Robotic Grasping in Clutter: Using a Hierarchy of Supervisors for Deep Learning from Demonstrations

Michael Laskey[1], Jonathan Lee[1], Caleb Chuck[1], David Gealy[1],Wesley Hseih[1],Florian T. Pokorny, Anca D. Dragan[1], and Ken Goldberg[1,2]

***Abstract*— Online learning from demonstration algorithms such as DAgger can learn policies for problems where the system dynamics and the cost function are unknown. However they impose a burden on supervisors to respond to queries each time the robot encounters new states while executing its current best policy. Traditionally there has only been one supervisor and it is an expert. However, an expert supervisor is a costly resource that could be impractical for large scale learning tasks. However, there an expert supervisor might not be needed for a large portion of the state space , which could be handled by less-skilled supervisors. We present the Gambler's DAgger, which instead of only using an expert supervisors presents a hierarchical supervisor model, that first leverages cheaper supervisors, such as analytical methods on approximate dynamics or crowdsourced Amazon Mechanical Turk workers to train the policy. Then only consults an expert supervisor for harder to learn states. We demonstrate this approach in training a visuomotor deep network policy in TensorFlow for grasping in clutter on a 4-DOF Zymark Robot. We were able to reduce the number of queries to the final expert supervisor by X and manage to collect XK examples by leveraging the hierarchical supervisor model.**

I. Introduction

Consider a robot trying to reach an object on a cluttered shelf in a shipping warehouse, as illustrated by the Amazon Picking Challenge at ICRA 2015 [**?**], this is an important task for being able to efficiently ship and process orders. The robot is presented with a visual feed of the environment and must plan a series of motions to push the cluttered obstacles and grasp a goal object. A challenge could arise when the robot encounters an unknown obstacle object that it does not have a known dynamics for and must manipulate the environment to reach goal.

One solution is to learn a manipulation policy across a variety of obstacle objects and try to generalize this approach to unknown obstacle in the environment. Reinforcement learning techniques are capable of being applied to this problem, however they could require a large amount of trials on the robot to accomplish the task when the dynamics are unknown and the state space is high dimensional image data [**?**].

Rather than using an unsupervised learning approach, we propose guiding the robot how on to interact with the environment via a supervisor, who knows how to accomplish the task [5]. Learning from demonstration (LfD) algorithms, which have been used successfully in recent year for a large number of robotic tasks, including helicopter maneuvering [2], car parking [3], and robot surgery [20], is a form of how to provide this guidance.

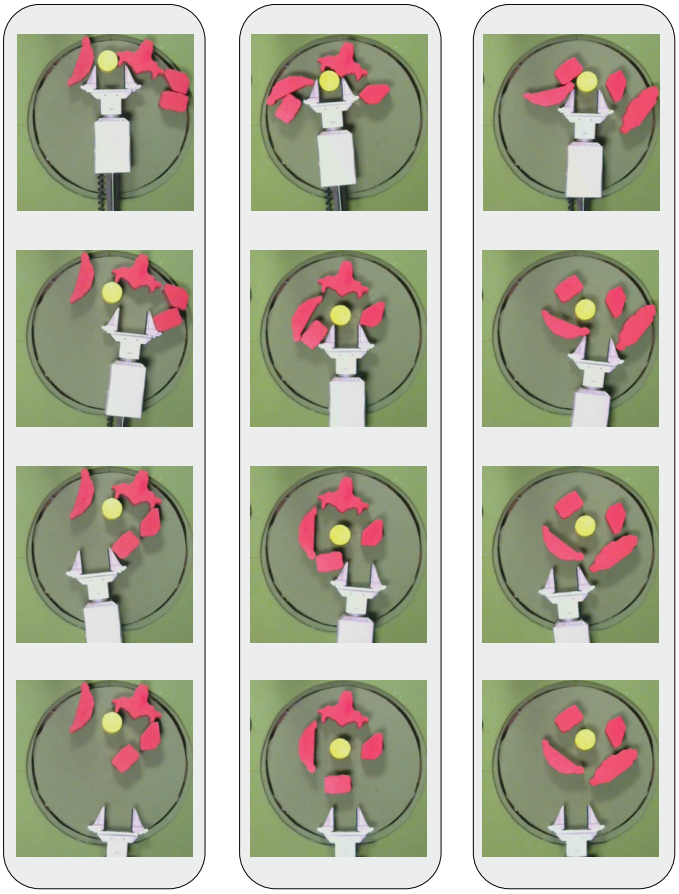


Fig. 1: Typical Initial States for our grasping in clutter task with a Zymark Robot. Red shapes indicate the clutter objects and the yellow circle indicates the goal object. The robot learns from 250x250 RGB image data to push the shapes and reach the yellow circle through our algorithm A Gambler's DAgger.

LfD algorithms can be categorized as offline, where the robot only observes demonstrations, or online, where the robot interacts with the world and receives feedback from the supervisor. In offline LfD, the robot learns the policy based on a batch of examples, and then executes it to achieve the task. During execution, a small control error can accumulate, leading the robot away from the region of the state space where it was given examples, leading to unrecoverable failures. For example, a robot trained on examples driving safely down the center of a lane, but even slight deviations will eventually put the robot into states near the side of the road where its policy could fail [15]. Ross and Bagnell showed the number

[1] Department of Electrical Engineering and Computer Sciences; {mdlaskey,iamwesleyhsieh,ftpokorny,anca}@berkeley.edu, staszass@rose-hulman.edu

[2] Department of Industrial Engineering and Operations Research; goldberg@berkeley.edu

[1−2] University of California, Berkeley; Berkeley, CA 94720, USA

of errors made by the robot, in the worst case, can scale quadratically with the time horizon of the task [16].

Online LfD addresses this issue by iteratively requesting more examples from the supervisor in states the robot encounters [9], [16], [17]. One such algorithm, DAgger, learns a series of policies. At each iteration, the robot computes a policy based on the existing examples, then rolls out (executes) that policy, and the supervisor provides demonstrations for all states the robot visits. The new state/control examples are aggregated with the prior examples for the next iteration. DAgger and related algorithms have been applied in a wide range of applications, from quadrotor flight to natural language to Atari games [10], [8], [18]. In DAgger, under certain conditions, the number of errors scales only linearly with the time horizon of the task [17].

One drawback is that DAgger imposes a substantial burden on the supervisor, who must label all states that the robot visits during training. Traditionally there has only been one supervisor and it is deemed an expert [16], [17], [18], [8]. However, when learning complicated tasks with high stochasticity in the environment this could require a large amount of the expert's time, which is a costly resource.

Learning can be bootstrapped with a sequence of increasingly skilled supervisors. In our grasping in clutter example, the robot spends a large portion of the time learning how to go towards the goal object. Analytical techniques such as forward kinematics, template matching and motion planning are readily available to provide examples in these parts of the state space.

We propose bootstrapping a policy from a hierarchy of supervisors ranked by cost: analytical models, crowd-sourced workers and an expert supervisor to learn the robot grasping in clutter task. We demonstrate this approach by training a deep neural network for the visuo-motor based task of grasping in clutter on a Zymark Robot. Our robot consists of an 2DOF arm and a gripper that lie in a planar workspace. The objects dataset consists of 20 extruded polygons taken from 2D projections of objects in the Dex-Net [**?**] database. Results show that by leveraging a hierarchy of less skilled experts, the total amount of cost incurred is reduced by X% and the task is 61% successful on our test set of 20 object configurations of shapes not trained on **ML: need to think of a better teaser statement**

II. Related Work

Below we summarize related work in Robotic Grasiping in Clutter, Online LfD setting and then the field of Curriculum learning, which is similar to the concept of hierarchical supervisors.

Robotic Grasping in Clutter Robotic grasping has been extensively studied over the past four decades [**?**]. A significant amount of prior work focuses on planning grasps given a known object. However, in unstructured environments clutter poses a significant challenge for reaching the planned grasp [**?**]. Prior work has addressed integrated perception and grasping in clutter where the objective is to grasp objects in an unorganized pile [**?**], [**?**], but these methods do not specifically aim to grasp a single target object in clutter. Leeper et al. [] use a human operator for assistance to guide the robot through clutter with the objective of grasping a target object. However, the robot required the human to be present at all times and did not attempt to learn to operate autonomously.We are interested in only querying the supervisor for examples until the robot is successful at the task.

Prior work has studied the issue of manipulating objects by performing pushing operations []. Berenson et al. [] use a sampling-based planner that considers clearance from obstacles in the environment to plan a grasp approach trajectory in cluttered environments. Cosgun et al. [3] and King et al. [**?**] consider the problem of planning a series of push operations that would place an object at a desired target location. Kiteav et al. planned a trajectory in a physics simulator using LQG based controllers [13]. However, all of these works assume a known model of the world is given to them, we are interested in learning manipulation polices via leveraging an supervisors intuition for the the task. **ML: We should discuss how to distinguish our work (Properties of our approach are we assume raw image data, real time and unknown dynamics)**.

Online Lfd with an Expert Supervisor Successful robotic examples of Online Learning From Demonstration where an expert has been employed in flying a quad-copter through a forest, navigating a wheel chair across a room and teaching a robot to follow verbal instructions and surgical needle insertion [18], [12], [8], [**?**].

However, to date these approaches have all used only one expert supervisor to provide examples for all parts of the state space. Our contribution consists of the use of a hierarchical structure of different skill level supervisors to reduce learning cost.

Reducing Supervisor Burden in Online LfD One approach that has been studied to reducing queries to the supervisor is applying active learning to only ask for queries one the robot is uncertain about the correct control to apply. Traditional active learning techniques like query-by-committee and uncertainty sampling have been applied by [7], [11], [9]

However, Kim et al. demonstrated that due to the non-stationarity of the problem traditional active learning techniques may not work because the underlying state distribution changes and thus the use of novelty detection is needed to be more conservative in determining uncertainty [12]. Laskey et al. introduced SHIV, for using active learning tailored to high dimensional and non-stationarity state distributions and provided a modified version One Class SVM to reduce the from density estimation, which requires regression, to the simpler regularized binary classification [**?**].

Novelty detection being needed to reduce supervisor burden via active learning implies that for problems with high stochasticity, such as grasping in clutter, the expert would be queried a significant number. Furthermore, we are interested in using deep neural architectures to learn task specific features for the task. SHIV applies novelty detection in a known feature space, however the feature space now changes each iteration. Thus, it is not straightforward to apply existing techniques.

Curriculum Learning **ML: TODO: add more curriculum** Our approach is similar to the area known as curriculum learning, where a neural network is trained via first easier to learn examples and then gradually increasing the difficulty

of the examples to learn [6].

Sanger et al. used curriculum learning in robotics to gradually train a neural network policy to learn the inverse dynamics of a robot manipulator. They then varied then data collection scheme such that easily learned trajectories where shown first and then gradually increased the difficulty [**?**].

Our approach is different than curriculum learning in that a supervisor can be provide low quality examples in parts of the state space visited by the robot, which means it isn't neccesarily easier for the robot to learn one supervisor's policy versus another in the hierarchy. Combining hierarchical supervisors and curriculum learning could be a promising avenue for future work though.

III. Problem Statement

The goal of this work is to learn a policy that matches that of an expert supervisor's while asking expert supervisor with fewer queries than DAgger.

Assumptions and Modeling Choices We assume a known state space and set of controls. We also assume access to a robot or simulator, such that we can sample from the state sequences induced by a sequence of controls.Lastly, we assume access to a set of supervisors who can, given a state, provide a control signal label. We additionally assume the supervisors can be noisy and imperfect, noting that a lower cost supervisor also has lower quality.

We model the system dynamics as Markovian, stochastic, and stationary. Stationary dynamics occur when, given a state and a control, the probability of the next state does not change over time. Note this is different from the non-stationary distribution over the states the robot encounters during learning. We model the initial state as sampled from a distribution over the state space.

Policies and State Densities. Following conventions from control theory, we denote by $\mathcal{X}$ the set consisting of observable states for a robot task, consisting, for example, of high-dimensional vectors corresponding to images from a camera, or robot joint angles and object poses in the environment. We denote by $\mathcal{U}$ a set of allowable control inputs for the robot, which can be discrete or continuous. We model dynamics as Markovian: the probability of state $\mathbf{x_{t+1}} \in \mathcal{X}$ can be determined from the previous state $\mathbf{x}_t \in \mathcal{X}$ and control input $\mathbf{u}_t \in \mathcal{U}$:

$$p(\mathbf{x}_{t+1}|\mathbf{u}_t, \mathbf{x}_t, \ldots, \mathbf{u}_0, \mathbf{x}_0) = p(\mathbf{x}_{t+1}|\mathbf{u}_t, \mathbf{x}_t)$$

We assume an unknown probability density over initial states $p(\mathbf{x}_0)$.

A demonstration (or trajectory)$\hat{\tau}$ is a series of $T+1$ pairs of states visited and corresponding control inputs at these states, $\hat{\tau} = (\mathbf{x}_0, \mathbf{u}_0, ..., \mathbf{x}_T, \mathbf{u}_T)$, where $\mathbf{x}_t \in \mathcal{X}$ and $\mathbf{u}_t \in \mathcal{U}$ for $t \in \{0, \ldots, T\}$ and some $T \in \mathbb{N}$. For a given trajectory $\hat{\tau}$ as above, we denote by τ the corresponding trajectory in state space, $\tau = (\mathbf{x}_0, ..., \mathbf{x}_T)$.

A policy is a function $\pi : \mathcal{X} \to \mathcal{U}$ from states to control inputs. We consider a space of policies $\pi_\theta : \mathcal{X} \to \mathcal{U}$ parameterized by some $\theta \in \mathbb{R}^d$. Any such policy π_θ in an environment with probabilistic initial state density and Markovian dynamics induces a density on trajectories. Let $p(\mathbf{x}_t|\theta)$ denote the value of the density of states visited at time t if the robot follows the policy π_θ from time 0 to time $t-1$. Following [17], we can compute the average density on states for any timepoint by $p(\mathbf{x}|\theta) = \frac{1}{T}\sum_{t=1}^{T} p(\mathbf{x}_t|\theta)$.

While we do not assume knowledge of the distributions corresponding to: $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, $p(\mathbf{x}_0)$, $p(\mathbf{x}_t|\theta)$ or $p(\mathbf{x}|\theta)$, we assume that we have a stochastic real robot or a simulator such that for any state $\mathbf{x}_t$ and control $\mathbf{u}_t$, we can sample the $\mathbf{x}_{t+1}$ from the density $p(\mathbf{x}_{t+1}|\pi_\theta(\mathbf{x}_t), \mathbf{x}_t)$. Therefore, when 'rolling out' trajectories under a policy π_θ, we utilize the robot or a simulator to sample the resulting stochastic trajectories rather than estimating $p(\mathbf{x}|\theta)$ itself.

Objective. The objective of policy learning is to find a policy that maximizes some known cumulative reward function $\sum_{t=1}^{T} r(\mathbf{x}_t, \mathbf{u}_t)$ of a trajectory $\hat{\tau}$. The reward $r : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is typically user defined and task specific. For example, in the task of inserting a peg into a hole, a function on distance between the peg's current and desired final state is used [14].

The reward function for a task like grasping in clutter, (i.e. successfully grasping the goal object) is delayed [13]. The idea behind DAgger and SHIV is to query a supervisor for appropiate actions, to provide the robot a set of N stochastic demonstrations trajectories $\{\hat{\tau}^1, ...\hat{\tau}^N\}$. This induces a training data set $\mathcal{D}$ of state-control input pairs.

We define a 'surrogate' loss function as in [17], $l : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$, which provides a distance measure between any pair of control values. In the continuous case, we consider $l(\mathbf{u}_0, \mathbf{u}_1) = ||\mathbf{u}_0 - \mathbf{u}_1||_2^2$, while in the discrete case $l(\mathbf{u}_0, \mathbf{u}_1) = 1$ if $\mathbf{u}_0 \neq \mathbf{u}_1$ and $l(\mathbf{u}_0, \mathbf{u}_1) = 0$ otherwise.

Given a candidate policy π_θ, we then use the surrogate loss function to approximately measure how 'close' the robot's policy's returned control input $\pi_\theta(\mathbf{x}) \in \mathcal{U}$ at a given state $\mathbf{x} \in \mathcal{X}$ is to the supervisor's policy's control output $\tilde{\pi}(\mathbf{x}) \in \mathcal{U}$. The goal is to produce a policy that minimizes the surrogate loss.

Following [17], our objective is to find a policy π_θ minimizing the expected surrogate loss with respect to the most skilled supervisor, where the expectation is taken over the distribution of states induced by the policy across any time point in the horizon:

$$\min_\theta E_{p(\mathbf{x}|\theta)}[l(\pi_\theta(\mathbf{x}), \tilde{\pi}(\mathbf{x}))] \tag{1}$$

If the robot could learn the policy perfectly, this state density would match the one encountered in user examples. But if the robot makes an error, that error changes the distribution of states that the robot will visit, which can lead to states that are far away from any examples and difficult to generalize to [15]. This motivates iterative algorithms like DAgger, which iterate between learning a policy and the supervisor providing examples.

We introduce the concept of their being a set of supervisors each with an associated cost and skill level. Denote $\tilde{\pi}_M$ the most skilled and costly supervisor in this set. This supervisor could for example be a person with a Phd in machine learning and robotics. We formally define how each supervisor in the set is related to each other in Sec. IV-B. Every time a supervisor is queried a cost, C_m, is incurred this could be an hourly pay rate or computational time.

While Eq. 1 is the primary objective of our algorithm a secondary objective is to reduce the cumulative cost associated

with training the robot. Thus, we are interested in solving the following objective

$$\min_{\theta, C_{m,j}} \sum_{j=1}^{J} C_{m,j} \tag{2}$$

$$\text{s.t. } E_{p(\mathbf{x}|\theta)}[l(\pi_\theta(\mathbf{x}), \tilde{\pi}_M(\mathbf{x}))] \leq \epsilon \tag{3}$$

where J is the number of queries need to satisfy the constraint on expected surrogate loss, $C_{m,j}$ denotes the cost incurred by querying the supervisor at that iteration and ϵ is a user-defined threshold on how close robot policy is to match the highest quality supervisor. We present our algorithm, LEATHERMAN, which aims to solve Eq. 2.

IV. Algorithm

In this section, we first recall the original DAgger algorithm. Then introduce the notion and formal definition of a hierarchical supervisor and finally describe how to combine DAgger with a hierarchical supervisor to create a Gambler's DAgger.

A. DAgger: Dataset Aggregation

DAgger [17] solves the minimization in Eq. 1 by iterating two steps: 1) compute a θ using the training data $\mathcal{D}$ thus far, and 2) execute the policy induced by the current θ, and ask for labels for the encountered states.

1) Step 1: The first step of any iteration k is to compute a θ_k that minimizes surrogate loss on the current dataset $\mathcal{D}_k = \{(x_i, u_i)|i \in \{1, \ldots, M\}\}$ of demonstrated state-control pairs (initially just the set $\mathcal{D}$ of initial trajectory demonstrations):

$$\theta_k = \arg\min_{\theta} \sum_{i=1}^{M} l(\pi_\theta(\mathbf{x}_i), \mathbf{u}_i). \tag{4}$$

This sub-problem is a supervised learning problem, solvable by estimators like a support vector machine or a neural net. Performance can vary though with the selection of a the estimator Selecting the correct function class depends on the task being consider and knowledge of the problem, see for a guide [19].

2) Step 2: The second step DAgger rolls out their policies, π_{θ_k}, to sample states that are likely under $p(\mathbf{x}|\theta_k)$. For every state visited, DAgger requests the supervisor to provide the appropriate control/label. Formally, for a given sampled trajectory $\hat{\tau} = (\mathbf{x}_0, \mathbf{u}_0, ..., \mathbf{x}_T, \mathbf{u}_T)$, the supervisor provides labels $\tilde{\mathbf{u}}_t$, where $\tilde{\mathbf{u}}_t \sim \tilde{\pi}(\mathbf{x}_t) + \epsilon$, where ϵ is a small zero mean noise term, for $t \in \{0, \ldots, T\}$. The states and labeled controls are then aggregated into the next data set of demonstrations $\mathcal{D}_{k+1}$:

$$D_{k+1} = \mathcal{D}_k \cup \{(\mathbf{x}_t, \tilde{\mathbf{u}}_t) \| t \in \{0, \ldots, T\}\}$$

Steps 1 and 2 are repeated for K iterations or until the robot has achieved sufficient performance on the task[1].

[1]In the original DAgger the policy rolled out was stochastically mixed with the supervisor, thus with probability β it would either take the supervisor's action or the robots. The use of this stochastically mix policy was for theoretical analysis. In practice, it is recommended to set $\beta = 0$ to avoid biasing the sampling [10], [17]

B. Hierarchy of Supervisors

Instead of one supervisor,we propose a hierarchy of M supervisors where for each supervisor $\tilde{\pi}_m$, there is an associated expected cumulative reward $R_m = \int \sum_{t=1}^{T} r(\mathbf{x}_t, \tilde{\pi}_m(x)) p(\tau|\tilde{\pi}_m) d\tau$, where $p(\tau|\tilde{\pi}_m)$ denotes the average distribution of states the supervisor encounters. We also denote some measure of cost C_m that is ascribed to the supervisor, such as computational time or monetary expense. Generally the rank of supervisors in terms of cost is equivalent to the rank of supervisor in the terms of expected cumulative reward. Thus, the least costly supervisor will receive the lowest expected cumulative reward.

Additionally for two supervisors $\tilde{\pi}_m$ and $\tilde{\pi}_{m+1}$ to be in the hierarchy, they must only disagree in expectation on a subset of the work space to some precision $\tilde{\epsilon}$. Denote the set of states two supervisors disagree as $\mathcal{X}_{m,m+1} = \{\mathbf{x}|||\tilde{\pi}_m(\mathbf{x}) - \tilde{\pi}_{m+1}(\mathbf{x})||_2^2 > \tilde{\epsilon}\}$ .We enforce the condition $\mathcal{X}_{m,m+1} \subset \mathcal{X}$ because if two supervisors never provided the same examples then all states would have to be relabeled by the next supervisor in the hierarchy.

We formally define an ordering on supervisors:

Definition Two supervisors $\tilde{\pi}_m$ and $\tilde{\pi}_{m+1}$ are in a hierarchy if $C_m < C_{m+1}$ and $\mathcal{X}_{m,m+1} \subset \mathcal{X}$

Violation of this definition could result in no additional benefit from the LEATHERMAN algorithm or potentially worst a higher budget incurred than without using a hierarchy of supervisors.

C. DAgger with Supervisor Hierarchy

ML: Anca and I wanted to justify the greedy allocation section, which we draw from the contextual arm bandit literature to do so... I can condense this The hierarchical supervisor problem can be framed as a contextual multi-armed bandit, which is like a traditional multi-armed bandit except the sampled reward depends on an additional state term [**?**]. At each iteration a supervisor is selected from M supervisors and the state is the current policy parameters or θ_k. Then a sampled reward is received which is how well the trained policy performed on the sample surrogate loss measured against the $\tilde{\pi}_M$ or the supervisor at the top of the hierarchy. Note we only measure against the supervisor at the top of the hierarchy because that is our primary objective (i.e. Eq. 1).

While a large amount of literature exists to solve contextual multi-armed bandit problems, a common assumption is that the sampled reward can be observed [**?**]. However, in our situation evaluation of the sampled reward requires querying the costliest supervisor every iteration. Thus, defeating the purpose of cost reduction. In light of this we propose a greedy allocation strategy where we train a policy with the cheapest supervisor first until convergence and then work up the hierarchy training with each supervisor.

Our algorithm is as follows: first iterate through Step 1 and 2 of DAgger for a given supervisor, however after iteration K or when the current policy, $\pi_{\theta_{m,k}}$ is able to achieve a loss ϵ on the sampled J states at that current iteration (i.e):

$$\frac{1}{J}\sum_{j=1}^{J}||\pi_{\theta_{m,k}}(\mathbf{x}_j) - \tilde{\pi}_m(\mathbf{x}_j)||_2^2 \leq \epsilon$$

Then iterate to the next supervisor in the hierarchy or $m = m+1$. The hyperparameter ϵ can be set as a measure of how different the controls applied between the robot's policy and the current supervisors a sequential process can tolerate. For example surgical sequential processes could warrant a very low threshold, but less precise sequential processes, like grasping in clutter, can tolerate a higher value.

An issue arises though when performing this iterations because now the current dataset $\mathcal{D}_K$ has examples from a different supervisor that receives a smaller expected cumulative reward. This could cause a learning algorithm to try and fit to contradictory examples and be worse than either supervisor trained with [19].

We are thus interested in not storing the dataset collected from the previous supervisor and only remembering the current $\theta_{m,K}$ weight parameters. Thus, we propose only performing DAgger for each individual supervisor in the hierarchy and using the resulting weight vector to bootstrap the learning of the next supervisor. We found empirically in Sec. VI that it also beneficial to add to the new dataset the states and controls applied by the current policy $\pi_{\theta_{m,k}}$ that the current supervisor agreed with by some euclidean distance $\tilde{\epsilon}$.Thus, acting as a regularization on the optimization since stochastic gradient on a section of the min-batch update would be zero for examples where the next supervisor agrees with the previous.

ML: we can formally prove the following statements or point to the web similar to SHIV... it will be alot easier to write these out though if we have space With respect to the theoretical analysis of DAgger's convergence rate, one can see that in our algorithm you are solving M individual DAgger's [17]. Thus, in the worst case you will converge to the supervisor at the top of the hierarchy $\tilde{\pi}_M$ in $\tilde{O}(MT)$ iterations instead of DAgger's original $\tilde{O}(T)$.

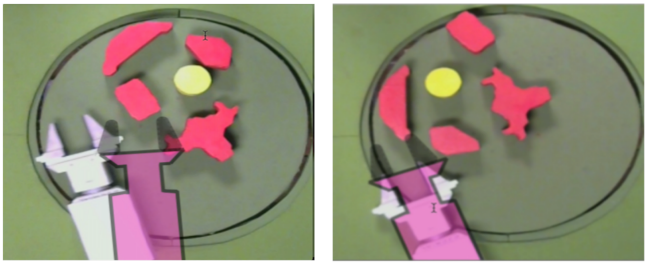
It should be noted though that as you iterate through the hierarchy of supervisors, each iteration starts out with an error bounded by $\int_{\mathcal{X}_{m,m+1}} ||\pi_{\theta_{0,m+1}}(\mathbf{x}) - \tilde{\pi}_{m+1}(\mathbf{x})||_2^2 p(\mathbf{x}|\theta_{0,m+1})d\mathbf{x} + \epsilon$. Thus, if the supervisors are able to agree on a large part of the state space, there should be less iterations with the expert supervisor than $\tilde{O}(T)$, which we experimentally show in our grasping in clutter task.

V. Robot Grasping in Clutter

We now describe the grasping in clutter problem, the supervisor hierarchy that we used to solve it and our deep learning policy architecture.

A. Grasping in Clutter

Consider a robot in an Amazon warehouse. The robot has an intended goal object for it to grasp on a shelf, however other objects could obstruct the path towards the goal object. These obstructions of movable objects prevent leveraging common motion planning techniques, because they do not posses a dynamics model of how the objects behave under different contact forces. [13], [**?**].



(a) Rotation (b) Extension

Fig. 2: The interface AMT workers see for providing feedback to the robot for the grasping in clutter task. The pink overlay indicates the desire change in the robot position with respect to the current robot state. Then AMT workers can use their intuition for how objects respond to force to provide examples of how the robot should behave. Each window shows a correction in a degree of freedom a) is the rotation of the robots base b) is extending the robot's arm.

Thus, the robot must reason about how its interactions with the environment will affect the intended outcome. This can be hard for two reasons 1) it becomes hard to find a low dimensional state representation for the task because the environment dynamics are affected by the global shape of each object 2) the dynamics of the environment involve modeling how K objects interact with each other under an arbitrary force [13].

In order to achieve real time grasping in clutter, which is important for a robot in a shipping warehouse, we leverage visuo-motor deep learning to learn a control policy for the task over a large number of different configurations working with a hierarchy of supervisors. Deep learning has the potential to take high dimensional image data of the scene and learn task specific features relevant for grasping in clutter. Additionally a trained neural network policy is can be computational inexpensive to evaluate and can result in real time performance [14].

B. Supervisor Hierarchy

In this section, we list each supervisor in the order they appear in our hierarchy. The first supervisor is deemed the least expensive.

Motion Planning Supervisor Our first supervisor is a an analytical method that computes the trajectory the robot should move in to reach the goal object, or yellow circle, with a relax dynamic model of the environment. Our method employs template matching to identify the goal object in the image and the using the forward dynamics of the robot to computes the relative change in direction the robot should apply as an control. The template matching is implemented in OpenCV and uses the normalized cross-correlation filter [**?**].

In this stage, the motion planning supervisors only tries to teach the robot to move towards the goal. Note that the motion planning supervisor's advice does not have any knowledge about how the cluttered objects will respond to the forces applied via the robot arm, which results in sub-optimal polices. However, our motion planning supervisor is both computationally and monetarily inexpensive to run which allows us to provide a large number of examples.

CrowdSourced Supervisor Our next level of supervisor relies on a crowd source service, called Amazon Mechanical Turk (AMT). Chung et al. demonstrated the reliable data

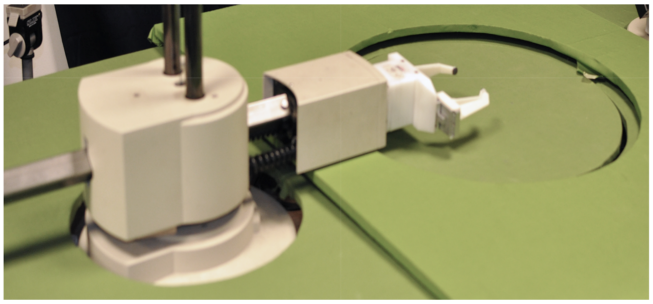


Fig. 3: Shown above is a Zymark robot. The robot consists of an 3DOF arm that lies in a planar workspace and the ability to rotate the turn table. The inscribed circle in the work space prevents the robot from learning the trivial policy of just pushing the objects off the table.

learning from demonstration data could be obtained from an AMT platform. The AMT platform makes readily available thousands of human workers that can perform remedial task for a price range of $0.1 per robot trajectory[**?**]. Thus potentially providing a higher quality supervisor who has an intuition for how the cluttered objects interact with the world, but at a slightly higher cost.

In order to get examples from a CrowdSourced Supervisor, we designed an interface shown in Fig. 2. The interface draws a transparent overlay that shows how the robot would respond provided the current control this allows for the AMT worker to see some of the effect of their control. We rely on their natural intuition for how objects to infer how the change in robot position will change the objects.

We also design a tutorial for the work that first has them perform designated motions on a virtual robot and introduces them to a problem. We additionally provide a video of an expert providing corrections on three robot trajectories.

Expert Supervisor Finally after training with the Motion Planning and Crowdsourced Supervisor we leverage the Expert supervisor, who is capable of achieving high cumulative reward but is a limited resource. An expert supervisor in this case is a Phd student in machine learning and robotics, which can cost on average ten times more than an AMT worker to employ [].

An expert supervisor can be used in a variety of scenarios.They first would have a better intuition of the physical limitations of the robot and environment, such as joint limits or how certain objects might behave under force. Furthermore, they would also understand how the examples given could lead to better training of the parameterized policy. For example, understanding the feature space the policy lies in can lead to the expert not providing contradictory examples.

It can still be hard for an expert to provide the correct feedback without knowing the magnitude of the controls suggested, so they use the same interface as the CrowdSourced Supervisor.

C. Neural Network Policy Architecture

Our policy is represented as deep neural network, which was trained using TensorFlow [1]. Our network architecture consists of 1 convolutional layer with 5 channels and filters with size 11x11, a fully connected layer with an output of 128 dimensions and a final layer that maps to a four dimensional control signal. We used ReLus to separate the different layers and a final tanh on the output to scale the output between -1 and 1.

The control examples was scaled between 1 and -1 for each dimension independently. To be robust to lighting and reduce the dimensionality of the problem, we applied a binary mask to each channel of the 250x250 RGB image, the mask would set to 1 values above 125 and 0 other wise. We then validated that all information (i.e. location of the gripper, cluttered shapes and goal object) where still visible in the masked image.

To determine our architecture we preformed a grid search ofver different architectures trained after 400 iterations with a batch size of 200. The set of architectures consist of different network architectures as well as different momentum terms, and weight initialization schemes. We trained all networks on a dataset of 6K images labeled with the Analytical Supervisor on a Nvidia Tesla K40 GPU, which is able to train each network in an average of 10 minutes.

VI. Experiments

For an experiment in the grasping in clutter domain, we are interested in training a Zymark robot to perform a grasping in clutter task on image data taken from a Logitech C270 camera. Examples of images from the camera can be seen in Fig. 1.

The objects in clutter are made Medium Density Fiberboard and each one is on average 4" in diameter. The objects deemed cluttered are painted Red and the goal object is painted yellow. There is an inscribed circle around the work space to keep the robot from pushing the objects outside of the work space. The inscribed circle can make the task more challenging because the robot cannot simply "sweep" the cluttered objects off the table.

The robot, shown in Fig. 3, has a 3 dimensional internal state of base rotation, arm extension and gripper. The robot is commanded via state position requests, which are tracked by a tuned PID controller. The policy π_θ outputs delta positions that are bounded by $15\,\mathrm{deg}$ for the gripper and turntable, 1 cm for the arm extension and 0.5 cm for the gripper at each time step. There is $T = 100$ time steps in a trajectory.

To test the performance of a policy, we created a test set composed of 20 different configurations each containing objects that were not trained on. The test set configurations varied in number of objects on the table, size of objects and relative pose of each object. We measure success as defined by three situations not successful, near successful, successful. We report total success as a score out of 2: 0 for not successful, 1 for near successful, 2 for successful.

The not successful situation corresponds to the case when the robot's gripper is not touching the goal object at the end of a rollout. Examples of when this happens is when the robot fails to push the objects away or does not head in the direction of the yellow circle. The near successful situation is defined as the robot touches the goal object with its gripper, but it is not enclosed in the gripper. Examples of when this happens is when the robot gets a small obstacle object trapped in the gripper first, or when the root slightly misses the yellow circle on its approach. Successful is defined as getting the goal object inside the gripper at the end of a trajectory.

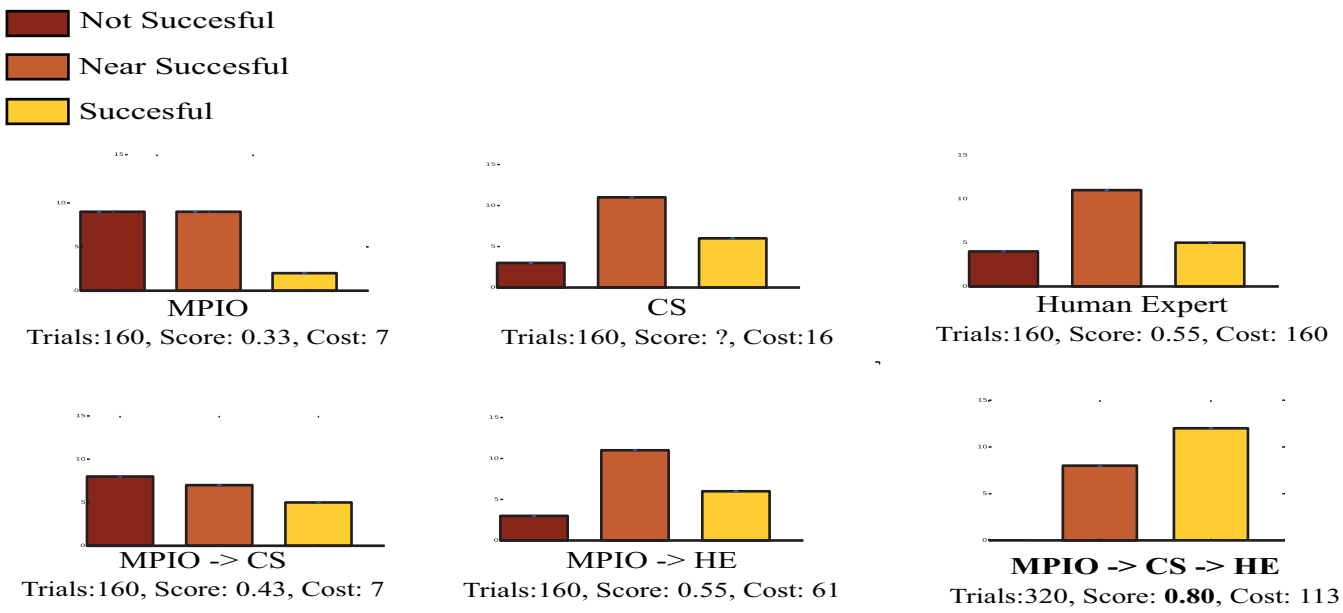


Fig. 4: Examples of different states a supervisor is ask to provide labels for. a) Represents states that are easy for the supervisor to provide labels for. b) Represents states that are difficult for a human to provide examples and can lead to contradictory examples in similar states

A. Hierarchical Supervisors

We first test that using a hierarchy of supervisors can reduce total cost needed to train a policy, but still maintain similar performance . We compare having an analytical only supervisor for a fixed amount of data of 160 demonstrations. The non-hierarchical policies trained with the expert supervisor only for 160 demonstrations or only the MPIO supervisor for 160 demonstrations. The policy trained with a hierarchical supervisor first receives 100 demonstrations from the MPIO supervisor and 60 demonstrations from the expert supervisor.

Our results, shown in Fig. 4, are the policy trained with just the MPIO supervisor achieves a score of 0.325%, the policy trained with just an expert achieves 0.54% and the policy trained with a hierarchy of supervisor achieves 0.55%. Thus, the hierarchical supervisor and expert supervisor achieve approximately the same performance. However,training with a hierarchical supervisor yields a 40% reduction in cost incurred compared to the policy trained with only an expert supervisor.

Thus demonstrating that by using a hierarchy of supervisors, we can reduce the cost of training a policy and achieve similar performance to the

B. Quality Crowdsourced Supervisor

We next evaluate the potential of a crowdsourced supervisor for a being part of the hierarchy. Thus, we perform an experiment using the AMT platform described in Sec. V-B. We test how well a crowdosurce supervisors performs as part of a hierarchy of supervisors, we had a policy train with 100 demonstrations from the MPIO supervisor , then had 60 demonstrations provided by AMT workers. We also compare how well the crowdsource supervisors performs just as well as a single supervisor, by having them provide all 160 demonstrations.

Our results, shown in Fig. 4, are the policy trained with just the crowdsourced supervisor achieves a score of $?\%$. The policy trained with the hierarchical crowdsourced and MPIO supervisor receives 0.43% on our test set compared to 0.55% with the motion planning and expert supervisors hierarchy. The total cost received with the only crowdsource supervisor was 15, while the hierarchical supervisor was 7.

ML: talk about crowdosuce statistics

C. Advancing in the Hierarchy

We further test how to advance between supervisors in the hierarchy.We examine three different strategies for dataset management when changing supervisors 1) aggregating the dataset of the data collected with both supervisors 2) transferring only the weights of the nerural network policy or θ vector between the supervisors and removing the first's supervisors dataset completely 3) transferring only the weights of the neural network policy but adding the values output by $\pi_\theta(\mathbf{x})$ instead of $\tilde{\pi}_m(\mathbf{x})$at parts of the states visited that the current supervisor is close to agreement with as measured by the following $||\tilde{\pi}_m(\mathbf{x}) - \pi_\theta(\mathbf{x})||_2^2 < 0.01$ **ML: need to get this number off the local code, will update tomorrow**. Thus, acting as a regularization on the optimization since stochastic gradient on a section of the min-batch update would be zero for examples where the next supervisor agrees with the previous.

We compare each strategy on a hierarchy of a MPIO supervisor trained with 100 iterations and then advancing to a human expert supervisor trained with 60 iterations. Our results reported in Fig.5 , show that aggregation strategy achieves 0.18%, the weight transfer strategy achieves 0.55% and the regularized stochastic gradient update strategy achieves 0.63%. Thus, suggesting that techniques to help the policy remain close to the previously trained supervisor are useful for effectively switching between supervisors.

D. Scaling the Hierarchy

The final experiment we ran is to run our algorithm on a hierarchy with 3 supervisors: MPIO, crowdsourced and human expert. We ran each policy until we observed the pay out in terms of reward achieved was not increasing. This results in MPIO for 100 demonstrations, crowdsourced for

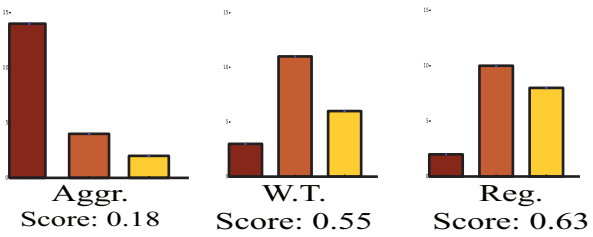


Fig. 5: Examples of different states a supervisor is ask to provide labels for. a) Represents states that are easy for the supervisor to provide labels for. b) Represents states that are difficult for a human to provide examples and can lead to contradictory examples in similar states

120 demonstrations and human expert for 100 demonstrations. Thus resulting in 320 trials on the robot and 320,000 annotated images of what control the robot should apply.

The results of our trained policy, as shown in Fig. 4, demonstrate that by leveraging a hierarchy we are able to achieve a score of 80% with a cost of only 113. We note that policy trained with the human expert supervisor, which incurred a cost of 160, was only able to achieve a score of 0.66%.

While our policy was able to do substantially better, to near successful mode a reasonable portion of the time. These failure modes where do to the robot not being able to accurately manipulate smaller objects, where slight perturbations could result them being pushed into the gripper, and accurately controlling the goal object to land in the circle (i.e. sometimes pushing against the side of the circle instead). We attribute this problem to a limit in the detail of precision our supervisors can currently give, future work will look at more precise analytical methods and fine tuning the policies via self-learning.

VII. Discussions and Future Work

VIII. Acknowledgments

This research was performed in UC Berkeley's Automation Sciences Lab under the UC Berkeley Center for Information Technology in the Interest of Society (CITRIS) "People and Robots" Initiative. This work is supported in part by the U.S. National Science Foundation under Award IIS-1227536, NSF-Graduate Research Fellowship, by the Knut and Alice Wallenberg Foundation and the National Defense Science and Engineering Graduate Fellowship Program. We thank Pieter Abbeel, Sergey Levine and Sanjay Krishnan for insightful feedback.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," *NIPS*, vol. 19, p. 1, 2007.

[3] P. Abbeel, D. Dolgov, A. Y. Ng, and S. Thrun, "Apprenticeship learning for motion planning with application to parking lot navigation," in *IROS 2008. IEEE/RS*. IEEE.

[4] P. Abbeel and A. Y. Ng, "Apprenticeship learning via reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[5] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[6] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 41–48.

[7] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *Journal of Artificial Intelligence Research*, vol. 34, no. 1, p. 1, 2009.

[8] F. Duvallet, T. Kollar, and A. Stentz, "Imitation learning for natural language direction following through unknown environments," in *ICRA*. IEEE, 2013, pp. 1047–1053.

[9] D. H. Grollman and O. C. Jenkins, "Dogged learning for robots," in *ICRA, 2007 IEEE*.

[10] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," in *NIPS*, 2014, pp. 3338–3346.

[11] K. Judah, A. Fern, and T. Dietterich, "Active imitation learning via state queries," in *Proceedings of the ICML Workshop on Combining Learning Strategies to Reduce Label Cost*, 2011.

[12] B. Kim and J. Pineau, "Maximum mean discrepancy imitation learning." in *Robotics Science and Systems*, 2013.

[13] N. Kitaev, I. Mordatch, S. Patil, and P. Abbeel, "Physics-based trajectory optimization for grasping in cluttered environments," pp. 3102–3109, 2015.

[14] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *arXiv preprint arXiv:1504.00702*, 2015.

[15] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," Carnegie-Mellon University, Tech. Rep., 1989.

[16] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 661–668.

[17] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," *arXiv preprint arXiv:1011.0686*, 2010.

[18] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *ICRA, 2013 IEEE*. IEEE.

[19] B. Schölkopf and A. J. Smola, *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[20] J. Van Den Berg, S. Miller, D. Duckworth, H. Hu, A. Wan, X.-Y. Fu, K. Goldberg, and P. Abbeel, "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations," in *ICRA, 2010 IEEE*. IEEE, 2010, pp. 2074–2081.