Improving Reinforcement Learning Pre-Training with Variational Dropout

Tom Blau, Lionel Ott, and Fabio Ramos

School of Information Technologies, The University of Sydney

Abstract— **Reinforcement learning has been very successful at learning control policies for robotic agents in order to perform various tasks, such as driving around a track, navigating a maze, and bipedal locomotion. One significant drawback of reinforcement learning methods is that they require a large number of data points in order to learn good policies, a trait known as poor data efficiency or poor sample efficiency. One approach for improving sample efficiency is supervised pre-training of policies to directly clone the behavior of an expert, but this suffers from poor generalization far from the training data. We propose to improve this by using Gaussian dropout networks with a regularization term based on variational inference in the pre-training step. We show that this initializes policy parameters to significantly better values than standard supervised learning or random initialization, thus greatly reducing sample complexity compared with state-of-the-art methods, and enabling an RL algorithm to learn optimal policies for high-dimensional continuous control problems in a practical time frame.**

I. Introduction

Automation of many real-world tasks requires being able to solve sequential problems. That is, problems where there is some state that evolves over time in response to agent decisions, and thus involve a sequence of decisions and states, rather than a single decision such as in the case of classification. Reinforcement learning has been very successful at learning control policies for agents on sequential tasks. Deep Q-Networks have shown human-level and even superhuman performance in Atari games [17], which are partially observable with discrete action spaces. Methods for continuous action spaces are able to learn gaits for various fully observable walkers [7]. One significant drawback is that such methods require a large number of data points in order to learn good policies. Since such data often has to be generated by executing policy rollouts on a physical system (or a physics simulation), getting training data can be quite expensive in terms of both computation and time. In the work by Mnih et al. [18], several days of computational time with 16 CPU cores were required to complete a single experiment in Atari [3] or TORCS [26] environments. State-of-the-art performance in robotic control is often achieved by systems with many processors and robotic agents collecting data in parallel. In work by Levine et al. [16], an agent learned to grasp objects with a robotic manipulator based on camera images, with a learning setup using as many as 14 manipulators working in parallel.

This problem of sample inefficiency is particularly noticeable in tasks that are partially observable, have continuous action spaces, have a high dimensionality or large state space, or all of the above. This is often the case with sensorimotor control of robotic agents, like grasping or pushing tasks. The ability to learn control policies for performing such tasks is highly desirable, as much of global human labor is locked up in repetitive, low-skill tasks that stand to be automated by robotic systems with sensorimotor capabilities.

A naive technique for reducing the sample complexity of a reinforcement learning problem is to initialize the model's parameters to good values by using supervised learning on demonstrations from an expert [4][8]. This has the problem that small errors accumulate over a sequence of actions, so that the state of the system gradually drifts away from the labeled data seen during training. In this work we address this problem of compounding errors by using the recently developed regularization technique of sparse variational dropout (VD)[19], which results in networks that generalize better beyond the training data.

The main contribution of this paper is demonstrating that supervised pre-training can be done effectively by using sparse variational dropout regularization, and that it greatly accelerates standard RL, compared with standard regularizers like L_2. Additionally, pre-training with sparse VD is less dependent on having a large number of expert demonstrations, or on having a training dataset that explores a large region of the state space, in order to achieve good results. This saves effort in generating expert demonstrations and tuning the exploration noise used for generating expert demonstrations, which is often either difficult (in the case of human demonstrations) or done by inefficient grid search. Finally, the pre-training procedure automatically prunes unneeded neurons from the neural network, resulting in sparser and more computationally efficient models.

The structure of the paper is as follows: in Section II we discuss previous work on pre-training RL models and the use of variational dropout. Section III then explains the theoretical background of the algorithm used in our proposed method, which is described in Section IV. The approach is assessed in several experiments in Section V. Finally, Section VI discusses conclusions arising from the results, as well as directions for future work.

II. Related Work

The high sample complexity of RL algorithms is often mitigated by adding a pre-training phase that initializes the model parameters to good values. One of the simplest ways of doing this is by executing a supervised learning algorithm on a set of expert demonstrations, a technique known as

Behavioral Cloning (BC). A behavioral cloning approach is often used independently of reinforcement learning, and indeed predates the current wave of RL algorithms [20]. However, BC is known to perform poorly when the expert demonstrations are not sufficiently similar to states encountered by the learned model, a condition known as *covariate shift* [13]. In complex environments with large state spaces, this requires various techniques to improve the quality of the training dataset.

Bojarski et al. [5] learned how to drive a car based on images from a front facing camera. Artificial shift and rotation was added to the input images to improve generalization. Duan et al. [8] used BC in the context of a meta-learning procedure, training a model that can learn to perform a new block-stacking task based on a single demonstration. A small amount of noise was injected into the expert demonstrations in order to generate training data that is more diverse. The authors report that this noise injection is critical to the performance of the learned model, but do not explain how to choose this noise. The DART [15] algorithm presents a more principled approach to shaping the noise injected into the expert demonstrations. An optimization procedure is used to find noise that simulates the errors a learned policy would make, thus reducing the covariate shift between the states seen at training time and the ones seen at test time. Our proposed approach also uses noise injection to improve the quality of the training data. However, the addition of variational dropout regularization reduces the sensitivity of training to choosing a correct magnitude of noise. This is particularly when injecting exact noise is impractical, such as with a human demonstrator, or in physical systems which might sustain physical damage if the noise is too large.

The DAgger algorithm [21] seeks to reduce the covariate shift between training and test data by iteratively alternating between exploring the state space with the learned policy and training the policy to mimic expert decisions on the encountered states. This procedure is computationally quite expensive, particularly if acquiring an expert demonstration is time-consuming. The approach proposed in this paper only requires generating demonstrations once, and these can be reused to train any number of policies.

Rusu et al. [22] used a BC approach to clone the Q-function of an expert model trained with deep Q-learning [17]. The student was trained to minimize the KL divergence between its output and that of the expert, in some cases resulting in a model that outperformed the expert. Model compression was also achieved by choosing smaller architectures for the student model, and even significantly compressed models were able to outperform the teacher. Our proposed method also achieves model compression, but it does so by pruning out neurons that are found to be unneeded, rather than by having to specify a smaller architecture.

In addition to the use of noise injection, we propose to improve the results of BC by the new regularization technique of variational dropout [19]. Dropout is a well known regularization technique that improves the generalization of learned models by preventing co-adaptation of parameters [25]. During training, in each forward pass, each weight is zeroed out with some probability p, which is usually 0.5. More recent works proposed frameworks for learning appropriate dropout probabilities from the data [2, 1, 24, 19], but for the most part are concerned only with classification and regression problems, and not with the reinforcement learning regime. Gal and Ghahramani [10][11] examined dropout as an approximation of Bayesian inference, and developed dropout methods that can be used to select exploration noise in reinforcement learning problems. Our work seeks to apply the regularizing power of adaptive dropout to reinforcement learning in a different way, by combining such techniques with a behavioral cloning approach.

III. Background

A. Behavioral Cloning

While supervised learning is concerned with single decision tasks, such as classification or regression, reinforcement learning is concerned with sequential decision tasks. In such tasks we have a system with some state, which changes with each decision, and to solve the task we must make a sequence of decisions that gradually bring the system to some goal state.

Behavioral cloning is an approach for learning control policies that solve sequential decision tasks using supervised learning techniques. Given a sequential decision task and some expert solver for that task, a training dataset can be generated by executing the expert and collecting pairs of state observations and expert decisions. An imitator policy is then trained on this dataset, using a supervised learning algorithm to minimize the discrepancy between the output of the imitator and the output of the expert. The weakness of this approach arises from the differences between the kind of sequential decision tasks dealt with in reinforcement learning and the single decision tasks which are the focus of supervised learning. In the context of single decision tasks, each prediction error is self-contained. However, if we execute a policy trained to imitate some expert, each error causes the state of the system to drift further away from parts of the state space represented in the training dataset, until the policy is required to make decisions for states entirely unlike those seen during training. This is often referred to as the *compounding error problem.*

Common approaches for mitigating the compounding error problem focus on providing better training data. Trivially, if the entire state space is seen in training, then there is no risk of drifting into previously unexplored regions. However, this requires an impractical amount of training data for all but the simplest tasks. More sophisticated approaches seek to explore a region of the state space that overlaps as closely as possible with the region that the trained policy will encounter. However, the compounding error problem is fundamentally one of generalization. If the learned policy generalizes poorly to unseen data, then compounding errors accumulate quickly, and the system soon reaches a state from which the learned policy is unable to recover. On the other hand, if the learned policy generalizes well, then it can correct for small errors in previous time steps and avoid drifting far from the training data, or possibly even reach the goal from unfamiliar states. For the most part, state-of-the-art BC algorithms do not address the issue of generalization, and serve only to reduce the probability of seeing novel states at test time, making them brittle. This is where regularization techniques such as sparse variational dropout come in.

B. Sparse Variational Dropout

This section briefly summarizes the principles of the sparse VD technique presented by Molchanov et al. [19]. Let $p(w)$ be some prior distribution over the weights of a neural network, and let $p(w|D)$ be the posterior after observing training dataset D of size N, given by Bayes rule. When $|w|$ is large, an analytical solution is intractable, so we approximate $p(w|D)$ using a parameterized distribution $q_\phi(w)$. We use variational inference to find parameters ϕ that minimize the Kullback-Leibler divergence $D_{KL}(q_\phi(w)||p(w|D))$ which is equivalent to maximizing the variational lower bound $\mathcal{L}(\phi)$, defined as:

$$\mathcal{L}(\phi) = L_D(\phi) - D_{KL}(q_\phi(w)||p(w)) \tag{1}$$

$$L_D(\phi) = \sum_{n=1}^{N} \mathbf{E}_{q_\phi(w)} \left[log\, p(y_n|x_n, w)\right] \tag{2}$$

Note that the KL divergence in this definition depends on $p(w)$ and not on $p(w|D)$. Using the reparameterization trick [14] we can get an unbiased Monte Carlo estimator of $L_D(\phi)$:

$$L_D(\phi) \simeq \frac{N}{M} \sum_{m=1}^{M} log\, p(\widetilde{y}_m|\widetilde{x}_m, f(\phi, \epsilon_m)) \tag{3}$$

where $(\widetilde{x}_m, \widetilde{y}_m)_{m=1}^{M}$ is a Monte Carlo sample of size M, and $f(\phi, \epsilon_m) = w$ is a deterministic function depending on a fixed non-parametric noise ϵ_m. The KL term of the lower bound can similarly be approximated from the same Monte Carlo sample. To apply this framework to neural network training, we need to define a probability distribution over network weights, as well as the function $f(\phi, \epsilon_m)$.

Binary dropout [12] is a common regularization mechanism for improving the generalizability of a neural network. In a binary dropout network, forward passes are stochastic, as the output of each neuron is multiplied by a realization sampled from a Bernoulli distribution $Bernoulli(1-p)$. This is equivalent to multiplying the neuron by zero with probability p, in which case it has been "dropped out" in that forward pass. Gaussian dropout extends the idea of multiplication by a random variable to use noise sampled from a Gaussian distribution $\mathcal{N}(1, \alpha)$. Applied to a network with parameters θ, this is equivalent to putting a distribution on the weights $w_{ij} \sim \mathcal{N}(\theta_{ij}, \alpha\theta_{ij}^2)$. This can be rewritten as:

$$w_{ij} = \theta_{ij}(1 + \sqrt{\alpha_{ij}}\epsilon_{ij}) \quad \epsilon_{ij} \sim \mathcal{N}(0, 1) \tag{4}$$

which is a suitable reparameterization to use as the function $f(\phi, \epsilon_m)$ in eq. (3), where $\phi = \{\theta, \alpha\}$

We now have a differentiable estimator of the lower bound, and can proceed to optimize eq. (1) using stochastic gradient descent. In addition to a regularizing effect, this procedure also has a sparsifying effect on the model. Srivastava et al. [25] have shown that multiplicative Gaussian noise with $\alpha = \frac{p}{1-p}$ corresponds to binary dropout with probability p. As any given α_{ij} becomes arbitrarily large, it therefore becomes equivalent to dropping out the corresponding weight with probability $p = 1$. This can be made explicit by zeroing out weights with α values that exceed a certain threshold.

IV. Method

Our proposed method represents a control policy using a neural network with multiplicative Gaussian noise. The network parameters, both W and α, are first tuned in a pre-training phase using a variational dropout procedure as described in section III. This is then followed by a standard RL procedure to produce the final policy. Figure 1 shows a high-level overview.

We treat a control problem as a Markov Decision Process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, \pi, \mathcal{P}, r, \gamma, T)$ where: $\mathcal{S}$ is the set of all possible states. $\mathcal{A}$ is the set of all possible actions. $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is some stochastic policy mapping state-action pairs (s,a) to the probability of choosing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. we use π_θ to denote a policy with parameters θ. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a state transition function mapping tuples (s_t, a_t, s_{t+1}) to the probability of arriving at state s_{t+1} after taking action a_t at state s_t. $r : \mathcal{S} \rightarrow \mathcal{R}$ is a reward signal assigning a scalar reward value to each state. $\gamma \in (0, 1)$ is a discount factor. $T \in \mathcal{N}$ is the time horizon. If a terminating state has not been reached within T state transitions, the current trajectory terminates regardless.

We define a parameterized stochastic policy of the form $\pi_\theta(s) = \mathcal{N}(\mu_\phi(s), \sigma_\psi(s))$, where s is a state observation, $\mu_\phi(s)$ and $\sigma_\psi(s)$ are parameterized functions with parameter sets ϕ and ψ respectively, and $\theta = \{\phi, \psi\}$. Executing the policy at state s is equivalent to sampling a realization from the Gaussian distribution. $\mu_\phi(s)$ and $\sigma_\psi(s)$ are represented by deep convolutional neural networks with multiplicative Gaussian noise on the weights. The networks have an architecture designed for sensorimotor control of robotic arms, shown in fig. 2. The architecture has two input sources- RGBD images and robot joint angles. Image inputs, as seen in the top half of fig. 2 are passed through 3 convolutional layers which learn to extract features from raw pixel data, and then through 3 fully-connected layers which learn system dynamics as a function of image features. Joint angle inputs, shown in the bottom of the figure, are passed through 3 fully-connected layers that learn system dynamics as a function of the robot configuration. The two streams are then concatenated, and passed through a final fully-connected layer which learns a mapping from the combined high-level features to policy moments. The output of a network is a vector of means or SDs of the stochastic policy, with one value for each dimension of the action space.

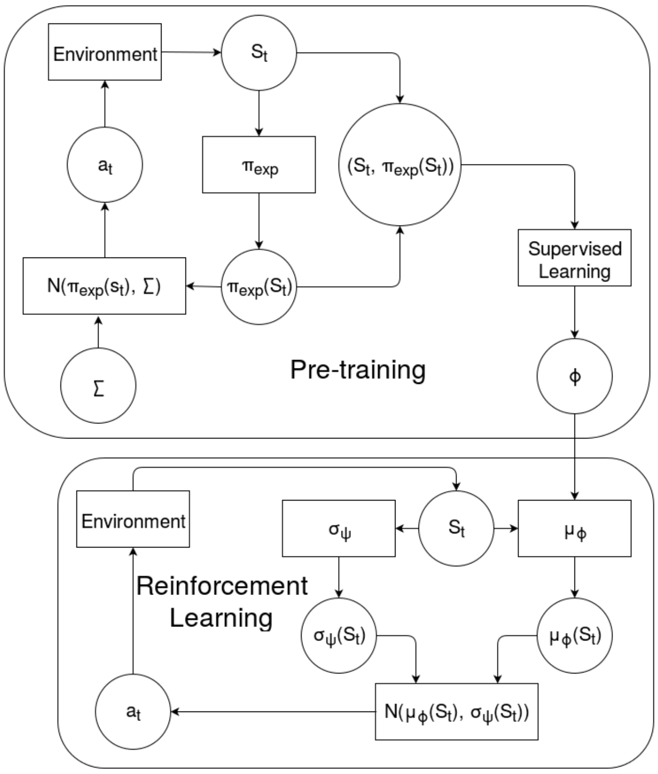

Fig. 1. Reinforcement Learning with Supervised Pre-training

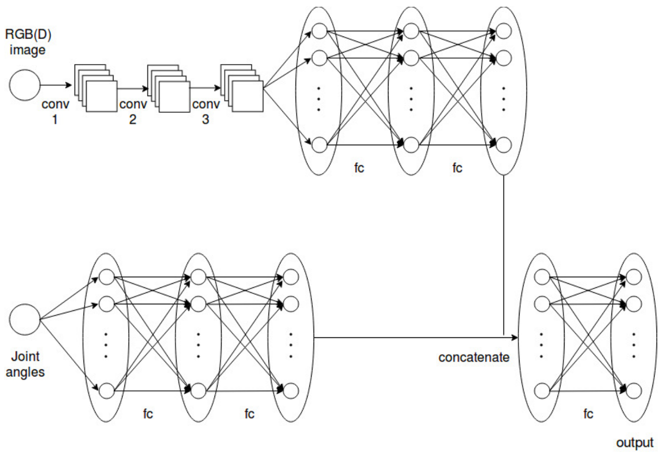

Fig. 2. Robot control DCNN architecture

Let $\pi_{exp}(s)$ be some expert that returns a good action for any state s. We execute $\pi_{exp}(s)$ to sample K expert trajectories where each trajectory is a sequence of states $T_i = [s_{i,0}, s_{i,1}, \ldots, s_{i,l_i-1}]$. As seen in the top half of fig. 1, in each trajectory, the state of the system is evolved according to the following equations:

$$s_{i,0} \sim p_0 \tag{5}$$

$$a_{i,t} \sim \mathcal{N}(\pi_{exp}(s_{i,t}), \Sigma) \tag{6}$$

$$s_{i,t} = \mathcal{P}(s_{i,t-1}, a_{i,t-1}) \tag{7}$$

Where p_0 is the distribution of initial states and the transition dynamics $\mathcal{P}(\cdot,\cdot)$ are assumed to be have very small variance. Note that in eq. (6) we add constant data generation noise Σ to the expert decision, which is distinct from the policy exploration noise σ_ψ. This is known to improve supervised learning from demonstration [8]. We define the training dataset D as:

$$D = \{(s_{i,t}, \pi_{exp}(s_{i,t})) : i \in [1, K], t \in [0, l_i - 1]\} \tag{8}$$

Note that in each element of D we pair a state with the noiseless decision of the expert, rather than with the noisy action defined by eq. (6).

Let w_{ij} be the j-th weight in the i-th layer of a network. For each $w_{ij} \in W$ there is a corresponding trainable parameter $\alpha_{ij} \in \alpha$ defining the variance of the Gaussian distribution over that weight, in accordance with eq. (4). In the pre-training phase, the output of a neuron in a forward pass is computed by sampling weights from the respective distribution, and both W and α are trained by stochastic gradient descent on the dataset D to maximize the lower bound defined in eq. (1). The log-likelihood in eq. (3) is computed based on the outputs of both $\mu_\phi(s)$ and $\sigma_\psi(s)$. However, only the parameters of $\mu_\phi(s)$ are trained in this stage, while those of $\sigma_\psi(s)$ remain fixed. In the RL phase, shown in the bottom half of fig. 1, we drop the weights w_{ij} such that $\alpha_{ij} > e^3$, which corresponds to binary dropout with probability $p > 0.95$. The output of a neuron in a forward pass is computed in the same way as a standard feed-forward network. We train W but not α of both $\mu_\phi(s)$ and $\sigma_\psi(s)$, using Trust Region Policy Optimization (TRPO) [23] in the same environment from which we derived the expert demonstrations.

V. Experimental Results

We conduct experiments on two different sensorimotor tasks- a reaching task and a grasping task- in order to assess the effectiveness of our algorithm relative to BC without dropout as well as standard reinforcement learning. For supervised learning we used ADAM with either L_2 or sparse VD regularization as appropriate. All networks were implemented using the Lasagne library [6]. For reinforcement learning we used TRPO. The implementation is based on the rllab library [7]. Experiments were carried out in the V-REP robot simulation environment [9].

A. Reaching Experiments

In the first set of experiments the task is to operate a 6-DOF Kinova Jaco robotic arm so as to position the gripper over a cup standing on a table. The cup itself is a non-interactable object, so that there is no concern of the arm colliding with it. Thus, this a reaching rather than a grasping task. The time horizon is $H = 25$, and each iteration of TRPO consists of 200 time steps. Observations include a 128x128 pixel RGBD image taken from a fixed position, as well as the angle values of each joint. Agent actions are vectors of angle deltas, prescribing a change to the angle of each arm joint, such that each individual delta is in the

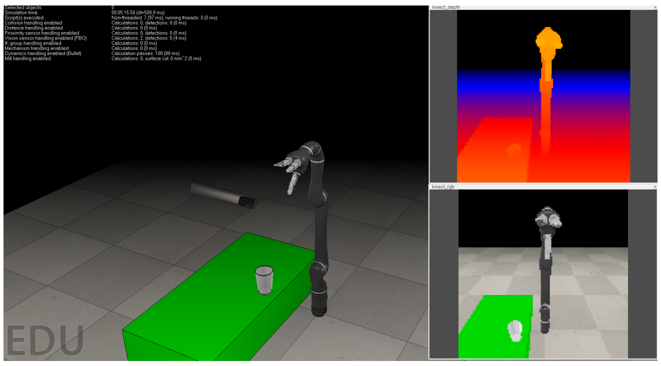

Fig. 3. A V-REP simulation of a robotic grasping task. Note on the right side a visualization of the policy inputs. The top window is a heat-map of depth-values, assigning warmer colors to pixels nearer to the sensor. The bottom window is an RGB image.

range $\left[-\frac{\pi}{5}, \frac{\pi}{5}\right]$. This range is a compromise between having optimal policies that induce very short trajectories (if the range contains $[-\pi, \pi]$ the entire state space is reachable in a single action), and having optimal policies that only induce very long trajectories (if the range is a subset of $\left[-\frac{\pi}{H}, \frac{\pi}{H}\right]$ then many cup positions can't be reached within the time horizon even by optimal policies). The arm is initialized to the default pose of a Jaco arm, while the cup is initialized to different locations on the table. Training samples are generated in the following manner:

We start by generating a set of uniformly distributed cup positions. For each cup position, inverse kinematics is used to compute a set G containing several distinct valid joint configurations, each of which results in the end effector being placed over the cup. Let $j(s)$ be the vector of arm joints angles at an arbitrary state $s \in \mathcal{S}$. The expert policy $\pi_{exp}(s)$ is defined by the following equations:

$$g^*(s) = \arg\min_{g \in G} |g - j(s)|_1 \tag{9}$$

$$\pi_{exp}(s) = \frac{\pi}{5} * \frac{g^*(s) - j(s)}{\max_i |g_i^*(s) - j_i(s)|_1} \tag{10}$$

Where the factor $\frac{\pi}{5}$ scales the action so that all joint angle deltas are in the range $\left[-\frac{\pi}{5}, \frac{\pi}{5}\right]$. We now have all the components required to execute eqs. (5) to (8) and generate the training data.

Rewards follow the equation:

$$R_t = -|a_{t-1}|_2 - d_t + c_1 \cdot succ_t - c_2 \cdot fail_t \tag{11}$$

Where d_t is the Cartesian distance of the gripper from the cup at time step t, $succ_t$ is 1 if the system is in a goal state and 0 otherwise, and $fail_t$ is 1 if the system is in a failure state and 0 otherwise. c_1 and c_2 are coefficients controlling the size of the success and failure rewards. We set $c_1 = 20$ and $c_2 = 50$ to prevent situations where it is more rewarding to fail quickly or to stop near the goal than to attempt to reach the goal itself.

Figure 4 compares the performance of models pre-trained with sparse VD and standard L_2 regularization, as well as

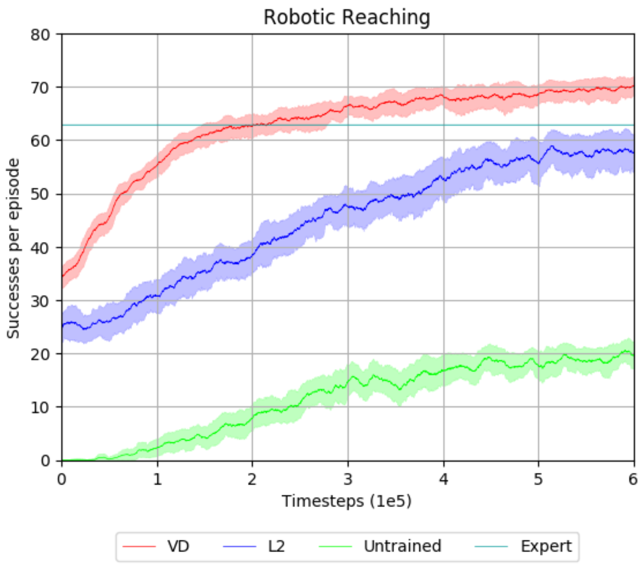

Fig. 4. Policy performance over time for policies different pre-training

a model that has no pre-training, and the inverse kinematics expert used to generate demonstrations for pre-training. The VD and L_2 models were trained on 8000 expert demonstrations with an expert noise $\Sigma = 0.5$. Performance is measured in the number of successful trajectories per episode, where episode length is 200 time steps. Sparse VD clearly outperforms the baselines by a significant margin, and at convergence even outperforms the IK expert. The sparse VD model overtakes the L_2 model's final performance at around $1.25e5$ time steps, meaning it requires less than one quarter as many samples. Figure 5 shows the effect of the training dataset's size on the performance of models in the RL stage. We consider datasets with 2000, 4000, and 8000 samples. Each sparse VD model outperforms its L_2 counterpart by a significant margin, demonstrating that variational dropout regularization provides an improvement independently of the amount of training data. Furthermore, even the worst-performing VD model, pre-trained with 2000 samples, shows comparable performance to the best L_2 model trained with 8000 samples. In other words, the use of variational dropout instead of L_2 regularization leads to a performance increase equivalent to quadrupling the size of the dataset used for pre-training. This gain is particularly valuable if acquiring expert demonstration for the training dataset is expensive or time-consuming. Figure 6 has a comparison between VD and L_2 models pre-trained on datasets with different levels of injected noise. Expert noise of magnitude $\Sigma = 0.05$, $\Sigma = 0.25$ and $\Sigma = 0.5$ was examined. For all noise levels, the trend lines for the VD models evince a quadratic shape at the beginning of training before tapering off. Of the L_2 models, only the trend line for noise with standard deviation of 0.05 shows this quadratic behavior, and the remaining models have a more linear shape. The implication of this is that VD models are much less dependent on having well-tuned noise in the expert demonstrations in order to

Experiment	Sparsity
8000 demonstrations; $\Sigma = 0.5$	0.677
4000 demonstrations; $\Sigma = 0.5$	0.526
2000 demonstrations; $\Sigma = 0.5$	0.359
8000 demonstrations; $\Sigma = 0.25$	0.685
8000 demonstrations; $\Sigma = 0.05$	0.766
Grasping	0.693

TABLE I

NEURON SPARSIFICATION RATIO FOR POLICIES TRAINED WITH VD IN DIFFERENT EXPERIMENTS.

achieve this convergence behavior. In table I we see the fraction of policy network neurons that were pruned by the sparse VD pre-training step, for each of the experiments. All experiments show a significant degree of sparsification, and in the most extreme case the number of weights is reduced to less than a quarter of the original architecture. Further, there is a trend of greater sparsification with an increasing number of demonstrations or with decreasing expert noise. In spite of this considerable reduction in network size, the VD models all either perform comparably or greatly outperform their L_2 counterparts.

B. Grasping Experiments

In the second set of experiments we use a grasping task that is similar to, but more difficult than, the reaching task of the previous section. The cup is now an interactable object that can cause collisions. The time horizon and action space remain unchanged from the previous task. Image observations have a resolution of 256x256, an increase found to be necessary in order to discern gripper orientation from the images. Training samples are generated in a manner similar to the reaching task, with some differences:

For each cup position, we compute a set G of joint configurations that place the end effector a small distance d_g from the cup and oriented towards the cup. In most states, the expert policy follows eqs. (9) to (10). When the gripper is within a distance of d_g from the cup and oriented towards it, inverse kinematics is used to compute a goal configuration that maintains the same orientation but places the gripper directly on the cup. If this is a valid configuration that results in no collision, we set it as $g^*(s)$ and follow eq. (10).

Rewards for the grasping task are sparse. a reward of -1 is given for each time step, a reward of -25 is given for a collision, and a reward of 25 is given for success. This ensures rewarding minimal length successful trajectories, and also avoids rewarding quick failure over longer trajectories. Figure 7 shows a comparison between a sparse VD and L_2 model for the grasping task. The VD model shows a clear improvement over L_2 regularization. However, after $3e5$ time steps both models still fall significantly short of the performance of an inverse kinematics expert, which achieves a total cumulative reward of almost 20. Neither model appears to have converged, and it is possible that both could match the expert's performance with sufficient training, but verifying this requires a prohibitive length of time. A model without pre-training was unable to learn a policy for this sparse-reward task, failing to achieve even a single successful trajectory.

VI. CONCLUSIONS

In this work we connected the fields of variational dropout and reinforcement learning through a behavioral cloning procedure with VD regularization that pre-trains an RL policy. Our results show that VD regularization provides significant gains in performance on high-dimensional continuous control tasks compared with the state of the art, converging more quickly and achieving higher performance. In a reaching task, the VD policy is even able to exceed the performance of an expert using inverse kinematics. The improvement is consistent across different quantities of training data, and in some tasks VD pre-training allows a considerable reduction in the number of expert demonstrations while still achieving comparable results. Further, results show that faster convergence during the RL stage is consistent across different levels of noise injected into the expert demonstrations. Finally, VD pre-training sparsifies the policies considerably, greatly reducing the computational cost of forward and backward passes.

While models trained with sparse VD are less dependent on having well tuned noise in the expert demonstrations in order to achieve fast convergence with a quadratic shape in the performance graph, they still suffers overall when they have poorly tuned noise. There is room for improvement by combining sparse VD pre-training with techniques for learning appropriate expert noise from the data. Further, the pre-training procedure trains only the mean function $\mu_\phi(s)$ of the policy, and leaves the SD function $\sigma_\psi(s)$ untouched. Since $\mu_\phi(s)$ is a dropout model, it inherently contains information about its own uncertainty. There is potential to use this uncertainty information to learn an appropriate SD function.

REFERENCES

[1] Alessandro Achille and Stefano Soatto. Information dropout: learning optimal representations through noise. *arXiv preprint arXiv:1611.01353*, 2016.

[2] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, 2013.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2013.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon

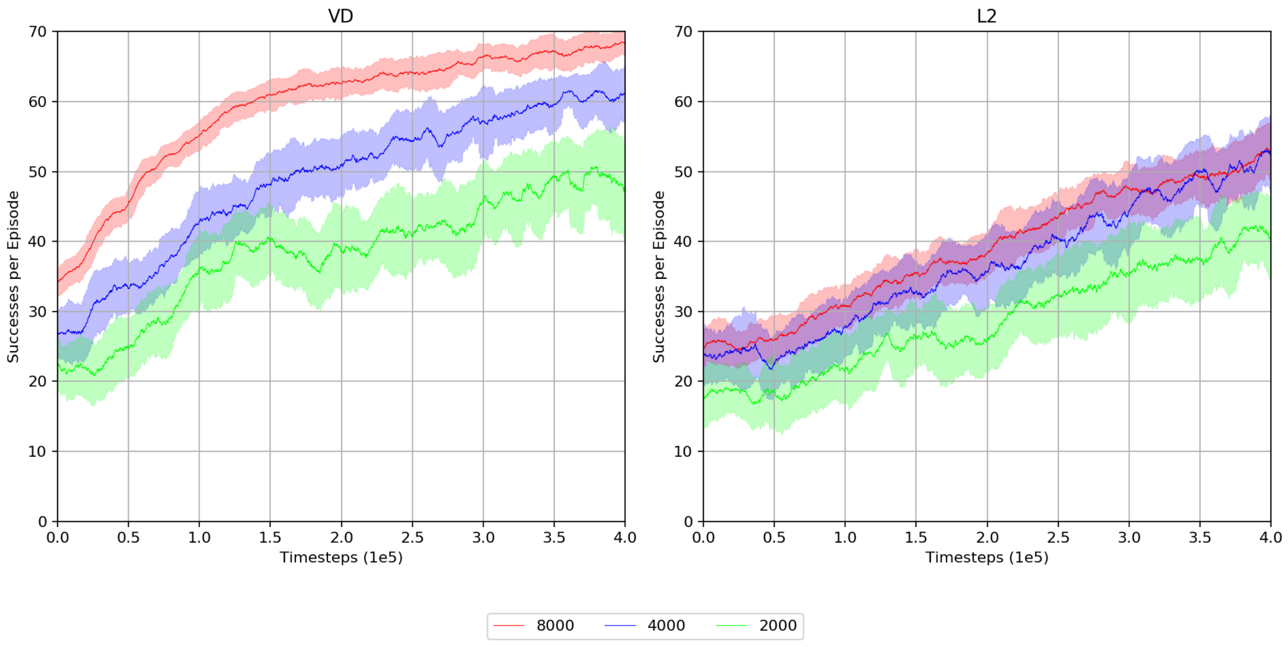

Fig. 5. Policy performance over time for policies pre-trained with training datasets of different sizes

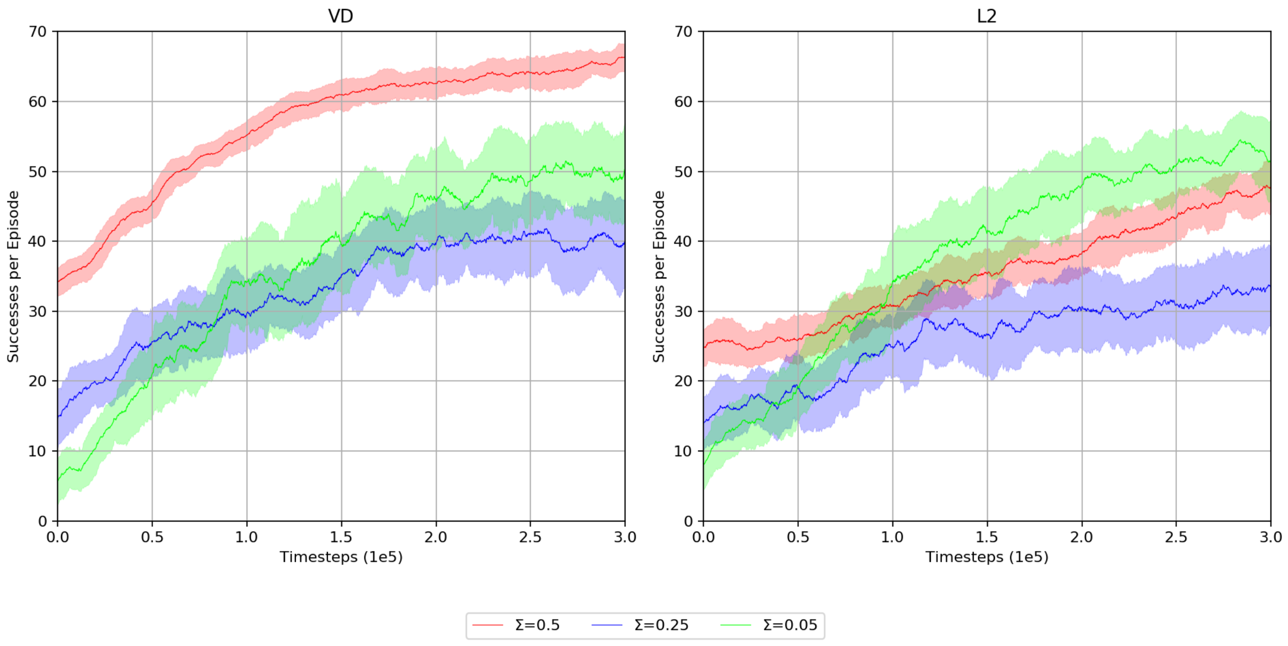

Fig. 6. Policy performance over time for policies pre-trained on datasets with different injected noise

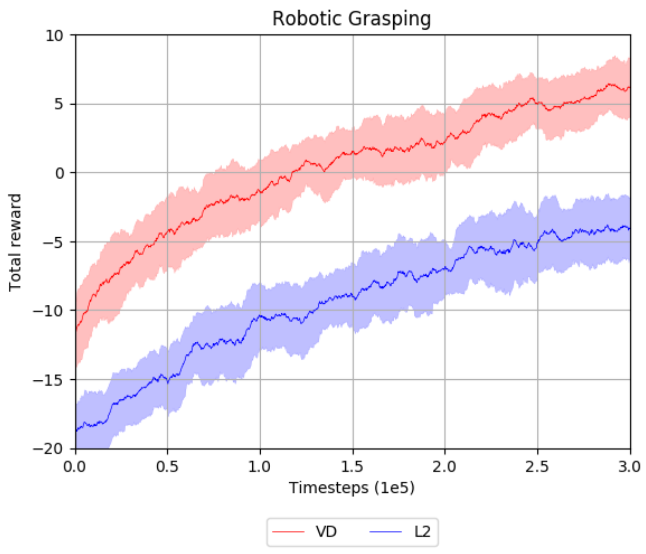

Fig. 7. Policy performance over time for a robotic grasping task

Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[6] Sander Dieleman, Jan Schlter, Colin Raffel, Eben Olson, Sren Kaae Snderby, Daniel Nouri, et al. Lasagne: First release., August 2015. URL `http://dx.doi.org/10.5281/zenodo.27878`.

[7] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *International Conference on Machine Learning*, 2016.

[8] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *NIPS*, 2017.

[9] M. Freese E. Rohmer, S. P. N. Singh. V-rep: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems*, 2013.

[10] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, 2016.

[11] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, 2017.

[12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[13] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NIPS*, 2016.

[14] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[15] Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on Robot Learning*, 2017.

[16] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *ArXiv e-prints*, 2016.

[17] 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*, 2015.

[18] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *International Conference on Machine Learning*, 2016.

[19] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, 2017.

[20] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 1991.

[21] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[22] Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[23] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *International Conference on Machine Learning*, 2015.

[24] Suraj Srinivas and R Venkatesh Babu. Generalized dropout. *arXiv preprint arXiv:1611.06791*, 2016.

[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 2014.

[26] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. TORCS The Open Racing Car Simulator. http://www.torcs.org, 2014.