Deep Bin Picking with Reinforcement Learning

Jeff Chen [1] **Tori Fujinami** [1] **Ethan Li** [1]

Abstract

We investigate an aspect of the robot bin picking problem which has not been well explored, namely the task of clearing a *deep* bin of items piled on top of each other. This task is challenging, as lifting one item may cause others to move into positions which are difficult to grasp, or it may cause extraneous lifts such that other items unintentionally fall out of the bin. We show that Pybullet and Dex-Net can be used to create the simulation environment for this task, and that heuristic approximation of action spaces enables computationally tractable training of a Q-learning model with function approximation. Experimental results demonstrate that our Q network achieves an average total reward of -2.0 with hand-crafted state-action features, that it significantly outperforms both random and greedy baselines with total rewards of -9.6 and -8.2, respectively. Further, preliminary results suggest that that more complete state-action representations without hand-crafted features may have potential to improve performance.

1. Introduction

Picking items from bins is a task that is primarily performed by humans today; Amazon alone employs more than 90,000 employees in its warehouses under suboptimal working conditions (Rittenhouse, 2017; Weindling, 2017). The use of robots to automate bin picking has the potential to revolutionize commerce to assist human workers directly or automate repetitive subtasks for human efforts.

There is immense interest to use robots for bin-picking and increase efficiency and throughput while reducing cost and labor injuries to workers. Amazon even launched the Amazon Robotics Challenge to encourage research in this area. (Correll et al., 2018; Ama, 2017)

Bin picking is the most critical step in bin clearing, where the goal is to clear a pile of items by repeatedly grasping one item at a time from a bin and placing these items into other containers. Research in this space has focused on decomposing the problem into two subproblems: 1) the poses of the items are detected (perception) and 2) best grasps are calculated for removing the items (planning). This approach is difficult due to sensor noise and occlusion resulting in imperfect estimates of the item poses (Mahler & Goldberg, 2017). The task is even more challenging when a deep bin of objects must be cleared, as items tend to be stacked on top of one another.

2. Related Work

Recent work has investigated the use of deep neural networks to calculate best grasps directly from point clouds of item piles. (Mahler & Goldberg, 2017) presents one such approach by hand-crafting a supervisor policy π^* given full state information and using this supervisor for imitation learning to train a policy which uses point cloud observations rather than item pose state information.

Formulating the supervisor policy π^* is non-trivial. First, picking up one item may accidentally lift up other items, which then causes the accidentally lifted item to fall out of the bin. Second, picking up one item may cause other items move and make them more difficult to grasp. These challenges were mitigated in (Mahler & Goldberg, 2017) by simulating mostly flat item piles on a plane, with items generally not stacked on top of each other. Additionally, (Mahler & Goldberg, 2017) notes that their supervisor policy only maximizes reward for the current time step, rather than the full time horizon.

In contrast to (Mahler & Goldberg, 2017), which uses a flat bin of items, we use a cube-shaped bin. This deeper bin results in items being piled on top of each other such that items from the bottom of the pile cannot be picked up by the gripper without collision. This means that the most stable grasps that have the highest probability of success are not necessarily feasible actions. We therefore have to use reinforcement learning to find an optimal policy for choosing the order in which items should be grasped.

[*]Equal contribution [1]Department of Computer Science, Stanford University, Stanford, California, USA. Correspondence to: Jeffrey Mahler <jmahl42@gmail.com>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

(Lillicrap et al., 2015) discusses one of our main issues. For robotics, the action space is generally continuous, but generally learning with a Deep Q Network (DQN) only does well with small discrete action spaces. Often, discretization of the action space may seem like an initially feasible solution, but issues remain with dimensionality and the coarseness of discretization needed for good control. (Lillicrap et al., 2015) solves this issue by instead taking the ideas underlying DQN and applying them to policy gradients. Particularly, they build an actor-critic, model-free, deterministic policy gradient to operate over the continuous action space.

(Gu et al., 2016) extends regular Deep Q-learning to apply to a continuous action space by outputting a value function term and an advantage function term. They then use model-guided exploration and rollouts with synthetic samples with fitted dynamics to improve sample efficiency.

Although we do not take the solutions presented in either (Lillicrap et al., 2015) or (Gu et al., 2016) directly, we adapt DQN to our continuous state and action space by leveraging the known set of items and all possible actions for each of those items to create a finite action space. We prune the action space and specially encode states and actions based on which features we think are important for learning. This allows us to minimize our state-action space representation to make learning feasible.

3. Approach

With a deep bin of items, removal of one item usually causes other contacting items to move, changing their available grasp locations with potential delayed consequences for how efficiently the remaining items can be removed. Furthermore, removing an item may cause other items resting on it to be pulled up and fall out of the box extraneously. Thus, optimal planning of the sequence of grasps for removing items may need to account for how grasps in the action space become feasible or infeasible due to physical interactions resulting from the grasp sequence. Given these challenges with delayed consequences and the difficulty of comprehensively modeling the consequences of item removal actions, we explore reinforcement learning approaches to train a policy to maximize reward over the entire time horizon of the deep bin picking task.

3.1. MDP Model

We formulate the problem as an MDP (S, A, P, R, γ):

- S (States): The state is the set of identifiers and poses of items currently in the pile. An item pose consists of its Cartesian position (x, y, z) and its orientation represented as quaternion (q_0, q_1, q_2, q_3).
- A (Actions): The actions for a state consist of the available grasps g for each item i in that state, along with their respective probabilities of grasp success p_g. The grasp locations for a single item are depicted in Figure 2. Each action corresponds to an attempt to remove an item from the pile with a specific grasp. Each grasp is a 4-DOF gripper pose $g = (p, \theta, d)$ where p is the grasp center location (x, y) of the vertical axis along which the gripper travels, θ is the orientation of the gripper about the grasp's vertical axis, and d is the grasp depth along the grasp's vertical axis. The action space is defined as a pre-computed sampled set of grasps for every item, extracted from the Dex-Net 2.0 database. Dex-Net also returns a grasp quality metric calculated using the Ferrari-Canny L1 score, which we use to approximate p_g as described in 3.3. At each time step, actions for the remaining grasps at the current state are pruned using collision checking through Dex-Net so that, for any given state, any actions which cause a collision between the gripper and another item are discarded. Further details on the motivation and approach for this are described in 3.4.

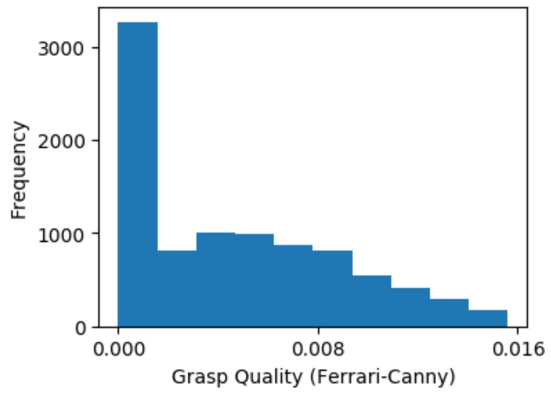

Figure 1. Distribution of grasp metrics across all grasps in the database. Grasp probability is calculated linearly rescaling and clamping the metrics so that grasps with a metric in the top 10% succeed with a probability of 0.9.

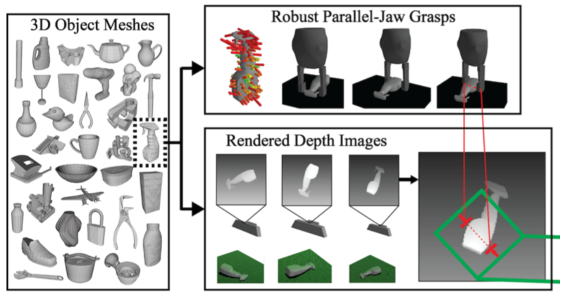

Figure 2. Image from (Mahler et al., 2017) showing grasp locations on an item returned from Dex-Net

- P (Transition Probabilities): The next state is the set of item locations and orientations simulated to static equilibrium after an attempted removal. Given the current pile of items and the grasp and item specified by the action, that item is successfully removed from the pile with the grasp success probability p_g; otherwise, the state remains the same.
- R (Rewards): We assign a reward of $+1$ for successfully removing an item from the environment and a penalty of -10 for every extraneous item which unintentionally falls out of the crate during removal of a different item.

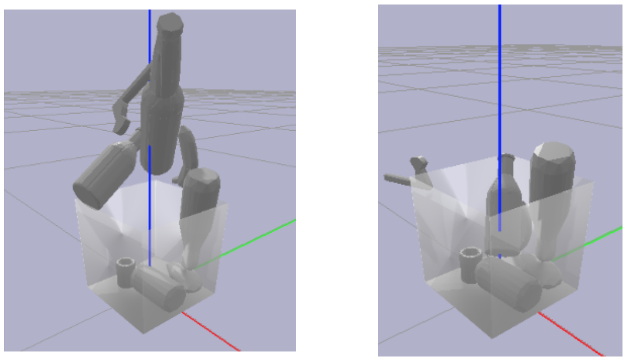
Figure 3. Taking out a bottle (left) causes a hammer to fall out of the crate (right)

- γ (Discount Factor): To encourage our policy to finish removing items efficiently, we introduce a discount factor of $\gamma = 0.9$.

We configure our model to terminate when no feasible, collision-free grasps are available or when 10 consecutive grasps have failed, indicating that the policy cannot find any grasps likely to succeed.

3.2. Simulator

We developed a custom simulator using the Pybullet physics simulator and Dex-Net's collision checking functionality to simulate environment dynamics. In our simulator, we use a cubic crate and 96 items exported from the Dex-Net's subset of the 3DNet database, selected such that items fit in the simulated crate and have valid properties for well-behaved physics simulation.

3.2.1. State Initialization

To sample from initial environment states, we generate a pile of items in the simulated crate by sequentially loading a randomized set of unique items with uniformly-sampled random friction coefficients, dropping them from uniformly-sampled random positions and orientations above the crate,

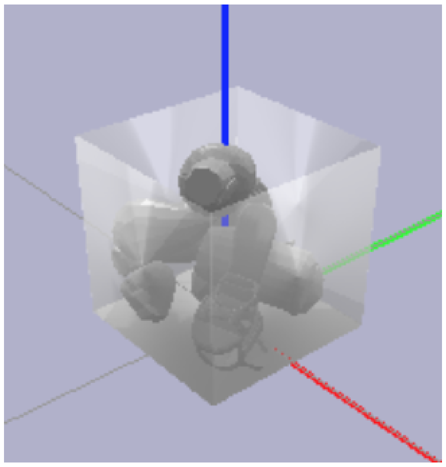
Figure 4. A crate of items as the initialized state

and simulating the environment until it reaches static equilibrium in our Pybullet simulator. Static equilibrium is determined by comparing the change in position and orientation of each item over a short time interval to thresholds for these displacements. The key difference between our approach and the approach in (Mahler & Goldberg, 2017) is our use of a crate-shaped bin, rather than a flat plane, for our picking environment. As described above, our approach makes environment dynamics and future rewards more sensitive to actions, as the stacking of items in the crate introduces potential consequences for items near the item removed by any action. Because we chose a coarse simulation time step in order to run simulations quickly, items occasionally get stuck intersecting the walls or floor of the crate. This presents challenges in simulating item removal, as described next.

3.2.2. Item Removal

We simulate removal of an item specified by action $a = (i, g, p_g)$ by first sampling a Bernoulli random variable with parameter p_g to determine whether the attempted grasp succeeds. If the grasp fails, we conclude the item removal attempt. If the grasp succeeds, we pull the item up at constant velocity until it reaches a threshold height. If the item is stuck due to intersecting the walls or floor of the crate, we delete it instead of pulling it up. We then simulate the environment until it reaches static equilibrium.

3.3. Action Space Approximation

Because the true sets of item and grasp poses are continuous, the true set of actions for picking up items is also continuous. We make this space manageable by sampling a large but finite set of actions from Dex-Net, and we avoid running relatively slow collision checks on all actions in this set by first pruning actions based on grasp success probability and by implementing multi-pass collision checking for all actions for each item.

3.3.1. Grasp Success Probability

We start by getting all grasps for every item in Dex-Net. Each grasp comes with an associated grasp quality metric calculated using the Ferrari Canny L1 score, which is proportional to the amount of force and torque perturbation required to knock an item out of the gripper jaws in grasp. However, in simulation, this metric is only indirectly related to the real probability of a successful grasp. For the purposes of learning in simulation, we convert the metric to a grasp probability by linearly rescaling and clamping the metrics so that grasps with a metric in the top 10% succeed with a probability of 0.9.

3.3.2. Sampling a Large Action Space

At each time step, we sample the resulting set of actions to generate a candidate list $A_{candidates}$ of the feasible actions, namely actions which would not cause collisions between the gripper and other items in the crate. The following techniques were adopted out of necessity to speed up simulation speed by an order of magnitude.

We pre-process grasps by pruning grasps with Ferrari Canny L1 scores of less than 0.001 (which correspond to success probabilities of less than 8%) because they are not likely to succeed, which helps to speed up simulation.

The first pass across the action space searches for a single collision-free action for each item. The items are first sorted based on item height (decreasing Z positions) so that we can guarantee that the first two actions in the list corresponding to the top two items in the bin are returned if any grasps exists. For the first two items, we thoroughly search all possible grasps to ensure grasps for highest positioned items are not missed.

After the first two items, we find actions for other items by sampling three actions. These actions are chosen as the 1st, 11th, and 22nd grasps from the list of all grasps for that item, sorted by decreasing p_g. Gripper-item collision checking is performed sequentially on these three grasps to check grasp feasibility, and the first feasible grasp found (if any) for that item is added to $A_{candidates}$, and the item is excluded from further searching. If none of the three actions are collision-free, the item is moved to a list of items which are unlikely to have successful grasps. If there are enough actions in $A_{candidates}$ after the first pass through the items, then it is returned and no further actions are checked for feasibility.

If there are not enough actions in $A_{candidates}$ after the first pass through the items, a second search is conducted to find up to one random feasible action for each remaining item. For every item in the unlikely list, a grasp is randomly chosen without replacement and checked for gripper-item collisions. If the grasp is collision free, this item is removed from the unlikely list, and the grasp's corresponding action is added to $A_{candidates}$. If not, then the item remains in the unlikely list and a random action for the next item is searched. The search through actions for the unlikely items continues until $A_{candidates}$ grows to the minimum number of actions, until there are no unlikely items left, or until a maximum of 10 iterations over the unlikely item list have been completed.

3.4. Dex-Net Collision Checking

Dex-Net 2.0 provides functionality for collision checking between items and grippers. This collision checking is both faster and performs better than collision checking through Pybullet simulation. (Mahler et al., 2017) Item meshes and grasps with item-to-world transforms are loaded into the Dex-Net collision checker so that each grasp can be evaluated for feasibility. The collision checker checks for collisions in both approach of the gripper to the item and in the closing of the gripper jaw on the item, using rigid body transformations of each item in the state given the item positions and orientations.

3.5. Baseline Heuristic Policies

We evaluate various hand-crafted heuristic baseline policies π on our MDP and analyze whether and how we can improve upon the heuristic algorithmic supervisor policy π^* presented in (Mahler & Goldberg, 2017) to maximize reward over the full time horizon in our simulated deep bin-picking environment.

Two unrealistic policies always successfully remove the specified item, regardless of action feasibility or grasp success probability. Thus, they act as extreme bounds on policy performance in our task:

- π_{lowest}: To estimate a lower bound on policy performance, this policy always successfully removes the lowest item in the crate. It tends to cause extraneous items to fall out of the box as each item is removed.
- $\pi_{highest}$: To estimate an upper bound on policy performance, this policy always removes the highest item in the crate.

Three baseline policies only select actions from $A_{candidates}$, the approximate set of feasible collision-free actions. The actions these policies choose will succeed with the grasp success probabilities p_g encoded in the actions:

- π_{random}: This baseline is equivalent to the algorithmic supervisor π^* from (Mahler & Goldberg, 2017), which randomly picks an action from $A_{candidates}$.

- π_{greedy}: This baseline chooses the action with the highest p_g from $A_{candidates}$.
- $\pi_{greedyhighest}$: This baseline chooses the action from $A_{candidates}$ which would remove the highest item in the crate removable by any action from $A_{candidates}$.

3.6. Reinforcement Learning

Since the dynamics of our environment are too complex to parameterize with an a priori model and the size of our state space makes solution of our MDP intractable, we investigate Q-learning with function approximation and experience replay for model-free learning. Specifically, we evaluate linear approximation and a neural network with two fully-connected hidden layers.

Because our complete sampled action space is very large, we implemented our function approximators and Replay Buffer from scratch in Tensorflow, with some snippets adopted from homework 2.

We construct state-action value function approximators $\hat{q}$ which take a state and action (s, a) as the input and output a single approximation of $q(s, a)$, rather than taking a state as input and outputting approximations of $q(s, a)$ for all a in our action space. Our action space approximation builds a list $A_{candidates}$ of at most one candidate action per item in the crate, so evaluating $\max_{a \in A_{candidates}} \hat{q}(s, a)$ given s is computationally efficient. Our replay buffer stores the current state and action, along with the current the variable-size $A_{candidates}$, which is represented as a fixed-size array with a mask array corresponding to the length of $A_{candidates}$.

3.6.1. Input Feature Representations

We evaluate function approximators using two different input representations of (s, a): a simpler representation $\phi_{simple}(s, a)$ consisting of features hand-crafted using our domain knowledge to evaluate whether our function approximator can learn to outperform $\pi_{greedyhighest}$, and a more complete representation $\phi_{complete}(s, a)$ to evaluate whether our function approximators can learn the relevant features from the complete state and action data.

In the simpler representation, $\phi_{simple}(s, a)$ is a vector consisting of:

- The item identifiers and z positions of the centers of all items, encoded as one-hot-like vectors $z_i \cdot \vec{e}_i$ when item i is in the crate, and $\vec{0}$ otherwise, where $\vec{e}_i$ is the one-hot vector label for item i. These 96 vectors, corresponding to the maximum of 10 are then concatenated.
- The grasp depth d of action a.

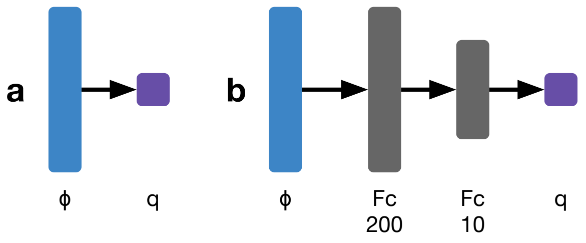

Figure 5. Function approximator architectures for (a) linear function approximation, (b) function approximation with a feedforward neural network consisting of two fully-connected layers . Note that ϕ_{simple} has approximately 960 elements, while $\phi_{complete}$ has approximately 1030 elements.

- The grasp success probability p_g of action a.
- The z position of the center of the item to be removed by the action.

In the complete representation, $\phi_{complete}(s, a)$ is a vector constructed by concatenating:

- A vector created by concatenating the vectors $s = (i, x, y, z, q_0, q_1, q_2, q_3)$ of all items in the crate, in arbitrary order. Item identifiers i are encoded as one-hot vectors.
- An action vector, excluding the item identifier for the item removed by the action.

3.6.2. Function Approximator Architectures

Each function approximator architecture takes either ϕ_{simple} or $\phi_{complete}$ as its input layer:

- Our linear model $\hat{q}_{linear}$ consists of a one-unit linear output layer (Fig 5a).
- Our neural network model $\hat{q}_{nn}$ consists of two fully-connected hidden layers with ReLU activations before a one-unit linear output layer. The first hidden layer has 200 units, while the second hidden layer has 10 units (Fig 5b).

4. Results and Evaluation

4.1. Baseline Policies

Figure 6 shows the results for all the baselines discussed above. As expected, π_{lowest} and $\pi_{highest}$ provide bounds on the rewards accumulated, while π_{random}, π_{greedy}, and $\pi_{greedyhighest}$ fall somewhere in between. The average total discounted return for π_{greedy} is significantly higher

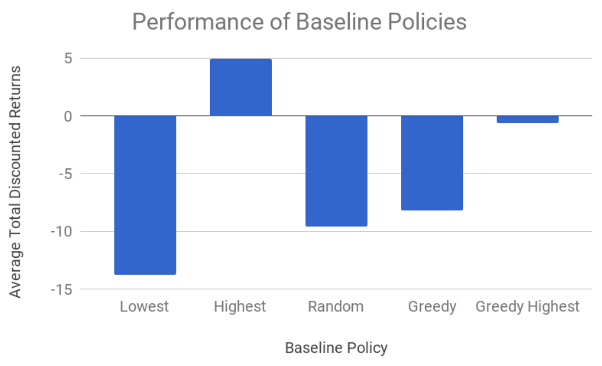

Figure 6. Results from running baseline policies in simulation for 500 episodes, by which time the average total discounted returns for all baselines have converged. Lowest: -13.8, Highest: 4.9, Random: -9.6, Greedy: -8.2, Greedy Highest: -0.7

from that of π_{random}, with a p-value of 0.042 for the one-tailed Student's t-test with independent samples and unequal variances. The average total discounted return for $\pi_{greedyhighest}$ is significantly different from those of the other baselines, with p-values less than 10^{-18} for the Student's t-test. Thus, we conclude that our novel heuristic $\pi_{greedyhighest}$ policy outperforms the greedy $\pi_{random} = \pi^*$ algorithmic supervisor from (Mahler & Goldberg, 2017) on the deep bin picking task.

4.2. Linear Approximation with $\hat{q}_{linear}$

We did not expect our linear approximator $\hat{q}_{linear}$ to learn or perform very well. The average reward plot for the linear model is displayed in Figure 7. The model achieves a maximum average total reward of -4.5 after 28,000 iterations.

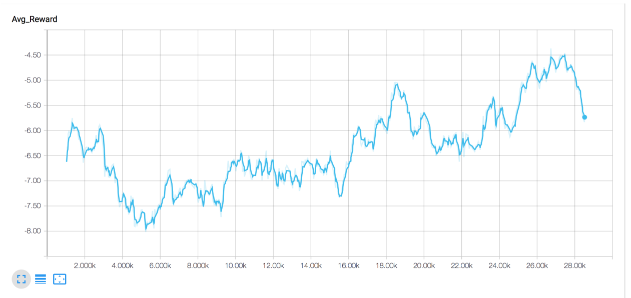

Figure 7. Average reward plot for the linear function approximator using ϕ_{simple} features. While the model is learning, it is only able to achieve a maximum total reward of -4.5 after 28,000 iterations, which took over 40 hours to train.

4.3. Neural Network Approximation with ϕ_{simple}

Figure 8 shows the average reward curve for the neural network function approximator $\hat{q}_{nn}$ with hand-crafted feature inputs ϕ_{simple}. This model performs significantly better than the Linear Approximator. A comparison of average total rewards for linear and neural net models can be seen in Table 2.

parameter	value
batch size	50
buffer size	100000
target update frequency	1000
learning frequency	4
initial learning rate	0.0025
final learning rate	0.00005
initial ϵ	1
final ϵ	0.1
number of steps between ϵ initial and final	20000
running average size	250

Table 1. Parameters used in neural network training

Table 1 details all the hyper-parameters tuned for training our neural networks.

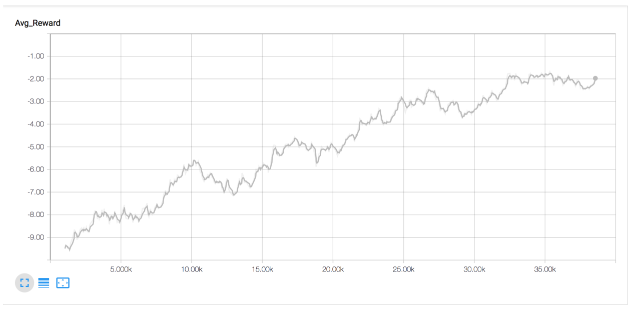

Figure 8. Average reward plot for the neural network approximator using ϕ_{simple} features. The model learns well, starting from an average reward of -9.5 and achieving -2.0 after 36,000 iterations, which took 46 hours to train.

4.4. Neural Network Approximation with $\phi_{complete}$

We additionally tried to train function approximator $\hat{q}_{nn}$ on the complete state-action input $\phi_{complete}$. The results are displayed in Figure 9. The learning curve here suggests that the network is learning. At this point in the learning curve for the neural network trained with ϕ_{simple}, the value was similarly sitting below -6.0, suggesting that more data is still needed to train the network using $\phi_{complete}$.

5. Discussion

We demonstrate the viability of training and evaluating deep bin picking using the Pybullet physics simulator with precomputed grasp metrics from Dex-Net 2.0. Simulation speed is paramount due to the large number of examples required. We show that simulation speed can be improved

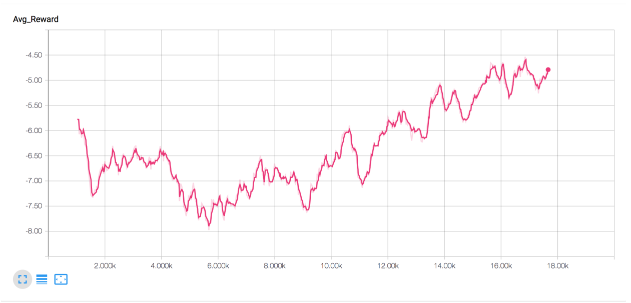

Figure 9. Average reward plot for the neural network approximator using $\phi_{complete}$ features. The model is learning, starting from a minimum average reward of -8.0 and achieving a maximum average reward -4.5 after 20 hours of training.

Model	Average Total Reward
$\pi_{highest}$ upper bound	4.9
$\pi_{greedyhighest}$ baseline	-0.55
NN Model + ϕ_{simple}	-2.0
NN Model + $\phi_{complete}$	-4.5*
Linear Model + ϕ_{simple}	-4.5
π_{greedy} baseline	-8.2
π_{random} baseline	-9.6
π_{lowest} lower bound	-13.8

Table 2. Performance of models against baselines. The neural network model $\hat{q}_{nn}$ with action input significantly outperforms greedy and random baselines. The greedy highest baseline uses domain knowledge and always attempts to grasp the highest item regardless of success probability. Highest upper bound is the theoretical total reward upper bound. *NN model $\hat{q}_{nn}$ with $\phi_{complete}$ has potential to outperform NN model $\hat{q}_{nn}$ with ϕ_{simple} given enough time to train, as it has not yet converged after 20 hours of training.

by approximately 10x through careful selection of simulation parameters and novel heuristics for grasp sampling and pruning in the search for collision-free grasps.

Deep bin picking is a difficult problem as demonstrated by the performance of our random and greedy baselines achieving average total rewards of -9.6 and -8.2, respectively, which is much closer to the lower bound policy (-13.8) than the upper bound policy (4.9).

We demonstrate the successful deployment of Q-learning with a function approximator taking an action input and a corresponding single output. Our neural network handcrafted features $\phi_{simple}(s, a)$ were used to achieve an average total reward of -2.0, which dramatically outperformed both the random and greedy baselines but did not outperform the $\pi_{greedyhighest}$ baseline, which includes human intuition that picking the highest item would result in the lowest rate of accidental removal of extraneous objects. The linear model also trained on the task but performed worse as expected, achieving a maximum average total reward of -4.5. This suggests that $\hat{q}(s, a)$ is not entirely a linear function of s and a. Final comparisons among all models are shown in Table 2.

Finally, we try to improve performance with the complete state representation $\phi_{complete}(s, a)$ instead of the handcrafted features of $\phi_{simple}(s, a)$. This is a much more challenging task for the function approximator because now the approximator has to infer the item to be acted upon through the position of the action. After training for 16,000 iterations, the neural network model achieves an maximum average total discounted reward of -4.5, which is comparable to that of $\hat{q}_{nn}$ with ϕ_{simple} at 16,000 iterations. However, due to the complexity of the task, we expect this model to require more data to converge, and it may need a deeper neural network architecture to fit the inputs.

6. Key Challenges and Future Work

One of our biggest challenges for this project was dealing with the speed of simulations and collision checks on a large action space. These factors resulted in simulated episodes being too slow to produce lots of training data. With our action space sampling and ϕ_{simple} feature encoding, we were able to increase performance speed by a factor of ten. However, these improvements only enabled us to run about 20,000 samples per day, which is still too slow to train models with more parameters. In order to attain results more comparable to deep Q learning, we would have to speed up the simulation rate by two more orders of magnitude, which would likely use of parallel methods to deep reinforcement learning, such as in (Nair et al., 2015).

Translating between mathematical representations and physics realism of grasp successes is also a challenge. Grasp success outcomes are sampled over grasp success probabilities, but these probabilities are heuristically estimated from a grasp quality metric by ignoring contact with other items. Thus, our grasp success probabilities are unlikely to transfer to the real world. An alternative is to simulate other items, and estimate all forces acting on the item, though this would cause additional slow downs in simulation.

Finally, there is a delicate balance between feeding a model with the entire state versus hand tuned features. While hand tuned features may require less parameters and train more quickly because of the reduced state space, giving the model the entire space may lead to higher performance at the cost of slower and more difficult training. Larger networks with full state representation and more training data should be attempted, as they have potential to surpass our $\pi_{greedyhighest}$ heuristic baseline policy.

7. Acknowledgements

We'd like to thank Jeff Mahler for his support in helping us understand the Dex-Net papers and the usage of Dex-Net 2.0. Further, we'd like to thank James Harrison, Emma Brunskill, Anchit Gupta, and Tian Tian for their guidance on the many challenges we faced throughout this project.

References

2017 Robotics Challenge Official Rules. Amazon Robotics LLC, 300 Riverpark Drive, North Reading, Massachusetts 01864, 2017.

Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M., and Wurman, P. R. Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1):172–188, Jan 2018. ISSN 1545-5955. doi: 10.1109/TASE.2016.2600527.

Gu, Shixiang, Lillicrap, Timothy, Sutskever, Ilya, and Levine, Sergey. Continuous deep q-learning with model-based acceleration. In Balcan, Maria Florina and Weinberger, Kilian Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2829–2838, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/gu16.html.

Lillicrap, Timothy P., Hunt, Jonathan J., Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. URL http://arxiv.org/abs/1509.02971.

Mahler, Jeffrey and Goldberg, Ken. Learning deep policies for robot bin picking by simulating robust grasping sequences. In Levine, Sergey, Vanhoucke, Vincent, and Goldberg, Ken (eds.), *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pp. 515–524. PMLR, 13–15 Nov 2017.

Mahler, Jeffrey, Liang, Jacky, Niyaz, Sherdil, Laskey, Michael, Doan, Richard, Liu, Xinyu, Ojea, Juan Aparicio, and Goldberg, Ken. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. 2017.

Nair, Arun, Srinivasan, Praveen, Blackwell, Sam, Alcicek, Cagdas, Fearon, Rory, Maria, Alessandro De, Panneershelvam, Vedavyas, Suleyman, Mustafa, Beattie, Charles, Petersen, Stig, Legg, Shane, Mnih, Volodymyr, Kavukcuoglu, Koray, and Silver, David. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015. URL http://arxiv.org/abs/1507.04296.

Rittenhouse, Lindsay. Amazon Warehouse Employees' Message to Jeff Bezos – We Are Not Robots, 2017. URL https://www.thestreet.com/story/14312539/1/amazon-warehouse-employees-discuss-grueling-work.html. [Online; accessed 1-March-2018].

Weindling, Jacob. 7 Examples of How Amazon Treats Their 90,000+ Warehouse Employees Like Cattle, 2017. URL https://www.pastemagazine.com/articles/2017/12/7-examples-how-amazon-treats-their-90000-warehouse.html. [Online; accessed 1-March-2018].