Learning Torque-Driven Manipulation Primitives with a Multilayer Neural Network

Sergey Levine, Nolan Wagener, Pieter Abbeel
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94709
{svlevine, nolan.wagener, pabbeel}@eecs.berkeley.edu

Abstract

Autonomous learning of robotic motion skills promises to allow robots to acquire large motion repertoires for interacting with the real world with minimal human intervention. However, most current robotic motion skill representations are compact and lacking in expressiveness. This limitation is due in large part to the difficulty of learning rich representations in the context of optimal control and reinforcement learning. We discuss our recent experiments on using the guided policy search method to learn multilayer neural network policies for robotic manipulation skills. The key idea in this approach is to turn the task of policy search into a supervised learning problem, so that known, reliable optimization algorithms can be used to train expressive function approximators with thousands of parameters. While our networks are small compared to those used for perception (two hidden layers with 40 units each), they are significantly more expressive than most controller representations in reinforcement learning. We present results for neural network control of a PR2 robot, and discuss future work on learning policies that directly map rich sensory input to joint torques using deep neural networks.

1 Introduction

Autonomous learning of robotic motion skills promises to allow robots to acquire large motion repertoires with minimal human intervention. Such capability would be a major step toward allowing robots to engage in the wide array of tasks required, for example, for household robotics, as well as other domains where robots must deal with uncontrolled, real-world environments designed primarily for human use. Although significant progress has been made in this area in recent years [8, 3], autonomous acquisition of robot motion skills remains a major challenge. One of the critical issues in this area is the question of representation. While alternatives such as fitted Q-iteration can handle richer function approximators [7], they typically do not scale as gracefully with task dimensionality, making them difficult to use for directly controlling a high-dimensional robotic system. Policy search is often the method of choice for learning direct robot control [3]. Successful applications of policy search to motion skill learning typically rely on carefully hand-crafted representations that expose a small number of free parameters to the learning algorithm [9, 21, 5, 4, 19, 17]. One of the reasons for this is that, for high-dimensional robotics problems, most reinforcement learning algorithms struggle with high-dimensional policy representations.

The need to provide low-dimensional policy representations limits the expressiveness of the resulting motion skills, restricting them, for example, to trajectory following behaviors [19] that cannot represent complex decision making strategies. Several authors have attempted to learn more general neural network policies with evolutionary algorithms [16, 10], but such methods typically rely on encoding the network via a low-dimensional, "compressed" representation with a few hundred parameters, and cannot be trained directly on a physical robot due to their exceedingly high sample complexity. In this work, we present some recent experiments on applying a guided policy search algorithm to directly learn robotic manipulation skills with multilayer neural networks that map sensory input to joint torques. This more general representation allows for a wide range of strategies to be acquired. An even more exciting possibility for this approach is to eventually tackle the problem of learning policies that simultaneously perform both perception and control, by feeding rich sensory inputs directly into the policy. This is generally impossible with simple, trajectory-centric policy representations, but may become feasible once the policy is made sufficiently expressive, for example by using deep neural networks. We discuss this possibility further in Section 4.

The key idea behind our approach is to turn the problem of policy search into a supervised learning task, which in general is much easier to solve (especially with high-dimensional, expressive function approximators) than a reinforcement learning task. In guided policy search, a high-dimensional, nonlinear policy is learned in supervised fashion from a set of simpler, trajectory-centric policies trained under different conditions (different initial states, different goals, etc.) [13, 14, 15]. The final, high-dimensional policy is then able to generalize to new conditions. We extend our recent work on guided policy search under unknown dynamics [12] to make it practical to apply on a real robotic platform, and show results for several manipulation tasks. The policy directly sets the torques on the robot at each time step based on the current observations. While our current control policies are small compared to state-of-the-art architectures in deep learning (two hidden layers with 40 units each and around 3500 parameters), the method itself is straightforward to apply to networks of any size, and compatible with popular optimization algorithms such as stochastic gradient descent. In light of this, we discuss our ongoing and future work on applying this method to control policies that act directly on raw sensory input, such as camera images, depth readings, and force sensors.

2 Guided Policy Search for Robot Motion Skill Learning

A control policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ is a probability distribution over actions $\mathbf{u}_t$, which might be the torques at a robot's joints, conditioned on the state $\mathbf{x}_t$, which might include the state of the world as well as the robot. At every time step, the robot samples an action from the policy and executes it. The goal of a policy search algorithm is to find the parameters θ of a parameterized policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ that minimize the expectation of the cost under that policy, given by $E_{\pi_\theta}[\sum_{t=1}^T \ell(\mathbf{x}_t, \mathbf{u}_t)]$.

In guided policy search, a complex policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ is trained by using one or more simple, local policies $p(\mathbf{u}_t|\mathbf{x}_t)$. The policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ is typically nonlinear and has a high-dimensional parameterization, while the simple policies $p(\mathbf{u}_t|\mathbf{x}_t)$ are time-varying and trajectory-centric: that is, they encode a trajectory and simple (linear) feedback rules for tracking that trajectory. For this reason, we will refer to $p(\mathbf{u}_t|\mathbf{x}_t)$ as a trajectory distribution. The form of $p(\mathbf{u}_t|\mathbf{x}_t)$ is chosen to make it easy to train, while the form of $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ is chosen to maximize generalization. To train $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, a training set of state-action pairs is sampled from each $p(\mathbf{u}_t|\mathbf{x}_t)$ and used in a supervised learning procedure. The complex policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ never interacts with the environment during training, which might seem odd from a reinforcement learning perspective, but is in fact quite natural: the role of $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ is to extrapolate from learned trajectories to new situations, while the simple policies $p(\mathbf{u}_t|\mathbf{x}_t)$ are tasked with learning how to act in the current training situations.

Unfortunately, naïve supervised training of control policies is known to be ineffective [20], and the key to making this procedure converge to a successful policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ is to alternate between optimizing $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ to match each trajectory distribution, and reoptimizing the trajectory distributions to obtains low cost and minimize deviation from the current policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$. At convergence, the trajectories visit the same states as $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ would if starting from the same initial conditions, making them a suitable training set for the complex nonlinear policy.

In this work, we extend and apply our recent work on combining guided policy search with a method for optimizing trajectory distributions under unknown dynamics [12]. The ability to handle unknown dynamics makes this method suitable for learning real-world robotic motion skills. We summarize this approach in the following sections, and then present some technical improvements that make it suitable for learning robotic manipulation skills with multilayer neural network policies, before describing experimental results for learning neural network policies for several challenging manipulation skills on a PR2 robot.

2.1 Trajectory Optimization under Unknown Dynamics

In order to apply the guided policy search framework to robotic control problems where the dynamics are unknown, such as in the case of interaction with unfamiliar objects, we must design a method for optimizing the simple trajectory-centric controllers $p(\mathbf{u}_t|\mathbf{x}_t)$. We use linear-Gaussian controllers $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\bar{\mathbf{u}}_t + \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t), \Sigma)$, which provide for an especially efficient optimization procedure based on the linear-quadratic-Gaussian regulator (LQG). The LQG dynamic programming method is an efficient approach for computing the optimal (time-varying) linear feedback controller under (time-varying) linear-Gaussian dynamics, but requires the dynamics to be known. In practice, the dynamics are often obtained by differentiating through a simulator of the system. In the case of unknown dynamics, we can estimate the time-varying linear-Gaussian dynamics from data: we generate samples from the current controller $p(\mathbf{u}_t|\mathbf{x}_t)$ by running it on the real robot, and then fit linear-Gaussian dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}t}\mathbf{x}_t + f_{\mathbf{u}t}\mathbf{u}_t, \mathbf{F}_t)$ at each time step. We can then solve the LQG dynamic programming problem to obtain a new linear-Gaussian controller that is optimal under these dynamics.

The difficulty with this approach is that the new controller might deviate arbitrarily far from the old one, and the linear-Gaussian dynamics that were fitted to data from the old controller might no longer hold. In practice, this would cause the method to divergence instead of discovering an effective controller. To prevent this, we limit the change in the trajectory distribution under the new controller by means of a KL-divergence constraint. This corresponds to solving the following constrained problem to obtain the new linear-Gaussian controller:

$$\min_{p(\tau)\in\mathcal{N}(\tau)} E_p[\ell(\tau)] \text{ s.t. } D_{\text{KL}}(p(\tau)\|\hat{p}(\tau)) \leq \epsilon,$$

where $\hat{p}(\tau)$ is the trajectory distribution under the previous controller. This type of policy update has previously been proposed by several authors in the context of policy search [1, 19, 18]. However, the special structure of the linear-Gaussian controller and dynamics admits a particularly simple solution in our case, which consists simply of solving the same LQG dynamic programming problem with a modified cost to include the KL-divergence term. This approach is described in detail in our recent paper [12], along with a technique for reducing the number of samples required to estimate the dynamics. Our previous results show that, in simulation, each iteration of this procedure can use as few as 5 sample trajectories on systems with up to 18 state dimensions, which is sufficient for practical applications on a real robotic platform.

2.2 Learning Neural Network Policies with Guided Policy Search

The simple, trajectory-centric controllers $p(\mathbf{u}_t|\mathbf{x}_t)$ trained using the method in the previous section can be used to generate a training set for supervised training of the neural network policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$. However, in order to ensure that this training procedure actually results in a policy with a good expected cost $E_{\pi_\theta}[\ell(\tau)]$, we must also adapt $p(\mathbf{u}_t|\mathbf{x}_t)$ so as to produce trajectories that can actually be reproduced by the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$. Even if the neural network is very large and expressive, it cannot capture all trajectory distributions. For example, if the nonstationary controller $p(\mathbf{u}_t|\mathbf{x}_t)$ takes different actions in two very similar states and different time steps, the neural network would be unable to reproduce its behavior.

We formalize this idea of trajectory adaptation using the following constrained optimization problem, which we solve approximately using an alternating optimization procedure:

$$\min_{\theta,p(\tau)} E_{p(\tau)}[\ell(\tau)] \text{ s.t. } D_{\text{KL}}(p(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\|p(\mathbf{x}_t, \mathbf{u}_t)) = 0 \ \forall t.$$

This constrained problem aims to find controllers $p(\mathbf{u}_t|\mathbf{x}_t)$ that match the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ at the states visited by $p(\mathbf{u}_t|\mathbf{x}_t)$. If this match is perfect (i.e. the constraint is satisfied), the state distributions of $p(\mathbf{u}_t|\mathbf{x}_t)$ and $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ are identical, and samples from $p(\mathbf{u}_t|\mathbf{x}_t)$ are suitable for training $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$. To allow $p(\mathbf{u}_t|\mathbf{x}_t)$ the freedom to explore without being fixed to always be equal to $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, we relax this constraint and solve the constrained problem using dual gradient descent (DGD), which consists of alternating betwen optimizing the Lagrangian with respect to the primal variables (the controllers $p(\mathbf{u}_t|\mathbf{x}_t)$ and the policy parameters θ) and taking a subgradient step on the Lagrange multipliers, where the subgradient is given by the amount of constraint violation.

Algorithm 1 Guided policy search with unknown dynamics

```
1: for iteration $k = 1$ to $K$ do
2:     Generate samples $\{\tau_i^j\}$ from each controller $p_i(\mathbf{u}_t|\mathbf{x}_t)$ by executing it on the robot
3:     Fit the dynamics $p_i(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ to the samples $\{\tau_i^j\}$
4:     Minimize $\sum_{i,t} \lambda_{i,t} D_{\text{KL}}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\|p_i(\mathbf{x}_t, \mathbf{u}_t))$ with respect to $\theta$ using samples $\{\tau_i^j\}$
       using any supervised learning procedure (see [15] for details)
5:     Update $p_i(\mathbf{u}_t|\mathbf{x}_t)$ using the trajectory optimization algorithm in Section 2.1
6:     Increment dual variables $\lambda_{i,t}$ by $\alpha D_{\text{KL}}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\|p_i(\mathbf{x}_t, \mathbf{u}_t))$
7: end for
8: return optimized policy parameters $\theta$
```

The Lagrangian of this problem is given by

$$\mathcal{L}_{\text{GPS}}(\theta, p, \lambda) = E_{p(\tau)}[\ell(\tau)] + \sum_{t=1}^{T} \lambda_t D_{\text{KL}}(p(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\|p(\mathbf{x}_t, \mathbf{u}_t)).$$

The optimization with respect to the primal variables is itself done in alternating fashion. Optimizing the Lagrangian with respect to the policy parameters θ corresponds to minimizing the KL-divergence. When this minimization is done using samples from $p(\mathbf{u}_t|\mathbf{x}_t)$, it corresponds simply to supervised learning, and can be performed with any standard optimization method, such as LBFGS or SGD. The optimization with respect to the controllers $p(\mathbf{u}_t|\mathbf{x}_t)$ corresponds to trajectory optimization, with an additional cost term to account for the KL-divergence against the policy. This is done using the algorithm described in the previous section. The alternating optimization is done for only a few iterations before each dual variable update, rather than to convergence, which greatly speeds up the method. Pseudocode is provided in Algorithm 1.

2.3 Generating Synthetic Samples for Neural Network Training

When optimizing the simple controllers $p(\mathbf{u}_t|\mathbf{x}_t)$ on the robot, we must minimize the amount of system interaction time, since every sample on the robot runs in real time, causes wear and tear, and can require manually resetting the system in some tasks (such as when the robot knocks an object off the table). Our method is well suited to this task, because it requires a small number of samples compared to most reinforcement learning algorithms. As few as 20-30 samples are sufficient to train a single trajectory, and four trajectories are sufficient to train an effective neural network policy. Unfortunately, the goal of minimizing sample count is at odds with the requirements for training rich, expressive function approximators, which generally require large amounts of training data. When using only samples from the real system for training the neural network policy, we observed extensive overfitting.

To alleviate this issue, we note that the neural network is trained on state-action tuples generated from a linear-Gaussian controller, which itself is optimized under linear-Gaussian dynamics. This corresponds to a Gaussian distribution over trajectories $p(\tau) = \prod_t p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)p(\mathbf{u}_t|\mathbf{x}_t)$, where the first term corresponds is the dynamics, and the second is the controller. We can marginalize this distribution at every time step to obtain $p(\mathbf{x}_t, \mathbf{u}_t)$. Using these marginals, we can generate as many samples as needed without running additional trials on the robot, creating an effectively unlimited dataset for training the policy. Within a stochastic gradient descent (SGD) optimization for the policy, we could in principle generate entirely new samples at every iteration. For simplicity, we optimized our policies with LBFGS, using a fixed dataset consisting of 50 samples per time step. With 4 trajectories of 200 time steps each, this correponded to 40,000 training samples.

Note that, although the resulting dataset can be as large as necessary, it is restricted the region around the training trajectories, each of which corresponds to a different training condition, such as the position of the target for block placement (four trajectories were used in our experiments). The policy can therefore still overfit to the training conditions, though it will always acquire a stable policy that matches the simple controllers for each of the conditions. This second type of overfitting (see [11]) causes problems when we use the neural network policy to generalize to new situations, and we describe one possible solution in the following section.

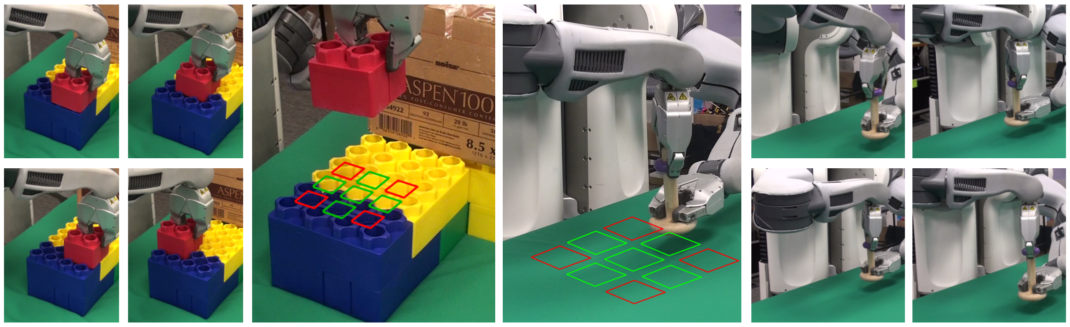
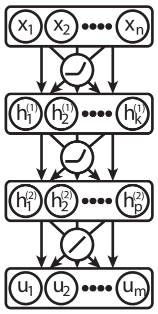


Figure 1: Objects used in our experiments: large lego blocks were stacked on a fixed base (left) and wooden rings were threaded onto a tight-fitting peg (right). The training positions are indicated in red, and the test positions are indicated in green. Videos of the results can be viewed on the project website at `http://rll.berkeley.edu/nips2014dlws/`

3 Experimental Results

We conducted a set of experiments to learn multilayer neural network controllers for two challenging robotic manipulation tasks: stacking large lego blocks and placing tight-fitting wooden rings onto pegs. These tasks are particularly challenging, since they involve objects with unknown physical properties and numerous collisions and contacts, which introduce discontinuities into the dynamics of the system. Images of these tasks are shown in Figure 1, and videos of the policies can be viewed at `http://rll.berkeley.edu/nips2014dlws/`.



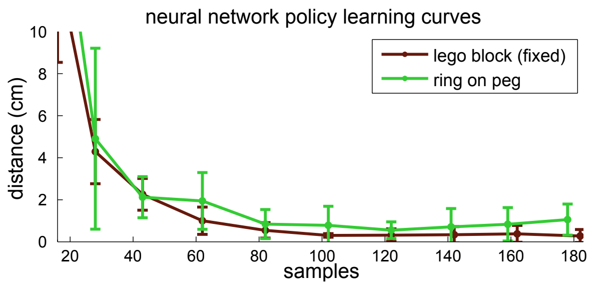
x_1 x_2 x_n
$h_1^{(1)}$ $h_2^{(1)}$ $h_k^{(1)}$
$h_1^{(2)}$ $h_2^{(2)}$ $h_p^{(2)}$
u_1 u_2 u_m

Both the trajectory-centric linear-Gaussian controllers and the final neural network policy directly commanded the torque on each of the seven joints of the robot's arm, and took as input the current joint angles and velocities, the velocities of two or three points on the manipulated object (two for the radially symmetric ring, three for the block), the vector from the target positions of these points to their current position, and the torque applied at the previous time step. The object was assumed to be rigidly attached to the end-effector, so that forward kinematics could be used to compute the object's current position. The network consisted of two hidden layers with 40 units each, with soft rectifying nonlinearities between the first two layers, of the form $a = \log(z + 1)$, and linear connections to the output, as shown on the right. Contrary to our prior experience (see, e.g., [11]), we found that a single-layer shallow architecture was unable to learn these tasks successfuly. A formal comparisons is left for future work.

The lego block policy was trained on four different target positions located at the corners of a square base of lego blocks (see Figure 1), and was then tested on all positions within that region. The ring policy was trained with the peg held in the robot's left gripper, which moved it between four positions, and then tested on five other positions. The training procedure required about 90 minutes per policy, of which only about 10 minutes were spent running the controllers on the robot, with the remainder split evenly between resetting to the initial state and computation time. Both the reset procedure and the MATLAB implementation could be made significantly faster.

Learning curves for the two policies are shown in Figure 2 in terms of the distance to the target relative to the total number of samples collected. Figure 2 further shows the success rate on the training and test positions for each policy. These results show that both policies were able to learn a generalizable motion skill that could be used to place the object at a variety of target positions.

On the ring task, the configuration of the robot differed significantly between the four training positions. This initially caused the neural network to overfit to the training positions, effectively learning a classifier for each of the targets, since the particular joint configuration was so strongly correlated with the target position. As a result, when given a new target, the network would simply go to the closest of the training locations. As discussed in Section 2.3, this type of overfitting cannot be solved by drawing more samples from the controllers: the policy overfit to the target positions, rather than to the samples, and no amount of extra samples for the same target positions would have helped.



lego block	training positions				
position	#1	#2	#3	#4	
success rate	5/5	5/5	3/5	4/5	
lego block	test positions				
position	#1	#2	#3	#4	#5
success rate	5/5	5/5	5/5	5/5	5/5

ring on peg	training positions				
position	#1	#2	#3	#4	
success rate	5/5	5/5	5/5	5/5	
ring on peg	test positions				
position	#1	#2	#3	#4	#5
success rate	5/5	5/5	5/5	4/5	5/5

Figure 2: Distance to target per iteration during neural network policy training. Note that the samples are divided across four training trajectories. Distance is measured for the trajectories only, since the neural network is not run on the robot during training. Performance of the trained networks was evaluated by testing the controller five times on each of the training and test positions. For lego blocks, the trial was considered a success if the block was placed at least halfway down on the teeth of the correct target block. On the ring and peg task, a success required that the ring slide down the peg when released (we did not release the ring each time, since the outcome was usually obvious).

To alleviate this issue, we introduced normally distributed noise with a standard deviation of 2 cm to the position of the peg. We were able to do this automatically, since the peg was already held in the robot's left gripper, which could automatically move the peg between every trial. Due to this noise, each of the linear-Gaussian controllers learned a policy that locally tracked the target position, and the network was able to observe that the target correlated more strongly with success than any particular configuration of joint angles. This allowed it to generalize to each of the test positions.

4 Discussion and Future Work

We presented a method for training neural network control policies for robotic motion skills with thousands of parameters. These policies directly control the torques on a robotic manipulator arm in response to the state of the system. The key idea in our approach is to transform the challenging task of reinforcement learning into a much simpler supervised learning problem, where standard optimization algorithms can be leveraged to optimize expressive function approximators such as large neural networks. To that end, a set of simple trajectory-centric controllers are trained on the real robot to serve as a training set for the final policy, and are then further optimized to conform to this policy and provide a better training set.

Since the neural network training is reduced to supervised learning, any standard algorithm can be used, including LBFGS and SGD. Furthermore, the input to the policy does not need to match the input to the simple trajectory-centric controllers, which are linear and nonstationary. For instance, the environment might be instrumented at training time with motion capture markers, with the positions fed to each of the linear controllers, but not provided to the neural network. The network would then be forced to do the task with only the inputs provided (such as camera images), and could execute it at test time in environments that are not instrumented. We experimented with this limited information setup in simulation in our previous work, where the neural network was forced to "search" for a slot position for an insertion task, which was provided to the controllers but not to the network [12]. In combination with visual inputs, this approach could be used to learn visual servoing policies that combine perception and action. Such joint learning would be able to automatically discover policies that, for instance, minimize perceptual errors along dimensions that are most significant for the task.

An important challenge in applying this method to policies with richer inputs is the size of the training set. In our current work, we augment the training set by sampling synthetic states and computing the corresponding actions that would be taken in those states in the trajectory distributions. However, when the policy is provided with rich sensory inputs such as camera images, sampling good camera images synthetically may be impractical. Techniques such as pretraining [2] and transfer learning [6] may become crucial in this case, and constructing an appropriate training set for pretraining a sensorimotor policy is an important direction for future work.

In the long run, simultaneous learning of perception and control networks can lead to improvements in both robot motor control and perception, since the control side of the policy can adapt to the limitations of perception, while the perception side can adapt itself in a goal-centric way, learning to represent those aspects of the world that are most relevant to the task.

Acknowledgments

This research was partly funded by a DARPA Young Faculty Award #D13AP0046 and the United States Army Research Laboratory (ARL) under the Micro-Autonomous Science and Technology Collaborative Technology (MAST) Alliance.

References

[1] J. A. Bagnell and J. Schneider. Covariant policy search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[2] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[3] M. Deisenroth, G. Neumann, and J. Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

[4] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng. Learning CPG-based biped locomotion with a policy gradient method: Application to a humanoid robot. *International Journal of Robotic Research*, 27(2):213–228, 2008.

[5] T. Geng, B. Porr, and F. Wörgötter. Fast biped walking with a reflexive controller and realtime policy searching. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[6] X. Glorot, A. Bordes, and Y. Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *International Conference on Machine Learning (ICML)*, 2011.

[7] R. Hafner and M. Riedmiller. Neural reinforcement learning controllers for a real robot application. In *International Conference on Robotics and Automation (ICRA)*, 2007.

[8] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotic Research*, 32(11):1238–1274, 2013.

[9] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *International Conference on Robotics and Automation (IROS)*, 2004.

[10] J. Koutnik, G. Cuccu, J. Schmidhuber, and F. J. Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *GECCO*, 2013.

[11] S. Levine. Exploring deep and recurrent architectures for optimal control. In *NIPS 2013 Workshop on Deep Learning*, 2013.

[12] S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[13] S. Levine and V. Koltun. Guided policy search. In *International Conference on Machine Learning (ICML)*, 2013.

[14] S. Levine and V. Koltun. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[15] S. Levine and V. Koltun. Learning complex neural network policies with trajectory optimization. In *International Conference on Machine Learning (ICML)*, 2014.

[16] G. Morse, S. Risi, C. R. Snyder, and K. O. Stanley. Single-unit pattern generators for quadruped locomotion. In *GECCO*, 2013.

[17] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *International Conference on Robotics and Automation (ICRA)*, 2009.

[18] J. Peters, K. Mülling, and Y. Altün. Relative entropy policy search. In *AAAI Conference on Artificial Intelligence*, 2010.

[19] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

[20] S. Ross, G. Gordon, and A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *Journal of Machine Learning Research*, 15:627–635, 2011.

[21] R. Tedrake, T. Zhang, and H. Seung. Stochastic policy gradient reinforcement learning on a simple 3d biped. In *International Conference on Intelligent Robots and Systems (IROS)*, 2004.