = \mathbb{E}_x \left[ L(\pi_\theta(x), y^*) \right] = \mathbb{E}_{\mathcal{A} \sim \mathcal{A}_{\text{seen}}} \left[ \mathcal{L}_s \pi_\theta(s, a) \right]
$$

(3)

Here $y^*$ denotes some optimal action distribution and $\varepsilon$ is the environment. The agent only have access to a limited set of known actions $\mathcal{A}_K \subset \mathcal{A}$ during training. Maximizing the training reward with embeddings $c_a$ drawn only from $\mathcal{A}_K$ is equivalent to minimizing the empirical risk of the policy $\pi_\theta$. A policy trained on reward maximization is prone to such overfitting to known actions during training. This problem becomes more severe for RL because the distribution of input data, $(s, c_a)$ used for training $\pi_\theta$ is governed by $\pi_\theta$ itself. Moreover, during training the policy acts over a fixed action space $\mathcal{A}_K$, whereas during evaluation, the policy is required to work with varying action sets $\mathcal{A} \subset \mathcal{A}_{\text{seen}}$.

To overcome these challenges, we propose the following regularizing objectives to augment the ERM objective so as to enable better generalization over action representations:

1. **Maximum entropy regularization**: For better generalization to unseen action spaces, the agent should learn to utilize diverse actions for a task. Maximum entropy objective (Ziebart et al., 2008) helps optimize the true risk in two ways: (a) it makes the data distribution used for training the policy
closer to uniform sampling over action representation inputs, and (b) it makes the least assumptions about the entire set of actions \( \mathcal{A} \), and hence by principle of maximum entropy leads to the best policy.

(2) **Changing action spaces**: Selecting the best action repeatedly for solving a task during training is not the best strategy for generalization, as an evaluation action set \( \mathcal{A} \) may be very different from the actions that the policy overfits to. Hence, in order to enforce a policy to utilize diverse actions for solving the task during training, we propose to sample action sets \( \mathcal{A} \subset \mathcal{A}_K \) during training as well. This trains the policy to solve the task even in the absence of well-known actions, and to learn to infer utility of less used actions, which may become optimal during evaluation.

(3) **Clustering similar actions**: Since many actions during training could be similar, a policy can still simply overfit to a group of actions, as always some member of the group can be present in the randomly sampled action space while training. However, since we already have the learned representations of training actions from section 3.2, we propose to cluster similar options together and perform cluster-based action set sampling. Such sampling ensures that certain groups of actions with similar attributes are blocked, encourage the policy to utilize other actions as well.

4 **Environments**

4.1 **Grid World**

In **Grid World** environment (Chevalier-Boisvert et al., 2018), an agent navigates a 2D 9x9 maze to reach a goal cell for a sparse reward. A column of lava is randomly placed in every episode, touching which ends the episode. The discrete action space consists of all 5-step macro actions, where each macro-action is defined by a 5-length sequence of left, right, up or down movement. The entire action space of size \( 4^5 = 1024 \) actions is randomly split into a train and test set of 512 actions. The action datasets are collected on an empty grid where the agent is initialized at random locations. Two kinds of data types are used to represent the state sequence of agent - one-hot vectors and continuous (x,y) grid coordinates.

4.2 **Recommender System**

The **Recommender System** environment (Rohde et al., 2018) simulates how users may respond to product recommendations. Every episode, the agent must recommend items to a new user with the objective of maximizing the click through rate (CTR) for the recommendations. This simulated environment uses randomly initialized embeddings for recommendations (actions), and we use the same to demonstrate policy generalization to new actions. Action space of size 1000 is randomly split equally into train and test actions.

4.3 **Chain REAction Tool Environment (CREATE)**

**CREATE** is a physics-based puzzle where the goal is to make a specified ball reach a goal position (blue), inspired by the popular video game The Incredible Machine. The agent must place tools in real time to manipulate the path of the ball to reach the goal position. The environment presents a challenging multi-step task, requiring the agent to select the tool to place as well as its position \((x, y)\) on the screen. The agent has access to a subset of diverse tools such as trampolines, see-saws, cannons, funnels, and conveyor belts (Appendix B.2). The position aspect makes this a parameterized action space [Hausknecht & Stone, 2015] with both discrete and continuous components. Our policy architecture consists of another head to output this continuous vector and it is trained jointly with the discrete action. We solve 3 different CREATE tasks: Push, Navigate and Obstacle. The tools evaluated at test time are completely unseen tool types from those seen during training.

4.4 **Shape Stacking**

In **Shape Stacking** the agent must drop blocks on a table to build the highest standing tower. Our objective is different from prior works (Groth et al., 2018; Lerer et al., 2016) in that we maximize the tower height in an RL setting, whereas the prior work predicts the stability of the tower. Similarly to CREATE, the action space in Object Stacking, consists of \((x, y)\) coordinates of where the object should be dropped above the table. This environment is shows our ability to generalize problem solving ability to a new action space in a complex 3D task. The action dataset here are images of
Figure 3: **Quantitative results**: displayed are 3 of the CREATE tasks, the Block Stacking task, the Recommender System task and the Grid World navigation task. The performance displayed is measured on generalization to the test set of actions. The details of these environments are described in 4. CREATE task results are averaged across 3 seeds. Grid World, Recommender System and shape stacking are averaged across 2 seeds. Performance of each seed is computed across 3,200 episodes.

the objects from various angles (or viewpoints). In this case the visual appearance of the object is sufficient to infer its functionality.

## 5 Experiments

### 5.1 Quantitative Results

The generalization performance of the policy to unseen actions across all environments and method variations is shown in 3. As seen from the results our method or ablations (Ours, RS, NE) of our methods have the strongest ability to generalize to unseen actions across a variety of environments. The difference among our ablations is smaller in simpler environments like Grid World, Recommender systems and Shape Stacking, where the unseen action spaces are very similar to training actions. The effect of clustering-based sampling and entropy regularization can be seen for Obstacle and Navigate environments, which require solving the task with quite different tools at testing. CREATE Push is solvable with a wide variety of tools, and hence the no-entropy policy trains to a higher reward, and is able to generalize as well as many unseen tools can solve the task easily. The performance of our method against its variant with non-hierarchical (VAE) embeddings shows the importance of hierarchy in latent space to represent actions.

We test the generalizability of our embedder and policy for the task of zero-shot generalization to unseen actions. Specifically, our primary experiments across all four environments, discussed in section 4, train a policy on a fixed set of actions, tune hyperparameters on a separate evaluation set, and then test the ability to generalize to a new set of actions. We further provide qualitative analysis on cases where this generalization succeeds and fails. Finally, we evaluate how our method’s generalizability varies with the degree of difference between seen and unseen.

### 5.2 Baselines & Ablations

As the problem of generalizing to unseen actions has not yet been explored we compare our method to various approaches in the policy architecture and embedding learning.

For the policy one baseline is looking up the closest action embedding during testing as seen during training, notated as NN. Another variation outputs a continuous prediction of action embedding and selects the closest, Dist, similar to Dulac-Arnold et al. (2015). We also ablate sampling from k-means with random sampling notated as RS. We ablate ablate using an entropy loss term with NE and train
Figure 4: **Varying difficulty of test action space:** (i) Each test action is at least a specific angle apart from all actions seen during training (ii) Each test action is at least a specific distance in embedding space apart from all actions seen during training (iii) Test set contains seen/unseen ratio over a fixed action space with $F_X$. We also compare to a baseline of our embedder that learns a VAE over all data points in the action dataset and then computes the action embedding as the average latent vector, notated as $VAE$.

5.3 **Further Analysis**

Qualitative results of the policy test performance are shown in 5. The left and middle column contain success cases. In the left column for CREATE we see the policy, despite never having used on of the tools before, still be able to solve the task. Likewise, for shape stacking we see the policy able to use novel shapes to build a tall and stable pile to maximize the height. We also show cases of failure to generalize in the right most column. In both cases the policy chooses the right types of actions and barely misses the objective.

We also analyze the conditions needed for generalization to unseen actions. We perform all analyses on CREATE Push task because of the large diversity of tool functionalities. We show generalization across changing physical tool parameters with angle and the embeddings the policy is trained and tested on. Finally, we show the effect of unseen versus seen actions on performance.

Figure 5: **Qualitative analysis:** shown are two success cases and one failure case for CREATE and Object Stacking. In CREATE the trace of the ball trajectory is outlined. All of the tools or objects in these results the policy is generalizing to select and was not trained over these actions.

6 **Conclusion**

Generalization to novel circumstances is an important ability to have, for robust and widely applicable artificial agents. In this paper we propose the problem of generalization of reinforcement learning policies to unseen spaces of actions, with the use of action representations learned in an unsupervised manner. Our two-phase framework demonstrates how representation learning can be combined with the downstream task of reinforcement learning, specifically to represent actions. We demonstrate the efficacy of our methods on four challenging environments, and discuss which variants work when. The key takeaway is that when unseen actions are quite different from known actions, then more regularization helps to train generalizable policies.
REFERENCES


Annie Xie, Frederik Ebert, Sergey Levine, and Chelsea Finn. Improvisation through physical understanding: Using novel objects as tools with visual foresight, 2019.


A Appendix

A.1 Analysis of Training Procedure and Models

A.1.1 Visualization of Hierarchical Embedding Spaces

Figure 6: T-SNE Visualization of learned embedding space for CREATE environment. Tools in CREATE are labeled by their properties which define them to be floor, trampolines, high-frictional, etc. The action embeddings clearly group similar actions together.

Figure 7: T-SNE Visualization of learned embedding space for Grid World environment.

B Environment Details

B.1 Grid World

Grid World environment consists of an agent and a lava wall with an opening as shown in figure 9. The lava wall can be either horizontal or vertical. The agent is spawned in a random position and can move in 4 directions (up, down, left, and right). The objective of the agent is to reach the goal in the bottom-right corner while avoiding lava.

Observations: The observation space is 9x9x3 where three channels represent (1) object id (agent, wall, goal, lava), (2) color of an object, and (3) zero in each cell. The agent gets a flattened vector of this 9x9x3 matrix.
Under review as a conference paper at ICLR 2020

(a) T-SNE Visualization of learned action embedding space for Shape Stacking environment. Similar shapes are close by in the embedding space.

Figure 8: placeholder

Figure 9: GRID WORLD environment. The agent is the red triangle and the goal is the green cell. The environment contains one row or column lava wall with a single opening. Each action consists of 6 consecutive moves in 4 directions.

**Actions**: An action of the agent is 5 consecutive moves in 4 directions. Hence, \(4^5 = 1,024\) actions are possible in total. Once the agent selects an action, it executes 6 sequential moves step-by-step. During an action execution, if the agent hits the boundary, it will stay in the current cell. If the agent steps on lava, the game will be terminated. The whole action set is divided into 4:1 split of train and test action sets.

**Rewards**: GRID WORLD provides a sparse reward, \(1 - 0.9 \times \text{step/\text{max\_step}}\), only when the agent reaches the goal. The reward is discounted based on the number of actions taken to encourage a shorter path to the goal.

**Termination**: Each game is terminated when the agent takes max_steps (64) actions or the agent reaches the goal or the lava wall.

**B.2 Chain REAction Tool Environment (CREATE)**

**Chain REAction Tool Environment (CREATE)** is a physics-based puzzle where the objective is to make a target ball reach a goal position by placing variety of tools, inspired by the popular video game “The Incredible Machine”. The environment contains two movable objects, a marker ball (green) and a target ball (red). When a game starts, the marker ball is falling off from the top of
the screen and an agent requires to place tools to redirect the kinetic energy of the marker ball to the
target ball so that the target ball reaches the goal position (blue) as illustrated in figure 10.

Figure 10: CREATE environment. In CREATE, the green ball is falling into the scene, which must
push the red target ball into the blue goal location. The top and bottom rows show actual evaluation
results when our model is tested on CREATE UP and CREATE DOWN, respectively.

Observations: An observation for each time-step is an 84x84x3 image of a game screen and we use
3 frame stack to provide information about velocity and acceleration of the balls. Initially, the game
contains 3 balls in the observation: marker ball (green), target ball (red), and goal ball (blue).

Actions: CREATE contains 2,000 tools of type: ramps, trampolines, walls, balls, floor, conveyor
belts, funnel, polygons of different shapes, cannons, fans and buckets. We also have an action for No-
Operation. The whole tool set is divided into two parts train and one part for evaluation and another
for testing. Every time-step, the agent outputs a parameterized action, i.e. a discrete-continuous action
which has three values \((\text{tool}, x, y)\), where \(\text{tool}\) specifies which tool to place and \((x, y)\) represents the
position of the tool in the screen.

Rewards: The agent gets +1 reward when the marker ball hits the target ball, and +10 reward when
the target ball passes the goal. In addition to reward for success, the environment provides some
intermediate rewards. For every time-step, +0.01 reward is given to encourage the agent to keep
balls inside the screen. At the same time, an invalid action is penalized by giving -0.01 reward (e.g.,
placing a tool outside of the screen or placing a tool on top of other tools).

Termination: Each game is terminated when the agent takes 20 actions, or the marker ball goes out
of the screen before it hits the target ball, or the target ball goes out of the screen before it passes the
goal.

B.3 Shape Stacking

SHAPE STACKING is a mujoco simulation environment where the agent is given a set of objects of
different shapes of varying sizes, for instance, cubes, rectangles, spheres, round cylinders, archs, etc.
The objective is to stack a tower as high as possible by choose the appropriate objects given in a
particular episode.

Observations: An observation for each time-step is an 84x84x3 image of the shapes laying on the
table.

Actions: In total there are 900 distinct shapes in the environment. 675 are used for learning the
policy. The remaining 225 are used for evaluating performance on unseen actions. The same types of
polygons do not appear in both train and test. Like CREATE the action space includes making a
discrete selection over the shape to drop and the \(x, y\) coordinates to drop the shape above the scene.

Rewards: The agent receives a reward for the positive difference in height of the tower. A penalty of
\(-0.25\) is given for every repeated shape.

Termination: The game is terminated either after the tower of shapes exceeds 3 in height or after 10
shape placements.