Deep Scoop

Samuel Clarke
Andrew ID: sclarke1
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
sclarke1@andrew.cmu.edu

Abstract

In this paper, I investigate the strength of using deep reinforcement learning techniques in the domain of robotic manipulation of granular media. I compare different popular algorithms in their performance on the task of controlling a robot as it scoops a varying desired mass of pellets from a tub. Each policy performed significantly better than a random policy, with the best performing policies both being based on sampling actions with distributions based on a critic network, rather than directly using an actor network to select actions. Variations in how each policy used data from past experiences are demonstrated to have significant effects on the performance of each policy over time.

1 Introduction

Using reinforcement learning to devise policies over real-world physical systems is often made possible through training an agent on high-fidelity simulations, restricting an agent to interact with a physical system with dynamics that are simple to learn or model, or using a good model for estimating the dynamics of the physical system. Unfortunately, the domain of granular materials fails to satisfy each of these constraints. Deep learning has proven itself to be quite effective in navigating complex relationships and structure where many other approaches fail. Using deep learning to capture the complex dynamics of granular media therefore seems to be a natural impulse, but practically speaking, sample efficiency is a great challenge to overcome when applying deep learning to such a complex "real-world" domain.

Manipulation of granular media is thus a domain for which deep learning approaches seem promising, but could be fraught with very tough challenges such as those previously detailed. This paper thus seeks to investigate the application of deep reinforcement learning to learning how to manipulate granular materials through a task in which a robot must scoop an arbitrarily specified mass of plastic pellets. The robot's perception of its state consists of an interpolated height map of the surface of the pellets, and its task is to select parameters of a scooping action that will successfully scoop a desired mass of pellets. Note that, as an open-loop action without the potential to make online adjustments from visual or tactile feedback, this task would likely be rather difficult for a human to complete, as it would require a rather strong ability to predict the complex dynamics of the granular material at hand. Different methods involving deep learning will be evaluated and compared in their competence to accomplish this task.

2 Related Work

With respect to robots and granular media, there is a sizable body of work on mobile robots interacting with granular media throughout the process of locomotion [5, 6]. Some work has approached building models and simulations specifically for modeling these types of interactions [9]. There is also work

in modeling and controlling scooping motions for large construction equipment [12]. Learning has been applied to the task of pouring granular materials [19, 10]. For example, Rozo et al. [10] used demonstration learning for a robot to learn how to pour a specific volume of small metal spheres from a bottle into a cup.

A similar problem to the problem of this paper was approached by Schenck et al. [15], in which the authors attempted to have a robot learn to scoop and dump a granular material into an empty container to make a desired shape within a specified number of actions. The authors trained a predictive model and used the Cross Entropy Method [2] to use their model to sample actions for their policy. An approach similar to this will be used for comparison in this paper. Also, the authors used some hand-engineered baselines and inputs for their model, whereas I intend to use such methods very sparingly in order to attempt to force my learning algorithms to perform the full exercise of learning from more raw input and independently extract any dynamics relationships.

Model-free learning over tasks with discrete action spaces can be accomplished by estimating a value function with a Deep Q-Network (DQN), with a memory buffer of previous experiences, *i.e.*, a *replay buffer*, using these previous experiences to stabilize learning over new experiences [8]. Prioritizing more significant previous experiences (where the definition of *significant* could vary based on application) in the replay buffer produced further improvements in performance on many reinforcement learning tasks [13]. Whereas a standard DQN is designed to estimate a value function with respect to a fixed goal, a Universal Value Function Approximator (UVFA) is able to estimate the value of a state and action through a function that generalizes over different goals [14]. Furthermore, Lillicrap et al. [7] introduced Deep Deterministic Policy Gradients (DDPG) as a method to train a model for a policy simultaneously with a value function in continuous actions spaces, using *target networks* to stabilize both a network mapping states to actions (the *actor network*) and a network mapping states and actions to values (the *critic network*).

Sample efficiency has often been a challenge in applying deep reinforcement learning techniques to solving problems in physical systems, especially with model-free approaches [3]. Hindsight Experience Replay (HER) was developed in order to mitigate the sample efficiency of learning UVFAs from sparse binary rewards, especially applicable in robotics applications [1]. However, it stands to reason that the underlying concepts of HER could be applied to tasks with rewards that are neither binary nor necessarily sparse, *e.g.* how I have designed the rewards for my task, as I will describe in Section 3.2. Another technique that has been shown to improve sample efficiency of deep networks in some cases is to use a single network with many layers shared by multiple tasks in *multi-task learning* [11]. Even in the case where there is only one task at hand for the network, developing *auxiliary tasks* can often improve sample efficiency [18], and many ground-breaking successful single-task networks have used auxiliary tasks and corresponding losses during training to make neural network parameters more expressive [17]. Unsupervised auxiliary tasks, their corresponding auxiliary losses, and *auxiliary rewards* have been directly applied to deep reinforcement learning algorithms to advance the performance of existing algorithms [4].

3 Methods

3.1 Experiment Setup

The physical setup for my experiment is shown in Figure 1. The robot is a Sawyer Robot Arm with a custom 3D-printed plastic scoop mounted to its end-effector plate. In front of the robot is a tub of round plastic pellets, resting on top of two USB scales for measuring the mass of the tub and the pellets currently contained therein. Hence, the mass is recorded before each scooping action and after each action to infer the mass of pellets in the scoop. The USB scales have been empirically observed to have ~ 4 gram worst-case and 2 gram average-case combined measurement resolution. An Intel RealSense RGBD camera (shown in the top left of the image), faces down into the tub of plastic pellets to capture a depth image of the surface. The camera is calibrated to world coordinates, and its depth image is used to construct a bilinearly interpolated surface height map in world coordinates, as shown in Figure 2.

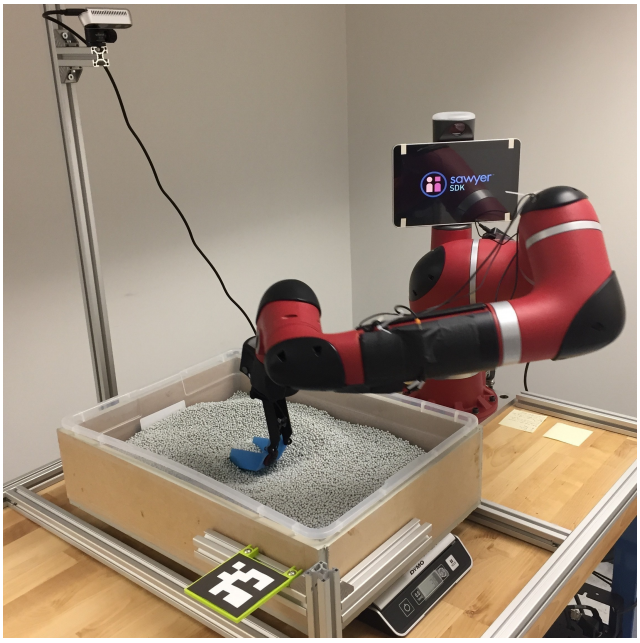

Figure 1: The robot scooping through pellets.

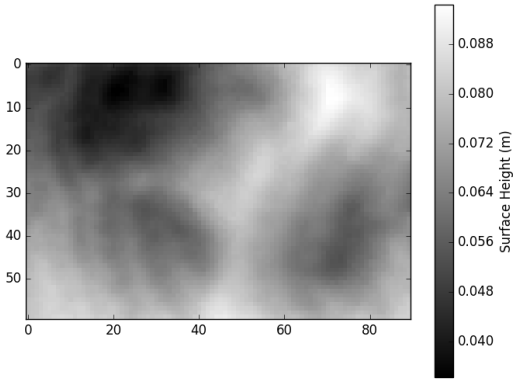

Figure 2: Example surface height map.

3.2 Task Specifications

From a reinforcement learning standpoint, the task is theoretically similar to a Contextual Bandit Problem. At each episode t, the robot is given goal g_t, a desired mass (in grams) for the robot to scoop. For each episode, a new goal g_t was sampled uniformly randomly on the interval of even integers [2, 100], such that the robot could realistically scoop the desired mass and that the average-case measurement resolution of the scales (2g) was sufficient to measure success. The robot first makes an observation of its state o_t, which is an interpolated height map of the material in the tub, discretized over half-centimeter width squares to make a 60x90 matrix. Based on this observation, it selects an action a_t, a 6D vector defining a parameterized scooping motion (with the 6 parameters diagrammed in Figure 3). Action a_t results in the robot scooping a certain mass m_t, and the robot then receives reward $r_t = -|m_t - g_t|$. For reasons that will be explained in Section 3.3, I also gave the robot an observation f_t, the observation of the final state of the tub after action a_t was taken, with the same specifications and structure as o_t.

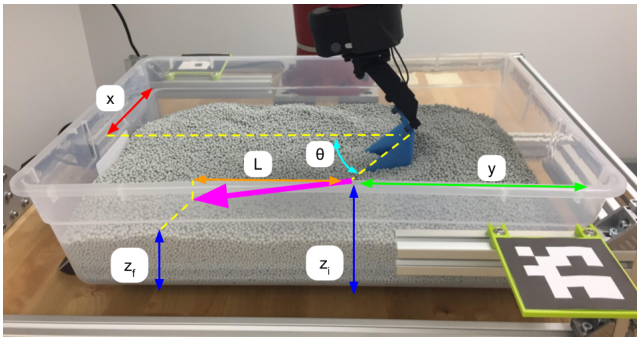

Figure 3: The parameters of the scooping action. The entire scooping motion can be described as follows: robot hovers high above the tub at the (x, y) coordinates, sets the pitch angle of the scoop to θ, then plunges the scoop straight down to height z_i. It then follows a linearly interpolated path to point $(x, y + L, z_f)$ while retaining the same angle θ. Upon reaching this final point, the robot then tilts the scoop up about its back edge to a constant angle in order to retain the scooped material, then lifts up out of the tub. Thus, in the example shown in the figure, the tip of the scoop would follow the magenta arrow.

3.3 Learning a Predictive Model

I first collected data from over 8000 trials (constituting approximately 65 robot-hours of data) of the robot selecting actions uniformly randomly within each parameter's bounds, for each sample recording the action parameters a_t, the initial and final observations o_t and f_t, as well as the scooped mass m_t.

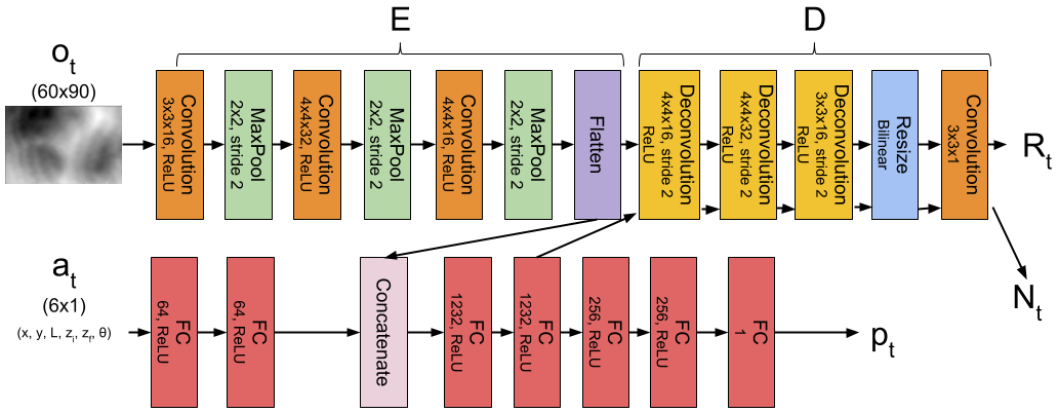

Figure 4: Architecture of the predictive model, also used for the critic.

To inform my design of a critic model, I first experimented with training predictive models mapping o_t and a_t to m_t. The architecture of the predictive model is shown in Figure 4, with a convolutional encoder structure E, corresponding decoder structure D, and predicted mass p_t. Also, dropout was applied between each pair of fully connected layers after the concatenation and before the final layer to mitigate the risk of overfitting to training data. To test the utility of auxiliary losses, I tested this predictive model architecture with three different weighted combinations of mean-squared error losses: one on only the prediction of mass m_t, one with this loss and an added loss on the encoder-decoder reconstruction of o_t (R_t in Figure 4), and one with both of these losses and a loss on the decoder reconstruction of f_t from the output of one of the fully connected layers downstream of both the encoder output and the action sub-network in the architecture (N_t in Figure 4). The premise of the loss on comparing R_t to o_t was to bias the encoder to continue to fully encode all of the descriptive information of the observation through a lower-dimensional representation. The premise of the loss on comparing N_t to f_t was to bias the encoder and preliminary fully connected layers to relate the parameters of action to their effects on the surface of the tub in image space. To

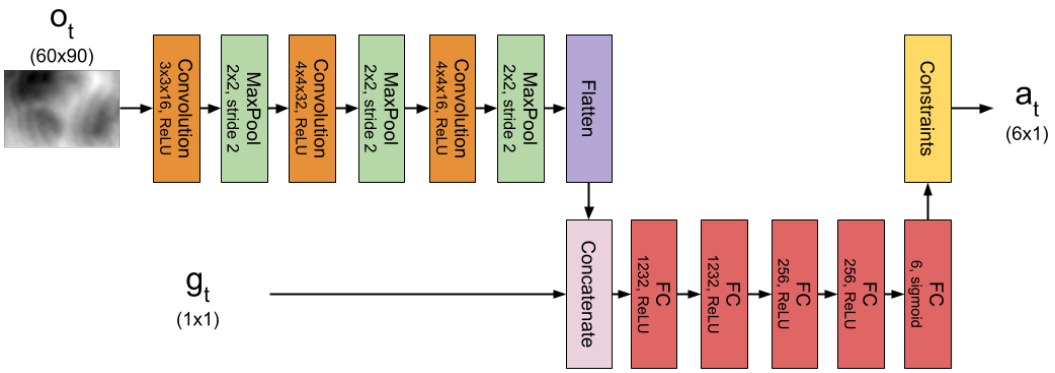

Figure 5: Architecture of the actor network. The output of the final fully-connected layer undergoes a sigmoid activation, and these activation values are converted into valid parameters of the scooping action by treating the activations as ratios for linearly interpolating between the valid bounds of each action parameter (at the layer labeled "Constraints" in the figure). Note that the bounds of the length of the scoop are dependent on the y value since the length is bounded by the distance of the entry point to the wall of the tub. The sigmoid output corresponding to the L parameter was thus used to linearly interpolate between 0 and the distance between the scoop entry point and the wall of the tub to which the scooping motion moves.

test sample efficiency, I trained each of these models on different-sized subsets of the data until their errors predicting m_t on a holdout validation set reached a minimum over 600 epochs, then computed their test error on a random subset that had been held out for testing. Hyperparameters used for this experiment are listed in Section A of the Appendix.

3.4 Learning Actor and Critic Models

Based on my findings that will be shown in Section 4.2 of training a predictive model, I decided not to use auxiliary losses on my critic model. My critic model used the exact same architecture as the predictive model shown in Figure 4, except with a final output computed as $-|g_t - p_t|$ to directly predict r_t. I trained this critic model on 90% of my full dataset, ensuring the validation error on the remaining 10% was satisfactory, for 200 epochs with an annealed learning rate. During each epoch of training, I randomly sampled a new goal g_t for each experience, according to the same distribution as that specified by the task in Section 3.2.

This critic model was all that was required for the Cross Entropy Method (CEM), but for Actor-Critic, I designed the actor network with the architecture shown in Figure 5. As suggested in [16], I used a target network instance of the actor network to stabilize learning the policy during training, and it was this target network which was used to select each action at test time.

Using the pretrained critic network and a newly initialized actor network, I prepopulated a replay buffer with the entire dataset of random trials, with 4 replicas of each experience based on 4 different random samplings of g_t and corresponding r_t, an approach conceptually similar to Hindsight Experience Replay. Before each test and trial, I pretrained the critic network and, as applicable, the actor network for one epoch on this buffer. Since the critic model had already been pretrained extensively and carefully, I used a very low learning rate for it during training on this buffer. After pretraining, I tested two different approaches with respect to handling past experiences on both CEM and Actor-Critic: retaining the replay buffer as normal or completely erasing the contents of the replay buffer. The latter case could essentially be considered a time-based prioritized replay.

After pretraining and immediately before running the policy, I slightly increased the learning rate on both the critic network and, if applicable, the actor network to allow each network to make faster adjustments to changes in the environment. Before beginning each trial, I manually flattened the surface of the pellets, then ran 100 episodes of each of these approaches (2 learning methods and 2 replay buffer approaches, permuting to 4 total experiments) without manually perturbing the contents of the tub between episodes. Hyperparameters used throughout this entire process are listed in Section B of the Appendix.

4 Results

4.1 Random Dataset

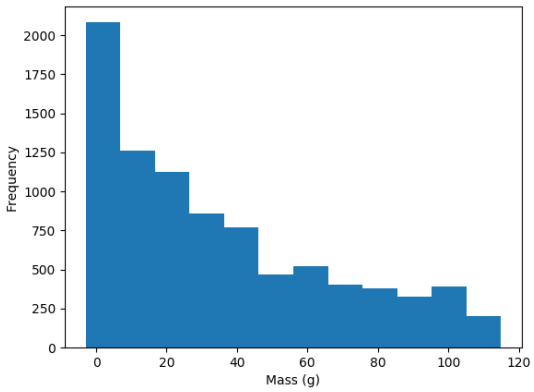

Figure 6: Histogram of scooped masses from dataset of trials of random actions.

The histogram of the scooped masses m_t from the ~ 8000 sample training dataset is shown in Figure 6. The histogram reflects that random actions produce a very right-skewed distribution, whereas for the actual task at test time, desired mass g_t is sampled uniformly randomly. Thus, the distribution of training data is relatively suboptimal for learning how to scoop higher masses. Perhaps more importantly, the distribution of observed height maps is also likely skewed within this dataset, which seemed to have very adverse effects on the performance of the algorithms over time, as will be detailed in Section 4.3.

4.2 Model Selection

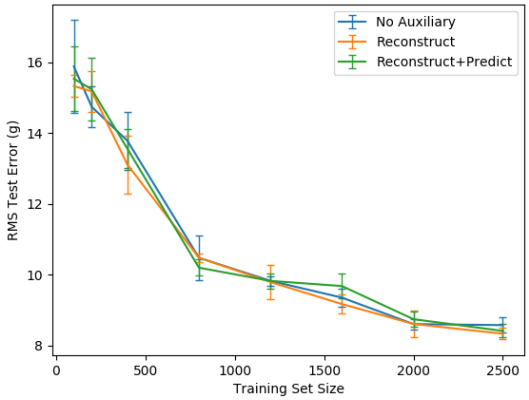

Figure 7: Test prediction errors for the predictive model with different combinations of auxiliary losses. Values are averaged over three trials.

The results of optimizing different auxiliary loss combinations jointly on my model predicting the scooped mass given an observation and action are visualized in Figure 7. Contrary to what I expected, jointly optimizing auxiliary losses showed no significant effect on the error of the data, neither in the low data nor in the high data regimes. The performance of each model was well within a standard deviation of the performance of each other model for each training set size. Since the auxiliary losses only introduced more model complexity and more hyperparameters to tune (*i.e.*, the coefficients on the different losses in their weighted linear combination), I decided not to use them in my critic or actor models for the remaining experiments. This may have been short-sighted though, as it later seemed important that my models generalize well to new distributions of observations. In this model selection experiment, my training, validation, and test sets were all drawn randomly from the same distribution as the training data from random actions. To test the efficacy of the auxiliary tasks more

rigorously, I should have experimented with training the models on the random data, then testing the models on data from a different distribution of data, *e.g.* a dataset composed of the experiences from following the policies I tested at test time. An even more ideal and rigorous approach than this would be to merely test the performance improvements these auxiliary losses offer directly while performing the task, but this would be unrealistically time-consuming to test online on the robot, as any specific configuration of hyperparameters, etc., would take at least on the order of an hour to test a single trial, let alone taking an average over multiple trials and/or experimenting with tuning hyperparameters.

4.3 Reinforcement Learning Algorithms

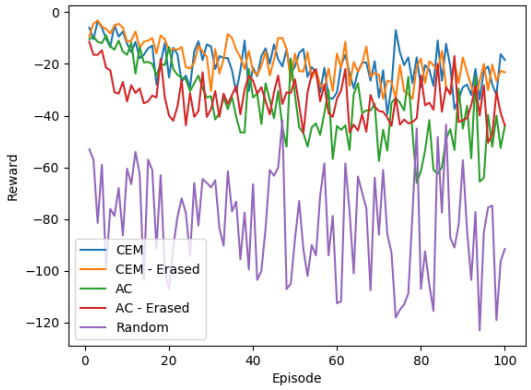

Figure 8: Rewards through first 100 episodes of each algorithm. Values are averaged over 4 trials.

Since the goals were sampled uniformly randomly on a large interval per each episode, there was a large variance in the performance of each algorithm at each episode, as can be seen in Figure 8. Note that the performance of each algorithm generally tends to degrade from the initial epochs. This is likely because flattening the surface of the pellets before beginning the 100 episodes made the task less challenging for the initial episodes. However, as the policies progressed, the surface of the pellets became much more irregular and therefore challenging to the algorithms.

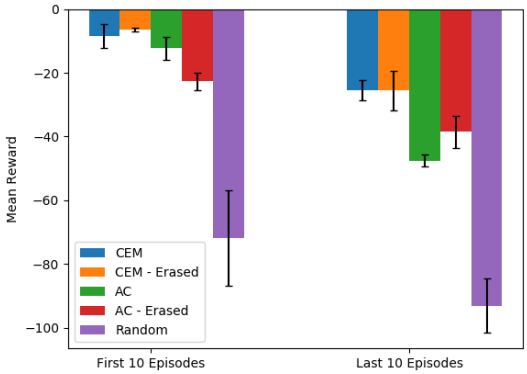

Figure 9: Mean reward per episode of each algorithm within a 100 episode trial during the first and last stages. Values are averaged over 4 trials. Note that all rewards are negative, so shorter bars reflect better performance.

Each learning algorithm substantially outperformed a random policy. However, both replay buffer variations of the Cross-Entropy Method generally outperformed both variations of my Actor-Critic method. This is especially apparent in Figure 9. Erasing the replay buffer seemed to have no significant effect on the performance of the CEM method. However, for the Actor-Critic method, erasing the replay buffer before beginning the episodes created an interesting trend in performance over time. Whereas the Actor-Critic with the full prefilled buffer performed significantly better at

the outset of the 100 episodes, the Actor-Critic with the empty buffer performed significantly better closer to the end of the 100 episodes.

Given that each algorithm used the same pretrained critic, it seems that there are at least two possible underlying explanations for the significant gaps in performance. Since the surface of the pellets was only flattened at the beginning of each trial, the structure of the surface changed throughout each trial. The random policy used to generate the initial 8000 samples likely induced a particular distribution of structures on the surface of the pellets. The skew of the distribution of masses in Figure 6 may provide some indirect evidence of this. However, the reinforcement learning algorithms tasked with scooping a desired mass, sampled uniformly randomly over the interval of possible masses, likely each induced different distributions of structures than that induced by the random policy. Thus, there are at least two possible reasons one policy could perform better than another at different time windows. First, the policy could induce a distribution of height maps that is more similar to or generalizable from the distribution of height maps from the random policy. And second, the policy may be more agile at adapting to the new distribution it has induced, perhaps reflecting that it is better at adapting to new distributions in general. Though I would be tempted to conclude that erasing the replay buffer and only training on new experiences at test time allowed Actor-Critic to adapt more quickly to the changing environment, especially given the difference in performance for the first 10 episodes, it may simply be that the Actor-Critic with the erased buffer just induced a relatively easier distribution of structure during the 100 episodes. It is thus not entirely clear which of these two possibilities was responsible for performance differences, or if it was some combination of both. Further experiments would need to be conducted to isolate each of these two possibilities, *e.g.*, manually structuring the surface of the pellets in a fixed novel shape before testing each policy.

Regardless of the differences in performance between the different learning algorithms, the drop in performance of each over time does not reflect well upon the critic network on which they are all based. One underlying hypothesis of this experiment was that the deep neural network would effectively be able to somewhat capture relationships from the underlying dynamics of the granular material. The results suggest, however, that the critic network was not especially robust at generalizing to new distributions that were, at least visually, only slightly diverged from the distribution of observations on which it had been trained. Therefore, it seems reckless to conclude that the deep network proficiently captured the underlying dynamics relationships of granular materials from its training data.

5 Conclusion

In this paper, I tested 4 different deep reinforcement learning policies for their competence on the task of a robot scooping a specified mass of a granular material. After collecting a dataset of the results of random actions to pretrain each of these policies, I evaluated each policy on the environment to compare their performances over time. The results show that the policies perform significantly better than the random policy on which they were trained, but interesting variations in performance arose as each policy was left to manipulate the environment over multiple episodes. The results suggest that, within the length of the trials I tested, the Cross Entropy Method achieved superior performance overall, with the Actor-Critic performing better initially when using a prepopulated replay buffer and performing better in later episodes when using a fresh replay buffer of only test-time experiences. Perhaps further tuning of hyperparameters of each method could have negated some of these differences in performance, and running longer trials could have confirmed or denied suspicions as to which learning algorithm adapted better over time. Thus, more experiments would need to be conducted to isolate specific explanations of differences in performance. Overall however, the results show the promise of using deep reinforcement learning in the rather challenging domain of manipulating deformable granular materials.

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[2] P. T. De Boer, D.P. Kroese, S. Mannor, and R.Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134, 2002.

[3] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[4] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement Learning with Unsupervised Auxiliary Tasks. *ArXiv e-prints*, November 2016.

[5] Chen Li, Paul B Umbanhowar, Haldun Komsuoglu, Daniel E Koditschek, and Daniel I Goldman. Sensitive dependence of the motion of a legged robot on granular media. *Proceedings of the National Academy of Sciences*, 106(9):3029–3034, 2009.

[6] Chen Li, Tingnan Zhang, and Daniel I Goldman. A terradynamics of legged locomotion on granular media. *science*, 339(6126):1408–1412, 2013.

[7] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[9] Rudranarayan M Mukherjee and Ryan Houlihan. Massively parallel granular media modeling of robot-terrain interactions. In *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 71–78. American Society of Mechanical Engineers, 2012.

[10] Leonel Rozo, Pablo Jiménez, and Carme Torras. Force-based robot learning of pouring skills using parametric hidden markov models. In *Robot Motion and Control (RoMoCo), 2013 9th Workshop on*, pages 227–232. IEEE, 2013.

[11] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017. URL `http://arxiv.org/abs/1706.05098`.

[12] Shigeru Sarata, Hisashi Osumi, Yoshihiro Kawai, and Fumiaki Tomita. Trajectory arrangement based on resistance force and shape of pile at scooping motion. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3488–3493. IEEE, 2004.

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized Experience Replay. *ArXiv e-prints*, November 2015.

[14] Tom Schaul, Dan Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1312–1320. JMLR.org, 2015. URL `http://dl.acm.org/citation.cfm?id=3045118.3045258`.

[15] Connor Schenck, Jonathan Tompson, Dieter Fox, and Sergey Levine. Learning robotic manipulation of granular media. *CoRR*, abs/1709.02833, 2017. URL `http://arxiv.org/abs/1709.02833`.

[16] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[17] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL `http://arxiv.org/abs/1409.4842`.

[18] N. Wahlström, T. B. Schön, and M. P. Deisenroth. From Pixels to Torques: Policy Learning with Deep Dynamical Models. *ArXiv e-prints*, February 2015.

[19] Akihiko Yamaguchi and Christopher G Atkeson. Differential dynamic programming for graph-structured dynamical systems: Generalization of pouring behavior with different skills. In *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, pages 1029–1036. IEEE, 2016.

Appendix A Model Selection Hyperparameters

Learning rate	5×10^{-4}
Learning rate decay coefficient	0.995 per epoch
Training epochs	600
Training dropout probability	0.5
Coefficient of MSE mass prediction loss	1
Coefficient of MSE observation reconstruction loss	1×10^{3}
Coefficient of MSE next observation reconstruction loss	1×10^{4}
Mini-batch size	32
Validation set size	700
Test set size	900

Appendix B RL Hyperparameters

Pretraining Critic on Full Dataset

Learning rate	2×10^{-4}
Learning rate decay coefficient	0.97 per epoch
Training epochs	200
Training dropout probability	0.5
Coefficient of MSE reward prediction loss	1
Coefficient of MSE observation reconstruction loss	0
Coefficient of MSE next observation reconstruction loss	0
Mini-batch size	32

Pretraining Critic on Replay Buffer

Learning rate	$< 2 \times 10^{-9}$
Training epochs	1
Training dropout probability	0.5
Coefficient of MSE reward prediction loss	1
Coefficient of MSE observation reconstruction loss	0
Coefficient of MSE next observation reconstruction loss	0
Mini-batch size	32

Pretraining Actor on Replay Buffer

Learning rate	1×10^{-6}
Training epochs	1
Training dropout probability	0.5
Target network coefficient τ	0.1
Mini-batch size	32

Critic Training During Performing the Task

Learning rate	5×10^{-5}
Training dropout probability	0.5
Coefficient of MSE reward prediction loss	1
Coefficient of MSE observation reconstruction loss	0
Coefficient of MSE next observation reconstruction loss	0
Mini-batch size	32

Actor Training During Performing the Task

Learning rate	5×10^{-5}
Training dropout probability	0.5
Target network coefficient τ	0.1
Mini-batch size	32